



ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA EN TELEMÁTICA

**TRABAJO FIN DE GRADO**

VISUALIZACIÓN EN REALIDAD VIRTUAL DE DATOS  
AERONÁUTICOS CON CONTEXTO GEOESPACIAL

Autor : Víctor Jesús Temprano Hernández

Tutor : Dr. Jesús M. González Barahona

Curso académico 2022/2023

©2023 Víctor Jesús Temprano Hernández

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución-CompartirIgual 4.0 Internacional” de Creative Commons,

disponible en

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

# Dedicatoria

En honor a mi padre, aunque no estés entre nosotros, siempre estarás presente en cada logro que alcanzo. Se que te sentirás orgulloso de mí, siempre te llevaré presente en mi corazón. Tu memoria y tu legado vivirá eternamente en mí.



# Agradecimientos

En primer lugar quiero expresar mi profundo agradecimiento a mi mujer por acompañarme en cada uno de mis desafíos, y especialmente en este, donde ha asumido el papel de padre y madre para permitirme enfrentar este reto. Este trabajo no habría sido posible sin tu apoyo incondicional.

También quiero dar las gracias a mi madre por recordarme que la vida es impredecible y que siempre es mejor tarde que nunca.

Y por último a mi tutor Jesús María González Barahona, por adaptarse a mis horarios caóticos y comprender mis necesidades. Su flexibilidad ha sido fundamental en el desarrollo de este proyecto.



# Resumen

Voy a desarrollar un conjunto de herramientas que simplifiquen la creación de escenarios 3D para la representación y visualización de datos relacionados con la aviación. Estas herramientas permitirán crear aplicaciones que muestren información en tiempo real o datos almacenados en el dispositivo para una visualización sin conexión.

Además, estas herramientas proporcionarán al usuario una experiencia inmersiva mediante el uso de tecnologías de realidad virtual. Esto permitirá al usuario sumergirse en el escenario y comprender de manera intuitiva el estado del espacio aéreo. Los usuarios podrán desplazarse a través del escenario sobre el terreno utilizando dispositivos de realidad virtual y explorar los datos de manera interactiva. Todo esto se logrará mediante una interfaz de usuario que integre un conjunto de componentes diseñados para la visualización de la información. Estos componentes formarán parte del escenario como elementos adicionales, ofreciendo una visualización completa de los datos en un entorno contextualizado.

Crearé un prototipo final que mostrará el conjunto de herramientas y componentes desarrollados. Este prototipo proporcionará un escenario cercano a un aeropuerto, donde el usuario podrá sumergirse en la escena y consultar datos de aviación en tiempo real. Además, el prototipo incluirá la representación de edificios y un terreno con alturas definidas, que se visualizará utilizando la textura de un mapa satelital de la zona seleccionada. Esto permitirá brindar mayor realismo a la escena y una experiencia inmersiva al usuario.





# Summary

The purpose of this project is to implement a solution based on virtual reality technologies that allows the user to immerse themselves in a scene and manipulate it to visualize information related to aeronautical and geospatial data.

By utilizing virtual reality technologies and leveraging aeronautical and geospatial data sources, the aim is to develop an immersive experience in a virtual environment that provides users with the ability to interactively and visually explore aviation and geographical information in an intuitive and engaging manner. This environment will represent a portion of the real world and include features such as flight path visualization and real-time aircraft position display, as well as the visualization and retrieval of details about buildings, terrain, and other relevant elements.



# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	4
1.2. Contexto . . . . .	4
1.3. Objetivo general . . . . .	7
1.4. Objetivos específicos . . . . .	7
1.4.1. Objetivos funcionales . . . . .	7
1.4.2. Objetivos técnicos . . . . .	8
1.5. Distribución del software . . . . .	8
1.6. Tecnologías Similares . . . . .	9
<b>2. Tecnologías relacionadas</b>	<b>11</b>
2.1. Sistema de Vigilancia Dependiente Automática (ADS-B) . . . . .	11
2.1.1. Introducción a la tecnología . . . . .	11
2.1.2. Historia y evolución . . . . .	12
2.1.3. Descripción técnica . . . . .	13
2.1.4. Contexto y aplicación en el prototipo . . . . .	14
2.2. Opensky-network API . . . . .	15
2.2.1. Introducción a la tecnología . . . . .	15
2.2.2. Historia y evolución . . . . .	15
2.2.3. Descripción técnica . . . . .	15
2.2.4. Contexto y aplicación en el prototipo . . . . .	15
2.2.5. Ejemplos de uso . . . . .	16
2.3. HTML5 . . . . .	17
2.3.1. Introducción a la tecnología . . . . .	17
2.3.2. Historia y evolución . . . . .	17

2.3.3.	Descripción técnica . . . . .	18
2.3.4.	Contexto y aplicación en el prototipo . . . . .	18
2.4.	DOM . . . . .	19
2.4.1.	Introducción a la tecnología . . . . .	19
2.4.2.	Historia y evolución . . . . .	19
2.4.3.	Descripción técnica . . . . .	20
2.4.4.	Contexto y aplicación en el prototipo . . . . .	20
2.4.5.	Ejemplos de uso . . . . .	20
2.5.	JavaScript . . . . .	21
2.5.1.	Introducción a la tecnología . . . . .	21
2.5.2.	Historia y evolución . . . . .	21
2.5.3.	Contexto y aplicación en el prototipo . . . . .	22
2.5.4.	Ejemplos de uso . . . . .	22
2.6.	WebGL . . . . .	22
2.6.1.	Introducción a la tecnología . . . . .	22
2.6.2.	Contexto y aplicación en el prototipo . . . . .	23
2.7.	WebXR . . . . .	23
2.8.	VR . . . . .	23
2.8.1.	Contexto y aplicación en el prototipo . . . . .	24
2.9.	Node JS . . . . .	24
2.9.1.	Introducción a la tecnología . . . . .	24
2.9.2.	Contexto y aplicación en el prototipo . . . . .	24
2.9.3.	Ejemplos de uso . . . . .	24
2.10.	Threejs . . . . .	25
2.10.1.	Introducción a la tecnología . . . . .	25
2.10.2.	Historia y evolución . . . . .	25
2.10.3.	Descripción técnica . . . . .	26
2.10.4.	Contexto y aplicación en el prototipo . . . . .	26
2.10.5.	Ejemplos de uso . . . . .	27
2.11.	A-Frame . . . . .	27
2.11.1.	Introducción a la tecnología . . . . .	27
2.11.2.	Historia y evolución . . . . .	28
2.11.3.	Descripción técnica . . . . .	28

2.11.4. Contexto y aplicación en el prototipo . . . . .	29
2.11.5. Ejemplos de uso . . . . .	29
2.12. Leaflet . . . . .	30
2.12.1. Introducción a la tecnología . . . . .	30
2.12.2. Contexto y aplicación en el prototipo . . . . .	30
2.12.3. Ejemplos de uso . . . . .	30
2.13. GDAL . . . . .	31
2.13.1. Introducción a la tecnología . . . . .	31
2.13.2. Contexto y aplicación en el prototipo . . . . .	31
2.14. Google Earth Engine . . . . .	31
2.14.1. Introducción a la tecnología . . . . .	31
2.14.2. Contexto y aplicación en el prototipo . . . . .	32
2.15. Overpass-api . . . . .	32
2.15.1. Introducción a la tecnología . . . . .	32
2.15.2. Contexto y aplicación en el prototipo . . . . .	32
2.16. GitHub . . . . .	33
2.16.1. Introducción a la tecnología . . . . .	33
2.16.2. Contexto y aplicación en el prototipo . . . . .	33
2.17. VSCode . . . . .	33
<b>3. Resultados</b>	<b>35</b>
3.1. Prototipos de demostración . . . . .	35
3.1.1. Escritorio . . . . .	35
3.1.2. Gafas de realidad virtual . . . . .	35
3.1.3. Móvil . . . . .	35
3.2. Progreso de desarrollo . . . . .	35
3.2.1. Sprint 0 . . . . .	37
3.2.1.1. Objetivos . . . . .	37
3.2.1.2. Desarrollo . . . . .	37
3.2.1.3. Resultado . . . . .	38
3.2.2. Sprint 1 . . . . .	38
3.2.2.1. Objetivos . . . . .	38
3.2.2.2. Desarrollo . . . . .	39

3.2.2.3.	Resultado	40
3.2.3.	Sprint 2	40
3.2.3.1.	Objetivos	40
3.2.3.2.	Desarrollo	41
3.2.3.3.	Resultado	42
3.2.4.	Sprint 3	42
3.2.4.1.	Objetivos	42
3.2.4.2.	Desarrollo	43
3.2.4.3.	Resultado	45
3.3.	Construcción de escenas	46
3.3.1.	Generación de datos para el terreno	46
3.3.2.	Generación de edificios para el terreno	46
3.3.3.	Almacenar metadatos de vuelos dentro del escenario (Opcional)	47
3.3.4.	Generación de fichero de configuración	47
3.3.5.	Creación de la página principal de nuestra aplicación	48
3.4.	Componentes reutilizables	50
3.4.1.	Componente de Información Contextual Interactiva	50
3.4.1.1.	ruta	50
3.4.1.2.	Guía del programador	51
3.4.1.3.	Ejemplo de uso	51
3.4.2.	Componente arrastrar entidades HUD	51
3.4.2.1.	ruta	51
3.4.2.2.	Guía del programador	51
3.4.2.3.	Ejemplo de uso	51
3.4.3.	Componente barra de herramientas 3D	52
3.4.3.1.	ruta	52
3.4.3.2.	Guía del programador	52
3.4.3.3.	Ejemplo de uso	53
3.4.4.	Geometría edificio	53
3.4.4.1.	ruta	54
3.4.4.2.	Guía del programador	54
3.4.4.3.	Ejemplo de uso	54

<b>4. Diseño e implementación</b>	<b>55</b>
4.1. Arquitectura general de la aplicación	55
4.2. Gestor de la escena principal	56
4.2.1. Componente main-scene	57
4.2.2. Movimiento fluido de aviones dentro de la escena	58
4.3. Gestión de la configuración de la aplicación	61
4.4. Sistema de información geográfica	62
4.4.1. Gestor de conversiones del mapa	62
4.4.1.1. De WGS84 a Mercator	69
4.4.1.2. De Mercator a WGS84	69
4.4.1.3. De MatorToWorld	69
4.4.1.4. De mundo 3D a Mercator	69
4.4.1.5. De WGS84 a mundo 3D	70
4.4.1.6. De mundo 3D a WGS84	70
4.4.1.7. Devuelve el tamaño del terreno	71
4.4.1.8. Crear entidades en los corner del terreno	71
4.5. Terreno en el mapa	72
4.5.1. Generación del fichero DEM	73
4.5.2. Generación de la capa de textura raster	75
4.5.3. Gestor de alturas	78
4.5.3.1. Calculo de altura para una coordenada 3D	79
4.5.4. Generación de datos geoespaciales de los edificios	81
4.5.5. Geometria edificio	82
4.5.6. Componente gestor de altura de cámara	83
4.6. Gestión de acceso a los datos	85
4.6.1. Datos caché de vuelo	85
4.6.2. Objeto de acceso a datos ADS-B (DAO)	86
4.6.3. Consulta y almacenamiento de datos a la carpeta caché	87
4.7. Gestión de la interfaz de usuario	89
4.7.1. Entidades en ampliación al alejarse	89
4.7.2. Componente de información contextual interactiva	90
4.7.3. Componente barra de herramientas de la interfaz de usuario	91
4.7.3.1. Animación de la barra de herramientas	93

4.7.4. Componente pantalla frontal de visualización (HUD) . . . . .	93
4.7.5. Desplazamiento de la barra de herramientas y pantalla frontal de visualización (HUD) . . . . .	95
4.7.6. Componente trayecto realizado por un vuelo . . . . .	98
<b>5. Conclusiones</b>	<b>99</b>
5.1. Consecución de objetivos . . . . .	99
5.2. Aplicación de lo aprendido . . . . .	100
5.2.1. A través de la titulación . . . . .	100
5.2.2. De forma autodidacta . . . . .	100
5.3. Trabajos futuros . . . . .	102
5.3.1. Evolutivos sencillos . . . . .	102
5.3.2. Evolutivos complejos . . . . .	103
<b>Bibliografía</b>	<b>105</b>



# Índice de figuras

1.1. Mapa de tecnologías . . . . .	3
2.1. Ejemplo de tipos de transpondedor ADS-B. . . . .	14
2.2. Petición a la API de OpenSky. . . . .	16
2.3. Escena principal del <i>Getting Started</i> de A-FRAME. . . . .	30
3.1. Planificación temporal del desarrollo del prototipo. . . . .	36
3.2. Versión inicial. . . . .	40
3.3. Evento clave que emite el componente de terreno. . . . .	41
4.1. Estructura de ficheros JAVASCRIPT. . . . .	56
4.2. Arquitectura general de la aplicación. . . . .	57
4.3. Animación para movimiento fluido interpolando posiciones de manera lineal. . . . .	59
4.4. Animación A-Frame movimiento fluido. . . . .	60
4.5. Configuración del escenario de Madrid. . . . .	61
4.6. Carga del fichero del módulo gestor de configuración y el fichero que configura el escenario de Madrid. . . . .	61
4.7. Coordenadas geográficas latitud y longitud. . . . .	62
4.8. Proyección cilíndrica Mercator. . . . .	63
4.9. Regla de la mano derecha. . . . .	64
4.10. Vista aérea de los sistemas de referencia. . . . .	65
4.11. Cálculo de dimensiones para el escenario Vatry. . . . .	66
4.12. Resultado del script. . . . .	66
4.13. Calculo del desplazamiento del escenario. . . . .	68
4.14. Calculo de la conversión de coordenadas cartesianas en MERCATOR a mundo 3D. . . . .	69
4.15. Calculo de la conversión vector 3D a coordenadas cartesianas Mercator. . . . .	70

4.16. Representación del contenido de un fichero DEM. . . . .	72
4.17. Visor online de ficheros DEM Copernicus. . . . .	74
4.18. Fichero de alturas smallMap.png visto como imagen en blanco y negro. . . . .	75
4.19. Código Javascript para la generación de la capa raster de textura de Madrid. . . . .	77
4.20. Raster de Madrid generado con Google Earth Engine. . . . .	77
4.21. Resultado final del componente con la textura de Madrid y el fichero DEM generado. . .	78
4.22. Calculo de conversión de coordenadas 3D al índice del array de alturas. . . . .	79
4.23. Diagrama de matriz de alturas. . . . .	80
4.24. Edificios extruidos sobre el terreno. . . . .	83
4.25. Código que configura el personaje principal, compuesto por la cámara y las manos. . . .	84
4.26. Diagrama del objeto de acceso a datos ADS-B (DAO). . . . .	87
4.27. Diagrama de consulta y almacenamiento de datos ADS-B en caché. . . . .	88
4.28. Ejecución con Node.js del almacenamiento de datos ADS-B en caché. . . . .	88
4.29. Calculo del factor de ampliación en función de la distancia. . . . .	89
4.30. Aviones sin ampliación. . . . .	90
4.31. Componente de ampliación. . . . .	90
4.32. Componente información contextual mostrando metadatos de la terminal. . . . .	91
4.33. Barra de herramientas desplegada. . . . .	92
4.34. Barra de herramientas plegada. . . . .	92
4.35. Maquetación del componente barra de herramientas. . . . .	92
4.36. función que genera los botón dentro de la barra de herramientas. . . . .	92
4.37. Animaciones para plegar la barra de herramientas. . . . .	93
4.38. Componente HUD con avión seleccionado. . . . .	94
4.39. Trayecto del avión en el escenario. . . . .	95
4.40. Visualización de la cámara de a bordo sobre el HUD. . . . .	95
4.41. Jerarquía de entidades. . . . .	96
4.42. Cálculo del vector desplazamiento de arrastre de entidades. . . . .	97

# Capítulo 1

## Introducción

El proyecto se enfoca en la creación de un conjunto de herramientas que proporcionen una plataforma para la visualización de datos aéreos en un entorno tridimensional, brindando al usuario una experiencia inmersiva.

Gracias al avance significativo de los dispositivos de realidad virtual y la disponibilidad de librerías de desarrollo que facilitan la creación de aplicaciones 3D, se creará una plataforma que ofrezca funcionalidades de visualización de datos aéreos en un entorno geoespacial 3D. Este entorno estará rodeado de entidades que crean un contexto que proporciona una experiencia más realista para el usuario.

Estas herramientas estarán dirigidas a un entorno web, con el objetivo de ser compatibles tanto en dispositivos móviles como en entornos de escritorio o en dispositivos de realidad virtual.

Los datos con los que voy a trabajar, procederán de servicios web gratuitos y podrán ser obtenidos en tiempo real o desde una caché local. Estos datos nos proporcionaran información sobre los vuelos y también información sobre las alturas del terreno y edificios presentes en la escena tridimensional.

Para lograr este propósito, construiré todo el prototipo utilizando el [Framework](#) de desarrollo llamado A-FRAME<sup>1</sup>. Este *framework* proporciona una capa de abstracción fácil de usar y comprender, lo cual facilita una implementación más sencilla de conceptos más complejos como la creación de escenas en realidad virtual y la manipulación e interacción de objetos 3D. Estos aspectos se analizarán con mayor detalle en secciones posteriores.

La razón principal de usar la tecnología A-FRAME es centrar nuestro esfuerzo en la codificación del modelo negocio acelerando el proceso de desarrollo de la gestión y generación de modelos 3D, que es un proceso complejo y laborioso.

Es importante destacar que la tecnología A-FRAME se basa en HTML5 para crear un ecosistema de componentes y geometrías utilizando elementos HTML. Esta tecnología permite aprovechar la poten-

---

<sup>1</sup><https://aframe.io/>

cia y flexibilidad de la librería THREE.JS a través de un lenguaje de marcado simplificado, acelerando así el desarrollo de aplicaciones 3D. A-FRAME facilita la creación de escenarios tridimensionales al proporcionar una forma más intuitiva y rápida de trabajar con THREE.JS.

A su vez, THREE.JS se construye sobre [WEBGL](https://www.khronos.org/webgl/)<sup>2</sup>, una API de gráficos en 3D basada en OPENGGL. Esto permite a los desarrolladores web acceder directamente a la capacidad de procesamiento de gráficos de la tarjeta gráfica del dispositivo mediante el uso de *JavaScript* para controlar la [Renderización](#) y la manipulación de los objetos 3D.

Esta arquitectura nos ofrece la ventaja de poder trabajar en un nivel de abstracción inferior cuando necesitamos abordar tareas más complejas relacionadas con objetos 3D. Además, podemos crear componentes reutilizables que serán útiles tanto en el proyecto actual como en proyectos futuros basados en la tecnología A-FRAME.

En resumen, la combinación de HTML5, A-FRAME, THREE.JS y WEBGL nos brinda un conjunto de herramientas poderosas para el desarrollo de experiencias 3D en entornos web, permitiéndonos aprovechar al máximo el potencial de la tarjeta gráfica del dispositivo y facilitando la reutilización de componentes en futuros proyectos.

A continuación en la figura [1.1](#), muestro un mapa donde se puede visualizar como se relacionan las tecnologías y los servicios de datos principales en las que se basará el prototipo que voy a realizar.

---

<sup>2</sup><https://www.khronos.org/webgl/>

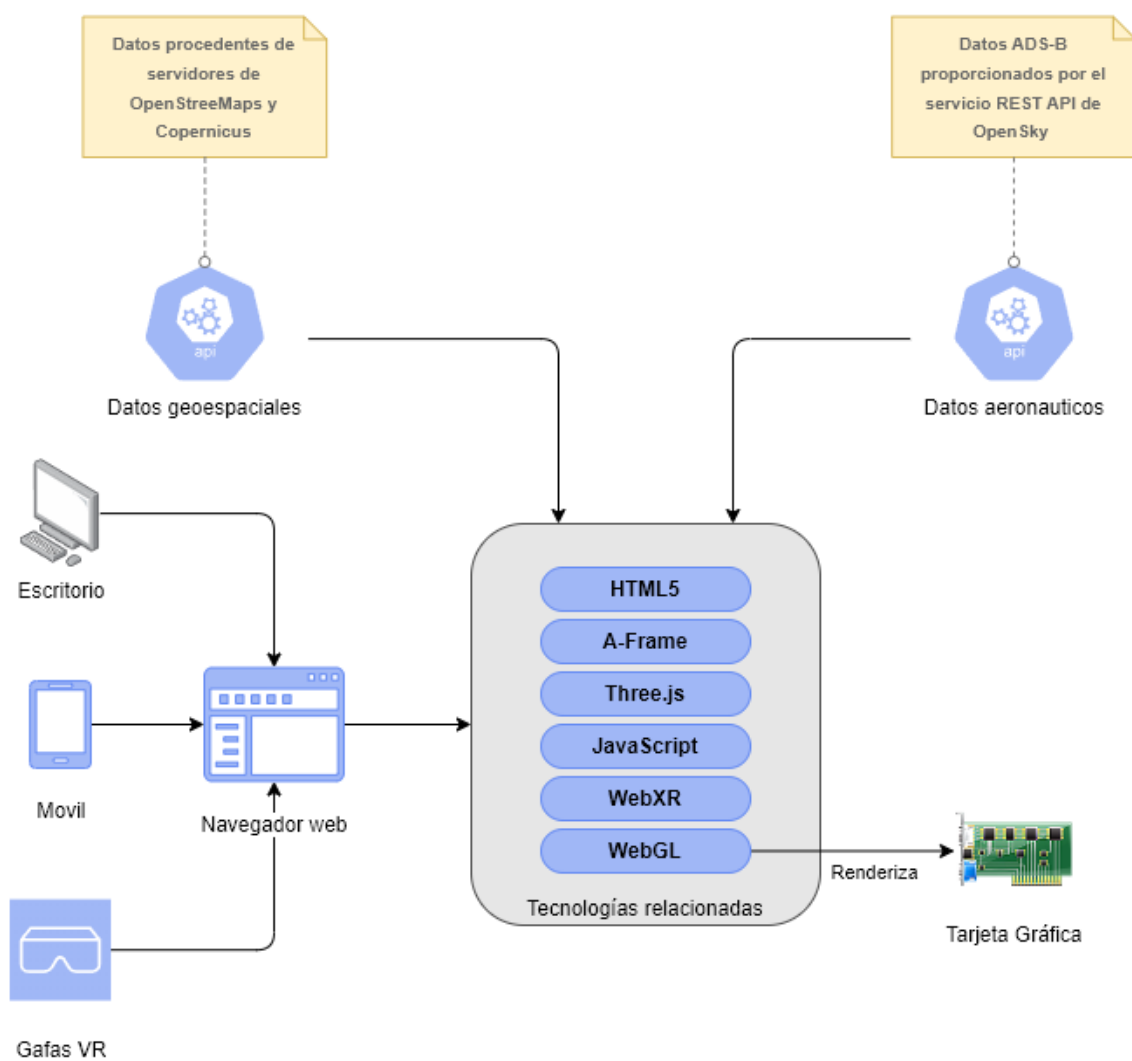


Figura 1.1: Mapa de tecnologías

## 1.1. Motivación

Después de trabajar durante seis años en el proyecto *fullstack* GIS llamado IGEA de la compañía INDRA, tuve la oportunidad de asumir la responsabilidad de integrar el núcleo del proyecto software IGEA<sup>3</sup> en una nueva aplicación de software para el proyecto iTEC, el cual se enfoca en la visualización y gestión de espacios aéreos europeos. Este desafío representó un hito muy interesante en mi carrera, ya que me permitió salir de mi zona de confort y experimentar un crecimiento de conocimientos sobre entornos de programación en 3D sobre la tecnología JAVA.

El objetivo principal de esta integración fue reutilizar los componentes centrales de IGEA para desarrollar una interfaz gráfica que facilitara a los usuarios administrativos la visualización y manipulación de datos aéreos de manera más intuitiva. Además, se aprovecharon las herramientas gráficas desarrolladas a lo largo de los más de veinte años de experiencia del proyecto IGEA para gestionar de manera efectiva los datos. Este enfoque permitió crear una solución poderosa y eficiente en poco tiempo para el proyecto iTEC.

Desde el momento en que comencé a buscar un proyecto para la finalización de mis estudios y comencé a analizar las posibilidades que me proporcionaba un proyecto basado en una aplicación 3D, me di cuenta de la interesante oportunidad de combinar mi experiencia laboral previa con las demandas de este proyecto. Explorando las posibilidades se me ocurrió la idea de visualizar información aeroespacial en un entorno de realidad virtual, ya que la representación bidimensional de datos aéreos que proporcionan la mayoría de las aplicaciones web actuales resulta limitada y la dimensión de altura es fundamental para la gestión de espacios aéreos. Además, la posibilidad de formar parte activa de la escena añade un valor significativo a las aplicaciones de consultas de datos. En este sentido, el futuro de estas aplicaciones parece prometedor.

## 1.2. Contexto

Durante la última década, la recopilación y transmisión de datos en tiempo real se han vuelto más accesibles gracias a la fusión de las tecnologías web junto a las tecnologías de vigilancia aérea, lo que ha creado nuevas oportunidades para el desarrollo de aplicaciones y servicios web que permiten a los usuarios visualizar el tráfico aéreo y los datos de vuelo en tiempo real.

Uno de los sistemas que ha experimentado un gran desarrollo y éxito en el ámbito aeronáutico es el sistema de vigilancia automática dependiente de difusión ADS-B 2.1. Este sistema permite que las aeronaves transmitan información precisa de sus sensores, como la posición, velocidad, altitud, orienta-

---

<sup>3</sup>[https://www.indracompany.com/sites/default/files/indra\\_in\\_grid\\_esp\\_baja.pdf](https://www.indracompany.com/sites/default/files/indra_in_grid_esp_baja.pdf)

ción, entre otros datos relevantes. El crecimiento de esta tecnología ha llevado a empresas y plataformas como FlightRadar24 y OpenSky a expandir sus redes y ofrecer servicios, incluso gratuitos, que proporcionan estos metadatos a través de la web. Esto ha impulsado el aumento de aplicaciones que muestran información de vuelos en tiempo real mediante mapas interactivos.

FlightRadar24, una empresa destacada en este campo, ofrece servicios web que muestran información en tiempo real sobre vuelos en un mapa bidimensional. Además del seguimiento de vuelos, proporciona detalles como el origen y destino de los vuelos, números de vuelo, tipos de aviones, altitudes, rumbos y velocidades.

OpenSky<sup>2.2</sup>, por su parte, es otra plataforma relevante que recopila y proporciona datos de vuelo en tiempo real. Además de mostrar información sobre los vuelos, OpenSky ha contribuido significativamente a la investigación en el ámbito de la aviación al proporcionar acceso a su base de datos para investigadores y desarrolladores interesados en analizar y utilizar datos aeroespaciales.

El avance en la recopilación y transmisión de datos en tiempo real, junto con el crecimiento de los sistemas ADS-B, ha sentado las bases para el desarrollo de aplicaciones y servicios web que permiten la visualización de información aeronáutica en tiempo real en dos dimensiones.

Sin embargo, el panorama se ha ampliado aún más con el aumento en los últimos años de servicios de sistemas de información geográfica para datos espaciales, como el programa COPERNICUS y los servicios ofrecidos por GOOGLE a través de su API GOOGLE EARTH ENGINE. Estos servicios nos brindan acceso a una amplia variedad de mapas y datos geoespaciales que podemos utilizar para crear escenas más realistas y contextualizadas.

Al aprovechar los datos proporcionados por estos servicios, podemos mejorar la visualización de información aeronáutica al agregar capas adicionales, como imágenes satelitales y datos de relieve. Esto nos permite crear escenarios más ricos y detallados que ofrecen una representación más precisa y completa de la información aeronáutica en tiempo real.

La migración de funcionalidades de las aplicaciones actuales en dos dimensiones hacia aplicaciones 3D inmersivas en realidad virtual tiene el potencial de llevar la experiencia de visualización de datos aeroespaciales a un nivel superior. Al hacerlo, se puede lograr una representación más completa y envolvente de los datos aeroespaciales.

Imaginar la inclusión de un controlador de espacios aéreos dentro de un entorno tridimensional, proporcionando la capacidad de gestionar aviones en tiempo real, es sumamente emocionante. Con un desarrollo adecuado, esta idea tiene el potencial de revolucionar la forma en que se controlan y gestionan los espacios aéreos.

La idea de poder estar en un aeropuerto, esperando a un familiar y tener la posibilidad de seguir el

aterrizaje y visualizar lo que están viendo desde el avión a través de una experiencia en realidad virtual también resulta emocionante. Esta perspectiva nos permitiría tener una conexión más inmersiva con la experiencia de vuelo y compartir virtualmente el viaje con nuestros seres queridos.



### 1.3. Objetivo general

El objetivo general del proyecto consiste en investigar y adquirir conocimientos sobre tecnologías de realidad virtual y el framework A-FRAME, el cual permite la creación de aplicaciones 3D para visualizar datos. Estas aplicaciones pueden ser ejecutadas en un navegador y ser utilizadas tanto en entornos de escritorio como en dispositivos móviles y dispositivos de realidad virtual, como los OCULUS QUEST.

En este contexto, se desarrollará un prototipo que constará de un conjunto de herramientas altamente configurable que sienta las bases para facilitar la creación de escenas 3D. Estas escenas permitirán la visualización de datos aeronáuticos, como vuelos en movimiento, en un contexto geoespacial. Además, se podrán visualizar entidades geoespaciales, como edificios y terrenos, que contendrán información sobre alturas. El objetivo es proporcionar una plataforma flexible y versátil que permita la visualización y exploración de datos aeronáuticos de forma interactiva en un entorno 3D.

### 1.4. Objetivos específicos

A continuación, se detallan los objetivos específicos del proyecto, clasificados en objetivos funcionales y objetivos técnicos.

#### 1.4.1. Objetivos funcionales

- Desarrollar un escenario que represente un rectángulo geodésico y que incluya un terreno con alturas reales. Además, se puede utilizar una imagen de satélite como textura para brindar mayor realismo al terreno.
- Introducir aviones que representen vuelos reales utilizando posiciones geodésicas. Estas posiciones se obtendrán de algún servicio web que ofrezca datos ADS-B 2.1, y se utilizarán estos datos para animar los aviones.
- Permitir la navegación por el escenario utilizando tanto el teclado y el ratón como dispositivos de realidad virtual, navegando sobre el contorno del terreno.
- Representar edificios sobre el terreno y mostrar datos de los mismos mediante el posicionamiento del ratón encima.
- Crear componentes HUD que formen parte de la escena y se muevan con el usuario dentro de su campo de visión para habilitar o deshabilitar funcionalidades a parte de mostrar información sobre entidades.

- Permitir al usuario mover los componentes de visualización de datos dentro de su campo de visión.
- Posibilitar la selección de aviones para visualizar su información sobre el componente HUD.
- Mostrar el trayecto que realiza un avión desde que entra en el escenario.
- Permitir la visualización de una cámara de a bordo del avión seleccionado.

#### 1.4.2. Objetivos técnicos

- Utilizar la librería A-FRAME como tecnología principal para la creación del prototipo.
- Desarrollar una API que brinde la capacidad de referenciar cualquier posición geodésica dentro del escenario.
- Desarrollar una API que proporcione las llamadas necesarias para calcular la altura del terreno en cualquier punto del escenario.
- Establecer un proceso de almacenamiento de datos de algún servicio web que ofrezca datos ADS-B para reproducir las animaciones de los aviones de manera local a través de una caché.
- Obtener metadatos de un servicio web que contenga información sobre contornos y alturas de edificios.
- Modularizar la aplicación mediante la creación de componentes reutilizables.
- Implementar un sistema configurable que permita la representación de escenarios en cualquier zona del mundo. Esto se logrará mediante la configuración de parámetros como las coordenadas geodésicas de la zona y la carga de datos específicos de esa zona.
- El prototipo final podrá ser ejecutado en entornos de escritorio, en dispositivos de realidad virtual como el dispositivo Oculus Quest que proporciona para las pruebas la universidad.

### 1.5. Distribución del software

Todo el código fuente, así como los componentes y las demos están alojados en el repositorio de GitHub en la siguiente dirección: <https://github.com/djprano/AFrameTFG>

## 1.6. Tecnologías Similares

En el mercado podemos encontrar aplicaciones líderes en el sector de visualización de datos aeronáuticos algunas de las cuales son:

- **FlightRadar24**<sup>4</sup>: Esta aplicación utiliza datos de fuentes como los transpondedores de las aeronaves, radares y sistemas de seguimiento de vuelo para mostrar la posición y la trayectoria de los aviones en tiempo real en un entorno bidimensional.

