UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA

Desarrollo de Software para visualización de modelos 3D en realidad virtual

Software development for 3D model visualization in virtual reality

Realizado por

D. Pablo Andrés Domínguez

Tutorizado por

Dr. Dña. Mónica Trella López

Cotutorizado por

Dr. D. Alfonso Gago Calderón

Departamento

LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

MÁLAGA, SEPTIEMBRE DE 2021

Esta hoja se ha dejado intencionadamente en blanco

Resumen

Este trabajo ha realizado un estudio comparativo entre varios motores gráficos de visualización en 3 dimensiones (3D). En base a sus resultados se ha desarrollado una aplicación genérica para poder ser usada en distintos dispositivos enfocados en la realidad virtual accesible a través un navegador y que cargará una escena con distintos modelos, animaciones, interacciones y otros componentes para mostrar la viabilidad de este tipo de motor.

Palabras clave: Realidad Virtual, Entornos 3D, Navegador Web, Gafas RV

Esta hoja se ha dejado intencionadamente en blanco

Abstract

This project has carried out a comparative study between several 3 dimensional (3D) visualization graphic engines. Based on its results, a generic application has been developed to be used in different devices focused on virtual reality accessible through a browser and that will load a scene with different models, animations, interactions and other components to show the viability of this type of engine.

Keywords: Virtual reality, 3D Playgrounds, Web Browser, VR headset

Esta hoja se ha dejado intencionadamente en blanco

Índice

Esta hoja se ha dejado intencionadamente en blanco

1

Introducción

El modelado 3D se está convirtiendo, en los últimos tiempos, en un atractivo cada vez mayor para la población general. Se han desarrollado diversos dispositivos como las gafas de realidad virtual Oculus para la interacción de usuarios con la representación virtual en 3 dimensiones, abierta a entornos tanto profesionales como domésticos.

En este contexto creciente, parece interesante el desarrollo de una aplicación que permita la interacción de modelos 3D que se pueda ejecutar de manera libre en distintos dispositivos, como móviles, ordenadores, tablets, las propias gafas virtuales, etc.

Para superar la barrera de la diferencia del hardware de cada dispositivo y los distintos sistemas operativos, dicho programa sería accesible desde un servidor externo y se permitiría su visualización a través de un navegador web (en concreto, vamos a trabajar con Chrome, navegador oficial de Google).

En la actualidad existen algunos proyectos que desarrollan esta filosofía. Uno de ellos, es específico sobre música, y utiliza PlayCanvas para renderizar datos, de tal forma que se puedan usar en distintos dispositivos. Igualmente otros proyectos tratan con motores de representación 3D similares a los se tratan en este trabajo, para crear, por ejemplo, videojuegos.

Este trabajo consiste en la evaluación e investigación de varios motores, la implementación de distintos modelos proporcionados por una base de datos, así como otros creados a través de editores con fórmulas geométricas (con ayuda de Blender), y la interacción de dichos programas a través de distintos dispositivos, haciendo especial hincapié en las gafas de inmersión virtual.

1.1 Motivación

La principal motivación para realizar este trabajo surge del interés personal por investigar las posibilidades de distintos motores gráficos que permitan la visualización de modelos de objetos y de instalaciones, animaciones e inmersión virtual en un entorno de espacio 3D.

Tras haber trabajado con motores gráficos como Unity, se pretende investigar las distintas alternativas, ventajas e inconvenientes de otros tipos de motores que existen. De éstos resultaron especialmente interesantes varios de ellos dedicados a la renderización en web de modelos 3D usando el elemento HTML de Canvas.

No se han podido encontrar referencias significativas de proyectos publicados que estén relacionados con este tema, en general o, específicamente, que desarrollen un software con capacidad de hacerlos funcionar a la vez en varios dispositivos con hardware diferentes. Se decidió profundizar en este objetivo y crear un programa que permitiera utilizar dichos motores gráficos ejecutándose en este tipo de entornos.