La aplicación permite hacer zoom, arrastrar y hacer clic en las aeronaves para obtener información adicional, como el número de vuelo, la aerolínea, el tipo de aeronave y los aeropuertos de origen y destino.

Esta aplicación ofrece un servicio de visualización de datos en 3D basado en la plataforma de desarrollo de aplicaciones geoespaciales 3D llamada Cesium<sup>5</sup>, que es similar a la funcionalidad que se desea desarrollar en el proyecto utilizando A-Frame.

Además, la aplicación cuenta con versiones móviles para dispositivos iOS y Android, lo que permite acceder a la información de seguimiento de vuelos desde teléfonos y tablets.

- **Plane Finder**<sup>6</sup>: Esta aplicación comparte muchas similitudes con *FlightRadar24* en términos de funcionalidad y características. Sin embargo la principal diferencia radica en la fuente de los datos, mientras que *FlightRadar24* tiene una amplia cobertura global y una densidad de datos considerable debido a su extensa red de receptores. *Plane Finder* ha aumentado su cobertura y densidad de datos en los últimos años, pero inicialmente tenía una cobertura más centrada en Europa.
- **FlightAware**<sup>7</sup>: FlightAware es una plataforma en línea que proporciona seguimiento de vuelos en tiempo real y datos de aviación. Ofrece una amplia cobertura de seguimiento de vuelos comerciales y privados en todo el mundo, así como información detallada sobre aeropuertos, rutas y aeronaves, se diferencia principalmente con los servicios anteriores en que no dispone de funcionalidad 3D y sus datos están mas centrados en América del Norte.
- **RadarBox**<sup>8</sup>: RadarBox es una herramienta de seguimiento de vuelos en tiempo real que ofrece datos y visualización de tráfico aéreo en todo el mundo. Al igual que el servicio anterior no ofrece una funcionalidad nativa de visualización en 3D como parte de su servicio principal de seguimiento

---

<sup>4</sup><https://www.flightradar24.com/>

<sup>5</sup><https://cesium.com/platform/cesiumjs/>

<sup>6</sup><https://planefinder.net/>

<sup>7</sup><https://es.flightaware.com/>

<sup>8</sup><https://www.radarbox.com/>

de vuelos. RadarBox se centra en proporcionar datos de seguimiento de vuelos en tiempo real y herramientas de análisis para el tráfico aéreo.

Estos ejemplos demuestran cómo las aplicaciones existentes permiten a los usuarios acceder a información valiosa sobre la ubicación de los aviones en tiempo real sobre un entorno bidimensional y tridimensional, lo que contribuye a mejorar la comprensión de la situación operativa.

## Capítulo 2

# Tecnologías relacionadas

En este capítulo vamos a describir el contexto tecnológico en el que se encuentra asentado el prototipo de aplicación web, repasando todas las herramientas tecnológicas usadas y estudiadas para el desarrollo de este proyecto. Por último se van a repasar tecnologías similares en el mercado.

### 2.1. Sistema de Vigilancia Dependiente Automática (ADS-B)

#### 2.1.1. Introducción a la tecnología

El Sistema de Vigilancia Dependiente Automática, conocido por sus siglas en inglés ADS-B (Automatic Dependent Surveillance Broadcast), es un sistema de vigilancia cooperativa en el cual una aeronave obtiene su posición a través de un sistema de navegación por satélite y emite periódicamente esta información para permitir un seguimiento de la aeronave. Esta tecnología no solo se encarga de transmitir información sobre la posición de la aeronave, sino que también puede proporcionar datos adicionales como velocidad, orientación, altitud, nombre de vuelo, identificador único de la aeronave, país de origen y otros datos relevantes.

- El término *Automatic* (automática) refleja el hecho de que las aeronaves equipadas con ADS-B transmiten información de manera automática y periódica sin requerir intervención humana directa.
- El adjetivo *Dependent* (dependiente) se emplea debido a que la información transmitida depende del equipamiento de sensores y sistemas presentes en la aeronave.
- La palabra *Surveillance* (vigilancia) alude al hecho de que el ADS-B proporciona datos que permiten el seguimiento y control de la aeronave.

- Por último, el término **Broadcast** (transmisión) indica que la información es enviada por la aeronave a través de señales de radio en modo unidireccional, posibilitando que cualquier receptor en su área de cobertura pueda capturar la señal.

### 2.1.2. Historia y evolución

En el pasado, los sistemas de vigilancia aeronáutica se fundamentaban en radares terrestres para detectar aeronaves en el espacio aéreo. Sin embargo, la OACI tenía como objetivo permitir que las propias aeronaves transmitieran su información de posición y otros datos relevantes mediante señales de radio, en lugar de depender exclusivamente de los radares terrestres.

En la primera década de los años 2000, se llevaron a cabo varios proyectos piloto para probar los primeros prototipos de sistemas ADS-B. Un ejemplo de ello es el proyecto *Capstone*<sup>1</sup> en Estados Unidos, que se llevó a cabo en Alaska entre 1999 y 2004. El objetivo de este proyecto piloto a gran escala era evaluar si la tecnología ADS-B mostraba mejoras significativas en la seguridad de la aviación en Alaska.

Conforme avanzaba la década de los años 2000, varios países comenzaron a adoptar la tecnología y los estándares de ADS-B. Además, se establecieron requisitos regulatorios más estrictos para equipar a más aeronaves con transpondedores ADS-B. Esto implicó un esfuerzo en la modernización de los sistemas de gestión del tráfico aéreo existentes, con el fin de aprovechar plenamente los beneficios de los sistemas ADS-B.

Hacia finales de la primera década del 2000, surgieron empresas como FLIGHTRADAR24 y proyectos de código abierto como OPENSKY, los cuales empezaron a desarrollar una red de receptores ADS-B. Estas organizaciones llegaron a acuerdos con instituciones académicas y proveedores de servicios de tráfico aéreo, lo que permitió que la red se expandiera con el tiempo. Además, se implementaron estrategias de *Crowdsourcing*, donde personas de todo el mundo instalaban receptores ADS-B en sus ubicaciones y enviaban los datos de las señales de las aeronaves a los servidores de estas plataformas. Esto permitió a dichas plataformas recopilar información y proporcionar servicios a través de sus servidores.

Se ha demostrado que esta tecnología contribuye al control y la seguridad aérea, lo cual llevó a que la Agencia de la Unión Europea para la Seguridad Aérea estableciera requisitos y fechas límite para la implementación de ADS-B OUT en el espacio aéreo europeo, estableciendo la fecha límite del 7 de junio de 2020 como se indica en el *reglamento de ejecución 2017/386*<sup>2</sup>. Actualmente, el uso de esta tecnología es obligatorio en las aeronaves registradas en la Unión Europea.

---

<sup>1</sup><https://www.aopa.org/news-and-media/all-news/2017/august/pilot/ads-b-in-alaska>

<sup>2</sup><https://www.boe.es/doue/2017/059/L00034-00036.pdf>

### 2.1.3. Descripción técnica

Para utilizar la tecnología ADS-B, una aeronave debe estar equipada con un dispositivo que admita el enlace **1090ES**, el cual permite un enlace de datos a través de VHF en la banda de *1090 MHz*, una frecuencia asignada para las comunicaciones aeronáuticas. Este dispositivo es el encargado de transmitir periódicamente la información.

Sin embargo, en Estados Unidos se permite el uso del estándar **UAT** (*Universal Access Transceiver*) en la banda de *978 MHz* para propósitos de ADS-B por debajo de los *18,000* pies. Por encima de esta altitud, se debe utilizar el enlace 1090ES. Esta diferencia se estableció con el objetivo de evitar la congestión debido al uso excesivo de la banda 1090ES.

Los transpondedores de sistemas **SSR** (*Secondary Surveillance Radar*) son utilizados en la aviación para complementar la vigilancia del tráfico aéreo proporcionada por el radar primario. Estos transpondedores ya están preparados para emitir en la banda 1090ES en *Modo S*. Sin embargo, dado que el sistema ADS-B requiere una transmisión más frecuente de información, se introdujo el *modo ES* (*Extended Squitter*). El modo es una modificación del transpondedor *Modo S* que permite una radiodifusión más frecuente de información.

Con esto, al tener un equipo de aviónica instalado que cumple los requisitos para operar con el sistema **SSR** y ADS-B al mismo tiempo, se presenta una gran ventaja operativa al utilizar un solo equipo para ambos propósitos.

Existen tres tipos de transpondedores en función de su capacidad:

- **ADS-B OUT**: es un transpondedor diseñado exclusivamente para la transmisión de datos. Cumple con el requisito mínimo establecido por muchas regulaciones y estándares de la aviación, en la figura 2.1 podemos apreciar que el avión de la derecha al ser de este tipo solo emite datos.
- **ADS-B IN**: es un transpondedor diseñado exclusivamente para la recepción de datos. Suelen ser utilizados por estaciones terrestres. Existen organizaciones como FLIGHTRADAR24 o OPENSky proporcionan transpondedores **ADS-B IN** de forma gratuita a cambio de completar un formulario. Estos transpondedores se utilizan para crear una red mundial que permite recibir información en tiempo real de todas las aeronaves, lo cual contribuye a alimentar los datos en sus servidores, especialmente en zonas con poca cobertura.
- **ADS-B IN & OUT**: es utilizado tanto en aviones como estaciones terrestres y presenta la capacidad tanto de emitir sus propios datos, como de recibir información de aviones y estaciones terrestres cercanas. Esta funcionalidad se implementa con el propósito de visualizar un radar en los controles de la aeronave, mostrando la posición de las aeronaves cercanas, así como para recibir información

meteorológica u otros tipos de metadatos de las estaciones terrestres. En la Figura 2.1, se puede observar que tanto el avión a la izquierda como la estación terrestre son de tipo **ADS-B IN & OUT**. Esto significa que el avión tiene la capacidad de recibir la información meteorológica transmitida por la estación terrestre, al mismo tiempo que transmite sus propios datos a dicha estación.

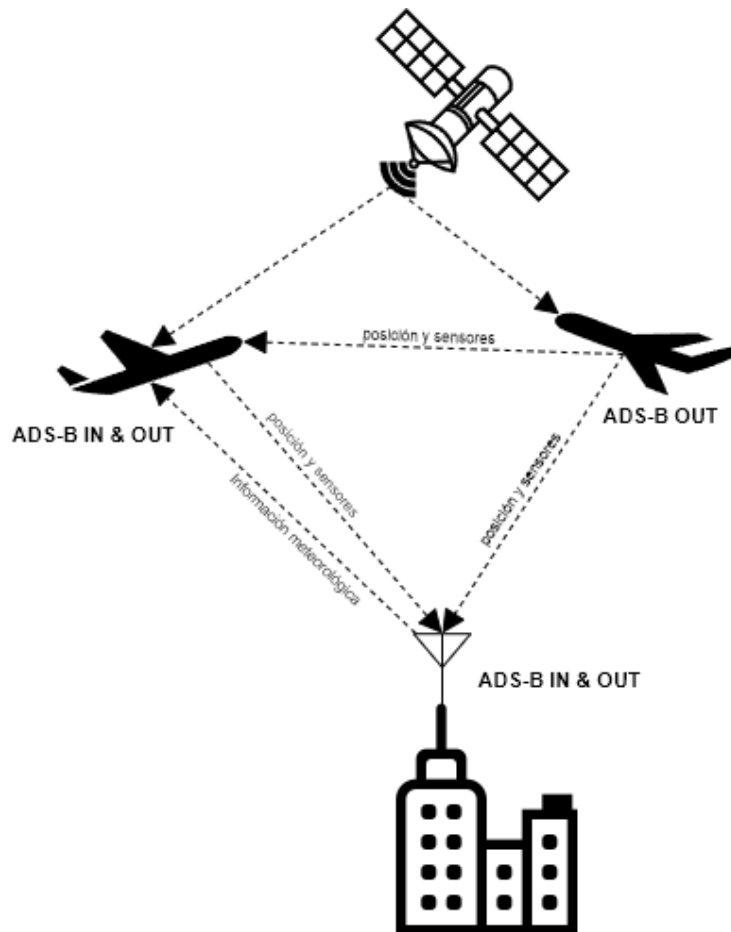


Figura 2.1: Ejemplo de tipos de transpondedor ADS-B.

#### 2.1.4. Contexto y aplicación en el prototipo

La tecnología ADS-B juega un papel fundamental en el contexto de este proyecto, el objetivo principal del prototipo es la visualización de datos aeronáuticos en un entorno 3D. Por lo tanto esta tecnología nos aporta la capacidad de obtener la información que necesita nuestra aplicación para acceder a los datos que proporcionan dinamismo, ya que las entidades que utilizan esta tecnología estarán constantemente cambiando de posición en el escenario gracias a la variación de los datos de posición. La tecnología estándar ADS-B nos ofrece una resolución temporal de un segundo entre muestras, aunque los servidores gratuitos normalmente ofrecerán solo la capacidad de acceder a los vectores de información cada



cinco segundos como es el caso de OPENSKY, resolución suficiente para crear una interpolación entre posiciones y recrear el vuelo de los aviones.

## 2.2. Opensky-network API

### 2.2.1. Introducción a la tecnología

OPENSKY es un proyecto colaborativo de código abierto que permite acceder a la información detallada de las aeronaves gracias a una plataforma o red de sensores distribuida por el mundo recopilando datos en tiempo real sobre el tráfico aéreo.

### 2.2.2. Historia y evolución

OPENSKY se inició sobre los años 2012 cuando un equipo de investigadores de *Armasuisse* (Suiza), la *Universidad de Kaiserslautern* (Alemania) y la *Universidad de Oxford* (Reino Unido) comenzaron a colaborar capturando datos de tráfico aéreo a través de la tecnología ADS-B. A medida que el proyecto fue creciendo crearon la plataforma online sin ánimo de lucro que hoy conocemos como OPENSKY NETWORK, con la finalidad de permitir a los investigadores, desarrolladores y entusiastas de la aviación acceder a los datos que se recopilan.

### 2.2.3. Descripción técnica

En la red de OPENSKY tan pronto como llega un mensaje ADS-B se crea un registro para el avión llamado *vector de estado*. El vector de estado del servicio contiene la información descrita en la tabla 2.1. En la sección 4.6.2 hablaremos como el prototipo utilizará la API proporcionada por los servicios de la organización de software libre OPENSKY para consumir datos en tiempo real o en forma diferida a través de una caché local. Esto permitirá crear representaciones en tiempo real del espacio aéreo dentro de un escenario definido por un cuadrado delimitado por coordenadas geodésicas.

### 2.2.4. Contexto y aplicación en el prototipo

En nuestro caso, utilizaremos la API gratuita que nos brindan los servidores de OpenSky con una resolución temporal de 5 segundos. Por lo tanto, aunque podríamos tener una mayor resolución temporal utilizando la tecnología ADS-B como vimos en la sección 2.1, nuestra aplicación se adaptará a la opción gratuita que nos ofrece el servicio REST de OpenSky.

### 2.2.5. Ejemplos de uso

En la figura 2.2, se muestra un ejemplo de una petición GET para las coordenadas de Madrid utilizando la API de OpenSky. En este ejemplo, se realiza una petición *GET* al *endpoint* `https://opensky-network.org/api/states/all` con los parámetros de consulta *lamin*, *lomin*, *lamax* y *lomax* especificados. Además, se incluye un encabezado de autorización con el valor (*Basic user:password*) para autenticar la solicitud donde la parte (*user:password*) puede codificarse en BASE64 para no ir los datos en claro.

```
GET /api/states/all?lamin=40.023417&lomin=-4.2041338&lamax=40.7441446&lomax=-3.2538165
Host: opensky-network.org
Authorization: "Basic user:password"
```

Figura 2.2: Petición a la API de OpenSky.

Indice	Propiedad	Tipo	Descripción
0	icao24	string	Dirección del transpondedor en hexadecimal.
1	callsign	string	Nombre del avión o vuelo de 8 caracteres.
2	origin_country	string	País de origen.
3	time_position	int	Marca de tiempo UNIX en segundos de la última actualización.
4	last_contact	int	Marca de tiempo UNIX en segundos de la última trama ADS-B recibida.
5	longitude	float	Coordenada longitud WGS-84.
6	latitude	float	WGS-84 Coordenada latitud WGS-84.
7	baro_altitude	float	Altitud barométrica en metros.
8	on_ground	boolean	Indica si la posición se recuperó de un informe de superficie.
9	velocity	float	Velocidad en metros por segundo.
10	true_track	float	Orientación del avión en grados respecto al norte
11	vertical_rate	float	Velocidad vertical en metros por segundo
12	sensors	int[]	Identificadores de los receptores ADS-B que han contribuido.
13	geo_altitude	float	Altitud geométrica en metros.
14	squawk	string	Código del transpondedor.
15	spi	boolean	indicador para vuelos de propósito especial.
16	position_source	int	Fuente del vector de estado.
17	category	int	Enumerado de categoría del avión.

Cuadro 2.1: Vector de estado de la API Rest OpenSky Network.

## 2.3. HTML5

### 2.3.1. Introducción a la tecnología

HTML5 es la última versión del lenguaje de marcado de hipertexto HTML, el lenguaje usado para estructurar y presentar contenido en una web. Junto con la tecnología DOM [2.4](#), forma la base tecnológica para el desarrollo de aplicaciones web modernas. Mientras que HTML5 proporciona las herramientas de marcado para estructura y presentar la información en la web, la tecnologías DOM representa la estructura de ese contenido en forma de árbol de objetos para hacer los datos accesibles y manipulables mediante lenguajes de script. La combinación de HTML5 y DOM permite a los desarrolladores acceder, modificar y actualizar de forma dinámica los elementos y estilos de una página web.

### 2.3.2. Historia y evolución

En 2004, el WHATWG, liderado por Ian Hickson, inició el desarrollo de lo que ahora conocemos como HTML5. Su objetivo principal era mejorar la semántica y estructura del lenguaje HTML, además de agregar nuevas funcionalidades y capacidades para el desarrollo web moderno. El enfoque que resultó exitoso fue el de abordar los problemas y limitaciones que HTML presentaba en ese momento. Paralelamente, la W3C también estaba trabajando en la evolución de HTML, presentando XHTML como su propuesta. Sin embargo, a medida que el proyecto avanzaba, quedó claro que XHTML no era compatible con el ecosistema web existente. En 2006, la W3C decidió abandonar el desarrollo de XHTML y unirse al grupo de trabajo WHATWG para colaborar en el desarrollo de HTML5. Durante el proceso de desarrollo de HTML5, se llevaron a cabo múltiples revisiones y actualizaciones con el fin de incorporar nuevas características, mejorar la semántica, definir APIs y abordar los desafíos técnicos y las necesidades emergentes de la web moderna.

HTML5 introdujo una serie de mejoras semánticas, tales como la introducción de los elementos `<header>`, `<nav>`, `<section>`, `<article>`, entre otros, que permiten una mejor estructuración y accesibilidad del contenido. Además, HTML5 incluyó capacidades multimedia nativas mediante la incorporación de elementos como `<video>` y `<audio>`, lo que eliminó la necesidad de utilizar tecnologías complementarias como FLASH. La especificación de HTML5 se finalizó oficialmente en octubre de 2014, marcando un hito importante en la evolución del desarrollo web. Desde entonces, HTML5 se ha convertido en el estándar predominante para la creación de páginas y aplicaciones web modernas.

### 2.3.3. Descripción técnica

HTML5 se basa en la combinación de las tecnologías HTML, CSS y JAVASCRIPT para proporcionar una plataforma de desarrollo web versátil y adaptada a las necesidades modernas.

HTML5 ofrece a los desarrolladores una amplia gama de APIs que les permiten acceder y manipular diversas características del navegador. Estas APIs han permitido a los desarrolladores crear experiencias avanzadas como por ejemplo hacer uso de la [API](#) de geolocalización para obtener la ubicación del usuario y ofrecer servicios personalizados de basados en su posición. Estas herramientas han brindado a los desarrolladores el ecosistema de tecnologías necesarios para el desarrollo de aplicaciones web más sofisticadas. La API de almacenamiento local permite a las aplicaciones web almacenar datos en el dispositivo del usuario, lo que acelera la experiencia del usuario debido al cacheo de muchos metadatos para mantener estados he incluso poder operar con aplicaciones de manera offline. Además, HTML5 ha introducido nuevas etiquetas semánticas y atributos que permiten una mejor estructuración del contenido y una mayor accesibilidad. Esto facilita a los motores de búsqueda y a los usuarios comprender la información presente en las páginas web de manera más precisa.

Una característica importante que HTML5 introdujo en el contexto del proyecto fue la capacidad de crear etiquetas personalizadas. Esta característica permite a los desarrolladores definir sus propias etiquetas HTML para especificar el comportamiento y apariencia utilizando JAVASCRIPT.

Esta capacidad de crear etiquetas personalizadas fue aprovechada por los creadores de A-FRAME para facilitar la creación de escenas en 3D. Utilizando estas etiquetas personalizadas [1], los desarrolladores de A-FRAME crearon etiquetas que ejecutan el JAVASCRIPT necesario para la inicialización de geometrías o componentes de THREE.JS[13]. Esto brinda una forma sencilla y rápida de crear escenas en 3D sin la necesidad de tener un amplio conocimiento de programación 3D, tan solo es necesario definir un `index.html` usando los elementos creados por la librería para la creación de una escena 3D.

### 2.3.4. Contexto y aplicación en el prototipo

Esta tecnología es la base sobre la cual A-FRAME crea la estructura del contenido de realidad virtual en el entorno web. Como veremos en la sección 2.11 A-FRAME utiliza una combinación de elementos HTML5 como `<a-scene>`, `<a-entity>` y `<a-camera>`, junto con componentes propios de A-FRAME, para construir y definir objetos 3D, cámaras, luces y la escena principal.

En resumen, A-FRAME utiliza tecnologías como HTML5 y componentes propios para definir la jerarquía del escenario y la [Renderización](#) de toda la escena.

## 2.4. DOM

### 2.4.1. Introducción a la tecnología

El *Modelo de Objetos del Documento* conocido como (DOM) es una representación estructurada y jerárquica de un documento HTML, XML o XHTML, que permite acceder y manipular los elementos y contenido del documento mediante programación. Representa la página de manera que los programas puedan cambiar la estructura, estilo y contenido del documento.

El navegador interpreta y construye el DOM a partir del código HTML con el fin de facilitar la manipulación y presentación de la página web. Mediante el uso del DOM, el navegador representa cada elemento del documento HTML como un objeto en la memoria. Este enfoque permite a los desarrolladores web acceder y manipular los elementos, atributos y contenido de la página utilizando lenguajes de programación como JAVASCRIPT.

El DOM no es un lenguaje de programación en sí, pero sin él, el lenguaje JAVASCRIPT 2.5 no tendría ningún modelo de objetos para una página HTML y sus componentes. El documento en su totalidad, la cabeza, las tablas dentro del documento, los encabezados de tabla, el texto dentro de las celdas de la tabla y todos los demás elementos en un documento son partes del modelo de objetos del documento. Todos ellos pueden ser accedidos y manipulados utilizando el DOM y un lenguaje de scripting como JAVASCRIPT.

### 2.4.2. Historia y evolución

El DOM surgió como un conjunto de objetos que representan a un documento HTML en forma de árbol. Fue creado inicialmente para el navegador NETSCAPE de la compañía NETSCAPE COMMUNICATIONS. A partir de mediados de la década de 1990, se convirtió en la interfaz entre el documento HTML y el lenguaje JAVASCRIPT 2.5, que se incorporó nativamente en los navegadores.

El DOM se originó en respuesta a la necesidad de acceder de manera sencilla a los datos estructurados de los elementos XML y HTML. Proporcionó una manera estándar de manipular y acceder a los elementos, atributos y contenido de un documento web.

Los primeros estándares del DOM, desarrollados por la W3C (*World Wide Web Consortium*), surgieron como un intento de poner fin a las guerras de los navegadores, donde cada uno ofrecía diferentes técnicas para modificar dinámicamente la estructura de las páginas web. Estos estándares buscaban establecer una forma común y coherente de interactuar con los elementos del documento a través del DOM.

### 2.4.3. Descripción técnica

El DOM se compone de varios conceptos:

- **Nodos:** Son los elementos fundamentales del árbol del DOM. Representan los elementos del documento, y cada nodo, excepto el nodo raíz, tiene un padre. La colección de todos los nodos refleja la jerarquía del documento.
- **Propiedades y métodos:** Cada nodo contiene propiedades que representan sus características, como su nombre, atributos, contenido o estilo. Además, los nodos tienen métodos que permiten realizar operaciones sobre ellos, como agregar o eliminar nodos hijos.
- **Eventos:** Los nodos del DOM también permiten el manejo de eventos, como cuando el usuario pasa el cursor sobre un nodo o hace clic en él.
- **Consulta:** A través de JAVASCRIPT, es posible acceder a los elementos del DOM y tener la capacidad de crear, modificar, eliminar y cambiar atributos del DOM de forma dinámica.

### 2.4.4. Contexto y aplicación en el prototipo

Esta tecnología dentro del contexto de la implementación es usada como base para generar entidades y componentes en tiempo real dando el dinamismo a la escena de A-FRAME. Es la tecnología usada para que las entidades contenidas en la escena sean manipuladas a través de la modificación de sus propiedades y el envío de eventos.

### 2.4.5. Ejemplos de uso

Desde cualquier parte del código de nuestra aplicación podemos manipular el DOM a través de los métodos que nos proporciona la [API](#) de JAVASCRIPT. Por ejemplo vamos a mostrar como podemos crear un elemento del DOM en nuestra aplicación que representa un avión y agregarlo dentro del elemento que representa la escena:

```
function createFlightElement(id) {  
  //Vuelo nuevo  
  let mainScene = document.querySelector('a-scene');  
  let entityEl = document.createElement('a-entity');  
  entityEl.setAttribute('id', id);  
  entityEl.setAttribute('gltf-model', "#plane");  
  entityEl.setAttribute('class', "clickable");  
  entityEl.setAttribute('scale', { x: configuration.scale, y:  
    ↪ configuration.scale, z: configuration.scale });
```

```
entityEl.setAttribute('hover-scale', 'limitDistance: 100');  
entityEl.addEventListener('mouseenter', evt => handleMouseEnter(evt));  
entityEl.addEventListener('click', evt => handleMouseClicked(evt));  
entityEl.addEventListener('mouseleave', evt => handleMouseLeave(evt));  
mainScene.appendChild(entityEl);  
  
return entityEl;  
  
}
```

## 2.5. JavaScript

### 2.5.1. Introducción a la tecnología

JAVASCRIPT es un lenguaje de programación ampliamente utilizado en el desarrollo web que permite agregar interactividad y dinamismo a las páginas y aplicaciones. Es el único lenguaje que permite ser ejecutado en el navegador web de forma nativa, sin necesidad de compilación. Su versatilidad, compatibilidad con *Frameworks* y capacidad para ejecutarse tanto en el lado del cliente como en el lado del servidor lo convierten en una herramienta fundamental para crear experiencias web modernas y funcionales.

### 2.5.2. Historia y evolución

La historia de JAVASCRIPT se remonta a 1995, cuando *Brendan Eich* un programador estadounidense diseñó el lenguaje para NETSCAPE NAVIGATOR en un tiempo récord de dos semanas. Inicialmente se llamó MOCHA y LIVESCRIPT, pero debido a una colaboración con NETSCAPE y SUN (la empresa propietaria por aquel entonces del lenguaje JAVA), por cuestiones de marketing ya que JAVA en aquel entonces era un lenguaje de programación muy popular se decidió establecer como nombre JAVASCRIPT. Sin embargo a pesar de la similitud entre el nombre de JAVA y el nuevo lenguaje JAVASCRIPT, las similitudes entre ambos lenguajes son pocas y no tienen mucho en común. Al poco tiempo del lanzamiento de JAVASCRIPT, MICROSOFT presentó un lenguaje más o menos compatible llamado JSCRIPT para INTERNET EXPLORER 3.0. Para conciliar ambos lenguajes, NETSCAPE presentó JAVASCRIPT a la *European Computer Manufacturers Association* (ECMA), con el objetivo de crear un estándar que unificase el lenguaje. Fué entonces cuando nació el estándar que se ha denominado ECMASCRIPT, que se adoptó en la versión 6 en 2015 (ES6 o ES2015 abreviado). Desde entonces se han ido agregando nuevas características año tras año por lo que se acordó no utilizar un número consecutivo de versión, sino simplemente enumerar el año respectivo en el número de versión.

Version Name	Year of Publication
ES1	1997
ES2	1998
ES3	1999
ES4	Not released
ES5	2009
ES6/ES2015	2015
ES2016	2016
ES2017	2017
ES2018	2018
ES2019	2019
ES2020	2020
ES2021	2021

### 2.5.3. Contexto y aplicación en el prototipo

En este proyecto JAVASCRIPT ha sido el lenguaje de programación base utilizado para la implementación de toda la lógica de negocio desarrollada para el prototipo. Es por lo tanto la herramienta principal para el diseño y desarrollo de componentes que compondrán la escena final . A parte de esto como veremos más adelante es la tecnología sobre la que está construida los *Frameworks* y librerías usados en este proyecto tales como A-FRAME 2.11 y LEAFLET 2.12

### 2.5.4. Ejemplos de uso

## 2.6. WebGL

### 2.6.1. Introducción a la tecnología

La tecnología de gráficos 3D es una tecnología que permite la *Renderización* de gráficos interactivos en tiempo real sobre un navegador web sin la necesidad de plugins adicionales. Está construida sobre una *API* basada en *OPENGL* para acceder a la funcionalidad de renderizado de la tarjeta gráfica del dispositivo.

WebGL<sup>3</sup> está basada en JAVASCRIPT 2.5 y se relaciona con los elementos de una página web mediante el uso de HTML5 2.3. Proporciona herramientas para crear gráficos 3D en el navegador de forma

<sup>3</sup><https://www.khronos.org/webgl/>



optimizada y eficiente gracias al uso de la [GPU](#). WebGL permite una renderización de objetos 3D, texturas, sombreado y efectos visuales avanzados.

### 2.6.2. Contexto y aplicación en el prototipo

Tanto A-FRAME como THREE.JS utilizan WebGL como tecnología para la renderización de escenas y objetos en 3D en un navegador web. Ambas bibliotecas se basan en esta tecnología para realizar operaciones sobre la tarjeta gráfica en el renderizado de sus escenarios.

## 2.7. WebXR

La tecnología WEBXR es una agrupación de estándares que se utilizan en conjunto para habilitar la representación de escenas en 3D en [Hardware](#) diseñado para la visualización de mundos virtuales (VR) o para agregar gráficos al mundo real (realidad aumentada). WEBXR proporciona una combinación perfecta entre realidad aumentada y realidad virtual.

Una de las ventajas principales de WEBXR es que permite crear experiencias inmersivas directamente en un navegador web gracias a que está construido sobre WebGL [2.6](#), sin necesidad de descargar ninguna aplicación adicional. Esto facilita el acceso a estas experiencias a través de múltiples dispositivos, ya que solo se necesita un navegador compatible para disfrutar de ellas.

WEBXR proporciona [API](#) y funcionalidades que permiten interactuar con dispositivos de realidad virtual y realidad aumentada, como gafas VR, cascos AR o incluso dispositivos móviles con capacidades AR. Estas funcionalidades incluyen el seguimiento de la posición y la orientación del dispositivo, renderización de escenas 3D, detección de gestos y eventos de interacción, entre otros.

## 2.8. VR

La realidad virtual (VR) es la tecnología responsable de recrear un entorno ficticio que simula la apariencia real, permitiendo al usuario sumergirse en él y sentir que forma parte de ese entorno. Para experimentar esta realidad virtual, se requieren dispositivos especiales como las gafas o cascos de realidad virtual. Estos dispositivos generan entornos virtuales utilizando la combinación de [Hardware](#) y software diseñados para engañar a nuestros sentidos. Las imágenes en el visor se muestran de manera estereoscópica, lo que significa que se crean dos imágenes ligeramente diferentes para simular una perspectiva diferente en cada ojo. Esto crea una sensación de percepción tridimensional, similar a cómo percibimos el mundo real con nuestros sentidos.

### 2.8.1. Contexto y aplicación en el prototipo

El prototipo de este proyecto está diseñado para funcionar con dispositivos que generan entornos virtuales, permitiendo una inmersión completa y realista en dicho entorno. Para lograr esto, se utilizan las herramientas proporcionadas por la plataforma A-Frame, como se describe en la sección 2.11.

## 2.9. Node JS

### 2.9.1. Introducción a la tecnología

NODE.JS es un entorno de código abierto y multiplataforma que permite la ejecución de código JAVASCRIPT de forma asíncrona, con capacidades de entrada y salida de datos, y una arquitectura orientada a eventos. Está basado en el *motor* V8 de GOOGLE y fue creado con el objetivo de desarrollar programas de red que puedan ser ejecutados en el lado del servidor.

Con NODE.JS, los desarrolladores pueden construir aplicaciones del lado del servidor utilizando JAVASCRIPT, lo que proporciona coherencia en el lenguaje de programación tanto en el lado del cliente como en el servidor. Además, cuenta con una amplia gama de módulos y bibliotecas disponibles a través de su gestor de paquetes [NPM](#), lo que facilita el desarrollo rápido y eficiente de aplicaciones web y de red.

### 2.9.2. Contexto y aplicación en el prototipo

Como mencionamos en la sección 4.6.3, hemos utilizado NODE.JS para desarrollar un *proceso batch* que se encarga de almacenar los datos de la API de OPENSky 2.2 en una caché local. Esta técnica nos permite ejecutar la aplicación en modo diferido, es decir, obteniendo previamente los datos necesarios y almacenándolos en la caché para su posterior uso.

### 2.9.3. Ejemplos de uso

A continuación se muestra un ejemplo de código que puede ser ejecutado por NODE.JS para almacenar los datos obtenidos de la API en un archivo [JSON](#) para su posterior uso:

```
function main() {  
    var endpoint = 'https://opensky-network.org/api/states/all?lamin=' +  
        ↪ configuration.latMin + '&lomin=' + configuration.longMin + '&lamax=' +  
        ↪ configuration.latMax + '&lomax=' + configuration.longMax;  
    var credentials = Buffer.from(configuration.apiUser + ':' +  
        ↪ configuration.apiPassword).toString('base64');  
    setInterval(() => {
```

```
fetch(endpoint, {
  method: 'GET',
  headers: { 'Authorization': 'Basic ' + credentials }
}).then(response => response.json()).then(json => {
  if (json != undefined && json != null &&
    ↪ !isEmptyObject(json)) {
    saveJson(json, index++);
  }
});
}, configuration.daoInterval);
}
```

## 2.10. Threejs

### 2.10.1. Introducción a la tecnología

THREE.JS es una biblioteca escrita en JAVASCRIPT que ofrece una API para interactuar con WEBGL 2.6 y crear de manera sencilla animaciones y escenas 3D sobre entornos web. A-FRAME 2.11 es una de las tecnologías usadas en el prototipo y se basa en THREE.JS creando una capa de abstracción que simplifica el desarrollo de entornos tridimensionales al permitir el uso directo de HTML5, entidades y componentes. A través de A-FRAME, es posible crear escenarios de manera rápida y sencilla, aprovechando la potencia de THREE.JS en combinación con la facilidad y familiaridad de HTML5.

### 2.10.2. Historia y evolución

El origen de THREE.JS se remonta a principios de la década de 2000, cuando el desarrollador *Ricardo Cabello*, también conocido como *Mr.doob*, comenzó a experimentar con gráficos y animaciones en tres dimensiones utilizando tecnologías web. En ese momento, WEBGL 2.6 aún no era muy utilizado y los navegadores no ofrecían un método sencillo para crear contenido 3D interactivo. *Cabello* optó por utilizar la especificación WEBGL recién lanzada en 2009, que permitía usar gráficos 3D acelerados por *Hardware* en los navegadores web. Comenzó a trabajar en una biblioteca JAVASCRIPT para que los desarrolladores web pudieran crear contenido WEBGL más fácilmente. El proyecto originalmente se llamaba *Canvas 3D* antes de cambiar su nombre a THREE.JS para reflejar el uso del lenguaje JAVASCRIPT como base de la librería. El objetivo de THREE.JS era simplificar la programación gráfica en tres dimensiones y facilitar la creación de gráficos interactivos en el navegador. A medida que más desarrolladores se dieron cuenta de su potencial y comenzaron a usar THREE.JS en sus proyectos, se hizo rápidamente popular. La creciente adopción de WEBGL por parte de los navegadores benefició a la biblioteca, que se convirtió

en una herramienta esencial para la creación de contenido 3D en la web.

### 2.10.3. Descripción técnica

Las principales características técnicas de THREE.JS son las siguientes:

- **Renderizado en tiempo real:** aprovechando la aceleración de *Hardware* proporcionada por WebGL, THREE.JS es capaz de renderizar gráficos en 3D de manera eficiente en los navegadores web modernos.
- **Geometrías y mallas:** la biblioteca ofrece una amplia variedad de geometrías predefinidas, como esferas, cilindros, cubos, entre otros. Estas geometrías pueden modificarse mediante atributos, e incluso es posible crear geometrías personalizadas utilizando técnicas de extrusión. Estas geometrías pueden combinarse con mallas para crear superficies y escenas complejas.
- **Materiales y texturas:** THREE.JS proporciona una amplia gama de materiales predefinidos, incluyendo materiales básicos, materiales refractivos, reflectantes y texturas. Estos materiales y texturas se utilizan para agregar detalles a las geometrías y mallas, logrando un mayor realismo en las escenas 3D.
- **Iluminación y sombreado:** la biblioteca admite diferentes técnicas de iluminación, como luces direccionales, puntuales y de área, lo que permite simular una iluminación realista en la escena. Además, se pueden aplicar técnicas de sombreado suave y sombreado de *Phong* para lograr efectos visuales más sofisticados.
- **Cámaras y controles:** THREE.JS proporciona cámaras que permiten definir la perspectiva y la vista de la escena. Además, incluye controles de cámara predefinidos, como los controles de órbita y los controles de vuelo, que facilitan la interacción del usuario para visualizar la escena 3D.
- **Animaciones:** THREE.JS ofrece un sistema de animación que permite crear animaciones de objetos mediante la transición de sus propiedades, como la posición o la escala.
- **Interactividad:** THREE.JS permite la interacción con los objetos de la escena a través de eventos de ratón, teclado y táctiles, lo que brinda una experiencia interactiva al usuario.

### 2.10.4. Contexto y aplicación en el prototipo

THREE.JS es la librería utilizada por A-FRAME como motor gráfico, lo que la convierte en la base para el desarrollo de componentes en el prototipo del proyecto. Además, THREE.JS se encarga del ren-

derizado en 3D y nos brinda las capacidades necesarias para crear componentes que luego pueden ser utilizados con una sintaxis declarativa similar a HTML, a través del uso de A-FRAME.

Por lo tanto, se ha utilizado como tecnología heredada por A-FRAME, y los componentes desarrollados en el prototipo hacen uso de la API proporcionada por esta librería.

### 2.10.5. Ejemplos de uso

En el siguiente ejemplo, podemos apreciar lo sencillo que resulta crear un material y una geometría de tipo línea. Estos serán utilizados para representar en el escenario el trayecto seguido por un vuelo:

```
const material = new THREE.LineBasicMaterial({
  color: 0x0000ff,
  linewidth: 1,
  fog: false
});
this.geometry = new THREE.BufferGeometry().setFromPoints(this.data.points);
this.geometry.userData = { points: this.data.points };
const line = new THREE.Line(this.geometry, material);
this.el.object3D.add(line);
```

## 2.11. A-Frame

### 2.11.1. Introducción a la tecnología

A-FRAME[1] es un *Framework* de código abierto construido sobre la tecnología THREE.JS de la que se habla en la sección 2.10, que permite la creación de experiencias de realidad virtual en entornos web. A diferencia de otros *Frameworks* 3D, A-FRAME ofrece un sistema de implementación sencillo, utilizando una sintaxis declarativa similar a la de HTML5. Esto facilita a los desarrolladores web la creación rápida e intuitiva de aplicaciones de realidad virtual, sin necesidad de poseer un profundo conocimiento ni adentrarse en la complejidad de la programación en entornos tridimensionales. Una de las principales ventajas de A-FRAME es que se basa en el *modelo entidad componente*, lo que permite el desarrollo de componentes reutilizables y su parametrización para configurar su comportamiento. Además, A-FRAME ya proporciona una amplia gama de geometrías y componentes para la creación de materiales, iluminación, sombras y la gestión de eventos de controladores de realidad virtual, como las gafas OCULUS. Esto permite que los proyectos puedan comenzar el desarrollo desde un punto avanzado.

### 2.11.2. Historia y evolución

A-FRAME fue desarrollado por el equipo de MOZILLA VR a finales de 2015. Este equipo fue líder en el desarrollo de herramientas para la tecnología WEBVR. Debido a la necesidad de crear contenido de manera más sencilla y rápida, se formó un equipo que incluía a los principales mantenedores de este *Framework*, como *Diego Marcos* y *Josh Carpenter*.

El objetivo de A-FRAME era permitir a los desarrolladores web y diseñadores de experiencias 3D y realidad virtual crear contenido utilizando HTML, sin necesidad de tener conocimientos profundos de *WEBGL*. El primer lanzamiento público de A-FRAME tuvo lugar el 16 de diciembre de 2015. En la actualidad, hay más de 75 contribuyentes en total.

### 2.11.3. Descripción técnica

Las principales características técnicas de A-FRAME son las siguientes:

- Configuración de la escena mediante el uso de lenguaje de marcado que nos permite configurar las luces, los controles, las cámaras, los eventos y todo lo relacionado con la configuración *WEBXR*.
- Es compatible con la mayor parte de las bibliotecas de desarrollo web basadas en *JAVASCRIPT* como *REACT*, *ANGULAR* o *VUE*.
- Arquitectura entidad-componente favoreciendo la reutilización de componentes complejos.
- Contiene una herramienta de inspector visual que se puede invocar desde el navegador mediante la combinación de teclas `control+alt+i`, ayudando a la depuración y la detección de errores, y permitiendo la visualización de la jerarquía de la escena, consultando sus componentes y permitiendo la modificación de sus atributos.
- Proporciona utilidades para optimizar el rendimiento, como es el caso de la función *throttled*, que permite que un código no sea ejecutado en cada refresco de la escena y solo sea ejecutado cada cierto intervalo de tiempo configurado.
- Proporciona un componente de animación que permite crear dinamismo de forma sencilla y configurable.
- Contiene componentes para los principales controladores de realidad virtual del mercado.
- Proporciona primitivas *HTML5* para las principales geometrías como son planos, cajas, círculos, conos, cilindros, esferas, texto, entre otros.

- Gestión de eventos mediante *Raycaster* que proporciona facilidad de interactuar con las entidades presentes en la escena.
- Capacidad de creación de componentes propios o extender alguno existente de manera sencilla e intuitiva.

En definitiva, las capacidades técnicas de A-FRAME están en continuo crecimiento gracias al gran ecosistema que hay de desarrollo a su alrededor, permitiendo nutrirse también del crecimiento de la tecnología principal, THREE.JS.

#### 2.11.4. Contexto y aplicación en el prototipo

A-FRAME es el motor principal de *renderizado* del prototipo, ya que toda la aplicación está construida haciendo uso de la *API* que proporciona esta librería. Al estar basada en JAVASCRIPT, el gestor principal sobrescribe el componente de la escena y define el comportamiento principal de nuestra aplicación mediante el uso de la API de JavaScript en conjunto con los componentes de A-FRAME y THREE.JS.

#### 2.11.5. Ejemplos de uso

```
<html>
<head>
<script src="https://aframe.io/releases/1.4.0/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>
```

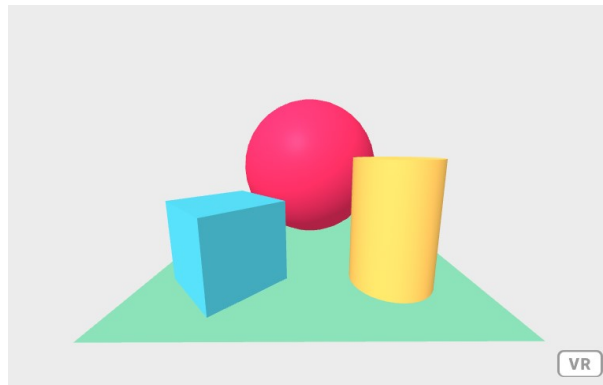


Figura 2.3: Escena principal del *Getting Started* de A-FRAME.

## 2.12. Leaflet

### 2.12.1. Introducción a la tecnología

LEAFLET es una biblioteca creada por *Volodymyr Agafonkin* en JAVASCRIPT y de código abierto. Su principal funcionalidad está orientada a la creación de aplicaciones de mapas web. Fue lanzada por primera vez en 2011 y es compatible con la mayoría de las plataformas móviles y de escritorio. Está tan extendida y es tan liviana que empresas como *FOURSQUARE*, *PINTEREST* y *FLICKR* la integran dentro de su arquitectura. LEAFLET permite a los desarrolladores mostrar de manera muy sencilla mapas web basados en teselas alojadas en un servidor público. Sobre estos mapas nos permite añadir capas que muestren información creando aplicaciones de manera sencilla y liviana.

### 2.12.2. Contexto y aplicación en el prototipo

Se ha utilizado esta librería para implementar las conversiones entre coordenadas geodésicas y cartesianas, como se describe en la sección 4.4.1. Al utilizar esta librería, nos beneficiamos del soporte técnico que proporciona, asegurándonos de contar con los ajustes necesarios para las fórmulas que desempeñan un papel importante en nuestra aplicación. Además, nos aseguramos de que las conversiones se realicen de la manera más eficiente posible para proporcionar un rendimiento óptimo en entornos móviles.

### 2.12.3. Ejemplos de uso

A continuación, se presenta un ejemplo de cómo realizamos la conversión de coordenadas geodésicas a cartesianas utilizando la proyección MERCATOR a través del objeto global *L* que nos proporciona un punto de entrada a las funcionalidades de la librería.

```
degreeToMeter(lat, long) {  
    let latLng = new L.LatLng(lat, long);
```



```

        return L.Projection.Mercator.project(latlng);
    }

```

## 2.13. GDAL

### 2.13.1. Introducción a la tecnología

GDAL<sup>4</sup> es una biblioteca de código abierto que proporciona un conjunto de herramientas para la lectura, escritura y manipulación de datos geoespaciales en diversos formatos. Su desarrollo inicial se centró en facilitar el acceso y procesamiento de datos geoespaciales en diferentes formatos y proyecciones. Esta biblioteca es ampliamente utilizada en numerosos proyectos que requieren acceder a datos geográficos, como MAPSERVER<sup>5</sup>, QGIS<sup>6</sup> y GRASS GIS<sup>7</sup>. Estas aplicaciones hacen uso de la funcionalidad proporcionada por GDAL para gestionar y visualizar datos geoespaciales de manera efectiva y eficiente.

### 2.13.2. Contexto y aplicación en el prototipo

Como se explica en la sección 4.5.1, utilizaremos esta tecnología para llevar a cabo las conversiones de los datos descargados del servidor de COPERNICUS<sup>8</sup>. Estos datos consisten en archivos binarios de alturas, y mediante el uso de esta biblioteca, realizaremos la unión y recorte de una zona específica que representará el terreno de nuestro escenario.

El objetivo es exportar los datos resultantes en el formato binario de alturas requerido por el *componente generador de terrenos*<sup>9</sup> de la biblioteca A-FRAME.

## 2.14. Google Earth Engine

### 2.14.1. Introducción a la tecnología

Es una plataforma geográfica basada en la nube que brinda a los usuarios la capacidad de operar, visualizar, analizar y exportar imágenes **Raster** de nuestro planeta. Esta plataforma es ampliamente utilizada por científicos y organizaciones sin ánimo de lucro para acceder a información geográfica en forma de datos raster y realizar preprocesamiento y exportación de los mismos.

<sup>4</sup><https://gdal.org/>

<sup>5</sup><https://mapserver.org/>

<sup>6</sup><https://www.qgis.org/es/site/>

<sup>7</sup><https://grass.osgeo.org/>

<sup>8</sup><https://land.copernicus.eu/imagery-in-situ/eu-dem/eu-dem-v1.1>

<sup>9</sup><https://github.com/bryik/aframe-terrain-model-component>

Los datos geográficos se almacenan en servidores en la nube y se puede acceder a ellos y analizarlos mediante una interfaz de programación de aplicaciones API y un entorno de desarrollo integrado IDE en la nube<sup>10</sup>. Esta plataforma ofrece a los usuarios una serie de herramientas y funcionalidades que les permiten trabajar con los datos de manera eficiente y precisa.

### 2.14.2. Contexto y aplicación en el prototipo

Como se mencionará en detalle más adelante en la sección 4.5.2, esta tecnología se ha utilizado para llevar a cabo la exportación de la capa raster que se utilizará como textura del terreno generado en nuestro escenario. Es importante destacar que esta capa raster estará georreferenciada, lo que permitirá a los usuarios identificar y comprender la ubicación específica en la que se están visualizando los datos dentro del escenario. La georreferenciación garantiza que el usuario puedan tener una referencia espacial precisa dentro del escenario. Además, las alturas, como montañas o cañones, estarán correctamente posicionadas, lo que proporcionará una sensación realista al visualizar el terreno generado en 3D.

## 2.15. Overpass-api

### 2.15.1. Introducción a la tecnología

Es una API que permite acceder a través de consultas a los datos alojados en los servidores de OPENSTREETMAP. Utilizando un lenguaje similar al de las bases de datos, podemos filtrar la información y extraer los datos en formato XML. Posteriormente, podemos utilizar herramientas como OSM-TOGEOJSON<sup>11</sup> para convertirlos en archivos GEOJSON, los cuales nuestra aplicación puede aprovechar para representar datos geoespaciales de manera efectiva.

### 2.15.2. Contexto y aplicación en el prototipo

Como se detalla en la sección 4.5.4, utilizaremos la página de consultas que nos proporciona la plataforma<sup>12</sup> para descargar y procesar los datos de los edificios ubicados en la zona específica de nuestro escenario y alojados en los servidores de OPENSTREETMAP. Nuestro objetivo será realizar un preprocesamiento de estos datos y subirlos en nuestra aplicación. De esta manera, el prototipo podrá generar las geometrías de los edificios durante la precarga, garantizando así una representación visual precisa de los edificios en el entorno 3D.

---

<sup>10</sup><https://code.earthengine.google.com/>

<sup>11</sup><https://github.com/tyrasd/osmtogeojson>

<sup>12</sup><https://overpass-turbo.eu/>

## 2.16. GitHub

### 2.16.1. Introducción a la tecnología

GITHUB es una plataforma de desarrollo colaborativo basada en la tecnología de control de versiones GIT<sup>13</sup>. Proporciona a los desarrolladores un entorno donde pueden compartir y colaborar en sus proyectos de software de manera eficiente. Una de las características clave de GITHUB es su capacidad para trabajar con ramas. Estas permiten que varios desarrolladores trabajen en diferentes aspectos de un proyecto de forma colaborativa y paralela. Cada desarrollador puede crear su propia rama para trabajar en nuevas funcionalidades, correcciones de errores o mejoras, sin interferir con el trabajo de otros colaboradores. Una vez que los cambios en una rama se consideran estables y completos, pueden fusionarse con la rama principal del proyecto.

### 2.16.2. Contexto y aplicación en el prototipo

Todo el desarrollo del prototipo de la aplicación 3D, incluyendo las demos y los componentes, se encuentra alojado en un repositorio de GITHUB<sup>14</sup>. Esta elección se ha realizado con el propósito de mantener un control de versiones y compartir el código fuente del proyecto. Además también cuenta con la opción de iniciar un servidor de aplicaciones, lo que facilita las demostraciones y la visualización de la aplicación en acción.

## 2.17. VSCode

VSCODE es la herramienta principal que se utiliza como entorno de desarrollo para generar el código del proyecto. Proporciona una variedad de funcionalidades gracias a sus plugins que facilitan el proceso de desarrollo. Además, nos permite arrancar un servidor local ligero para ejecutar nuestra aplicación en modo de depuración, lo que nos permite identificar y corregir errores de manera eficiente. Una de las ventajas de VSCODE es su integración con herramientas de control de versiones como la que analizamos en la sección 2.16. Además, VSCODE ofrece características útiles como autocompletado de código y resaltado de sintaxis, lo que agiliza el desarrollo del prototipo y mejora la productividad.

---

<sup>13</sup><https://git-scm.com/>

<sup>14</sup><https://github.com/djprano/AFrameTFG>



## Capítulo 3

# Resultados

### 3.1. Prototipos de demostración

#### 3.1.1. Escritorio

#### 3.1.2. Gafas de realidad virtual

#### 3.1.3. Móvil

### 3.2. Progreso de desarrollo

Para modelar el proceso de desarrollo del proyecto, me he basado en la metodología de planificación SCRUM. Esta metodología hace uso de un marco de trabajo ágil, introducido por primera vez en la década de los 90 por *Ken Schwaber* y *Jeff Sutherland* como una herramienta para abordar proyectos complejos. El enfoque principal de SCRUM se fundamenta en la realización de iteraciones incrementales. Esto implica dividir el proyecto en tareas más pequeñas y de corta duración conocidas como *sprints*. Cada *sprint* tiene una duración estimada en la cual se llevan a cabo actividades como la planificación, el desarrollo, las pruebas y la revisión.

En el ámbito laboral, es muy común utilizar esta metodología en proyectos tecnológicos, y muchas empresas ofrecen productos para la planificación de proyectos. Uno de los más ampliamente utilizados es JIRA<sup>1</sup> de ATlassian, que cuenta con una guía de la metodología muy bien documentada [2].

Algunas de las características que utilizaremos de *Scrum* son las siguientes:

- **Roles:** se define tres roles principales: *Product Owner* (Dueño del producto), el *Scrum Master* (responsable de que se cumpla la metodología) y el equipo de desarrollo (desarrolladores).

---

<sup>1</sup><https://www.atlassian.com/software/jira>

- **Backlog:** registro de tareas pendientes, que deberán estar estimadas y priorizadas.
- **Artefactos:** se utiliza varios artefactos para gestionar el trabajo. El más importante es el *Product Backlog*, que es una lista priorizada de todas las funcionalidades, mejoras y tareas pendientes. Otros artefactos incluyen el *Sprint Backlog*, que contiene las tareas seleccionadas para el *sprint* actual, y el Incremento, que es la versión funcional del producto al finalizar cada *sprint*.
- **Reuniones:** la metodología incluye varias reuniones estructuradas para facilitar la colaboración y la toma de decisiones. Estas reuniones incluyen la *Sprint Planning*, donde se define el alcance del SPRINT actual, la *Daily Scrum*, una breve reunión diaria de seguimiento, y la *Sprint Review*, donde se revisa y se demuestra el trabajo realizado durante el *sprint*.

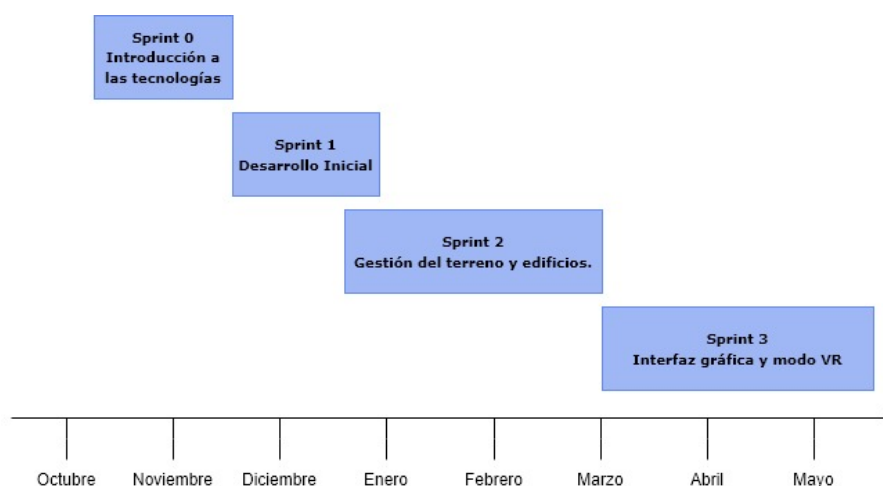


Figura 3.1: Planificación temporal del desarrollo del prototipo.

En la figura 3.1 podemos visualizar en el tiempo como se han distribuido las tareas principales. A continuación se presentará una breve introducción de cada uno de los sprints completados en el proyecto:

- **Sprint 0:** En esta etapa, se lleva a cabo una investigación exhaustiva de las tecnologías disponibles para abordar los requisitos del proyecto. Puesta en marcha del entorno de desarrollo y realiza formación para familiarizarse con las librerías y el lenguaje de programación, se realizan experimentos y se busca componentes reutilizables.
- **Sprint 1:** En esta etapa, se sientan las bases para el inicio del prototipo. Se comienza a crear los módulos y componentes que tendrán la funcionalidad principal de mover aviones utilizando datos de [APIs](#).

- **Sprint 2:** En esta etapa, se enfoca en el desarrollo de la gestión del terreno. Se crea el suelo del escenario con representación realista mediante alturas y texturas. Se desarrollan los módulos encargados del cálculo de alturas y la generación de edificios utilizando datos de OPENSTREETMAPS.
- **Sprint 3:** En esta etapa, se pone énfasis en la interfaz gráfica. Se crea un ecosistema de componentes reutilizables que permiten al usuario visualizar información, activar y desactivar funcionalidades de la aplicación. Además, se desarrollan los componentes necesarios para la interacción con el escenario, ya sea mediante ratón y teclado o mediante gafas y mandos de realidad virtual.

### 3.2.1. Sprint 0

#### 3.2.1.1. Objetivos

- Montar un entorno de programación para aplicaciones web con un servidor liviano para poder arrancar aplicaciones web en local y poder depurar con inspección de variables y puntos de ruptura.
- Investigar tecnologías que vamos a necesitar para el desarrollo del prototipo.
- Leer documentación de las tecnologías y realizar los ejemplos de la documentación para familiarizarse con la API que nos proporcionan.
- Leer libro sobre JAVASCRIPT y Documentación para refrescar conocimientos sobre el lenguaje.

#### 3.2.1.2. Desarrollo

Se tomó la decisión de utilizar el editor de código VSCODE como entorno de desarrollo, debido a la familiaridad previa con esta herramienta en cursos de JAVASCRIPT. Se instaló VSCODE y el servidor de aplicaciones LIVE SERVER para ejecutar la aplicación en modo de depuración local.

El uso de la librería A-FRAME fue un requisito del *Product Owner*, por lo que se siguieron los ejemplos de la documentación y se realizó el proceso de inicio rápido (*Getting started*). Se instaló GIT y se creó un repositorio en GITHUB para mantener un control de versiones y compartir el código fuente del prototipo. Además, se configuró un *GitHub Pages* para mostrar la evolución del desarrollo al *Product Owner*.

Se leyó la documentación de THREE.JS para conocer las herramientas disponibles para futuras implementaciones de componentes de A-FRAME. Se realizó el curso interactivo de A-FRAME disponible en <https://mozilla.pe/aframe-school/>, lo cual ayudó a resolver muchas dudas y obtener un mejor entendimiento de la tecnología. Se comenzó a crear animaciones y realizar pruebas con la ges-

ción de eventos del ratón en A-FRAME. Se consultó un *workshop* interesante sobre A-FRAME y WEBXR en <https://github.com/german-alvarez-dev/workshop-webvr-aframe>.

Para abordar el requisito de la creación de un terreno se investigó y se encontraron varios ejemplos de creación de terrenos con A-FRAME que podrían ser útiles para el prototipo. Algunos de ellos son:

- <https://github.com/DougReeder/aframe-atoll-terrain>
- <https://github.com/jesstelford/aframe-map>
- <https://github.com/bryik/aframe-terrain-model-component>
- <https://bryik.github.io/aframe-terrain-model-component>
- <https://github.com/anselm/aterrain>
- <https://cesium.com/ion>

Después de realizar varias pruebas, se concluyó que el componente *aframe-terrain-model-component* era el más adecuado debido a su facilidad de uso y capacidad para manejar archivos DEM en formato ENVI.

Se abordó el requisito de obtener datos de los aviones de una API en línea, y se estudiaron las dos principales opciones de datos ADS-B: *FlightRadar24* y *OpenSky*. Tras evaluar las limitaciones y considerando que *OpenSky* era gratuita y tenía menos restricciones, se decidió utilizar esta API. Se instala NODE.JS y se desarrolla un programa simple para probar el servicio web y para almacenar datos localmente y poder trabajar de forma independiente de las limitaciones impuestas por la API.

### 3.2.1.3. Resultado

Después de realizar todas estas actividades, se llevó a cabo una reunión con el *Product Owner*, quien mostró satisfacción con el progreso y consideró que el equipo estaba listo para abordar el siguiente *sprint*, que implicaría el comienzo del desarrollo del prototipo y cerraría la etapa de investigación de tecnologías.

## 3.2.2. Sprint 1

### 3.2.2.1. Objetivos

- Desarrollar un módulo que pueda leer los datos locales almacenados por el proceso batch. Además, este módulo debe ser configurable de manera que podamos acceder a datos locales o a datos en tiempo real del servidor.



- Encontrar un modelo GLTF de un avión con una licencia que permita su uso y que sea ligero para no afectar el rendimiento. Este modelo debe ser incorporado en la escena y comenzar a posicionarlo en función de los datos ADS-B que se han leído.
- Crear animaciones que permitan interpolar linealmente la posición del modelo de avión, evitando cambios bruscos de posición.
- Desarrollar un módulo para gestionar la configuración de la escena de manera que se pueda crear aplicaciones para diferentes escenarios utilizando la misma plataforma, simplemente modificando la configuración.
- Implementar la capacidad de seleccionar aviones dentro de la escena.

### 3.2.2.2. Desarrollo

Se buscó en la página de SKETCHFAB un modelo de avión y se encontró un modelo GLTF<sup>2</sup> de uso libre compuesto por tan solo 416 triángulos, el cual se utilizará como icono para los vuelos. Se creó el componente principal que sobrescribe el comportamiento de la escena principal y actúa como el gestor principal de la aplicación. Para obtener más detalles sobre el diseño de este componente, consulta la sección 4.2.1. Se realizaron las primeras animaciones de los aviones, pero surgieron algunos problemas. El sistema A-FRAME está diseñado para trabajar con unidades en metros. Sin embargo, al utilizar coordenadas absolutas, se encontraron dificultades al manejar posiciones con vectores de dimensiones muy grandes, como el caso del aeropuerto en Madrid, cuyas coordenadas cartesianas son  $(-396861, altura, 4937984)$ . Estas dimensiones resultan difíciles de manejar. Además, se identificó que uno de los ejes está invertido por el sistema de referencia que posee A-FRAME, como se explica detalladamente en la sección 4.4.1. Después de una reunión con el *Product Owner*, se decidió crear un módulo dedicado a gestionar las conversiones entre las coordenadas geodésicas y las coordenadas en el mundo 3D. En este módulo, se invirtió el eje problemático y se realizó una traslación para centrar el escenario en las coordenadas  $(0, 0, 0)$ . Además, se aplicó un escalado para manejar distancias en escenarios de grandes dimensiones, como cientos de kilómetros. Todas estas transformaciones se realizaron de manera parametrizada a través de la configuración. Se comenzaron a crear las primeras animaciones de los aviones con éxito sobre un escenario que representa el aeropuerto de Barajas y sus alrededores. También se generaron las primeras siluetas sin extruir en el suelo para comprobar que las posiciones de los aviones coincidían con las posiciones de los edificios, como se muestra en la figura 3.2. Además, se inició la captura de un *Raster* para incorporar una referencia en forma de mapa de OPENSTREETMAP en el suelo, lo cual permitirá verificar si se

<sup>2</sup><https://sketchfab.com/3d-models/low-poly-plane-151517395bae4d849b30ca53a5e3c5a8>

están creando las geometrías en el lugar correcto. Se implementan en el gestor principal de la escena las primeras interacciones con eventos. Se ha creado una entidad en forma de cono que se posiciona encima del avión para indicar visualmente el avión sobre el cual tenemos el ratón. Esta funcionalidad permite al usuario identificar fácilmente qué avión está siendo seleccionado o resaltado.

### 3.2.2.3. Resultado

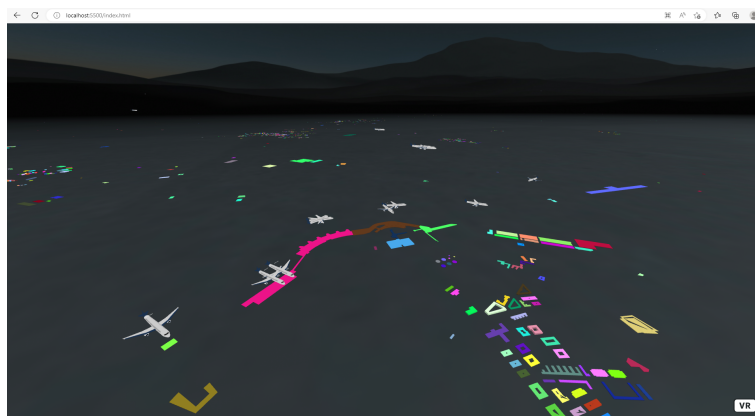


Figura 3.2: Versión inicial.

Durante la reunión final del *sprint* con el *Product Owner*, se ha llegado a la conclusión de que se han cumplido satisfactoriamente los objetivos establecidos. Se considera que se ha logrado una buena base para continuar el desarrollo, especialmente con la incorporación de las siluetas en el terreno de la escena que es parte del desarrollo para el próximo *sprint* donde abordaremos la generación de un módulo que gestiona el terreno.

### 3.2.3. Sprint 2

#### 3.2.3.1. Objetivos

- Generar un archivo DEM y un *Raster* para los escenarios de Madrid y Vatry.
- Integrar en la plataforma el componente *aframe-terrain-model*<sup>3</sup>, el cual genera un mallado a partir de un archivo binario de alturas.
- Desarrollar un módulo para la gestión de terrenos.
- Crear un segundo escenario adaptando el gestor de configuración de la aplicación para permitir la cargar toda la plataforma tan solo modificando un fichero en el HTML principal.

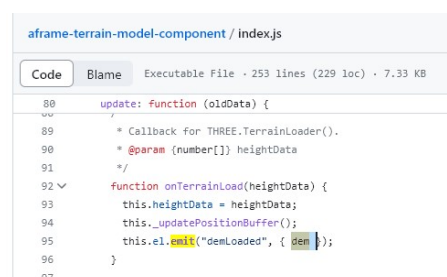
<sup>3</sup><https://github.com/bryik/aframe-terrain-model-component>

- Utilizar THREE.JS para crear geometrías complejas y extruir los edificios en alturas específicas, teniendo en cuenta la altura del terreno.
- Adaptar el gestor principal para considerar la altura del terreno al posicionar los aviones.

### 3.2.3.2. Desarrollo

Se generan los ficheros DEM y *Raster* con éxito siguiendo el procedimiento descrito en las secciones 4.5.1 y 4.5.2. Para integrar el componente de terrenos en la plataforma, hemos hecho que los parámetros del componente binario de alturas y *raster* del terreno sean configurables, como también la anchura, altura y la magnificación del eje de altura.

Uno de los problemas actuales es que nuestro gestor de alturas debe conocer el valor de altura en cada punto generado por el componente de terreno. Hemos llevado a cabo una investigación consultando el código fuente del componente, y hemos encontrado un evento interno que podemos reutilizar para acceder a la variable de alturas guardada dentro del objeto DEM, como se muestra en la figura 3.3. Después de ver que los datos almacenados en la variable eran valores extraños e incomprensibles, logramos entender la lógica del *array* de datos. El problema residía en que el *array* era del tipo *Uint16Array* y las alturas aplicadas seguían una lógica de normalización del valor del archivo binario multiplicado por el factor de magnificación de alturas. Al insertar esta lógica dentro del nuevo gestor de alturas, realizamos una prueba generando esferas en cada punto del archivo y observamos que se posicionaron correctamente sobre el mallado. Hemos agrupado toda esta lógica en el archivo *heightManager.js*, que contendrá la instancia única encargada de gestionar la carga del terreno y la API de alturas. Descargamos los datos de edificios y los preprocesamos según se detalla en la sección 4.5.4 para poder usarlos en el gestor de terreno.



```
afre-terrain-model-component / index.js
Code Blame Executable File · 253 lines (229 loc) · 7.33 KB
88 update: function (oldData) {
89   //
90   * Callback for THREE.TerrainLoader().
91   * @param {number[]} heightData
92   */
93   function onTerrainLoad(heightData) {
94     this.heightData = heightData;
95     this._updatePositionBuffer();
96     this.el.emit("demLoaded", { dem: this.heightData });
97   }
98 }
```

Figura 3.3: Evento clave que emite el componente de terreno.

Se realizan todos las modificaciones finales en el modulo de configuración para poder parametrizar la plataforma para otro escenario, y se descargan vuelos, rater y fichero de alturas y edificios tal y como se detalla finalmente en el manual de usuario para generar el escenario de Vatry (Francia) y poner a prueba

la versatilidad de la plataforma.

### 3.2.3.3. Resultado

Podemos observar en la figura 4.21 como cargamos una demo simple con los ficheros generados del binario de alturas con su respectivo **Raster**.

Se puede visualizar la demo en el siguiente enlace [https://djprano.github.io/AFrameTFG/demos/madrid\\_terrain.html](https://djprano.github.io/AFrameTFG/demos/madrid_terrain.html). Se realiza la reunión final con el *Product Owner* concluyendo que se han cumplido todos los objetivos cerrando el desarrollo y definiendo los próximos objetivos para el siguiente sprint.

### 3.2.4. Sprint 3

#### 3.2.4.1. Objetivos

- Insertar una geometría que envuelva el cuerpo del avión, indicando cuál está seleccionado y que siempre quede mirando hacia la cámara.
- Realizar un panel **HUD** donde visualizar la información de los vuelos seleccionados, refrescándose cuando haya cambios mediante eventos.
- Implementar una barra de herramientas para habilitar y deshabilitar funcionalidades.
- Encontrar una solución al problema de selección de aviones muy lejanos que son demasiado pequeños.
- Almacenar el trayecto que siguen los aviones en una caché desde que entran al escenario y desarrollar una funcionalidad en el **HUD** para visualizar el trayecto.
- Insertar una funcionalidad que permita visualizar la perspectiva de lo que se está viendo desde el vuelo.
- Permitir al usuario mover el panel **HUD** para colocarlo en una posición donde no le moleste.
- Hacer que la aplicación sea compatible con el modo VR y que sea funcional con unas gafas de realidad virtual.
- Realizar una implementación que permita al usuario visualizar los metadatos de los edificios.

### 3.2.4.2. Desarrollo

Se crea el componente **HUD** dentro del fichero *hud.js*, el cual va a concentrar toda la lógica del panel de información contextual interactiva y gestión de eventos de selección de aviones, así como la transmisión de información y actualización de los metadatos del vuelo que se visualizan en el panel.

Finalmente, se decide usar la primitiva *Ring* para rodear al avión seleccionado, tal y como se muestra en la figura 4.38. Para cumplir con el requisito de que siempre mire a cámara, encontramos el componente *look-at*<sup>4</sup>, el cual ofrece los resultados esperados, por lo tanto, se decide no realizar ninguna implementación adicional al respecto.

Para el panel contextual, vamos a implementar un panel de A-FRAME donde se irán insertando componentes de texto delante, tal y como se detalla en la sección 4.7.4.

Aquí nos topamos con el primer problema en este *sprint*, del cual nos damos cuenta afortunadamente debido a que cuando posicionamos el ratón sobre un elemento de la clase *clickable*, el cursor cambia a una mano. Esto nos hace darnos cuenta de que en ciertas zonas de la pantalla el cursor cambia a mano, pero no hay nada visible que obstaculice la interacción con el elemento. Después de verificar que el panel estaba oculto en la misma zona, llegamos a la conclusión de que el culpable es el panel en modo invisible.

Hemos elegido una estrategia en la que el panel del **HUD** es un elemento *clickable* y, por tanto, intercepta los eventos del *Raycaster*. Cuando el panel no está visible y está delante del usuario tapando un avión, aunque no vemos que el panel está tapando, al estar invisible, el panel consume los eventos y no podemos seleccionar los aviones.

Solucionamos este problema al posicionar el panel detrás del usuario en lugar de hacerlo invisible. En resumen, cuando el panel recibe el evento de cerrar, se coloca detrás del usuario en una posición donde nunca puede ser visible. De esta manera, deja de interferir en los eventos del controlador principal y podemos seleccionar los objetos sin problemas. Además, cuando el panel recibe un evento de elemento seleccionado y cambia su posición de oculto a visible, guarda la última posición en la que estaba posicionado antes de recibir el evento de ocultar. De esta manera, cada vez que se oculta el panel, la instancia almacena la última posición para su posterior uso cuando reciba el evento de avión seleccionado.

Desarrollamos una barra de herramientas de dos dimensiones para gestionar la activación y desactivación de funcionalidades. En un principio, esta barra se implementó como un elemento *div* en el HTML principal. Sin embargo, más adelante, cuando probé el prototipo en modo VR utilizando las gafas OCU-LUS, me di cuenta de que los elementos *div* desaparecían en dicho modo.

Por lo tanto, cambiamos nuestra estrategia y comenzamos a desarrollar una solución en la que in-

---

<sup>4</sup><https://www.npmjs.com/package/aframe-look-at-component>

tegraremos una barra de herramientas 3D que estará presente como entidades geométricas dentro de la escena, siguiendo la misma filosofía que el HUD vease la sección 4.7.3. Esta barra de herramientas contendrá dos botones para activar o desactivar opciones, es decir, serán conmutables, y tendrá un tercer botón cuya función será plegar el panel con una animación para ocupar menos espacio y no interferir en el campo visual del usuario.

En esta etapa del desarrollo, también abordamos otro problema de la aplicación, que es la limitación de seleccionar vuelos que se encuentran demasiado lejos en el escenario. Para resolver esto, desarrollamos un componente llamado *hover-scale*, del cual proporcionamos una descripción técnica en la sección 4.7.1. Configuraremos este componente en todas las entidades de aviones, y será responsable de ajustar el factor de escala del avión en función de la distancia con el usuario. De esta manera, cumplimos con el requisito establecido de facilitar al usuario la selección de los vuelos.

Para implementar el requisito de mostrar el trayecto que realiza un avión, creamos un nuevo componente que se agrega a la entidad avión. Este componente se suscribe a los eventos de cambios de posición, y haciendo uso de las ventajas de implementar un DTO (*Data Transfert Object*) para manejar el cacheo de los aviones, llamado *FlightCacheData*, insertamos la lógica en este DTO para que cuando se realiza un cambio de posición, envíe un evento. De esta manera, el componente recibe el evento y actualiza la geometría del trayecto. Tan solo tendremos que enviar otro evento desde el panel para visualizarlo, mediante un botón conmutable para mostrar u ocultar el trayecto.

Para ver con más detalle la implementación de esta parte, puedes consultar la sección 4.6.1 y la sección 4.7.6.

Para implementar el requisito de visualizar una perspectiva de cámara a bordo del avión seleccionado, decidimos hacer uso de un artículo y unos componentes disponibles en la guía de introducción de A-FRAME de Jesús María González Barahona [7]. En esta guía, se explica cómo crear texturas con la visualización de cámaras secundarias.

Haciendo uso del componente *camrender*, cuando el usuario pulsa en el botón conmutable que activa esta opción, creamos una cámara dentro del avión con dicho componente. Luego, utilizando animaciones, creamos un panel al que le asignamos el material de la cámara de a bordo y realizamos una ampliación para crear un efecto de despliegue. Finalmente, colocamos el panel en el HUD. De esta forma, implementamos el requisito de visualizar la cámara a bordo de una manera visualmente atractiva, como se muestra en la figura 4.40.

Para el requisito de visualizar los datos de los edificios, se ha desarrollado un componente reutilizable llamado *tooltip-info*. Este componente recibe como argumento un objeto de tipo cadena *String* que se mostrará cuando se reciba un evento de posicionamiento del cursor encima de la entidad correspondiente.

Además, el componente modifica el material para indicar cuál es la entidad que se está visualizando, puede consultarse la sección 4.7.2 para más detalle sobre la implementación.

En resumen esta funcionalidad, al crear los edificios, se procesan los metadatos provenientes de la información de OPENSTREETMAP. Se genera una cadena de texto que se pasa como parámetro al componente *tooltip-info*, asignándolo a la entidad correspondiente. El componente *tooltip-info* calcula la posición más alta de la geometría del edificio y crea un texto que se muestra de manera visible y siempre mirando hacia la cámara cuando el cursor se posiciona encima del edificio.

El último desafío al que nos enfrentamos fue el de mover las entidades HUD que existen dentro de la estructura del usuario, ya sea a través del ratón o de los controladores de realidad virtual de las gafas OCULUS que nos proporcionó el universidad. En un principio, consideramos utilizar el componente *aframe-super-hands-component*<sup>5</sup>, ya que es ampliamente utilizado en la comunidad de desarrollo de A-FRAME. Comenzamos a integrar este componente en nuestro prototipo, pero resultó ser un fracaso. Los movimientos solo funcionaban cuando la cámara miraba en una dirección, y a medida que se movía la cámara, los movimientos se volvían incoherentes.

Después de investigar el código fuente de la biblioteca, llegamos a la conclusión de que el componente no era adecuado para nuestro caso, ya que estábamos intentando mover una entidad que no tiene coordenadas absolutas, sino coordenadas relativas a la cámara. Además, los componentes *movement-controls* establecidos en la entidad *Rig* en combinación con el componente *look-controls* en la cámara hacían que las posiciones relativas de la cámara tampoco fueran absolutas. También debíamos asegurarnos de que al arrastrar una entidad, no moviéramos la cámara, ya que el movimiento natural del ratón se utiliza para desplazar la orientación cámara.

Para solucionar este requisito, implementamos un componente personalizado que tuviera en cuenta todas estas peculiaridades y nos permitiera arrastrar objetos ubicados jerárquicamente dentro de la entidad de la cámara. También, para evitar la rotación de la cámara mientras se arrastra una entidad, deshabilitamos el componente *look-controls* mientras se está realizando el arrastre del elemento HUD.

Para obtener más detalles sobre cómo se implementó este componente *custom-draggable*, consulta la sección 4.7.5.

### 3.2.4.3. Resultado

Se lleva a cabo una reunión final con el *Product Owner* para mostrarle el prototipo y se finalizan todos los desarrollos, cumpliendo con los requisitos establecidos en este *sprint*. Posteriormente, se procede a cerrar las etapas de desarrollo con el objetivo de completar la documentación final.

---

<sup>5</sup><https://github.com/c-frame/aframe-super-hands-component>

### 3.3. Construcción de escenas

Aquí presentaremos un resumen del procedimiento necesario para crear un escenario y configurar la plataforma proporcionada por el prototipo. Además, en la sección 3.4, describiremos los componentes que se pueden reutilizar en otros proyectos y cómo hacerlo.

#### 3.3.1. Generación de datos para el terreno

El primer paso consiste en determinar las coordenadas geodésicas para nuestro escenario. Es importante intentar definir un rectángulo con coordenadas que formen un plano lo más cuadrado posible. Esto se debe a que, aunque el cálculo se basa en ambas dimensiones, la cúpula del cielo se construye utilizando la coordenada más corta para asegurarse de que no tengamos una parte del área sin cubrir en el horizonte. Un escenario con proporciones cuadradas evitará tener un exceso de terreno fuera de la cúpula, lo cual podría afectar al rendimiento. En resumen, se recomienda generar coordenadas cuadradas en la medida de lo posible. Una vez que tengamos las coordenadas geodésicas de nuestro escenario, procederemos a extraer el archivo DEM (Modelo de Elevación Digital) y el archivo [Raster](#) para configurar correctamente los archivos necesarios para crear el terreno. A continuación, seguiremos los pasos enumerados en el manual 4.5.1 para generar un archivo binario de alturas basado en las coordenadas seleccionadas. Para la generación del archivo raster, utilizaremos el script proporcionado en la figura 4.5.2, el cual se basa en el servicio de GOOGLE [5]. Asegúrate de reemplazar las coordenadas establecidas en el script con las correspondientes a tu escenario.

#### 3.3.2. Generación de edificios para el terreno

El siguiente paso consiste en generar los edificios que queremos que aparezcan dentro de nuestro escenario. Para ello, seguiremos estos pasos:

1. Utilizando el *endpoint* 4.5.4 y sustituyendo las coordenadas por las del escenario, descargaremos los metadatos del servidor de OPENSTREETMAPS en formato *OSM*. Esta consulta se puede realizar desde el Navegador o utilizando un software para hacer peticiones, como POSTMAN, que nos permita guardar el resultado de la consulta. Es importante tener en cuenta que el tamaño del archivo dependerá del escenario, y si se trata de una zona con muchos edificios, podría llegar a pesar cientos de *Megabytes*.

Para evitar problemas de rendimiento en la aplicación, se recomienda aplicar algún tipo de filtrado adicional para obtener una lista reducida de edificios.



2. El siguiente paso consiste en procesar el archivo para generar un formato de archivo GEOJSON compatible con nuestra plataforma. Para ello, podemos instalar la librería OSMTOGEOJSON utilizando *npm* y ejecutar el comando en la consola, tal como se indica en el script 4.5.4.

### 3.3.3. Almacenar metadatos de vuelos dentro del escenario (Opcional)

El siguiente paso es necesario solo en caso de que deseemos ejecutar el escenario en modo *offline* (sin conexión). Siempre que queramos utilizar la aplicación en modo tiempo real, este paso puede omitirse. Si deseamos levantar la aplicación en modo *offline*, es necesario contar con un archivo de vectores de posición de vuelos guardados en una carpeta configurada. Para guardar los datos de vuelos para las coordenadas del escenario, ejecutaremos el proceso por lotes *openSkyDataSaver.js*. En la sección 4.6.3, se describe el proceso realizado, pero veamos más detalles a continuación.

Debemos crear un archivo principal que cargue la configuración necesaria para ejecutar el proceso por lotes y que llame al método principal de la clase para inicializar el proceso. En la siguiente figura se muestra un ejemplo explicado con comentarios en las líneas principales.

```
import * as configuration from './configuration/configurationModel.js';
import * as openSkyDataSaver from './openSkyDataSaver.js';

//Establecemos las coordenadas del escenario.
configuration.setMerConfig(40.0234170, 40.7441446, -4.2041338, -3.2538165);
//Ruta absoluta a la carpeta donde queremos almacenar los datos de vuelo
configuration.setFlightLocalFolder('C:\\Users\\djpra\\Documentos\\workspaceTFG\\.....');
//Usuario de la API OpenSky si no se posee uno simplemente hay que registrarse.
configuration.setApiUser('xxxxxx');
//Contraseña de la web OpenSky.
configuration.setApiPassword('xxxxxxx');
//Intervalo entre peticiones, recordar que gratuitamente nos e permite vectores de
↔ posición menos a 5seg.
configuration.setDaoInterval(5100);
//Lanzamos el proceso.
openSkyDataSaver.main();
}
```

Después para lanzar el proceso por lotes simplemente ejecutar el siguiente comando en consola contra el fichero anteriormente explicado:

```
node ficheroPrincipal.js
```

### 3.3.4. Generación de fichero de configuración

Ahora vamos a crear un archivo de configuración en formato *JavaScript* donde estableceremos todos los parámetros de nuestra plataforma. A continuación, se muestra un ejemplo detallado con cada uno de

los parámetros:

```
import * as configuration from './configurationModel.js';
//Coordenadas del escenario latmin, latmax, longmin, longmax.
configuration.setMerConfig(40.0234170,40.7441446,-4.2041338,-3.2538165);
//Coordenadas de la posición inicial del usuario.
configuration.setCamPosition(40.4893, -3.52254);
//Nombre del fichero de edificios sin extensión ubicado en la carpeta
//"data" del proyecto.
configuration.setBuildingFileName('madrid_building');
//sufijo de la carpeta que contiene los vuelos, debe contener el prefijo "flightData"
//Opcional solo para uso offline.
configuration.setFlightLocalFolder('_madrid');
//fichero con la capa raster del terreno ubicado en la carpeta
//"data" del proyecto.
configuration.setMapRaster('Madrid_raster.jpg');
//fichero binario de alturas para el terreno ubicado en la carpeta
//"data" del proyecto.
configuration.setMapDem('madrid_dem.bin');
//Establece si queremos lanzar la aplicación en modo offline con los datos de la carpeta
//caché.
configuration.setLocalApiMode(true);
//Intervalo de refresco de los datos, cada cuanto se realiza una petición.
configuration.setDaoInterval(2000);
//En caso de modo offline por cual fichero queremos empezar la reproducción.
configuration.setDaoLocalIndex(0);
//Usuario de la API OpenSky si no se posee uno simplemente hay que registrarse.
configuration.setApiUser('xxx');
//Contraseña de la web OpenSky.
configuration.setApiPassword('xxxxx');
```

Es fundamental tener identificado este archivo, ya que será el que especificaremos en el cuerpo de nuestro archivo *index.html* para ejecutar la configuración de nuestra aplicación.

### 3.3.5. Creación de la página principal de nuestra aplicación

En esta sección, crearemos la página principal que iniciará nuestra aplicación. Esta página solo varía en función del escenario seleccionado y del archivo de configuración que se cargue. Por lo tanto, solo necesitamos copiar la estructura principal del archivo *index.html*, como se muestra a continuación. En la línea 20 donde se encuentra *configuraciónEscenario.js*, debemos establecer la ruta al archivo de configuración que creamos en la sección anterior 3.3.4. A continuación, se muestra la estructura genérica de la página principal de nuestra aplicación:

```
1 <!DOCTYPE html>
2 <html lang="es">
```

```

3 <head>
4 <meta charset="UTF-8" />
5 <meta name="viewport" content="width=device-width, initial-scale=1" />
6 <title>Vatry</title>
7 <link rel="stylesheet" type="text/css" href="css/aframe.css">

8 <script src="https://aframe.io/releases/1.4.0/aframe.min.js"></script>
9 <script src="https://cdn.jsdelivr.net/gh/c-frame/aframe-extras@7.0.0/dist/
10 aframe-extras.min.js">
11 </script>
12 <script
13 src="https://unpkg.com/aframe-terrain-model-component@1.0.1/dist/
14 aframe-terrain-model-component.min.js"></script>
15 <script src="https://unpkg.com/aframe-look-at-component@0.8.0/dist/
16 aframe-look-at-component.min.js"></script>
17 <script src="https://unpkg.com/leaflet@1.9.2/dist/leaflet.js"
18 integrity="sha256-o9N1jGDZrf5tS+Ft4gbIK7mYMipq9lqpVJ9lXHsyKhg=" crossorigin=""></script>
19 <script src="js/configuration/configurationModel.js" type="module"></script>
20 <script src="js/configuration/configuraciónEscenario.js" type="module"></script>
21 <script src="js/gui/hud.js" type="module"></script>
22 <script src="js/gui/custom-draggable.js" type="module"></script>
23 <script src="js/gui/hover-scale.js" type="module"></script>
24 <script src="js/map-ground/building-geometry.js" type="module"></script>
25 <script src="js/mainscene.js" type="module"></script>
26 <script src="js/map-ground/camera-height.js" type="module"></script>
27 <script src="js/gui/oculus-test.js" type="module"></script>
28 <script src="js/gui/tooltip-info.js" type="module"></script>
29 <script src="js/gui/toolbar3d.js" type="module"></script>
30 <script src="js/gui/track.js"></script>
31 <script src="js/gui/camrender.js"></script>
32 <script src="js/gui/canvas-updater.js"></script>

33 </head>

34 <body>
35 <a-scene main-scene>
36 <a-assets>
37 
39 
40 <a-asset-item id="plane" src="plane/scene.glTF"></a-asset-item>
41 <canvas id="cameraOnBoard"></canvas>
42 </a-assets>
43 <!-- Camera -->
44 <a-entity id="rig" position="0 0 0" movement-controls terrain-height>

```

```

44 <a-entity id="camera" hud camera look-controls="reverseMouseDown:false"
    ↳ cursor="rayOrigin: mouse; fuse: false"
45 raycaster="far: 4000; objects: .clickable" position="0 0 0" toolbar3d>
46 </a-entity>
47 <a-entity id="left-hand" oculus-touch-controls="hand: left" laser-controls="hand: left"
48 raycaster="far: 4000; objects: .clickable"></a-entity>
49 <a-entity id="right-hand" oculus-touch-controls="hand: right" laser-controls="hand: right"
50 raycaster="far: 4000; objects: .clickable"></a-entity>
51 <a-entity id="cameraOnBoardEntity" camera="active: false" camrender="cid:cameraOnBoard;fps:25"
    ↳ position="0 0 0"
52 rotation="0 -180 0"></a-entity>
53 </a-entity>
54 <a-sky id="sky" src="#skyImage" theta-length="90" radius="1000"></a-sky>

55 <a-sphere id="moon" material="shader: flat; color: #fef7ec" radius="10"
    ↳ position="-96 350 -238"
56 light="type: directional; color: #fef7ec; intensity: 0.65"></a-sphere>

57 <a-entity light="type: ambient; color: #fef7ec; intensity: 0.3"
    ↳ position="0 0 300"></a-entity>
58 </a-scene>
59 </body>

60 </html>

```

## 3.4. Componentes reutilizables

En esta sección, se presentarán en detalle los componentes de la aplicación que pueden ser reutilizados en otros proyectos a través de la configuración. Además, se presentarán aquellos componentes que pueden integrarse en otros proyectos a través de pequeñas modificaciones. También se incluirán instrucciones detalladas sobre cómo llevar a cabo la reutilización de estos componentes.

### 3.4.1. Componente de Información Contextual Interactiva

Este componente tiene la función de mostrar información contextual sobre una entidad específica. Permite al usuario obtener detalles adicionales o aclaraciones sobre el elemento seleccionado. Si deseas obtener más información sobre cómo implementarlo y utilizarlo, te recomiendo consultar la sección 4.7.2 donde se brindarán más detalles al respecto.

#### 3.4.1.1. ruta

<https://github.com/djprano/AFrameTFG/blob/main/js/gui/tooltip-info.js>

### 3.4.1.2. Guía del programador

Para que este componente funcione es un requisito que la entidad sea de la clase *clickable* por lo tanto , debes establecer la clase de la siguiente forma:

```
entidadEl.setAttribute('class', "clickable");
```

Después simplemente debes calcular el literal que quieres establecer como texto contextual para la entidad y pasarlo como argumento del componente cuando se configura en la entidad.

### 3.4.1.3. Ejemplo de uso

```
entidadEl.setAttribute('tooltip-info', { info: "Texto contextual" });
```

## 3.4.2. Componente arrastrar entidades HUD

En esta sección hablaremos del componente que permite al usuario cambiar de posición una entidad relativa a la cámara , moviendose a lo largo de un plano perpendicular a la cámara es decir que no permite arrastrar la entidad en la dimensión de profundidad solo en altura y anchura.

### 3.4.2.1. ruta

<https://github.com/djprano/AFrameTFG/blob/main/js/gui/custom-draggable.js>

### 3.4.2.2. Guía del programador

Para asegurarnos de poder interactuar con todos los componentes mediante eventos del ratón o del controlador de las gafas de realidad virtual, es necesario que la entidad en la que vamos a configurar este componente sea de la clase *clickable* (vease 3.4.1.2 para obtener más detalles al respecto). Además, otro requisito importante que mencionamos previamente en la descripción, pero es necesario resaltarlo, es que este componente está diseñado específicamente para mover o arrastrar entidades en altura o anchura, pero no en profundidad. Además, estas entidades deben estar ubicadas dentro de la jerarquía de la cámara. Por último, es recomendable configurar el componente *look-at*<sup>6</sup> en la entidad que vamos a mover. Esto permitirá que la entidad siempre se oriente hacia la cámara.

### 3.4.2.3. Ejemplo de uso

```
//Establecemos la clase clickable
hudEl.setAttribute('class', "clickable");
//Configuramos el componente look-at para que mire al elemento camera
hudEl.setAttribute('look-at', '#camera');
```

---

<sup>6</sup><https://www.npmjs.com/package/aframe-look-at-component>

```
//Configuramos el componente que nos permite desplazar la entidad con los controladores  
hudEl.setAttribute('custom-draggable', '');
```

### 3.4.3. Componente barra de herramientas 3D

En esta sección vamos a detallar como reutilizar el componente de barra de herramientas que junto con el componente *custom-draggable* del que hablamos en la sección anterior proporciona un elemento en el mundo de realidad virtual con botones conmutables que viajan con la cámara permitiendo activar o desactivar funcionalidades de la aplicación al usuario.

#### 3.4.3.1. ruta

<https://github.com/djprano/AFrameTFG/blob/main/js/gui/toolbar3d.js>

#### 3.4.3.2. Guía del programador

Para reutilizar este componente, es necesario agregarlo a la entidad de la cámara. Se trata de un componente diseñado para viajar con la cámara y añadir una superficie desplazable que proporciona botones con funcionalidades para la aplicación. Para ello, puedes crear un componente que herede del código existente y sobrescribir el método `init`. También es posible copiar el código y simplemente definir tu propio método `init`.

Dentro de este método, debes añadir los botones y establecer su funcionalidad. Para crear un botón, utiliza la función interna llamada `createToolBarButton`, la cual recibe los siguientes parámetros:

- ***id***: Identificador del elemento que se va a crear.
- ***width***: Ancho del botón en unidades del mundo 3D.
- ***height***: Alto del botón en unidades del mundo 3D.
- ***depth***: Grosor del botón en unidades del mundo 3D.
- ***text***: Texto del botón en caso de ser pulsable.
- ***textSize***: Tamaño de la letra del botón en unidades del mundo 3D.
- ***position***: Vector de posición relativa al panel de la barra de herramientas.
- ***toggle***: Booleano que indica si es un botón conmutable.
- ***enableFunction***: Función que se ejecuta cuando el botón conmutable es activado.

- ***disableFunction***: Función que se ejecuta cuando el botón conmutable es desactivado.
- ***enableColor***: Color del botón conmutable cuando está activado.
- ***disableColor***: Color del botón conmutable cuando está desactivado.
- ***enableText***: Texto del botón conmutable cuando está activado.
- ***disableText***: Texto del botón conmutable cuando está desactivado.

De esta manera, al crear los botones con su funcionalidad y agregar este componente dentro de la cámara, obtendremos una barra de herramientas en la aplicación que se necesite.

### 3.4.3.3. Ejemplo de uso

A continuación mostramos un ejemplo para la creación de un botón:

```
//Creamos un botón para habilitar el hud
this.hudEnableButton = this.createToolBarButton('hudEnableButton',0.3, 0.15,
↪ null,null, { x: -0.09, y: 0, z: 0.06 }, true,
() => this.sceneEl.emit('hud-enable', null),
() => this.sceneEl.emit('hud-disable', null),
'#0a0', '#a00', 'Hud enable', 'Hud disable');
this.toolbar.appendChild(this.hudEnableButton);
```

Para agregar el componente, tan solo debemos configurar *toolbar3d* dentro del elemento cámara como podemos ver en el siguiente ejemplo:

```
<a-entity id="camera" hud camera look-controls="reverseMouseDown:false"
↪ cursor="rayOrigin: mouse; fuse: false"
raycaster="far: 4000; objects: .clickable" position="0 0 0" toolbar3d>
```

## 3.4.4. Geometría edificio

El siguiente componente es una implementación personalizada de geometría en THREE.JS, diseñado específicamente para representar edificios utilizando las coordenadas de su contorno. Este componente se encarga de extruir el contorno proporcionado y realizar las transformaciones afines necesarias para posicionar la geometría en el lugar correcto en la escena tridimensional. Este componente personalizado resulta especialmente útil para proyectos que involucren la visualización de edificios y estructuras arquitectónicas en entornos 3D.

#### 3.4.4.1. ruta

<https://github.com/djprano/AFrameTFG/blob/main/js/map-ground/building-geometry.js>

#### 3.4.4.2. Guía del programador

Para usar esta geometría necesitamos pasarle los siguientes parámetros:

- **points**: Lista de puntos en coordenadas 2D que definen las posiciones X y Z del contorno donde vamos a dibujar el edificio.
- **terrainHeight**: Altura del terreno, donde se asienta la geometría del edificio.
- **height**: Altura del edificio.

Con estos datos y creando una entidad con la primitiva está geometría se creará un edificio en el escenario.

#### 3.4.4.3. Ejemplo de uso

```
let waypoints = [];  
waypoints.push({ x: 0, y: 1 });  
waypoints.push({ x: 1, y: 1 });  
waypoints.push({ x: 1, y: 0 });  
waypoints.push({ x: 0, y: 1 });  
let buildingProperties = { primitive: "building", points: waypoints, terrainHeight: 0  
↵ , height:20};  
let item = document.createElement("a-entity");  
item.setAttribute("geometry", buildingProperties);  
item.setAttribute("material", { color: '#88e9fd', roughness: 0.8, metalness: 0.5 });
```



## Capítulo 4

# Diseño e implementación

### 4.1. Arquitectura general de la aplicación

Se ha diseñado una arquitectura modular tal y como podemos ver en la figura 4.2, donde se ha dividido la lógica de la aplicación en distintos ficheros que se han organizado por responsabilidad a través de en las siguientes carpetas: A continuación, se presenta un diagrama 4.1 que muestra la estructura de

Sección	Descripción
<i>configuration</i>	Ficheros responsables de precargar la configuración del escenario y configurar el acceso da los datos.
<i>data</i>	Ficheros que gestionan la lógica responsable de acceder a los datos y mantenerlos en memoria.
<i>map-ground</i>	Ficheros responsables de cargar el suelo, los edificios y gestionar las alturas de las entidades.
<i>gis</i>	Contiene la lógica que gestiona las transformaciones geoespaciales.
<i>gui</i>	Ficheros que gestionan y contienen todos los componentes relacionados con la interfaz gráfica del usuario.

Cuadro 4.1: Tabla que describe las secciones en las que se ha dividido la lógica de la aplicación.

archivos en la cual hemos organizado la lógica de nuestra aplicación. Esta lógica está estrechamente relacionada con la arquitectura general de la aplicación, la cual se muestra en la figura 4.2. En este diagrama, se puede observar la distribución de las secciones y la relación entre los componentes creados. Además, se puede apreciar la comunicación de eventos entre las diferentes partes de la arquitectura.

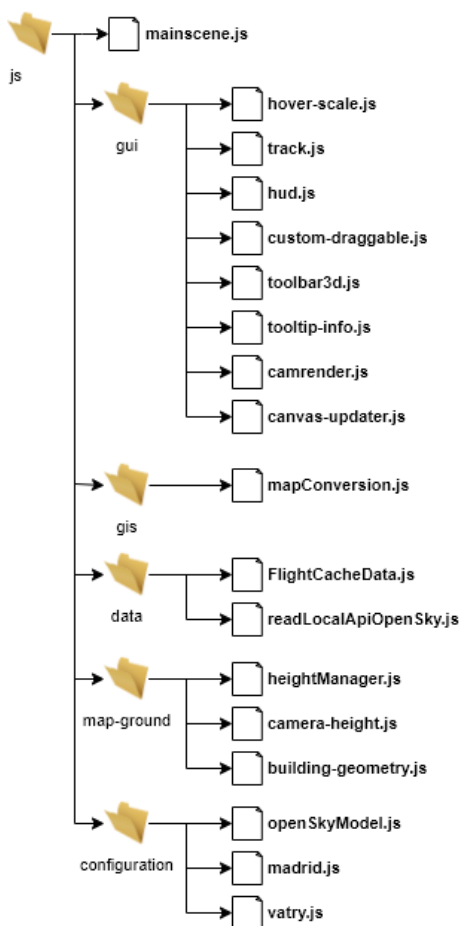


Figura 4.1: Estructura de ficheros JAVASCRIPT.

## 4.2. Gestor de la escena principal

El archivo *mainScene.js* desempeña un papel fundamental en la aplicación, podemos describirlo como el núcleo central de la aplicación. Si nos referimos a la figura 4.2, podemos apreciar su importancia. Su principal responsabilidad es sobrescribir el método de inicialización de la escena para agregar una función adicional que se invoca periódicamente según el intervalo de tiempo configurado.

El propósito de esta función adicional es obtener datos aeronáuticos de una API. Para lograr esto, se utiliza el archivo *readLocalApiOpenSky.js*, que contiene la lógica para acceder a los datos en tiempo real o desde una caché local, dependiendo de la configuración establecida. Una vez que se obtienen los nuevos datos, se actualiza dinámicamente el escenario creando las entidades necesarias para representar la información aeronáutica en tiempo real. En resumen, este archivo desempeña un papel central al proporcionar la funcionalidad esencial para la obtención y representación de datos aeronáuticos en el entorno 3D de la aplicación.

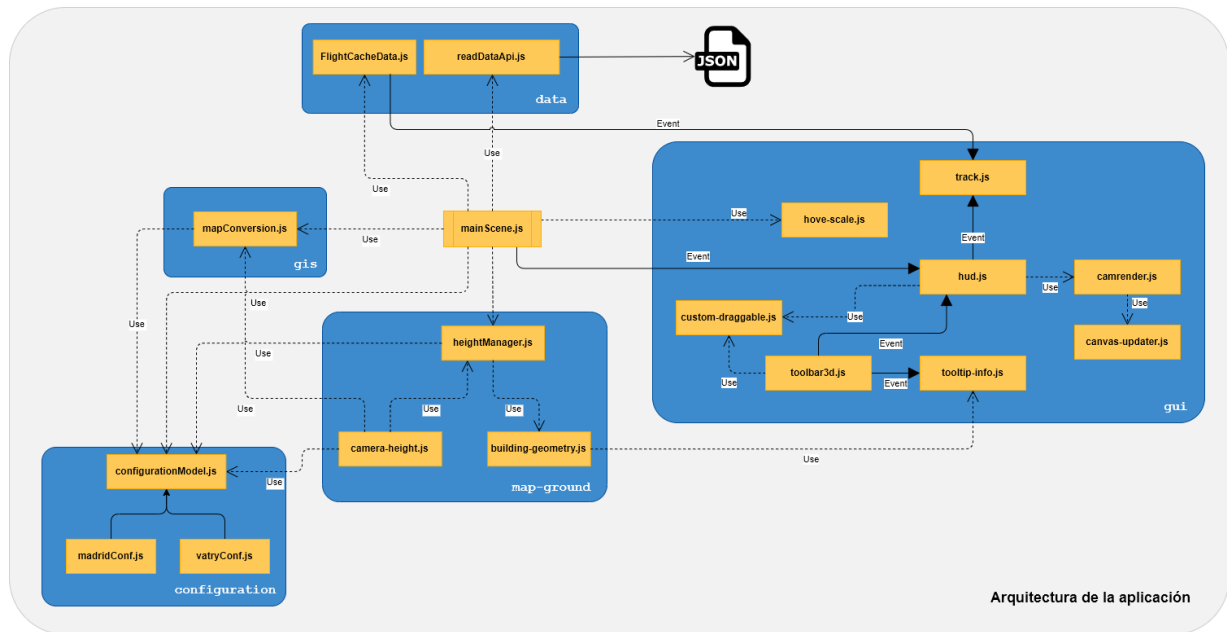


Figura 4.2: Arquitectura general de la aplicación.

#### 4.2.1. Componente main-scene

Como mencionamos en la introducción, este archivo tiene la responsabilidad de sobrescribir las funciones *init* y *tick* del componente principal *main-scene* en la escena de A-FRAME. Esto permite agregar funcionalidades relacionadas con la obtención de datos de vuelo, la actualización de elementos en la escena y la interacción con las representaciones de aviones en el entorno *3D*.

En la función *init*, utilizaremos la instancia única del convertidor de mapas de la clase *MapConversion* que se encuentra en el archivo *mapConversion.js*. Esto nos permitirá convertir la posición inicial de la cámara configurada a la posición que debe usar dentro del entorno *3D*. Estableceremos las coordenadas *X* y *Z*, mientras que la altura se calculará posteriormente utilizando el componente *heightManager* responsable de gestionar las alturas de la cámara a medida que se desplaza por el terreno.

Después, llamaremos a la instancia única del gestor de alturas *HeightManager* para generar el suelo de la escena y los edificios correspondientes. Finalmente, se establecerá una función que se invocará periódicamente según el intervalo de tiempo configurado para la actualización de los datos aeronáuticos. Es crucial que este intervalo de tiempo sea parametrizable, ya que representa el intervalo en el cual los datos pueden ser obtenidos cuando la consulta se realiza en modo *online*. En el caso de utilizar una caché local, este intervalo de tiempo determinará la frecuencia con la que se han obtenido los datos, lo que permitirá que su representación se acerque lo más posible a la realidad. En caso de establecer

un intervalo más corto al que se configuró al capturar los datos, estaremos representando la realidad de forma acelerada.

Es crucial comprender el concepto de una función con límite de frecuencia, también conocida como *throttledFunction*, ya que no solo se utilizará en la actualización de datos aeronáuticos, sino que generalmente se empleará al implementar la función *tick* de un componente. Este método se ejecuta automáticamente en cada cuadro renderizado por el motor de *A-Frame*. Esto significa que el código dentro de la función *tick* se ejecutará continuamente en cada repintado. Es por eso que es importante tener en cuenta que el uso excesivo o ineficiente del método *tick* puede afectar negativamente el rendimiento de la aplicación. Por lo tanto, la idea principal detrás de una *throttledFunction* es limitar la cantidad de veces que una función puede ser llamada dentro de un período de tiempo determinado. Al utilizar una *throttledFunction* en la actualización de datos aeronáuticos, nos aseguramos de que la obtención y procesamiento de los datos se realice de manera controlada y eficiente. Esto evita llamadas excesivas a la [API](#) de datos y reduce la carga en el sistema, lo que mejora el rendimiento general de la aplicación en entornos *3D*. Después de comprender el concepto de la *throttledFunction*, la última tarea que realiza el método de inicialización es establecer una función que utilizará esta técnica. Esta función se ejecutará periódicamente cada intervalo de tiempo configurado y utilizará el servicio *readLocalApiOpenSky* para obtener los datos aeronáuticos. Una vez que se obtengan los nuevos datos, la función invocará a *updateData*, que está definida dentro del archivo *mainScene.js*. En este punto, se realizará el análisis y parseo del archivo [JSON](#) que contiene los datos aeronáuticos. Si se encuentran nuevos componentes que deben ser creados en la escena, se generarán las entidades necesarias. Para los componentes existentes, se crearán animaciones que permitirán que los aviones se muevan de manera fluida dentro del entorno *3D*. Esta estrategia garantiza que los datos aeronáuticos se actualicen regularmente y que la representación de los aviones en la escena sea realista y dinámica.

Es importante remarcar que el componente *main-scene* también será responsable de registrar los eventos necesarios para el envío de eventos sobre los elementos creados como aviones, para que el componente que explicaremos más adelante responsable de mostrar los datos reciba la información y el evento de mostrarse.

#### 4.2.2. Movimiento fluido de aviones dentro de la escena

Para lograr un movimiento fluido de los aviones dentro de la escena, vamos a crear una interpolación lineal utilizando las animaciones proporcionadas por *A-FRAME*, tal como se muestra en el código de la Figura 4.3. En esta implementación, cada vez que se ejecute el método de actualización de la escena en intervalos de tiempo regulares, crearemos una animación que interpolará de forma lineal la posición

anterior y la nueva del avión. Esta animación tendrá una duración igual al intervalo de actualización asegurando de tal manera que la animación termina cuando se recupera un nuevo dato para generar la siguiente.

```
if (cacheData.lastPosition !== cacheData.newPosition) {  
    entityEl.setAttribute('animation__000', {  
        property: 'position',  
        from: cacheData.lastPosition,  
        to: cacheData.newPosition,  
        autoplay: true,  
        loop: 0,  
        easing: 'linear',  
        dur: intervalTime  
    });  
}
```

Figura 4.3: Animación para movimiento fluido interpolando posiciones de manera lineal.

Para llevar a cabo esta tarea, es necesario mantener una caché de las entidades presentes en la escena y poder consultar rápidamente si existen y cuáles son sus datos antiguos. Además, esta caché nos será útil más adelante para implementar la funcionalidad que muestra el trayecto de un avión seleccionado en tiempo real, véase la sección 4.7.6 para más información.

Dentro del archivo *mainScene.js*, encontraremos una variable llamada *flightsCache*, la cual es un mapa que contiene una instancia de la clase *FlightCacheData* 4.6.1 definida en el archivo *FlightCacheData.js*, este mapa indexa el identificador del vuelo con su objeto caché para encontrar los datos de manera eficiente. Este mapa de objetos en caché será responsable de mantener el estado de los aviones en el mapa y nos permitirá consultar si un avión existe en la escena y cuál fue su posición anterior para la creación de la animación.

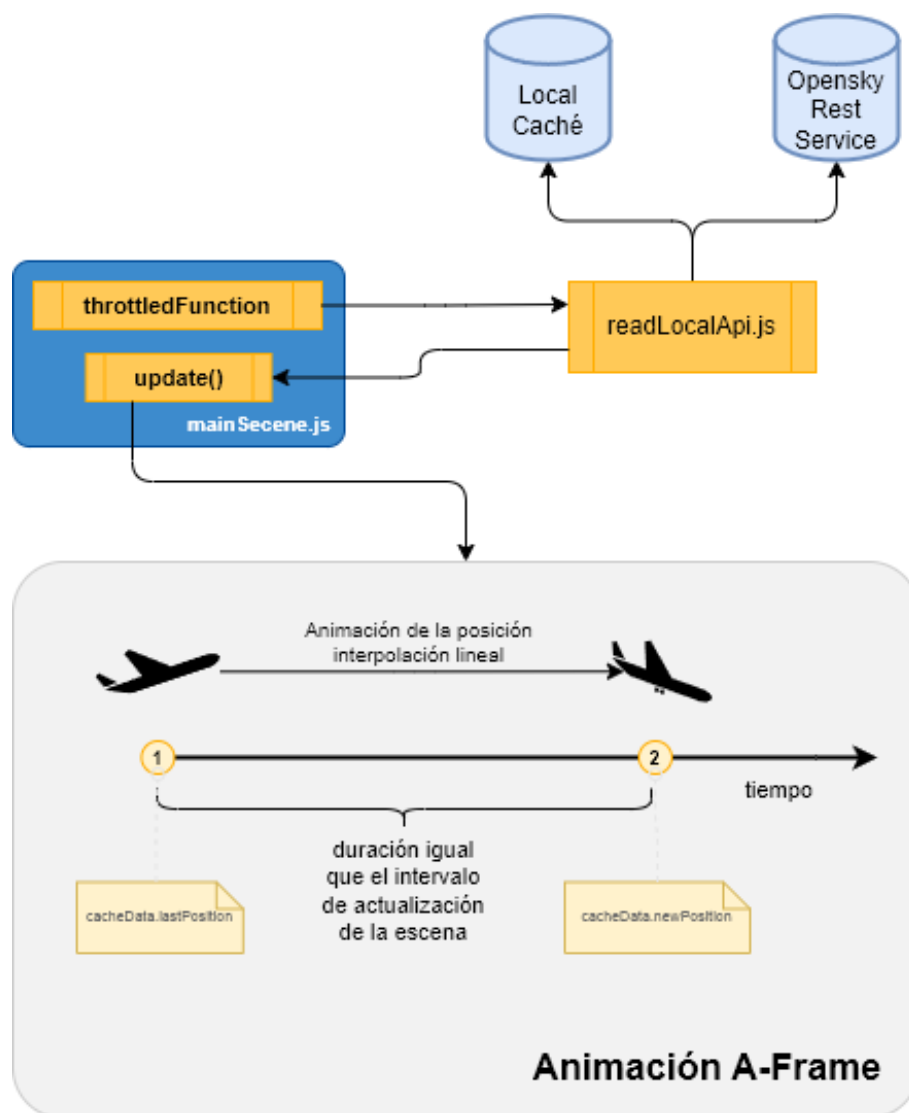


Figura 4.4: Animación A-Frame movimiento fluido.

### 4.3. Gestión de la configuración de la aplicación

Debido a la naturaleza altamente reutilizable y configurable de nuestra aplicación, hemos dedicado especial atención a la parametrización de todos los componentes. Nuestro objetivo es que la creación de cualquier tipo de escenario se centre en un único archivo de configuración, simplificando así el proceso de establecer los parámetros necesarios para su correcto funcionamiento.

Para lograr esto, hemos desarrollado una pieza clave llamada *configurationModel.js*, que contiene un módulo utilizado por todos los componentes de la aplicación. Este módulo almacena todos los parámetros específicos de un escenario en particular. Para establecer los valores internos de *configurationModel.js*, creamos un archivo JAVASCRIPT que invoca los métodos *set* del módulo de configuración, estableciendo así todas las propiedades necesarias para el escenario tal y como podemos ver en la figura 4.5. Como resultado, el archivo HTML principal de nuestro escenario carga el módulo de configuración y el archivo JAVASCRIPT que establece las propiedades requeridas para su correcto funcionamiento como se puede apreciar en la figura 4.6.

```
import * as configuration from './configurationModel.js';  
//Establece las coordenadas geodésicas del escenario (latmin, latmax, longmin, longmax).  
configuration.setMerConfig(40.0234170, 40.7441446, -4.2041338, -3.2538165);  
configuration.setCamPosition(40.50, -3.54); //posición de la cámara.  
configuration.setBuildingFileName('madrid_building'); //carpeta de edificios.  
configuration.setFlightLocalFolder('_madrid'); //carpeta caché de vuelos.  
configuration.setMapRaster('Madrid_raster.jpg'); //fichero raster del terreno  
configuration.setMapDem('madrid_dem.bin'); //fichero de alturas del terreno  
configuration.setLocalApiMode(true); //establece el modo offline datos cacheados.  
configuration.setDaoInterval(1000); //intervalo de refresco.  
configuration.setDaoLocalIndex(0); //índice del fichero por donde comienza.  
configuration.setApiUser('xxxx'); //Establece el usuario para el modo online  
configuration.setApiPassword('xxxxx'); //Establece la contraseña para el modo online
```

Figura 4.5: Configuración del escenario de Madrid.

```
<script src="js/configuration/configurationModel.js" type="module"></script>  
<script src="js/configuration/madridconf.js" type="module"></script>
```

Figura 4.6: Carga del fichero del módulo gestor de configuración y el fichero que configura el escenario de Madrid.

## 4.4. Sistema de información geográfica

En esta sección, se describirá en detalle la implementación de toda la lógica de conversiones utilizada para georreferenciar las entidades dentro del escenario. También se abordarán los problemas que han surgido durante el proceso y las soluciones aplicadas, así como las librerías utilizadas.

### 4.4.1. Gestor de conversiones del mapa

En este fichero, se han implementado todas las funciones necesarias dentro de la aplicación para transformar coordenadas geoespaciales, en particular el objetivo principal de este gestor es la conversión de coordenadas expresadas en latitud y longitud en grados, a un vector válido para el escenario 3D (x, y, z). Antes de adentrarnos en la descripción técnica, es importante hacer una breve introducción sobre los sistemas de referencia que vamos a utilizar y por qué es necesario realizar estas transformaciones. En el mundo de los Sistemas de Información Geográfica (GIS), es común utilizar sistemas de referencia geoespaciales representados por coordenadas geográficas y sistemas de proyección cartográfica. Una coordenada geográfica es un par de valores numéricos que representan la ubicación de un punto en la superficie de la Tierra. Como podemos observar en la figura 4.7 la *latitud* y una *longitud*, indican la posición definiendo la relación con los meridianos y paralelos terrestres.

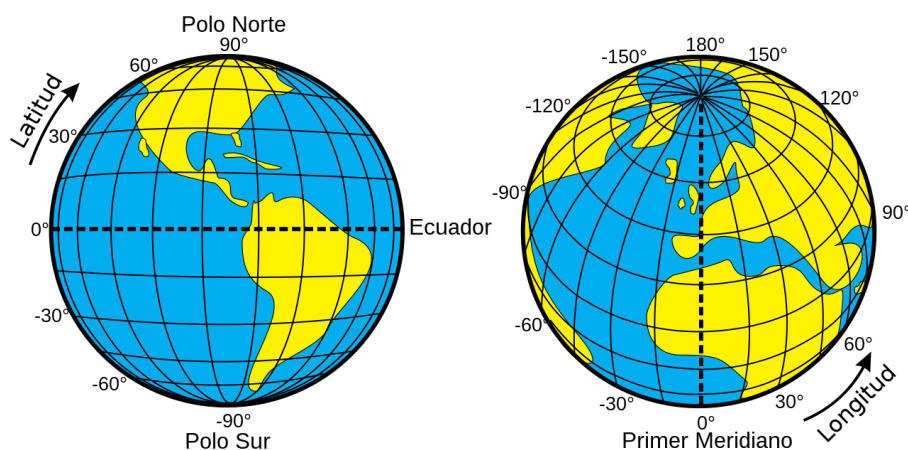


Figura 4.7: Coordenadas geográficas latitud y longitud.

En los sistemas de referencia geoespaciales, es común utilizar un datum que es un modelo matemático que describe la forma y la orientación de la Tierra, así como un punto de referencia. Muchos servicios y aplicaciones que proporcionan datos espaciales a nivel mundial, como GOOGLE MAPS y OPENSky, utilizan el datum WGS84 como su sistema de referencia estándar. El datum WGS84 (Sistema Mundial Geodésico de 1984) es ampliamente aceptado y utilizado en todo el mundo. Fue desarrollado por el Departamento de Defensa de los Estados Unidos y la Agencia Nacional de Inteligencia Geoespacial



(NGA) como un sistema de referencia global para el intercambio de datos geoespaciales. WGS84 se basa en un modelo matemático que representa la forma de la Tierra como un elipsoide de revolución y establece un conjunto de parámetros para la conversión precisa entre coordenadas geográficas y sistemas de coordenadas proyectadas. Dado que muchos servicios y datos geoespaciales se basan en WGS84, es importante que nuestra aplicación sea capaz de trabajar en este datum para garantizar la interoperabilidad y la correcta interpretación de los datos. Esto nos permite realizar conversiones precisas entre coordenadas geográficas en formato de *latitud* y *longitud*, y las coordenadas del escenario 3D en el que estamos trabajando.

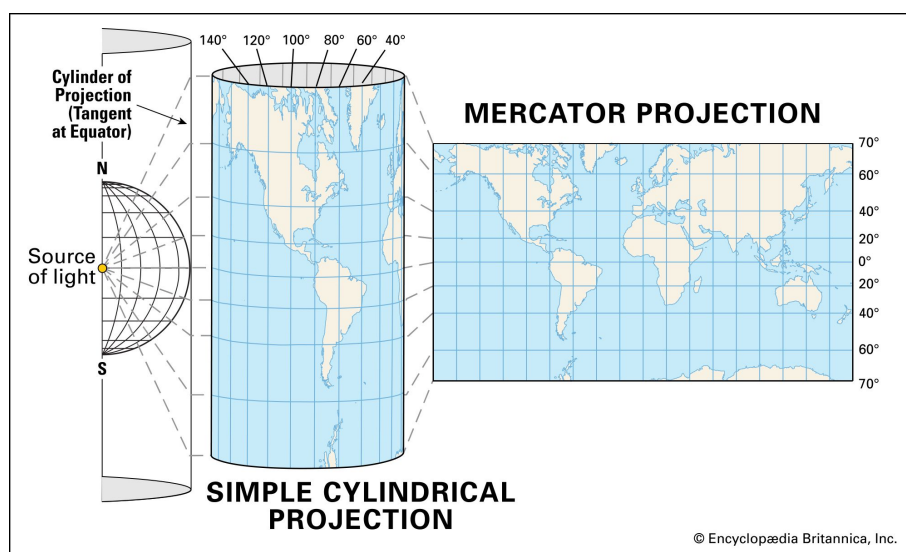


Figura 4.8: Proyección cilíndrica Mercator.

Por otro lado, un sistema de proyección cartográfica es una metodología que permite representar la forma curva de la Tierra en un plano. Debido a que nuestro planeta es un objeto tridimensional, es necesario utilizar proyecciones cartográficas para convertir la superficie esférica de la Tierra en un plano bidimensional. Estas proyecciones utilizan fórmulas matemáticas y parámetros específicos para lograr una representación precisa y adecuada de la Tierra en un mapa plano. En nuestra aplicación, para convertir las coordenadas geoespaciales en un plano bidimensional, utilizaremos la proyección Mercator. Esta proyección es una de las proyecciones cartográficas más comunes y ampliamente utilizadas en internet. Fue desarrollada por Gerardus Mercator en el siglo XVI y se caracteriza por preservar los ángulos rectos y las formas de las áreas pequeñas, lo que la hace adecuada para representar regiones con menor extensión latitudinal. Se basa en envolver con un cilindro la superficie de la tierra y trazar una línea desde el centro terrestre hasta el plano del cilindro creando una relación entre el punto de intersección de la tierra con el del plano del cilindro tal y como podemos ver en la figura 4.8.

Hablemos ahora del sistema de coordenadas de A-Frame, en el contexto del sistema de referencia 3D utilizado en A-Frame, se sigue la regla de la mano derecha. Esta regla establece la orientación de los ejes X, Y y Z en el espacio tridimensional. Según la regla de la mano derecha, si extendemos el pulgar, el índice y el dedo medio de nuestra mano derecha de manera que sean perpendiculares entre sí, podemos asignar los ejes de la siguiente manera:

- El pulgar representa el eje X, apuntando hacia la derecha.
- El índice representa el eje Y, apuntando hacia arriba.
- El dedo medio representa el eje Z, apuntando hacia adelante.

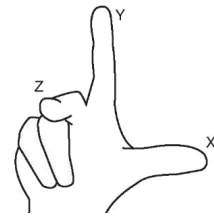


Figura 4.9: Regla de la mano derecha.

Esta convención es ampliamente utilizada en gráficos 3D y programación, incluido el entorno de desarrollo A-FRAME. Al seguir la regla de la mano derecha, podemos establecer una consistencia en la orientación de los ejes y facilitar la comprensión y manipulación de los objetos en el espacio tridimensional. En este punto, nos enfrentamos al primer problema que debemos abordar en las conversiones. Si observamos una vista aérea tanto del suelo de nuestro escenario como de un mapa representado por un datum WGS84 y proyectado con MERCATOR, notaremos una diferencia en las coordenadas. La coordenada Y, que representa la latitud en el sistema [Goespacial](#), aumenta hacia arriba, mientras que en el sistema tridimensional, se representa a lo largo del eje Z y aumenta en dirección opuesta. Por lo tanto, esto es algo que debemos tener en cuenta al diseñar nuestras fórmulas de conversión. Si observamos la figura 4.10, podemos apreciar cómo la posición con coordenada ( $x=8$ ,  $y=-7$ ) en el sistema [Goespacial](#) se representa de manera diferente en ambos sistemas de referencia.

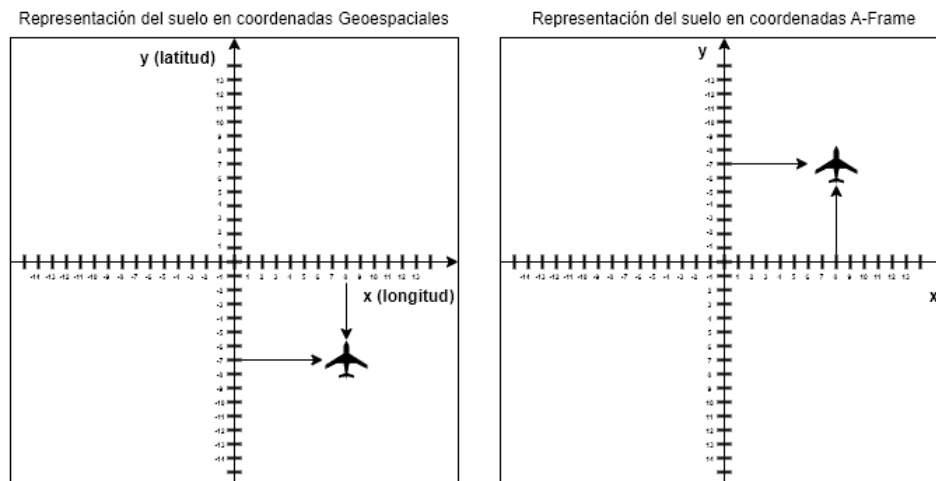


Figura 4.10: Vista aerea de los sistemas de referencia.

El siguiente problema al que nos enfrentamos está relacionado con las dimensiones de nuestra aplicación. Si consideramos cuidadosamente los escenarios que vamos a representar, el radio de alcance del escenario en kilómetros es considerablemente grande. A-FRAME trabaja con unidades de metros, lo cual es adecuado para aplicaciones de realidad aumentada donde se busca que las dimensiones en el mundo 3D sean equivalentes a la realidad. Sin embargo, en el caso de nuestra aplicación, nuestro objetivo es asegurar que los aviones se puedan visualizar correctamente incluso a distancias de varios kilómetros. Manejar dimensiones tan grandes plantea problemas en el renderizado de objetos lejanos y en la interacción con entidades distantes, además de dificultades en el manejo de coordenadas 3D con unidades tan grandes. Para ilustrar este concepto, consideremos el escenario de la demo de *Vatry*, donde se configura un rectángulo con las siguientes coordenadas geoespaciales que representan  $[x_{min}, x_{max}, y_{min}, y_{max}] = (48.491151723988, 49.027963936994, 3.7545776367187, 4.6238708496094)$ . Utilizando un pequeño *script* en GOOGLE EARTH ENGINE, podemos calcular las distancias de altura y anchura para este rectángulo:

```

// Cargar el rectángulo de coordenadas geodésicas
var rectangle = ee.Geometry.Rectangle(
[3.7545776367187, 48.491151723988, 4.6238708496094, 49.027963936994]);
// Agregamos el rectángulo al mapa
Map.addLayer(rectangle, {}, 'Rectángulo');
// Centrar el mapa en el rectángulo
Map.centerObject(rectangle, 10);
// Obtener los límites del rectángulo
var bounds = rectangle.bounds();
var coordinates = bounds.coordinates().get(0);
var firstCoordinate = ee.List(coordinates).get(0);
var secondCoordinate = ee.List(coordinates).get(1);
var thirdCoordinate = ee.List(coordinates).get(3);
// creamos geometrías de tipo punto para calcular las distancias
var punto1 = ee.Geometry.Point(firstCoordinate);
var punto2 = ee.Geometry.Point(secondCoordinate);
var punto3 = ee.Geometry.Point(thirdCoordinate);
// Calcular la distancia
var altura = punto1.distance(punto2).divide(1000);
var anchura = punto1.distance(punto3).divide(1000);
// Imprimir la altura y la anchura en la consola
print('Altura del rectángulo:', altura);
print('Anchura del rectángulo:', anchura);

```

Figura 4.11: Cálculo de dimensiones para el escenario Vatry.

```

Altura del rectángulo:
64.25269118063613
Anchura del rectángulo:
59.78757908704291

```

Figura 4.12: Resultado del script.

Como se puede observar en la figura 4.12, este *script* nos arroja que estamos manejando distancias de aproximadamente 60 km. Por lo tanto, debemos tener en cuenta que cuando los aviones se alejan, se verán muy pequeños si representamos las unidades con una anchura de 60.000 unidades en el mundo 3D de A-Frame. Además, las operaciones como la intersección del rayo utilizado para seleccionar los aviones pueden presentar problemas y afectar el rendimiento de nuestra aplicación. Para abordar esta situación, vamos a realizar una operación de escalado en nuestras ecuaciones. Esto nos permitirá dimensionar nuestro escenario de una manera más óptima, ajustando las unidades a una escala adecuada para asegurar

una representación visual adecuada y un rendimiento eficiente.

El tercer problema al que nos enfrentamos es la asimetría en el escenario de representación. A menos que estemos representando una zona cercana al meridiano de GREENWICH y al ecuador, al trabajar con coordenadas en metros es común que estemos manejando unidades excesivamente grandes. Por ejemplo, en la ciudad de *Vatry*, las coordenadas se centran en aproximadamente  $x = 472476$  e  $y = 6245020$ . A medida que nos alejamos, podemos encontrarnos con zonas de Rusia donde las coordenadas se expresan en millones de metros, lo cual hace que las operaciones resulten poco prácticas. Para abordar este problema, se ha tomado la decisión de desplazar el centro del mapa al centro del escenario. Esto implica que, además del escalado mencionado anteriormente, realizaremos un desplazamiento de tal manera que el centro del escenario coincida con el centro del mapa. Esta solución nos permite trabajar con dimensiones más manejables y evita la asimetría en la representación. Al centrar el mapa en el escenario, las operaciones y cálculos se realizan de manera más eficiente y se logra una representación equilibrada de los elementos geoespaciales en el entorno 3D. Recapitulando, podemos resumir los siguientes puntos:

- Nuestras transformaciones se centrarán en establecer las equivalencias de ejes entre los sistemas de referencia. En este caso, el eje X Geoespacial se mapeará al eje X en el mundo 3D, el eje Y geoespacial se mapeará al eje Z en el mundo 3D, y la altura se representará en el eje Y en el vector 3D.
- Habrá una inversión en el eje que representa la latitud. Esto significa que la coordenada Y geoespacial se mapeará al eje -Z en el mundo 3D debido a la inversión.
- Se realizará un desplazamiento para centrar el escenario en la coordenada  $(0, 0, 0)$  en el mundo 3D. Esto asegurará que el centro del escenario coincida con el origen de coordenadas en el mundo 3D.
- Además, se aplicará un escalado para facilitar el dimensionado del escenario en el mundo 3D. Esto permitirá trabajar con dimensiones más manejables y evitar problemas de representación y rendimiento.

En resumen, estas transformaciones nos permitirán establecer una correspondencia adecuada entre los sistemas de referencia geoespacial y el entorno 3D, asegurando una representación coherente y optimizada de los elementos geoespaciales en nuestro escenario.

Para la conversión de coordenadas de WGS84 en grados a metros, hemos optado por utilizar la librería LEAFLET 2.12 en lugar de las fórmulas clásicas de la proyección MERCATOR. Las razones principales que respaldan esta elección son las siguientes:

- **Facilidad de uso:** LEAFLET proporciona una interfaz intuitiva y sencilla para trabajar con diversos sistemas de coordenadas y proyecciones. La librería admite múltiples sistemas de proyección, incluyendo la proyección MERCATOR ampliamente utilizada. Además, ofrece funciones integradas para realizar conversiones de coordenadas y proyecciones, lo cual simplifica la transformación de datos geoespaciales en el contexto del mapa.
- **Optimización de rendimiento:** LEAFLET está diseñado para ofrecer un rendimiento óptimo en entornos web en tiempo real. Sus conversiones están optimizadas para asegurar un alto rendimiento.
- **Mantenibilidad y actualizaciones regulares:** LEAFLET cuenta con una comunidad activa de desarrolladores que constantemente trabajan en mejorar la biblioteca, solucionar problemas, agregar nuevas características y garantizar la compatibilidad con las últimas tecnologías y estándares. Esto brinda confianza en la estabilidad y continuidad del proyecto, al tiempo que reduce la carga de mantenimiento para nuestro prototipo.

A continuación, en la Figura 4.13, se presenta el cálculo del desplazamiento. Se puede observar que primero se calculan las coordenadas geodésicas del centro del escenario. Para ello, se toma el punto medio de la latitud y la longitud. Dado que deseamos desplazar el centro del escenario al punto  $(0,0,0)$  en el mundo 3D, es necesario tener previamente calculado el desplazamiento en metros que se aplicará a cada coordenada.

Posteriormente, se utiliza una función que realiza la conversión a metros mediante la proyección de MERCATOR 4.4.1.1. Esto da como resultado las coordenadas cartesianas del centro del escenario expresadas en metros. Estas coordenadas representan el desplazamiento que se aplicará a cada coordenada para lograr que el mapa esté centrado en el punto  $(0,0,0)$ .

$$\begin{aligned} long &= \frac{LONG\_MIN + LONG\_MAX}{2} \\ lat &= \frac{LAT\_MIN + LAT\_MAX}{2} \\ displacement &= degreeToMeter(lat, long) \end{aligned}$$

Figura 4.13: Cálculo del desplazamiento del escenario.

Por lo tanto, el constructor de nuestra clase *MapConversion* realizará estos cálculos y almacenará el resultado en una *instancia única* para que pueda ser utilizado en todas las funciones de conversión. De esta manera, evitamos repetir los cálculos y garantizamos que todas las transformaciones se realicen utilizando el mismo desplazamiento y escala.

#### 4.4.1.1. De WGS84 a Mercator

Esta función utiliza la librería LEAFLET para crear una coordenada geodésicas en el sistema de referencia WGS84 a partir de una *latitud* y *longitud* dadas. Luego, realiza la proyección en MERCATOR para convertir la coordenada geodésica en una coordenada cartesiana expresada en metros.

#### 4.4.1.2. De Mercator a WGS84

Esta función realiza la operación inversa a la función descrita en 4.4.1.1. Convierte una coordenada cartesiana en el sistema de proyección MERCATOR a una coordenada geodésica en el sistema de referencia WGS84 haciendo uso de la librería LEAFLET. Esta función aprovecha sus capacidades integradas para realizar la desproyección de coordenadas.

#### 4.4.1.3. De MatorToWorld

Esta función se encargará de realizar la conversión de una coordenada cartesiana en proyección MERCATOR a una coordenada en el mundo 3D. Su objetivo principal es aplicar las transformaciones mencionadas anteriormente para abordar los problemas de representación, que incluyen la inversión del eje Z, el desplazamiento y el escalado. Para lograr esto, la función hará uso del desplazamiento calculado en el constructor de la clase. A continuación, se presentan las fórmulas que serán aplicadas:

$$X_{3DWorld} = \frac{\text{mercatorVector}_x - \text{displacement}_x}{\text{FACTOR}}$$

$$Y_{3DWorld} = \frac{\text{mercatorVector}_z - \text{displacement}_y}{\text{FACTOR}}$$

$$3DWorld = \left[ X_{3DWorld}, \frac{\text{height}}{\text{FACTOR}}, -Y_{3DWorld} \right]$$

Figura 4.14: Calculo de la conversión de coordenadas cartesianas en MERCATOR a mundo 3D.

#### 4.4.1.4. De mundo 3D a Mercator

Esta función tiene la responsabilidad de realizar la operación inversa a la función mencionada en ???. Su objetivo es convertir un vector del escenario 3D en una coordenada cartesiana en proyección MERCATOR, aplicando todas las operaciones inversas al escalado, la inversión de ejes y el desplazamiento realizados por la función *Mercator a mundo 3D* ???. A continuación, se presentan las fórmulas que serán aplicadas:

$$\begin{aligned}
X_{\text{Mercator}} &= (\text{world3DVector}_x \times \text{FACTOR}) + \text{displacement}_x \\
Y_{\text{Mercator}} &= (\text{world3DVector}_z \times \text{FACTOR}) + \text{displacement}_y \\
\text{Mercator} &= \left[ X_{\text{Mercator}}, \text{world3DVector}_y \times \text{FACTOR}, -Y_{\text{Mercator}} \right]
\end{aligned}$$

Figura 4.15: Calculo de la conversión vector 3D a coordenadas cartesianas Mercator.

#### 4.4.1.5. De WGS84 a mundo 3D

Esta función se basará en la función *de WGS84 a Mercator* 4.4.1.1 para convertir las coordenadas de *latitud* y *longitud* en formato WGS84 a una proyección MERCATOR. A continuación, utilizará la función *Mercator a mundo 3D* ?? para transformar la coordenada MERCATOR resultante en una coordenada tridimensional en el mundo 3D. En resumen, esta función permitirá convertir una coordenada geodésica en una coordenada 3D sin altura, lo que facilitará la ubicación de las entidades del mundo real dentro del escenario.

Esta función será útil para posicionar entidades geoespaciales, como edificios o aviones, dentro del escenario. Además, en un futuro, podría utilizarse para crear cualquier tipo de entidad que se base en datos geodésicos, ya que la amplia mayoría de servicios utiliza el sistema WGS84 y esta función nos devuelve el vector de posición para el escenario A-FRAME.

#### 4.4.1.6. De mundo 3D a WGS84

Esta función se encargará de realizar la operación inversa a la función *WGS84 a mundo 3D* descrita en la sección 4.4.1.5. Su objetivo es obtener la coordenada geodésica a partir de un vector tridimensional del mundo 3D. Para lograr esto, primero se realizará la conversión de la coordenada 3D a una coordenada MERCATOR utilizando la función *mundo 3D a Mercator* mencionada en la sección 4.4.1.4. Luego, se aplicará la conversión de la coordenada cartesiana a WGS84 utilizando la función *de Mercator a WGS84* descrita en la sección 4.4.1.2.

Aunque esta función no se utiliza actualmente en la aplicación, resulta útil para futuras funcionalidades. Por ejemplo, podría emplearse para mostrar la ubicación de la cámara principal en el HUD, proporcionando las coordenadas geodésicas correspondientes a la posición de la entidad de la cámara. Su implementación sería sencilla gracias al uso de esta función para convertir el vector de posición en el mundo 3D en una coordenada geodésica en formato WGS84.



**4.4.1.7. Devuelve el tamaño del terreno**

La función proporcionará las dimensiones del escenario en el entorno 3D al utilizar las constantes que definen la *latitud* y *longitud* máxima del rectángulo del escenario geodésico. Mediante la transformación de estas coordenadas al sistema de coordenadas del mundo 3D y considerando que el escenario está centrado, podremos calcular con precisión las dimensiones de anchura y altura en el entorno tridimensional (Mundo 3D). Esta función será de gran utilidad en etapas posteriores para el gestor del terreno del escenario, ya que se requieren estas dimensiones para el correcto funcionamiento del componente de terreno descrito en la sección [4.5.3](#).

**4.4.1.8. Crear entidades en los corner del terreno**

Esta función no tiene ninguna aplicación en el prototipo final, su único propósito es facilitar la depuración y verificar el correcto funcionamiento de las transformaciones de esta clase. Crea entidades en las esquinas del escenario y proporciona una referencia visual del terreno en el escenario para comprobar si está correctamente centrado en el entorno tridimensional, son funciones necesarias que se han creado para una depuración visual en el inspector del escenario.

## 4.5. Terreno en el mapa

En esta carpeta se encuentran todos los archivos responsables de gestionar la lógica relacionada con el suelo y las entidades asociadas. El objetivo de estos archivos es gestionar todas las operaciones relacionadas con el suelo, desde la generación del mallado hasta el posicionamiento y desplazamiento de las entidades sobre la superficie del suelo. Para lograr esto, se utilizan diversas tecnologías y librerías que permiten realizar las operaciones necesarias de forma eficiente y precisa. También podremos encontrar las entidades que serán colocadas sobre el suelo, como la cámara principal que se moverá sobre la superficie del terreno y los edificios que deben estar alineados con el suelo. Para modelar las superficies, se utilizan conjuntos de datos [Raster](#). Un raster es una matriz compuesta por celdas o píxeles dispuestos en filas y columnas que cubren una determinada región geográfica. Cada celda en la matriz representa una unidad de área cuadrada y contiene un valor numérico que representa una altura tal y como podemos apreciar en la figura 4.16.

Un archivo DEM es un tipo de archivo raster con celdas que representan un valor de elevación correspondiente a una ubicación geodésica específica en relación a un datum de referencia. En resumen, un archivo DEM es una representación muestreada de una superficie terrestre limitada, donde las alturas se almacenan en una matriz.

10.44	10.53	10.62	10.69	10.85	10.95	11.02	11.08	11.09	11.14
10.56	10.68	10.64	10.91	10.82	10.61	10.58	10.58	10.71	10.7
10.64	10.87	10.91	10.88	10.89	10.73	10.59	10.58	10.63	10.7
10.8	10.84	11.08	10.99						
11.19	11.23	10.96	11.08	10.44	10.53	10.62	10.69	10.85	10.95
11.23	11.42	11.31	10.91	10.56	10.68	10.64	10.91	10.82	10.61
11.36	11.42	11.32	11.06	10.64	10.87	10.91	10.88	10.89	10.73
11.28	11.34	11.25	11.07	10.8	10.84	11.08	10.99	10.96	10.66
11.17	11.16	11	10.97	11.19	11.23	10.96	11.08	10.97	10.89
10.7	10.78	10.81	10.82	11.23	11.42	11.31	10.91	11	10.83
				11.36	11.42	11.32	11.06	10.81	10.89
				11.26	11.34	11.25	11.07	10.78	10.65
				11.17	11.16	11	10.97	10.68	10.7
				10.7	10.78	10.81	10.82	10.78	10.54

Figura 4.16: Representación del contenido de un fichero DEM.

Para crear la entidad del terreno, este componente requiere varios parámetros. En primer lugar, se necesita un archivo DEM que contenga la información de elevación del terreno. Además, se deben proporcionar los siguientes argumentos:

- las dimensiones de la entidad en el escenario para lo cual se creó la función descrita en la sección [4.4.1.7](#).
- la textura que se utilizará para pintar el terreno.
- la resolución del terreno (dimensiones de la matriz de alturas).
- un parámetro de magnificación de alturas que controla la escala de las elevaciones del terreno en relación con la entidad visual resultante, de esto hablaremos más adelante pero este parámetro define el valor máximo en el mundo 3D para la altura más alta del fichero de alturas, esto es debido a que se normalizan los valores.

Con estos parámetros, el componente *afreame-terrain-model-component* genera y renderiza el terreno en el escenario, permitiendo así la representación realista de la superficie terrestre.

#### 4.5.1. Generación del fichero DEM

En primer lugar, vamos a detallar el proceso utilizado para generar el archivo de alturas DEM en formato ENVI, que luego se inyectará en el componente *afreame-terrain-model-component*. Basándonos en la experiencia del tutor del proyecto, el *Dr. Jesús M. González Barahona*, quien ha investigado y experimentado con la carga de terrenos del programa COPENICUS de la Unión Europea a través transformaciones con las herramientas proporcionadas por GDAL, una biblioteca de código abierto ampliamente utilizada en el campo de los sistemas de información geográfica que permite la manipulación de datos geoespaciales. Vamos a describir los pasos para generar el fichero DEM de Madrid como ejemplo:

1. Primero, procedemos a descargar la zona de interés de los servidores de COPENICUS, que contiene el área correspondiente a las coordenadas donde generaremos el archivo DEM. Para realizar esto, visitamos la página <https://land.copernicus.eu/imagery-in-situ/eu-dem/eu-dem-v1.1> y observamos en la Figura 4.17 los cuadrantes relevantes para extraer el archivo DEM de Madrid, que son E30N10 (30° Este 10° Norte) y E30N20 (30° Este 20° Norte). Por lo tanto, procedemos a descargar los binarios correspondientes a estas áreas.
2. Utilizamos el comando `gdalbuildvrt` para crear un archivo VRT llamado *map.vrt* que contiene un mosaico virtual de los archivos [Raster eu\\_dem\\_v11\\_E30N10.TIF](#) y [eu\\_dem\\_v11\\_E30N20.TIF](#). Esto permite acceder y trabajar con los datos de elevación contenidos en ambos archivos como si fueran un solo conjunto de datos, sin necesidad de fusionar físicamente los archivos originales. El archivo VRT proporciona una vista virtual de los datos raster y permite un acceso eficiente y flexible a los mismos.

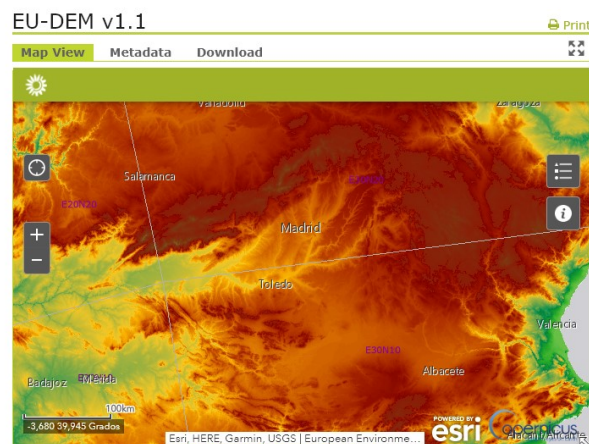


Figura 4.17: Visor online de ficheros DEM Copernicus.

```
gdalbuildvrt map.vrt eu_dem_v11_E30N10.TIF eu_dem_v11_E30N20.TIF
```

- Los ficheros que hemos descargado están proyectados en el sistema de referencia EPSG:3035<sup>1</sup>, que utiliza la proyección UTM (Universal Transverse MERCATOR). Para realizar la transformación, utilizamos el comando `gdalwarp` que toma un archivo VRT llamado *map.vrt* y lo convierte en un nuevo archivo TIFF llamado *smallMap.tif*. Durante este proceso, los datos se reproyectan al sistema de referencia espacial EPSG:4326<sup>2</sup>, que utiliza el DATUM WGS84 y es el sistema de referencia que utilizamos para los datos de las API. Luego, los datos se recortan para abarcar la extensión definida por los límites de *latitud* y *longitud* proporcionados al comando.

```
gdalwarp -t_srs EPSG:4326 -te -4.204133836988655 40.023417003380956  
-3.253816454176155 40.744144594569384 map.vrt smallMap.tif
```

- Comprobamos las coordenadas del fichero resultante.

```
gdalinfo -mm smallMap.tif
```

Corner Coordinates:

```
Upper Left ( -4.2041338, 40.7441446) ( 4d12'14.88"W, 40d44'38.92"N)  
Lower Left ( -4.2041338, 40.0234170) ( 4d12'14.88"W, 40d 1'24.30"N)  
Upper Right ( -3.2538165, 40.7441446) ( 3d15'13.74"W, 40d44'38.92"N)  
Lower Right ( -3.2538165, 40.0234170) ( 3d15'13.74"W, 40d 1'24.30"N)  
Center ( -3.7289751, 40.3837808) ( 3d43'44.31"W, 40d23' 1.61"N)
```

<sup>1</sup><https://epsg.io/3035>

<sup>2</sup><https://epsg.io/4326>

```
Band 1 Block=3671x1 Type=Float32, ColorInterp=Gray  
Computed Min/Max=461.652,1886.917  
NoData Value=-3.4028234663852886e+38
```

5. Podemos utilizar el comando `gdal_translate` para generar un archivo PNG que nos permita visualizar los datos generados a partir del archivo de alturas. Esto nos brindará una representación en blanco y negro de la superficie como se puede observar en la figura 4.18.

```
gdal_translate -scale 0 2522 0 255 -outsize 200 200 -of PNG smallMap.tif smallMap.png
```

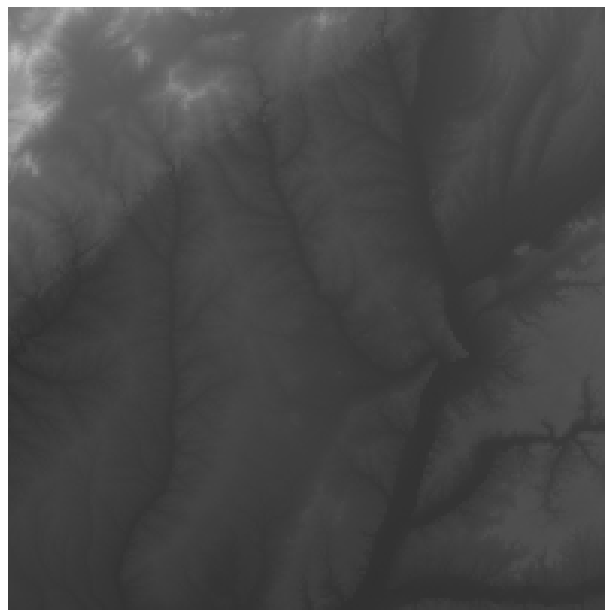


Figura 4.18: Fichero de alturas `smallMap.png` visto como imagen en blanco y negro.

6. A continuación, haremos uso de nuevo del comando `gdal_translate` para realizar un escalado de valores. Transformaremos el rango original de  $(0,2522)$  a un nuevo rango de  $(0,65535)$ , que corresponde al rango del tipo de datos `UInt16` que usa el componente para almacenar los datos. El resultado será guardado en un archivo binario ENVI llamado *Madrid.bin*.

```
gdal_translate -scale 0 2522 0 65535 -ot UInt16 -outsize 200 200 -of ENVI smallMap.tif Madrid.bin
```

#### 4.5.2. Generación de la capa de textura raster

Vamos a describir cómo utilizando la tecnología que nos brinda GOOGLE EARTH ENGINE podemos extraer la capa [Raster](#) correspondiente a las coordenadas del rectángulo para el cual hemos generado

el archivo bin en formato ENVI. Esto nos permitirá superponer la textura del terreno con una imagen satelital que esté georreferenciada con las mismas coordenadas, lo que nos dará una representación más realista del terreno. En la figura 4.19 se puede observar que estamos usando el API que ofrece el [Framework](#) de GOOGLE EARTH ENGINE, el objeto `ee` (Earth Engine) nos ofrece la capacidad de crear geometrias añadirlas al mapa y realizar operaciones con ellas, a continuación paso a describir el proceso que seguimos para realizar la extracción del raster:

1. Creamos una geometría rectangular con coordenadas WGS84 que nos permitirá realizar una operación de intersección con el raster para la obtención de una porción.
2. Añadimos el rectángulo como capa para visualizar la zona que vamos a exportar y realizar una comprobación visual de los datos.
3. Cargamos una colección de imágenes satelitales Sentinel-2, filtrando por las capturadas entre el 1 de enero de 2019 y el 28 de febrero de 2019.
4. Creamos un mosaico de las imágenes de la colección. Esto combina todas las imágenes en una sola imagen, seleccionando los píxeles en función de su importancia o calidad. Añadimos este mosaico como capa para visualizar el resultado.
5. Creamos una imagen RGB utilizando las bandas B4, B3 y B2.
6. Exportamos la imagen RGB en formato GEOTIFF, utilizando una escala de 20 metros por píxel y el sistema de coordenadas EPSG:3857. Utilizamos EPSG:3857 porque utiliza el DATUM WGS84 y proyecta en MERCATOR, que es el sistema de coordenadas comúnmente utilizado por GOOGLE MAPS y OPENSTREETMAPS. De esta manera, obtenemos el raster proyectado en MERCATOR directamente, que es lo que necesitamos. En la figura 4.16, se muestra el resultado de la exportación. Mientras que en la figura 4.21, se presenta una escena simple en A-Frame donde visualizamos el raster exportado sobre una malla con las alturas obtenidas del fichero DEM de Madrid. En esta representación, se pueden apreciar claramente las montañas y las alturas alrededor del aeropuerto de Madrid.

```
var geometry = ee.Geometry.Rectangle(-4.204133836988655,40.023417003380956,
  ↪ -3.253816454176155,40.744144594569384);
// Convierte la geometría a un objeto Feature y establece un nombre
var rectangulo = ee.Feature(geometry, {nombre: 'Mi rectángulo'});
// Añade el rectángulo a la vista del Mapa
Map.addLayer(rectangulo, {}, 'Rectángulo');
var ColeccionSentinel = ee.ImageCollection('COPERNICUS/S2').filterDate('2019-01-01',
  ↪ '2019-02-28');
var Mosaico = ColeccionSentinel.mosaic();
Map.addLayer(Mosaico, {max: 5000.0,min: 0.0,gamma: 1.0,bands: ['B4', 'B3', 'B2']},
  'Composicion RGB');
// Crear una imagen RGB utilizando las bandas B4, B3 y B2
var RGB = Mosaico.visualize({bands: ['B4', 'B3', 'B2'], max: 5000, min: 0, gamma: 1.0});
// Crea un objeto Projection a partir de la identificación EPSG
var epsg3857 = 'EPSG:3857';
// Descargar la imagen RGB en formato GeoTIFF
Export.image.toDrive({image: RGB,description: 'Sentinel2_RGB',scale: 20,crs:epsg3857, region:
  ↪ geometry, maxPixels: 28710052848});
```

Figura 4.19: Código Javascript para la generación de la capa raster de textura de Madrid.



Figura 4.20: Raster de Madrid generado con Google Earth Engine.





Figura 4.21: Resultado final del componente con la textura de Madrid y el fichero DEM generado.

### 4.5.3. Gestor de alturas

El gestor de alturas es una clase en JAVASCRIPT encargada de administrar el terreno en el escenario. En nuestra aplicación, utilizaremos una única instancia de este gestor para aprovechar el almacenamiento de los datos de las alturas durante la carga del terreno. Además, el gestor proporcionará funcionalidades para obtener las alturas de puntos específicos en el escenario 3D. En la inicialización del escenario principal descrita en la sección 4.2, el gestor de la escena invocará los métodos para cargar el terreno y los edificios. Un problema común al reutilizar componentes de A-FRAME creados por otros desarrolladores, como en este caso el componente utilizado para generar el mallado del terreno con THREE.JS, es la dificultad para acceder a los datos leídos del archivo binario que contiene las alturas. Esta falta de documentación y de una [API](#) específica para acceder a los datos puede ser un obstáculo. Sin embargo, al analizar la implementación del componente *afame-terrain-model-component*, descubrimos que después de leer el archivo se emite un evento que contiene el objeto con los datos de alturas. Aprovechando esta información, nos suscribimos al evento una vez que creamos el componente del terreno. De esta manera, podemos recibir y almacenar la matriz de alturas preparando nuestra instancia única para atender peticiones que requieran la altura para un punto del mundo 3D.

Es importante tener en cuenta que otros componentes, como el gestor de altura de la cámara o las geometrías de los edificios, necesitarán acceder a la altura en puntos específicos del escenario y podrán hacerlo a través de la API proporcionada por la instancia única de esta clase.



#### 4.5.3.1. Cálculo de altura para una coordenada 3D

En esta subsección, se describirán las operaciones aritméticas que se realizan para calcular y obtener la altura correspondiente a un punto o vector en el escenario 3D. El gestor de alturas tiene una referencia a la matriz de alturas utilizada para generar el mallado del terreno, lo cual nos proporciona la información necesaria sobre las alturas del archivo DEM que se está representando. No obstante, la transformación de un vector 3D a su posición correspondiente en la matriz de alturas no es trivial. En el gestor de alturas, se han implementado dos métodos para calcular la altura de un punto en el terreno. Uno de ellos es un método eficiente que utiliza fórmulas matemáticas para establecer la relación entre el vector 3D y un índice en la matriz de alturas. El segundo método es más preciso y emplea operaciones comunes en entornos 3D, como intersecciones de geometrías. Esto se ha hecho así para solucionar los problemas de resolución, debido a que si establecemos pocos puntos de resolución entre una muestra de altura y otra existirá mucho espacio y debemos usar alguna técnica para detectar la altura interpolada. A continuación, se explicarán ambos métodos y las fórmulas que se han deducido y aplicado:

1. En el método de alto rendimiento, se está haciendo:

$$index_x = \text{round} \left( \frac{\text{vector3D}_x + \frac{\text{terrainWidth}_{3D}}{2}}{\text{cellWidth}} \right)$$

$$index_y = \text{round} \left( \frac{\text{vector3D}_z + \frac{\text{terrainHeight}_{3D}}{2}}{\text{cellHeight}} \right)$$

$$index = (index_y \times (\text{gridWidth} + 1)) + index_x$$

$$height = \frac{\text{magnification}_y \times \text{heightMatrix}[index]}{65535}$$

Figura 4.22: Cálculo de conversión de coordenadas 3D al índice del array de alturas.

2. En el método de alta precisión, se busca abordar los problemas de cambios bruscos de altura que pueden ocurrir en los terrenos debido a la resolución de alturas. La precisión del terreno depende del número de celdas que se establezcan al extraer las alturas utilizando el método descrito en la sección 4.5.1. Es posible que al desplazarnos por el terreno, experimentemos cambios bruscos de altura en ciertas posiciones. Para mitigar esto, se añade un paso adicional al método de alto

rendimiento. Este paso consiste en establecer la altura del terreno de alto rendimiento más un factor constante de altura que asegure que el personaje siempre esté por encima del plano del mallado. Luego, se crea un rayo perpendicular al suelo y se calcula la intersección con la superficie real que cubre el mallado. El punto de intersección representará la altura del terreno, de esta manera no se tienen alturas discretas, sino que se calcula la posición de la superficie real que interpola los puntos discretos de las alturas.

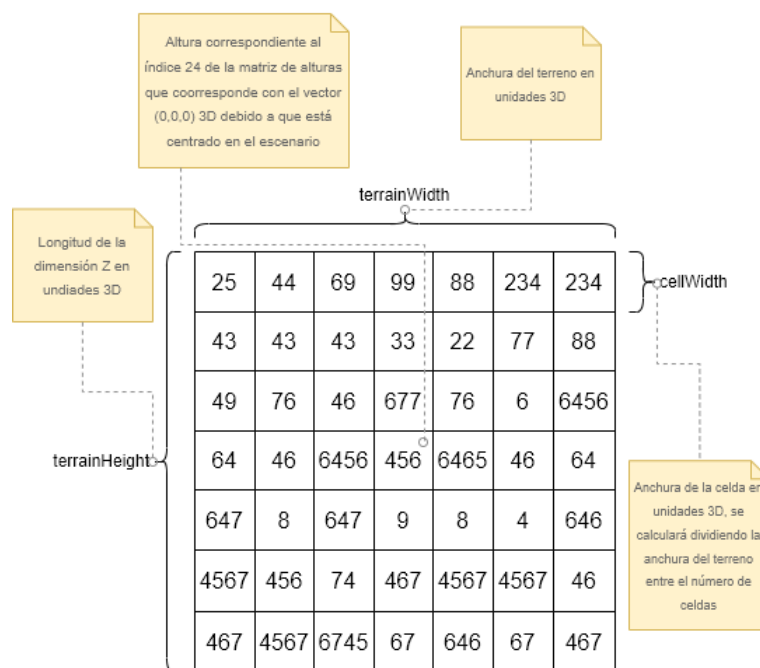


Figura 4.23: Diagrama de matriz de alturas.

Dado que el terreno está centrado en el vector  $(0,0,0)$ , es importante tener en cuenta que el cálculo de la posición en el mundo 3D estará centrado en torno al índice de la matriz que representa centro del escenario. Dado que la matriz empieza a contar los índices desde la esquina superior izquierda existe un desplazamiento con respecto al escenario que está centrado en el eje  $(0,0,0)$ . Para tener en cuenta este desplazamiento, calculamos el centro de la matriz dividiendo la anchura total del terreno por la mitad y sumando el resultado al eje x del vector 3D. Para determinar la columna correspondiente en la matriz, dividimos la posición x entre el ancho de cada celda y redondeamos el resultado para obtener un índice exacto. Esto nos dará la columna correspondiente en la matriz. De manera similar, calculamos la fila correspondiente en la matriz utilizando el eje Z. Estos cálculos nos proporcionarán la fila y columna

adecuadas en la matriz de alturas. En la figura 4.22 podemos ver las fórmulas de cálculo de índices en columna y fila que acabamos de describir.

Finalmente, es importante mencionar que estamos utilizando una representación de la matriz de alturas en formato de array unidimensional. Aunque conceptualmente es una matriz bidimensional, se almacena en una variable de tipo array de una sola dimensión. Para determinar el índice en el array al que corresponde una determinada fila y columna en la matriz, realizamos una operación básica para convertir el índice bidimensional en un índice unidimensional. La fórmula utilizada para este cálculo se muestra en la Figura 4.22. Esta fórmula nos permite acceder correctamente al valor de altura correspondiente en el array, teniendo en cuenta la relación entre la posición bidimensional y la representación unidimensional de la matriz de alturas.

Una vez obtenido el índice de la matriz, es necesario normalizar el valor correspondiente en el array de alturas. Dado que el array utiliza el tipo de dato `Uint16`, cuyo valor máximo es 65535, al dividir el valor obtenido se normaliza el rango de alturas entre  $[0, 1]$ . Si se utiliza un factor de magnificación configurado en el componente, se puede obtener la altura real utilizada en el entorno 3D. Esta relación se explica en el contexto de la última fórmula en la Figura 4.22.

En resumen para obtener la altura correspondiente a un vector de posición en el mundo 3D se realizan las siguientes operaciones:

1. Se calcula la fila y la columna relacionada con el vector de posición 3D a través de las fórmulas descritas en la figura 4.22.
2. Se transforma la fila y la columna a un índice del array unidimensional de alturas.
3. Se transforma el valor del array al valor normalizado usado por el componente, dando como resultado la altura pintada en el escenario.

#### 4.5.4. Generación de datos geoespaciales de los edificios

Para extraer la información **Geoespacial** de los edificios, nos hemos basado en el servicio proporcionado por OVERPASS<sup>3</sup>, el cual ofrece acceso a datos geoespaciales almacenados en la base de datos de OPENSTREETMAPS<sup>4</sup>. Overpass nos brinda una API que nos permite realizar consultas a la base de datos y también proporciona una herramienta en línea llamada OVERPASS TURBO<sup>5</sup>, que nos permite visualizar consultas a la API, así como visualizar y exportar los resultados de manera interactiva. Inicialmente, consideramos cargar los datos en línea en tiempo real. Sin embargo, debido a que el formato

---

<sup>3</sup><https://overpass-api.de/>

<sup>4</sup><https://www.openstreetmap.org/>

<sup>5</sup><https://overpass-turbo.eu/>

del resultado de la consulta requería un preprocesamiento adicional y a la gran cantidad de entidades que se recuperaban en cada consulta, llegamos a la conclusión de que, para el prototipo, sería mejor trabajar con una cantidad reducida de entidades preprocesadas previamente y almacenadas localmente dando así prioridad al rendimiento de la aplicación. Para descargarnos los datos haremos uso de POSTMAN<sup>6</sup> y generaremos la siguiente petición para la API:

```
https://overpass-api.de/api/interpreter?
data=(way[building] ["building:levels"]
(40.023417003380956,-4.204133836988655,40.744144594569384,-3.253816454176155);
relation[building] ["building:levels"]
(40.023417003380956,-4.204133836988655,40.744144594569384,-3.253816454176155);
);out;>;out skel qt;
```

Esto nos proporciona un archivo XML con todos los metadatos de los edificios en la zona del escenario de Madrid. El siguiente paso consistirá en convertir este archivo XML en un archivo GEOJSON que podamos utilizar para trabajar con él. Esto se debe a que la búsqueda de información en el formato de OPENSTREETMAPS es complicada debido a que los metadatos están distribuidos a través de identificadores y no se encuentran todos juntos en una única entidad, como sucede en el caso de GEOJSON.

Para llevar a cabo esta conversión, utilizaremos el proyecto OSMTOGEOJSON<sup>7</sup>, una biblioteca de código abierto disponible en GITHUB que nos permitirá convertir los datos de formato OPENSTREETMAP (OSM) al formato GEOJSON. Para realizar este proceso, simplemente debemos instalar la biblioteca utilizando NODE.JS y ejecutar su *script* correspondiente.

```
npm install -g osmtogeojson
osmtogeojson data\buildings.xml > data\buildings.geojson
```

#### 4.5.5. Geometría edificio

En esta sección, vamos a describir brevemente la implementación de la generación de edificios utilizando la información proporcionada por la API de OVERPASS.

El gestor de alturas utiliza un archivo geoespacial en formato GEOJSON para representar los edificios. Cada edificio se define usando una lista de coordenadas geoespaciales en el sistema de referencia WGS84, que describe su contorno. En ocasiones también se incluye un metadato que indica el número de pisos del edificio dependiendo la zona que consultemos, sobre todo encontraremos más metadatos de alturas en ciudades grandes. Para la generación de la geometría del edificio, necesitaremos la lista

<sup>6</sup><https://www.postman.com/>

<sup>7</sup><https://github.com/tyrasd/osmtogeojson>

de coordenadas ya convertida al espacio 3D utilizando nuestro conversor descrito en la sección 4.4.1, la altura del terreno y la altura del edificio calculada a partir del número de pisos.

Para calcular la altura del terreno, obtenemos el [Centroide](#) de la lista de puntos que definen el contorno del edificio. Luego, aplicamos las fórmulas descritas en 4.22 para determinar la altura del terreno en el punto del centroide.

En nuestra implementación, comenzamos creando una geometría bidimensional en THREE.JS a partir de la lista de coordenadas del edificio. Sin embargo, surge un problema al realizar la extrusión, ya que las coordenadas que utiliza la API de THREE.JS para extruir una forma son en 2D y al devolver la geometría 3D tendrá la siguiente forma  $(x, y, 0)$ , cuando en realidad deberían ser  $(x, 0, y)$ . Por lo tanto, después de crear la forma, aplicamos una rotación de 90 grados alrededor del *eje X* para colocar el edificio en su posición original. Luego, ajustamos la altura del edificio aplicando una traslación en el *eje Y* que incluye la altura del terreno y la altura del edificio. De esta manera, el edificio queda correctamente posicionado y anclado al terreno.

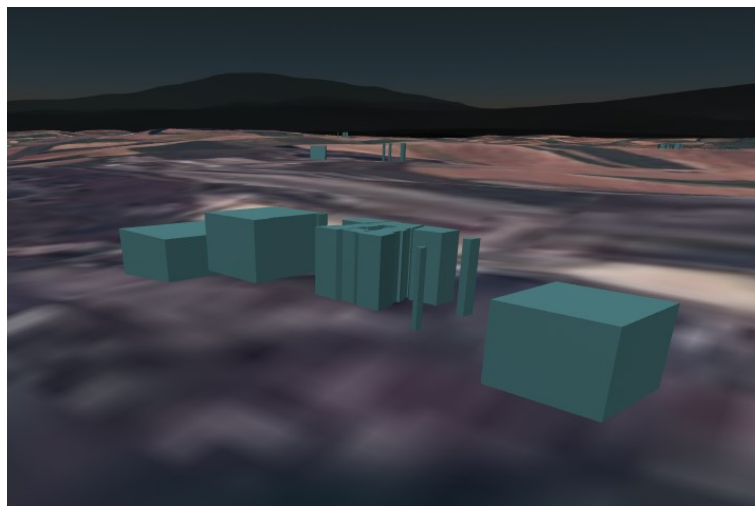


Figura 4.24: Edificios extruidos sobre el terreno.

#### 4.5.6. Componente gestor de altura de cámara

Este componente se configura en la entidad agrupadora que contiene la cámara principal y es responsable de ajustar la altura a medida que el usuario se desplaza por el escenario. Es un componente que simula el desplazamiento sobre la superficie del terreno para crear una experiencia inmersiva al caminar sobre un terreno con relieve.

La altura del personaje se mantiene constante de manera relativa con respecto a la altura del terreno. En otras palabras, la posición vertical del personaje se ajusta para mantener una altura constante con

respecto al terreno. Es importante tener en cuenta que el tamaño del personaje no sigue la misma escala que el escenario. Esto se debe a que el objetivo de la aplicación es visualizar datos a gran escala, y si el personaje principal se estableciera en la misma escala, sería difícil que se desplace rápidamente en un escenario que representa cientos de kilómetros. Estas diferencias de escala y velocidad del personaje son configurables.

El componente inicialmente se suscribe al cargador del terreno para establecer la altura correspondiente a la posición inicial del personaje una vez que el terreno se haya inicializado. También crea una función de *throttledFunction*, como se explica en 4.2.1, para no penalizar el rendimiento por introducir lógica en la función `tick`. Por lo tanto se ha configurado la función para que actualice la posición de la cámara cada 200 ms, lo que implica que la altura de la cámara se actualiza 5 veces por segundo. Esto proporciona una experiencia satisfactoria con una pequeña penalización en el rendimiento.

Para calcular la altura, se obtiene el vector actual del objeto 3D de la cámara y del objeto que agrupa al personaje y que contiene la cámara. Para obtener la posición real de la cámara en el mapa, se realiza un cálculo teniendo en cuenta la configuración establecida en el proyecto para el personaje principal, como se muestra en la figura 4.25.

En la configuración, podemos observar que el esqueleto del personaje principal contiene un componente

```
<!-- Camera -->
<a-entity id="rig" position="0 0 0" movement-controls terrain-height>
  <a-entity id="camera" hud camera look-controls="reverseMouseDown:false"
    ↪ cursor="rayOrigin: mouse; fuse: false"
      raycaster="far: 4000; objects: .clickable" position="0 0 0" toolbar3d>
</a-entity>
<a-entity id="left-hand" oculus-touch-controls="hand: left" laser-controls="hand: left"
  raycaster="far: 4000; objects: .clickable"></a-entity>
<a-entity id="right-hand" oculus-touch-controls="hand: right" laser-controls="hand: right"
  raycaster="far: 4000; objects: .clickable"></a-entity>
<a-entity id="cameraOnBoarEntity" camera="active: false"
  ↪ camrender="cid:cameraOnBoard; fps:25" position="0 0 0"
  rotation="0 -180 0"></a-entity>
</a-entity>
```

Figura 4.25: Código que configura el personaje principal, compuesto por la cámara y las manos.

*movement-controls* del proyecto A-FRAME EXTRAS<sup>8</sup>, que permite un movimiento del personaje compatible con el teclado y los joysticks de varios dispositivos de realidad aumentada, como las gafas de realidad aumentada OCULUS.

<sup>8</sup><https://github.com/c-frame/aframe-extras/tree/master>

Por otro lado, la cámara principal, que representa la cabeza del personaje, contiene el componente *look-controls* de A-FRAME, que nos permite mover la cabeza tanto con el ratón como con los sensores de posición de las gafas de realidad aumentada. Debido a esto, la posición real de la cámara es relativa al componente del personaje principal. Es por eso que, en nuestro cálculo, extraemos ambos vectores y los sumamos para obtener la posición absoluta de la cámara en el escenario.

Una vez calculada la posición de la cámara principal en el escenario, utilizamos la instancia del gestor de alturas descrito en la sección 4.5.3 para llamar a la función que nos devuelve la altura a través de un vector 3D del escenario. Con esa altura, sumada a la altura del personaje, establecemos la coordenada Y del vector del objeto 3D de la cámara, ajustando la altura en función de la posición en el escenario un total de 5 veces por segundo.

## 4.6. Gestión de acceso a los datos

En esta sección, vamos a analizar los componentes clave de la arquitectura principal encargados del acceso a los datos. En el prototipo desarrollado, se pueden configurar dos tipos de comportamientos.

El primero se refiere a la escena principal funcionando como una representación de datos en caché. En este caso, se muestra un histórico de datos obtenidos a través de la pieza responsable de guardar datos de la API dentro del prototipo o de otra fuente de datos históricos. El único requisito es que estos datos mantengan el mismo formato **JSON** proporcionado por la API de OPENSKY. Sin embargo, en el futuro, se podrían desarrollar fácilmente componentes adicionales que puedan leer otros formatos de datos, ya que toda la información está referenciada a la misma clase, la cual es precargada por el módulo de configuración. Por lo tanto, si cambian las posiciones de la información proveniente de otro servidor, solo se requeriría reconfigurar las posiciones en el archivo de configuración que precarga la escena.

El segundo comportamiento se refiere a la representación en tiempo real del tráfico aéreo. En este caso, los datos provienen directamente del servicio OPENSKY REST API, que nos proporciona datos ADS-B en tiempo real. Estos datos se representarán en el escenario, lo que permitirá mostrar el espacio aéreo en un entorno 3D en tiempo real. Esta funcionalidad resulta especialmente útil como punto de partida para aplicaciones de control de espacios aéreos.

### 4.6.1. Datos caché de vuelo

En esta sección, vamos a describir cómo se gestiona la caché a través de la clase *FlightCacheData*, que se encuentra en el archivo *FlightCacheData.js* dentro del módulo de gestión de datos. Estos objetos se instancian cuando un vuelo aparece en el escenario y se mantienen siempre y cuando el vuelo esté

presente en cada lectura de datos de la API. Actúan como objetos **DTO** (*Data Transfer Object*) para mantener el estado del avión. Estos objetos son necesarios para el cacheo de la posición antigua dentro del escenario, lo cual es crucial para generar animaciones que proporcionen un movimiento fluido de los aviones. Además, cada vez que se actualiza la posición de un avión, se emite un evento al componente responsable de pintar el trayecto del vuelo (descrito en la sección 4.7.6). De esta manera, el componente puede actualizar su geometría y mostrar el trayecto de los aviones desde su entrada en el escenario hasta su posición actual.

Como ya se describió en la sección 4.2.2, estos objetos se almacenan en un mapa indexado a través de su identificador para que su recuperación sea eficiente, servirán para mantener el estado de un vuelo específico en el escenario. Esto permite que el gestor del escenario genere una animación utilizando el vector de la caché y el nuevo vector leído de la API para lograr un movimiento fluido, y también actualiza la geometría responsable de representar el trayecto del avión en el escenario mediante un evento emitido sobre el escenario.

#### 4.6.2. Objeto de acceso a datos ADS-B (DAO)

El objeto de acceso a datos, comúnmente conocido como **DAO** (*Data Access Object*), es el objeto responsable de la obtención de los datos ADS-B. Como se muestra en la figura 4.26, el DAO accede al gestor de configuración para verificar si está configurado para acceder en tiempo real o utilizar datos en caché. En el caso de estar configurado para usar datos en caché, el DAO consulta al gestor de configuración la carpeta y el índice desde donde debe comenzar a leer. En cada evento de actualización, el DAO lee un archivo **JSON** de la carpeta especificada y lo proporciona al gestor principal de la escena.

Si la configuración del DAO está establecida en tiempo real, se realizará una petición a una API REST para obtener los datos en formato JSON. El DAO consultará al módulo de configuración para obtener el usuario y la contraseña necesarios para autenticarse en la API. Es importante destacar que la API nos permite realizar consultas dentro de una zona delimitada por coordenadas geodésicas. Por lo tanto, la petición se realizará dentro de los límites geodésicos establecidos en la configuración del escenario. La respuesta JSON de la API contendrá los vuelos que se encuentren dentro de la escena especificada.

El gestor principal de la escena utiliza los datos proporcionados por el DAO para crear las instancias en caché de cada vuelo. Como se muestra en la figura 4.26, el gestor principal mantiene un mapa que almacena la caché de los vuelos, indexados por su identificador único de vuelo (ICAO24). En cada evento temporal, el gestor principal se encarga de mantener la tabla de caché. Si un vuelo no ha sido actualizado, se elimina de la tabla. Si ha sido actualizado, se actualiza la instancia de datos en caché



correspondiente. Si el vuelo no existe en la tabla, se inserta en ella. De esta manera, la tabla de caché representa de manera precisa los vuelos presentes en la escena en ese momento específico.

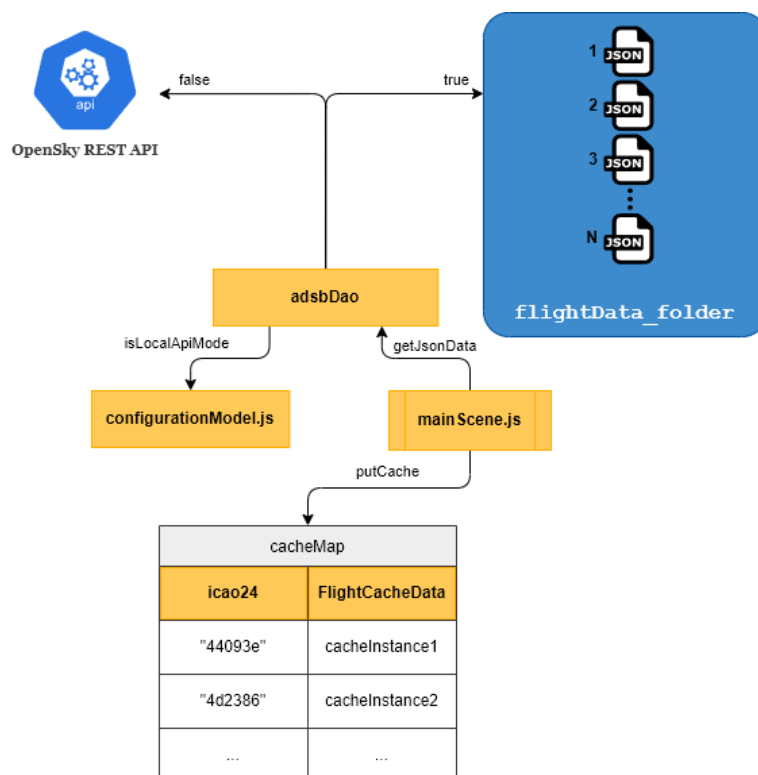


Figura 4.26: Diagrama del objeto de acceso a datos ADS-B (DAO).

#### 4.6.3. Consulta y almacenamiento de datos a la carpeta caché

Para asegurar el correcto funcionamiento del componente de acceso a los datos mencionado en la sección anterior 4.6.2, es necesario contar con la implementación de una pieza responsable del almacenamiento de los datos en una carpeta de caché local. Así posteriormente, el DAO puede consultar esta carpeta en caso de que esté configurado en el modo local. Este componente está implementado en JAVASCRIPT y se ejecuta en el entorno de NODE.JS. Su función principal es cargar previamente la configuración de un escenario específico, que incluye parámetros como el rectángulo geodésico para la obtención de datos y el intervalo de tiempo en el que se debe establecer el temporizador para realizar una petición a la API en cada evento. Además, este componente se encarga de almacenar el resultado de cada petición en un archivo dentro de la carpeta de caché, como se muestra en la figura 4.27.

Como podemos apreciar en la figura 4.28, en cada petición almacenamos en la carpeta local que usaremos de cache, un fichero JSON con los vectores de posición de los vuelos en ese instante temporal. Estos ficheros podrán ser leídos por el DAO, el componte anteriormente descrito en la sección 4.6.2.

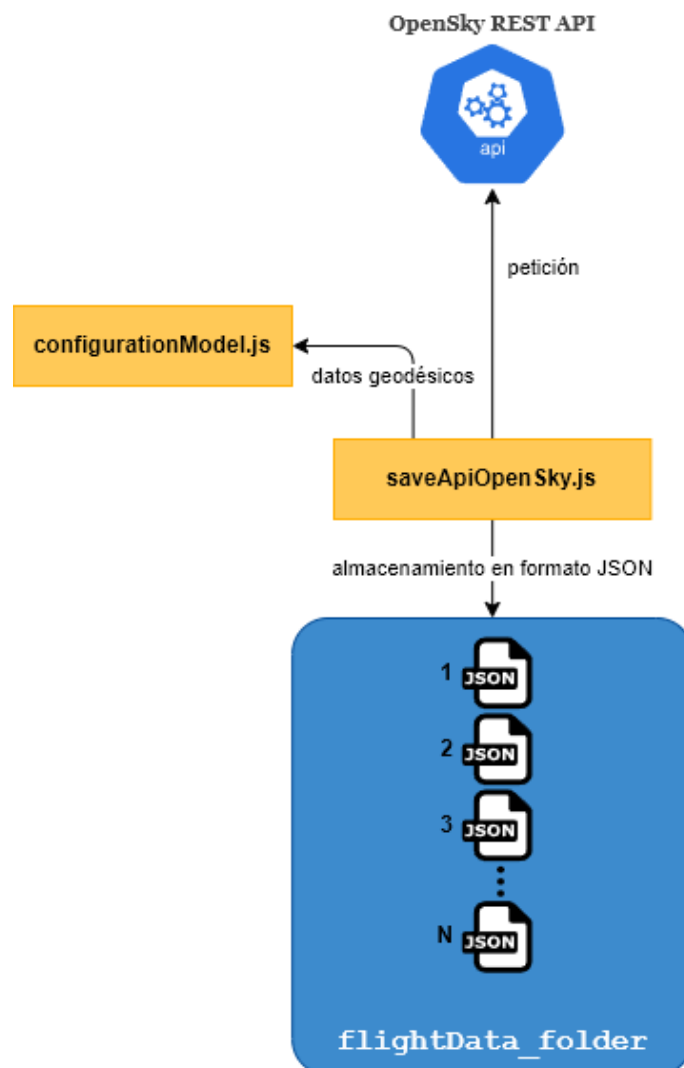


Figura 4.27: Diagrama de consulta y almacenamiento de datos ADS-B en caché.

```

PS C:\Users\djpra\Documents\workspaceTFG\AFrameTest\js> node .\saveMadrid.js
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response0.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response1.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response2.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response3.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response4.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response5.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response6.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response7.json
saving filename:C:\Users\djpra\Documents\workspaceTFG\AFrameTest\flightData_madrid2\response8.json

```

Figura 4.28: Ejecución con Node.js del almacenamiento de datos ADS-B en caché.

## 4.7. Gestión de la interfaz de usuario

En esta sección, vamos a examinar las soluciones y componentes implementados para la interacción del usuario con la aplicación, tanto en el modo de escritorio como en el modo de realidad virtual. En el modo de realidad virtual, se hace uso de gafas y mandos como las gafas OCULUS VR para manipular las entidades en el entorno tridimensional.

El objetivo principal de todos los componentes desarrollados en esta aplicación es proporcionar al usuario la capacidad de consultar toda la información presente en la escena y visualizarla de una manera que sea equivalente a las posiciones reales de las entidades en el mundo real, pero al mismo tiempo, que sea manejable por el usuario. Esto significa que, aunque en el mundo real no podemos ver con claridad una entidad que represente un avión situado a 60 km de distancia, la aplicación tiene como objetivo seguir permitiendo el acceso a todas las entidades del escenario, independientemente de su ubicación en relación al usuario.

### 4.7.1. Entidades en ampliación al alejarse

Uno de los desafíos encontrados en la usabilidad de la aplicación se refiere a la selección de aviones para visualizar sus datos. Dado que los aviones en la realidad se encuentran a distancias significativas en términos de altura y dimensiones del escenario, sin aplicar técnicas para visualizar objetos distantes, sería imposible seleccionarlos y visualizarlos correctamente.

Para abordar este problema, se ha optado por una solución que sacrifica un poco de realismo dentro del escenario. Cuando un objeto supera cierto umbral de distancia, se aplica un proceso de ampliación que simula que el tamaño del símbolo del avión se mantiene más o menos constante. Con esta solución, facilitamos al usuario la capacidad de seleccionar aviones distantes. Puedes observar las capturas de pantalla que ilustran la escena con el componente de ampliación y sin el en las figuras 4.31 y 4.30.

$$\text{Factor de Ampliación} = \left( \frac{\text{distancia} - \text{distancia umbral}}{\text{divisor ampliación}} \right) + 1$$

Figura 4.29: Calculo del factor de ampliación en función de la distancia.

El componente se agrega a cada entidad que representa un vuelo y calcula la distancia entre la cámara principal y la entidad. Si la distancia supera un umbral configurado, se aplica un escalado a la entidad en cada repintado. La fórmula utilizada para el escalado se muestra en la figura 4.29. Esta fórmula calcula

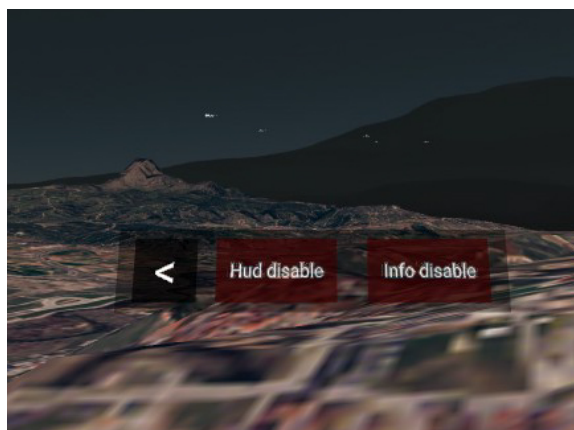


Figura 4.30: Aviones sin ampliación.

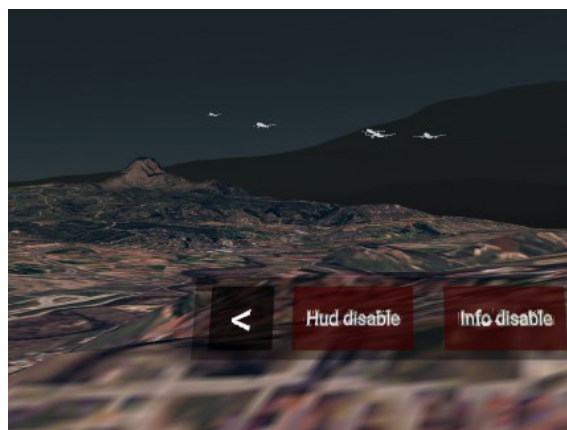


Figura 4.31: Componente de ampliación.

un factor de ampliación en función de la distancia, donde se asegura que el factor sea 1 en el umbral, lo que significa que no se realiza ningún escalado. A medida que la distancia aumenta, el factor se incrementa linealmente, y se puede ajustar el incremento mediante un factor divisor configurable. Todos estos parámetros, como el divisor del factor de ampliación y la distancia a partir de la cual se comienza a aplicar el escalado, son configurables, lo que brinda al usuario la capacidad de ajustar la visualización de los tamaños en diferentes distancias según sus preferencias.

#### 4.7.2. Componente de información contextual interactiva

En esta sección, describiremos el componente encargado de mostrar los metadatos de las entidades geoespaciales presentes en el escenario. Dado que el objetivo de la aplicación es visualizar información de entidades georeferenciadas, resulta interesante desarrollar un componente que inserte texto sobre las entidades del escenario que contengan metadatos.

Como se explicó anteriormente en la sección 4.5.5, al procesar la información de la capa de edificios extraída de los datos de OPENSTREETMAP, tenemos la capacidad de extraer los metadatos asociados a cada edificio generado. Por lo tanto, se ha creado un componente al cual se le puede proporcionar un texto como argumento y tiene la responsabilidad de generar un texto que siempre mire hacia la cámara y se posicione sobre la entidad correspondiente. Este texto solo es visible cuando el ratón o el *raycaster* están posicionados sobre la entidad. Además, el componente genera un material de color rojizo para indicar el elemento seleccionado del cual se muestra la información. De esta manera, podemos visualizar los metadatos de edificios, como se muestra en la figura 4.32.

Como veremos más adelante, este componente tiene la capacidad de ser activado o desactivado mediante eventos enviados a través de la barra de herramientas (vease la sección 4.7.3), la cual está presente

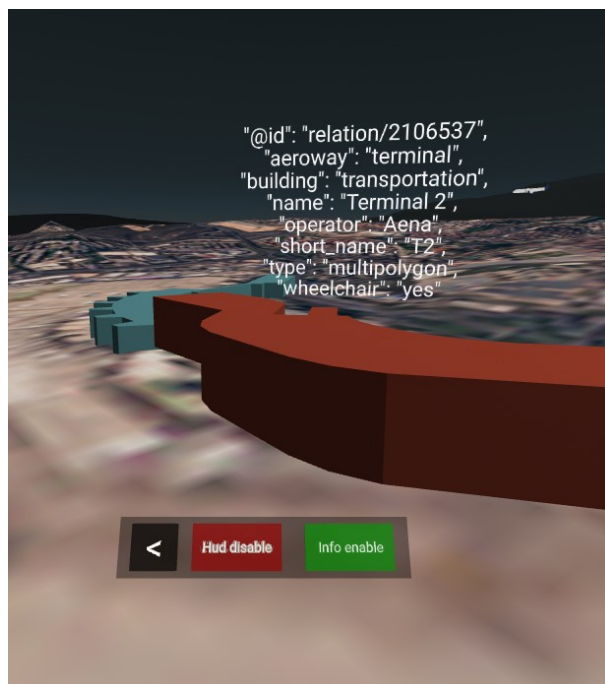


Figura 4.32: Componente información contextual mostrando metadatos de la terminal.

como una interfaz de usuario en la cámara principal. Esto brinda al usuario la flexibilidad de controlar la visualización de los metadatos según sus necesidades y preferencias.

### 4.7.3. Componente barra de herramientas de la interfaz de usuario

En esta sección, exploraremos el componente que acompaña al usuario y permite habilitar o deshabilitar funcionalidades. La barra de herramientas es una entidad 3D que se despliega como un **HUD** (*Head-Up Display*) en forma de pantalla frontal de visualización. Proporciona al usuario una interfaz para interactuar con las diversas funcionalidades de la aplicación.

Dado que nos encontramos en un entorno 3D, hemos implementado la barra de herramientas como un componente 3D dentro de la escena, anclado a la cámara principal. Sin embargo, nos hemos enfrentado al desafío de que esta barra puede resultar molesta al bloquear parte de la vista del usuario. Para solucionar este problema, hemos diseñado el componente de manera que pueda plegarse y arrastrarse, brindando al usuario la capacidad de colocarlo en una posición menos intrusiva y ocupando un espacio mínimo en la pantalla. En la figura 4.33 podemos ver la barra de herramientas en su tamaño desplegado mientras que en la figura 4.34 se aprecia el poco espacio que ocupa una vez la plegamos.

La solución técnica implementada consiste en la creación de un componente que se configura dentro de la jerarquía de la cámara principal. Durante la inicialización de este componente, se agregan geometrías cuadradas de A-FRAME de tipo *a-plane* a la cámara principal para formar tanto el panel como los boto-

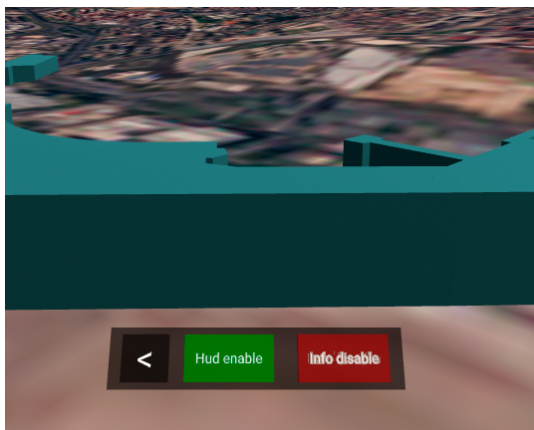


Figura 4.33: Barra de herramientas desplegada.



Figura 4.34: Barra de herramientas plegada.

nes conmutables. Esto se puede observar en la figura que muestra la disposición de los componentes en la barra de herramientas (4.35). La barra de herramientas está representada por un plano principal con

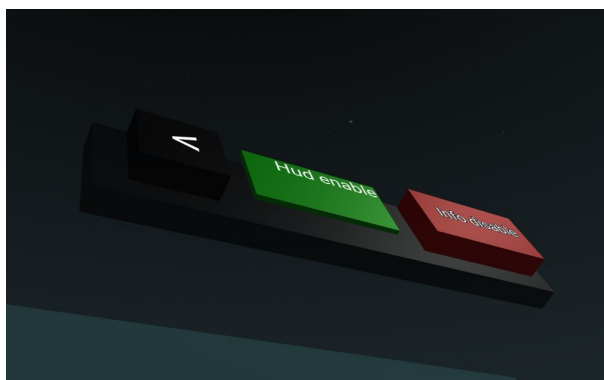


Figura 4.35: Maquetación del componente barra de herramientas.

transparencia, mientras que los botones se representan mediante tres planos adicionales con texto superpuesto. Estas geometrías planas tienen añadidos controladores de eventos para permitir la interacción con el usuario.

Cada botón se crea mediante una función interna (ver Código 4.36) que recibe varios argumentos. Estos

```
createToolbarButton: function (id,width, height, text,texSize, position,
↪ toggle, enableFunction, disableFunction, enableColor, disableColor,
↪ enableText, disableText)
```

Figura 4.36: función que genera los botón dentro de la barra de herramientas.

argumentos incluyen un identificador único utilizado para hacer referencia al botón en el contexto del

**DOM**, las dimensiones de la entidad que determinan su ancho y alto, el texto, la posición dentro de la barra de herramientas, si el botón es conmutable o no, la acción a ejecutar cuando el botón está activado y la acción a ejecutar cuando el botón está desactivado. Además, se pueden especificar propiedades visuales para el botón cuando está en estado activado y cuando está en estado desactivado, como el color y el texto correspondiente. Esta configuración flexible y modular hace que nuestra barra de herramientas sea altamente reutilizable para otros proyectos y fácilmente extensible para añadir más funcionalidades dentro de la aplicación. En la sección 3.4.3.2 mostramos un ejemplo de como reutilizar la barra de herramientas con mas detalles sobre la función que acabaos de describir. Nuestra aplicación, haciendo uso del método anteriormente descrito nuestro componente barra de herramientas crea dos botones conmutables que permiten habilitar la funcionalidad de selección de aviones y visualizar su información a través de un panel **HUD** que contiene acciones adicionales. Además, hemos incluido un botón que activa y desactiva la funcionalidad de texto contextual, como se describe en la sección anterior 4.7.2.

#### 4.7.3.1. Animación de la barra de herramientas

Para lograr la funcionalidad de plegado, se han implementado animaciones que actúan sobre la escala y opacidad de los componentes de la barra de herramientas. Cuando el usuario presiona el botón de plegado, la barra de herramientas oculta los botones mediante una animación que los desvanece gradualmente a través de la opacidad. Luego, la barra de herramientas se comprime horizontalmente, lo que también afecta al botón de plegado, haciendo que se comprima en tamaño. Para contrarrestar esta compresión, se realiza una animación inversa en el botón de plegado, expandiéndolo horizontalmente y restaurándolo a su tamaño original. Esto crea un efecto visual que contrarresta la compresión causada por el plegado de la barra de herramientas tal y como aclaramos en el diagrama 4.37.

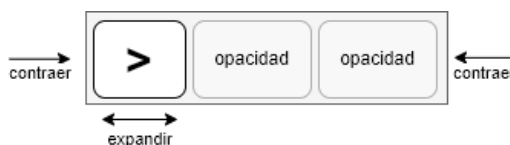


Figura 4.37: Animaciones para plegar la barra de herramientas.

#### 4.7.4. Componente pantalla frontal de visualización (HUD)

El elemento **HUD** es esencial en nuestra aplicación, ya que proporciona al usuario la visualización de datos de los aviones, así como funcionalidades adicionales relacionadas con los aviones seleccionados.

Uno de los principales objetivos de nuestra aplicación es permitir al usuario ver la información de los aviones en tiempo real o en diferido. Cuando habilitamos la funcionalidad del **HUD** en la barra de



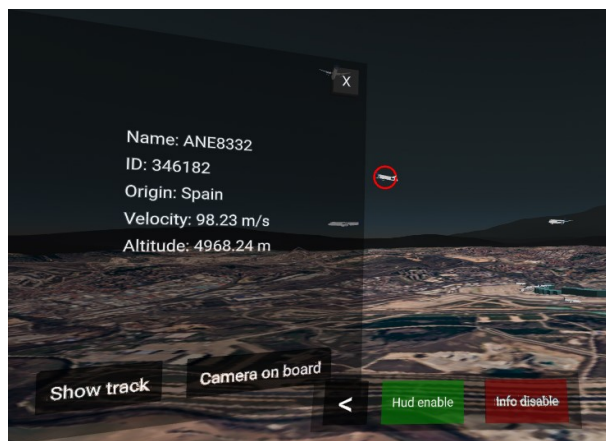


Figura 4.38: Componente HUD con avión seleccionado.

herramientas, le brindamos al usuario la capacidad de seleccionar un avión y desplegar un panel informativo. Este panel contiene una serie de datos obtenidos a través de metadatos ADS-B, y se actualiza mediante eventos en cada consulta a la API. En resumen, al activar la funcionalidad del HUD en nuestra barra de herramientas, permitimos al usuario seleccionar aviones y ver información actualizada de los mismos, como la altura y velocidad, basada en metadatos ADS-B obtenidos a través de consultas a la API. En la figura 4.38, podemos observar que cuando seleccionamos un avión, se agrega una entidad en forma de anillo que rodea al avión y siempre está orientada hacia la cámara. Esta entidad se desplaza junto con el avión y permite al usuario realizar un seguimiento visual del avión seleccionado. Además, proporcionamos al usuario dos acciones adicionales relacionadas con el avión seleccionado.

La primera acción consiste en visualizar el recorrido del avión en el escenario. Como se muestra en la figura 4.39, cuando un avión está seleccionado, se muestra un botón que habilita una línea azul que representa el trayecto que ha seguido el avión desde que ingresó al escenario hasta el momento actual. Esta línea muestra todas las posiciones que la entidad del avión ha ocupado durante su presencia en el escenario.

La segunda funcionalidad que ofrece el panel HUD es la capacidad de visualizar una cámara de a bordo a través del panel. Esto brinda al usuario la experiencia de visualizar lo mismo que pueden ver los pasajeros del vuelo. La cámara de a bordo se proyecta en una pantalla que se despliega con una animación justo frente al panel HUD, como se muestra en la figura 4.40.



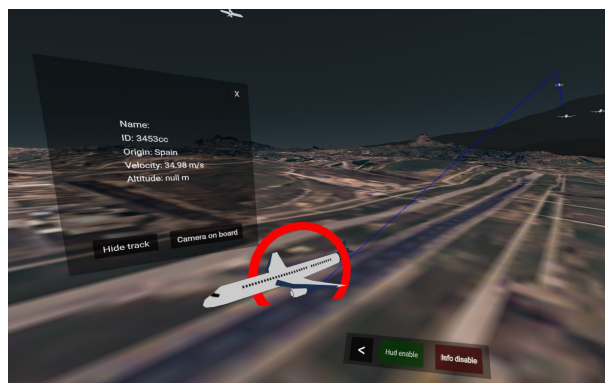


Figura 4.39: Trayecto del avión en el escenario.

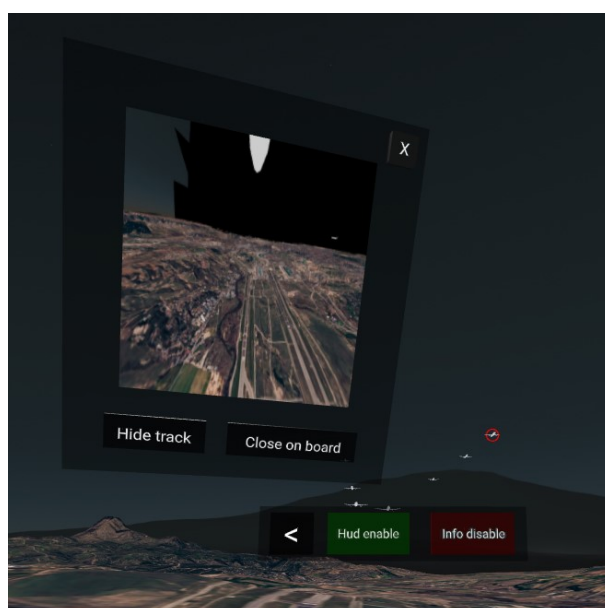


Figura 4.40: Visualización de la cámara de a bordo sobre el HUD.

#### 4.7.5. Desplazamiento de la barra de herramientas y pantalla frontal de visualización (HUD)

Como mencionamos anteriormente, es de vital importancia brindar al usuario principal la capacidad de desplazar los componentes HUD de nuestra aplicación. Por lo tanto, en esta sección abordaremos el componente que nos permite arrastrar y mover nuestros componentes HUD a cualquier parte de nuestra área visual. Tanto el componente pantalla frontal de visualización 4.7.4 como el componente barra de herramientas 4.7.3 configuran en su entidad principal el componente detallado en esta sección llamado *custom-draggable* que será responsable de proporcionar al usuario principal la capacidad de desplazar las entidades HUD donde no le moleste.

Implementar esta funcionalidad requirió mucho tiempo y esfuerzo, probando diversas estrategias e

incluso explorando bibliotecas como *A-frame-super-hands*<sup>9</sup>, aunque sin éxito. Esto se debe a que nuestro caso difiere significativamente de las aplicaciones convencionales. El componente *super-hands* está diseñado para agarrar y mover componentes que se encuentran en una posición absoluta, pero no está preparado para componentes que se encuentran en una posición relativa dentro de la estructura del esqueleto del usuario principal. Esta es la principal razón por la que no fue posible reutilizar ningún componente de las bibliotecas probadas, y se tuvo que desarrollar un componente propio. En este proceso, se encontró una solución ingeniosa para calcular la posición de manera eficiente, adoptando una estrategia diferente a cómo lo hacen las bibliotecas mencionadas. Además, se invirtieron muchas horas de investigación para garantizar que el componente fuera compatible tanto con gafas de realidad virtual como con un ratón convencional.

En el archivo *custom-draggable.js*, se ha creado un componente que permite arrastrar entidades que están contenidas dentro de la cámara principal. Es importante mencionar que en A-FRAME, cuando agregamos una entidad con geometría dentro de otra entidad, la posición de la entidad será relativa a su entidad padre. Nuestras entidades HUD se basan en este concepto, por lo tanto, tanto la barra de herramientas descrita en la sección 4.7.3 como nuestra pantalla frontal de visualización descrita en la sección 4.7.4, son entidades secundarias del esqueleto principal que representa al usuario (*rig*) tal y como podemos ver en la figura 4.41.

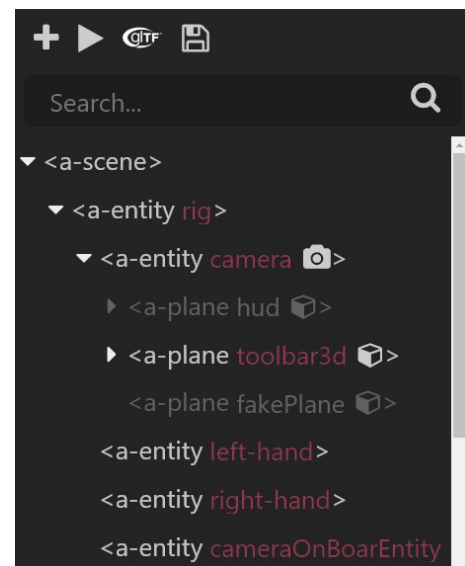


Figura 4.41: Jerarquía de entidades.

Para permitir que los componentes que viajan con la cámara principal puedan ser arrastrados, vamos a aclarar el proceso. En primer lugar, es importante tener en cuenta que la complejidad radica en garantizar que el elemento se mueva siempre en el mismo plano, sin cambios en la profundidad, solo en las coordenadas horizontales y verticales. Además, cuando arrastramos el componente con el ratón o el mando de las gafas de realidad virtual, es fundamental que siga el movimiento realizado para dar la sensación de arrastre. Después de explorar diversas estrategias, llegamos a la conclusión de que la técnica más sencilla y eficiente es utilizar un *Raycaster*. Un componente *raycaster* es una herramienta fundamental en entornos 3D que permite realizar intersecciones y detecciones de colisiones en la esce-

<sup>9</sup><https://github.com/c-frame/aframe-super-hands-component>

na. Su funcionamiento se basa en el trazado de un rayo desde un origen en una dirección determinada. Cuando el rayo colisiona con las entidades envía eventos a los que los componentes pueden suscribirse. Cuando recibimos el evento de presionar el botón para arrastrar en la entidad, guardamos el componente

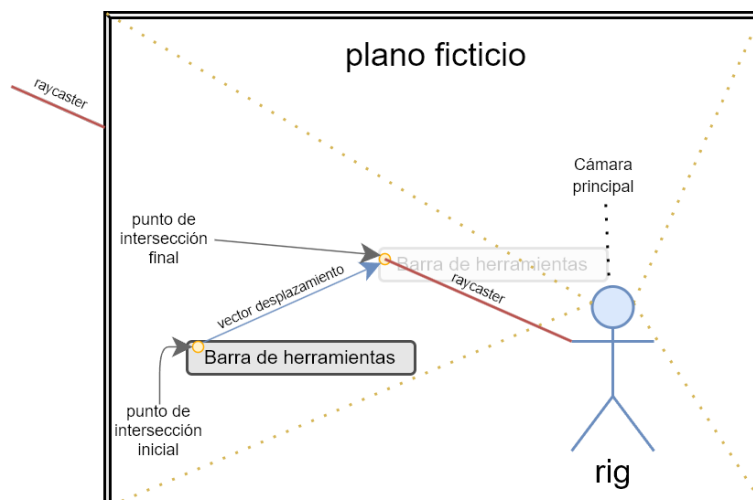


Figura 4.42: Cálculo del vector desplazamiento de arrastre de entidades.

*raycaster* que generó dicho evento, dentro de la instancia para recuperarlo en los eventos posteriores. También registramos la posición de intersección entre el *raycaster* y el elemento HUD y hacemos transparente la entidad para dar la sensación de que la tenemos agarrada. Después, cambiamos al estado de presionado y creamos una entidad plana ficticia e invisible que colocamos justo frente a la cámara principal, a la misma distancia que el elemento HUD tal y como vemos en la figura 4.42. A partir de este punto, mientras estemos en el estado de "presionado" (es decir, arrastrando), en cada evento de la función *tick*, calculamos la intersección entre el plano ficticio y el *raycaster* que generó el evento de agarre. A partir de esta intersección, determinamos el desplazamiento realizado por el controlador, que es un vector que contiene los incrementos de las coordenadas horizontales y verticales. Luego, convertimos este vector en uno relativo a la cámara y lo sumamos al vector de posición del elemento agarrado. De esta manera, logramos arrastrar y mover cualquier entidad que esté dentro de la jerarquía de la cámara en el plano deseado. Finalmente, cuando soltamos el botón del mando o el ratón, cambiamos al estado de no presionado para fijar la posición de la entidad y evitar que siga siendo desplazada en cada evento de la función *tick*.

Es importante destacar que todas las entidades HUD contienen el componente *look-at*<sup>10</sup> de A-FRAME, el cual garantiza que siempre estén orientadas hacia la cámara.

<sup>10</sup><https://github.com/supermedium/superframe/tree/master/components/look-at/>

#### 4.7.6. Componente trayecto realizado por un vuelo

El componente *track*, contenido en el archivo *track.js*, es un componente que se suscribe a los eventos del objeto *DTO FlightCacheData* descrito en el apartado 4.6.1 y del panel *HUD* descrito en el apartado 4.7.4. Es responsable de dibujar una línea que muestra el trayecto del avión en el escenario en caso de que el botón conmutable del *HUD* que activa la funcionalidad del vuelo seleccionado esté habilitado. Este componente solo escucha eventos de su vuelo, por lo tanto, solo usará el *tag* correspondiente al identificador de vuelo, escuchando eventos de los siguientes prefijos a través de eventos en el escenario:

- *flightCacheData.push*\_: Evento recibido por el objeto *FlightCacheData* para actualizar la lista de puntos que componen la línea.
- *flightCacheData.show*\_: Evento recibido por el panel *HUD* emitido por el botón que activa la funcionalidad.
- *flightCacheData.hide*\_: Evento recibido al desactivarse el botón conmutable presente en el *HUD*, que desactiva la funcionalidad.
- *flightCacheData.clear*\_: Evento emitido cuando el vuelo desaparece del escenario para borrar el elemento de la escena.

El dibujado del trayecto se realizará mediante la creación de una línea de *THREE.JS* mediante un *buffer* de puntos.

## Capítulo 5

# Conclusiones

### 5.1. Consecución de objetivos

Se ha desarrollado una plataforma de herramientas para la construcción de escenarios que brinda al usuario una experiencia más inmersiva en comparación con otras aplicaciones bidimensionales similares. He recreado parte de la funcionalidad ofrecida por las aplicaciones y servicios web de visualización de datos aeronáuticos, como se menciona en la sección 1.6, pero en un entorno tridimensional mediante la implementación de un ecosistema de componentes que permite al usuario visualizar datos interactuando con el escenario aprovechando las tecnologías de realidad virtual.

He creado un conjunto de componentes que permiten ser reutilizados para otras aplicaciones de manera sencilla y he documentado como se pueden reutilizar, proporcionando una licencia para su reutilización o modificación, se pretende que todos los avances de este proyecto puedan servir para futuras aplicaciones en otros proyectos, como es el ejemplo de todos los avances en la interfaz de usuario que pueden ser de gran utilidad independientemente de la aplicación que se requiera realizar.

He creado dos escenarios que representan el entorno que rodea a los aeropuertos de *Adolfo Suárez* Madrid y el aeropuerto de *Châlons-Vatry* en Francia. Esto ha sido posible gracias a la creación de una plataforma sólida que permite reproducir escenarios que representen aviones y entidades georeferenciadas en cualquier parte del mundo y en las dimensiones necesarias. Esta plataforma proporciona una base sólida que se puede ampliar fácilmente en términos de funcionalidad.

Este ecosistema de componentes implementados en el prototipo, no se limita únicamente a aplicaciones de datos aéreos, sino que puede ser utilizado para cualquier tipo de aplicación GIS, brindando un escenario donde se puede consultar información georeferenciada. Por ejemplo, podría ser utilizado para la consulta de una red eléctrica por parte de una compañía que necesite visualizar entidades como acometidas, postes, cables de alta tensión, centros de transformación, entre otros, todo en un entorno

tridimensional que permite consultar metadatos sobre una entidad seleccionada. Esto proporciona una herramienta de gran valor añadido a un técnico, permitiendo visualizar donde se encuentra una avería o como planificar una obra para una ampliación de la red.

En resumen, esta plataforma es adecuada para cualquier tipo de aplicación de Sistemas de Información Geográfica que requiera visualizar entidades geoespaciales.

A nivel personal, ha sido una experiencia muy gratificante avanzar en cada iteración del proyecto y aprender nuevas bibliotecas y tecnologías que enriquecen mi conocimiento profesional. Cada obstáculo superado durante el desarrollo del proyecto ha representado una lección aprendida, la cual seguramente será de gran utilidad en mi futuro profesional.

## 5.2. Aplicación de lo aprendido

### 5.2.1. A través de la titulación

- **Fundamentos de la Programación:** Esta asignatura sentó las bases fundamentales de la programación, proporcionándome los conocimientos necesarios para desenvolverme en cualquier lenguaje de programación. Aprendí las herramientas principales utilizadas para implementar la lógica de negocio en diversos lenguajes de programación.
- **Aplicaciones Telemáticas:** Durante esta asignatura, adquirí experiencia en el desarrollo de aplicaciones del lado del *Front-End*, utilizando lenguajes como *JavaScript* y familiarizándome con el paradigma de la programación funcional. Además, amplí mis conocimientos en el desarrollo de componentes orientados a eventos, donde la lógica se encuentra desacoplada y la comunicación se realiza a través de eventos en el *DOM*.
- **Prácticas Académicas Externas:** Realicé mis prácticas en la empresa Indra, donde trabajé en un software de sistemas de información geográfica. Esta experiencia fue fundamental para mi desarrollo en el proyecto actual, especialmente en el aspecto *Geoespacial*. Adquirí conocimientos sobre proyecciones, datums y transformaciones necesarias para el manejo de ficheros binarios de alturas, donde una matriz define un espacio georeferenciado.

### 5.2.2. De forma autodidacta

- **A-Frame:** Durante el proceso de desarrollo, he adquirido conocimientos a través de la documentación de A-FRAME [1]. Esta documentación me ha permitido aprender a crear una amplia variedad

de componentes y estrategias relacionadas con la solución de problemas típicos en programación de entornos tridimensionales.

- **Three.js:** He adquirido conocimientos de forma autodidacta sobre cómo extruir geometrías complejas para la creación de edificios utilizando la librería THREE.JS.
- **Google Earth Engine:** Con el fin de lograr escenarios más realistas, he tenido que investigar y aprender a utilizar por mi cuenta la API proporcionada por GOOGLE para obtener imágenes satelitales georreferenciadas. Estas imágenes me han permitido obtener texturas realistas para el terreno de mi aplicación.
- **Oculus Quest:** He tenido que aprender a utilizar dispositivos de realidad virtual y adaptarlos a A-FRAME para integrar los controladores en la aplicación. Esto ha permitido que el usuario pueda utilizar sus manos para interactuar con las entidades que representan la interfaz gráfica, como agarrar objetos y realizar acciones dentro de la aplicación.
- **OpenSky:** He aprendido de forma autodidacta a utilizar la API de OPENSky mediante peticiones REST y a interpretar todos sus metadatos.
- **Node JS:** He aprendido a utilizar *npm* y NODE.JS para crear procesos por lotes que me permiten descargar datos de las API de OPENSky y almacenarlos en una caché local.
- **Overpass-api:** Aprendí de manera autodidacta a utilizar las sentencias principales para consultar y obtener datos en los servidores de OPENSTREETMAP. Además, aprendí a descargar y convertir los datos para poder utilizarlos dentro del prototipo.
- **Leaflet:** He aprendido a utilizar una librería para proyectar coordenadas y posteriormente realizar transformaciones afines, lo que me ha permitido realizar conversiones entre sistemas de referencia en el entorno 3D y sistemas geoespaciales.
- **Entorno de programación JavaScript:** He aprendido a instalar y configurar un entorno de desarrollo utilizando el editor de código VSCODE y un servidor local. Esto me ha permitido depurar el código JAVASCRIPT en tiempo de ejecución, lo que facilita la solución de errores y proporciona una plataforma sólida para el desarrollo.
- **L<sup>A</sup>T<sub>E</sub>X:** He aprendido a documentar utilizando L<sup>A</sup>T<sub>E</sub>Xy a instalar todos los plugins y complementos necesarios para la creación de una memoria técnica.
- **Draw.io:** He aprendido a utilizar *draw.io*<sup>1</sup> para crear diagramas que ayuden a los lectores de mi

---

<sup>1</sup><https://app.diagrams.net/>

memoria a comprender mejor el contenido.

### 5.3. Trabajos futuros

El desarrollo de una aplicación es un proceso continuo, ya que siempre existen aspectos que se pueden mejorar y nuevas funcionalidades que se pueden añadir. Por esta razón, a continuación enumeraremos una serie de mejoras y nuevas funcionalidades que podrían aplicarse al prototipo en el futuro.

Estas mejoras y funcionalidades se han dividido en dos secciones según su complejidad. Los evolutivos complejos son tareas que requieren una estimación de tiempo prolongado o el desarrollo de varios componentes, mientras que los evolutivos sencillos son tareas simples que aportan un gran valor al proyecto con un costo temporal reducido. Cabe destacar que el prototipo está preparado para recibir modificaciones de manera sencilla.

#### 5.3.1. Evolutivos sencillos

1. Lectura de varios ficheros de edificios, daría mas versatilidad a la carga de edificios, siendo posible guardar varios ficheros por cada consulta, y que el software cargue todos los ficheros presentes en la carpeta configurada, de esta manera podemos dividir las entidades geoespaciales clasificandolas por tipo y comenzar a añadir incluso datos que no sean solo edificios.
2. Añadir un conmutable a la barra de herramientas que active la funcionalidad de mostrar en un HUD las coordenadas geodésicas en las que se encuentra el usuario.
3. Añadir un botón conmutable en la barra de herramientas que desactive el componente gestor de altura de la cámara permitiendo volar al usuario para visualizar otra perspectiva diferente del escenario.
4. Realizar un componente que limite al usuario poder desplazarse fuera del escenario.
5. Añadir un botón conmutable en la barra de herramientas que desactive el componente que amplifica los aviones con la distancia para tener una visualización más real cuando el usuario lo requiera.
6. Una funcionalidad adicional de alto valor añadido sería la inclusión de una interfaz gráfica que permita al usuario ingresar un identificador de vuelo y seleccionar el avión correspondiente en la escena. Esto brindaría al usuario la capacidad de consultar información sobre los vuelos directamente desde la visualización 3D.



7. Se puede implementar fácilmente que el gestor de terreno lea un archivo JSON que permita configurar entidades mediante una posición en el sistema de coordenadas WGS84, un texto informativo y un archivo *gltf* que represente el modelo de esa entidad. De esta manera, se pueden cargar y añadir modelos 3D de edificios importantes o monumentos al terreno añadiendo más realismo. Esto permite que el usuario configure de manera sencilla dichas entidades sobre el terreno y además proporcione información adicional cuando son seleccionadas por el usuario.

### 5.3.2. Evolutivos complejos

1. Se ha detectado que en escenarios grandes con velocidades de desplazamiento lentas ayudaría mucho añadir un conmutable a la barra de herramientas que despliegue un panel con el mapa visto desde arriba y nos permita seleccionar un punto del escenario donde desplazar al usuario de un salto.
2. Usar estimaciones con las marcas de tiempo de los vectores de posición de OPENSKY para hacer estimaciones reales de si las muestras nuevas. Si la nueva muestra tiene un intervalo temporal más pequeño que el utilizado por la aplicación, la posición mostrada reflejará un desplazamiento menor al real, lo que dará la sensación de que el avión se frena y luego acelera repentinamente en la siguiente muestra. En ese caso, podemos utilizar la información de la velocidad para descartar la posición y realizar una estimación basada en el trayecto y la velocidad del avión.
3. Una mejora interesante sería habilitar la navegación entre escenarios directamente desde la aplicación. Sería muy útil comenzar la escena con una vista panorámica aérea en una escala amplia, similar a lo que hacen aplicaciones como *FlightRadar24* ó *Plane Finder*. Podríamos superponer entidades rectangulares transparentes que representen las áreas de los escenarios disponibles. Cuando el usuario pase el ratón por encima de una de estas entidades, podría seleccionarla y dirigirse a la aplicación correspondiente de ese escenario, manteniendo la misma marca temporal si se está visualizando la aplicación en el modo datos cacheados. Esta funcionalidad proporcionaría una manera atractiva y global de visualizar todos los escenarios disponibles para ejecutar.



# Bibliografía

- [1] A-Frame. API documentación. <https://aframe.io/docs/>.
- [2] Atlassian. Guía de la metodología scrum: qué es, cómo funciona y cómo empezar. <https://www.atlassian.com/agile/scrum>, 2021.
- [3] Chacon, Scott and Straub, Ben. *Pro Git*. Apress, second edition edition, 2014. Disponible en: <https://git-scm.com/book/en/v2>.
- [4] D. Flanagan. *JavaScript: The Definitive Guide, 7th Edition*. O'Reilly Media, Sebastopol, CA, 2012.
- [5] Google. Google Earth Engine API documentación. <https://developers.google.com/earth-engine/guides>.
- [6] International Civil Aviation Organization. ADS-B Implementation and operations guidance document. 2018. Disponible en: [https://www.icao.int/APAC/Documents/edocs/Revised%20ADS-B%20Implementation%20and%20Operations%20Guidance%20Document%20\(AIGD\)%20Edition%2014.pdf](https://www.icao.int/APAC/Documents/edocs/Revised%20ADS-B%20Implementation%20and%20Operations%20Guidance%20Document%20(AIGD)%20Edition%2014.pdf).
- [7] Jesus M. Gonzalez-Barahona. Some notes while learning about A-frame. <https://github.com/jgbarah/aframe-playground>.
- [8] Leaflet. API documentación. <https://leafletjs.com/reference.html>.
- [9] Mozilla. Developer Network: JavaScript API documentación. <https://developer.mozilla.org/es/docs/Web/JavaScript>.
- [10] OpenSky Network. API documentación. <https://opensky-network.org/apidoc/>.
- [11] OpenStreetMap. Overpass API. [https://wiki.openstreetmap.org/wiki/Overpass\\_API](https://wiki.openstreetmap.org/wiki/Overpass_API).
- [12] Overleaf Team. Documentación L<sup>A</sup>T<sub>E</sub>X. <https://www.overleaf.com/learn>.

- [13] Three.js. API documentación. <https://threejs.org/docs/>.

# Glosario

**1090ES** El 1090ES es un estándar de comunicaciones utilizado en la aviación. Se refiere a una forma de transmisión de datos basada en el modo de transpondedor de vigilancia dependiente automática (ADS-B) en la banda de 1090 MHz.. 22

**3D** Se refiere a la representación tridimensional de objetos o entornos en un espacio virtual. En el contexto de gráficos y visualización, el término 3D se utiliza para describir la capacidad de representar objetos con altura, anchura y profundidad, añadiendo así una dimensión adicional a la imagen o escena.. 1, 13, 24, 30, 32, 35, 43, 47, 52

**API** Siglas de "Application Programming Interface" (Interfaz de Programación de Aplicaciones). Se refiere a un conjunto de reglas y protocolos que permiten la comunicación y la interacción entre diferentes software o componentes de un sistema. Una API define las formas en que los programas pueden solicitar servicios o funcionalidades de otro software y cómo pueden intercambiar datos entre sí. 1, 10, 15, 25, 28, 30, 32, 33, 39, 42, 48, 64, 68, 72

**Centroide** En matemáticas y geometría, el centroide es un punto que representa el centro de masa o centro de gravedad de un objeto o una figura geométrica.. 73

**Crowdsourcing** El crowdsourcing es un modelo de colaboración en línea que utiliza una comunidad en línea para obtener recursos. Se basa en la idea de que un grupo diverso de individuos puede generar soluciones innovadoras o realizar tareas complejas más eficientemente que una sola persona o una organización tradicional.. 22

**DAO** DAO (Data Access Object) es un patrón de diseño utilizado en el desarrollo de software para abstraer y encapsular la lógica de acceso a una fuente de datos, como una base de datos o un servicio web.. , 76, 78

**DOM** El DOM (Document Object Model) es una representación estructurada y jerárquica de un documento HTML o XML.. 83, 102

**DTO** DTO (Data Transfer Object) es un patrón de diseño utilizado en el desarrollo de software para transferir datos entre diferentes capas o componentes de una aplicación.. 18, 76, 89

**ECMA** ECMA (European Computer Manufacturers Association) es una organización internacional de estándares técnicos. ECMA es conocida por su trabajo en la estandarización de lenguajes de programación, como JavaScript (ECMAScript).. 31

**Framework** Conjunto estructurado de herramientas, bibliotecas, componentes y estándares que proporciona una base para el desarrollo de software. 1, 31, 32, 37, 38, 66

**Geoespacial** El término geoespacial se refiere a la integración de datos geográficos y espaciales en sistemas de información.. 53, 54, 57, 71, 102

**GIS** Un GIS (Sistema de Información Geográfica), también conocido como SIG en español, es un sistema diseñado para recopilar, almacenar, analizar y visualizar datos geoespaciales.. 3, 52

**GPU** "Graphics Processing Unit"(Unidad de Procesamiento Gráfico). Se trata de el componente de computación diseñado para realizar tareas relacionadas con la renderización para la visualización de gráficos en tiempo real en una computadora.. 33

**Hardware** El hardware se refiere a los componentes físicos y tangibles de un sistema informático. Incluye dispositivos como la unidad central de procesamiento (CPU), la memoria, el disco duro, la tarjeta gráfica, el monitor, el teclado, el mouse y otros periféricos.. 33, 35, 36

**HUD** Siglas de "Head-Up Display"(Pantalla de visualización frontal). Se refiere a una tecnología de visualización que proyecta información directamente en el campo de visión del usuario. El HUD permite al usuario ver información relevante independientemente de la dirección donde mire la cámara.. 6, 16–19, 60, 82, 84–89

**JSON** JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y legible por humanos. Se basa en una estructura de pares clave-valor y es ampliamente utilizado para transmitir datos entre aplicaciones web.. 34, 48, 75, 76, 79

**Modo S** El modo S es un protocolo de comunicación utilizado en la aviación para la transmisión de información entre aeronaves y controladores de tráfico aéreo.. 23

**npm** npm (Node Package Manager) es el administrador de paquetes predeterminado para Node.js, una plataforma de desarrollo de aplicaciones basada en JavaScript.. 34, 92

**OACI** Organización de Aviación Civil Internacional. 22

**OpenGL** Una API de gráficos en 2D y 3D de código abierto, que proporciona una interfaz estándar para interactuar con la GPU. Es una de las librerías de gráficos 3D más utilizadas en la industria de desarrollo de aplicaciones y juegos.. 32

**Raster** Un raster es una representación de una imagen o datos espaciales en forma de una cuadrícula de píxeles o celdas. Cada píxel o celda contiene un valor que representa una propiedad o atributo específico. Los datos raster son comúnmente utilizados en sistemas de información geográfica (GIS) para representar información geoespacial, como mapas y fotografías aéreas.. 13–16, 42, 62, 63, 65, 91

**Raycaster** Un raycaster es un algoritmo utilizado en gráficos por computadora para determinar la intersección entre un rayo y un objeto en una escena tridimensional. El rayo se origina en un punto específico (como la posición de la cámara) y se proyecta en una dirección determinada.. 17, 39, 88

**Renderización** La renderización es el proceso de generar una representación visual a partir de datos en bruto o de una descripción digital. En el contexto de los gráficos por computadora, la renderización implica convertir datos de geometría y atributos en una imagen o animación final.. 1, 30, 32

**SSR** SSR (Secondary Surveillance Radar) es un sistema de radar secundario utilizado en la aviación para el seguimiento y la identificación de aeronaves. A diferencia del radar primario que detecta objetos por la señal reflejada, el SSR utiliza transpondedores en las aeronaves para transmitir una respuesta electrónica a las solicitudes del radar.. 23

**UAT** UAT (Universal Access Transceiver) es un sistema de comunicaciones utilizado en la aviación para la transmisión de datos basada en el modo de transpondedor de vigilancia dependiente automática (ADS-B) en la banda de frecuencia de 978 MHz. A diferencia del 1090ES, que opera en la banda de 1090 MHz.. 22

**WebGL** WebGL es una API de gráficos en 3D basada en JavaScript. Permite la renderización de gráficos interactivos en tiempo real en navegadores web compatibles sin necesidad de complementos adicionales.. 1, 38