Este objetivo supone complementar la formación recibida en el grado, desarrollando conocimientos previos y adquiriendo competencias nuevas en dicha materia mediante investigación y aprendizaje.

De este modo, este proyecto supone un reto de desarrollo personal, atractivo y complejo por la necesidad de combinar lenguajes, motores y librerías para renderizar software en navegadores web.

Desde el Área de Proyectos de Ingeniería de la Escuela de Ingeniería Industriales (a través del cotutor Dr. D. Alfonso Gago) se ha aportado directrices para incluir, en las especificaciones del desarrollo, los elementos necesarios para que el resultado final permita la visualización de desarrollos de productos e instalaciones, propios de esta ingeniería para que puedan ser utilizados como una herramienta docente e, incluso, como una herramienta de interés para empresas de ingeniería.

1.2 Objetivos

Los objetivos parciales o complementarios en base a los que se desarrolla este Trabajo de Fin de Estudios se exponen a continuación en el siguiente listado:

- Crear un software que permita la visualización de modelos 3D de distintos archivos.
- Este software funcionará con un motor que permita la renderización por web de los modelos aportados.

- Dicho software se aplicará sobre los modelos de manera que sean interactivos con el cliente, además de producir distintos efectos y animaciones.
- El software será accesible desde los navegadores de una amplia gama dispositivos, como ordenadores, móviles, o gafas de visión 3D como las Oculus.
- Específicamente, se tratará el tema de la inversión virtual, en concreto con las gafas virtuales.
- Realizar este mismo proyecto en otros motores para comparar los diferentes rendimientos.

1.3 Estructura de la memoria

La memoria está estructurada en cinco capítulos con sus correspondientes subcapítulos. En cada capítulo se relata un pilar de este trabajo, pero desde un punto de vista diferente al anterior.

En la introducción se ha presentado, en los distintos subcapítulos, la idea que se va a tratar en este proyecto.

En la segunda parte, las Herramientas y Tecnologías, se detalla cada uno de los elementos que se han usado. Desde los distintos motores gráficos y extensiones web, hasta el hardware utilizado.

En la tercera parte, Análisis y diseño, se explica el tipo de estructura que ha seguido este trabajo, junto al análisis de sus modelos, y la metodología ágil, que es la que se ha usado en este trabajo.

En el cuarto apartado se hará un resumen de los distintos sprints, siendo cada uno su propio subcapítulo, que se han llevado a cabo en cada uno de los softwares que se han desarrollado, desde cómo se han llevado a cabo, hasta los problemas que han ido surgiendo y sus soluciones.

Por último, hay un quinto capítulo donde se exponen las conclusiones a las que se han llegado tras realizar dicho trabajo, posibles mejoras, y futuras líneas de trabajo.

Más tarde, se muestran las referencias que se han usado para consultar y lograr las distintas características de los softwares que se han llevado a cabo, junto a los anexos con la documentación generada del trabajo para ahondar en los distintos detalles.

Esta hoja se ha dejado intencionadamente en blanco

2

HERRAMIENTAS Y TECNOLOGÍAS

2.1 Programas y extensiones para modelos

Una parte imprescindible para este trabajo es tener desarrollados una serie de modelos que puedan ser implementados para poder interactuar con ellos.

Para saber qué tipos de programas se van a utilizar, hay que considerar cuales extensiones se usarán:

- FBX
- GLB

2.1.1 FBX

Las extensiones de estos archivos son dibujos 2D o 3D creados por una herramienta de Autodesk FBX. Están ajustados para ofrecer interoperabilidad entre distintos programas. [1]

Estos tipos de archivos son los que serán usados a través de un banco de datos.

2.1.2 GLB

Este archivo contiene un modelo 3D guardado en “Formato de transmisión GL”, en una versión 3D. Dichos archivos están optimizados para su descarga y tiempo de carga durante una ejecución, lo que nos facilitará su uso en web. [2]

Por las ventajas que se aprecian visto de tiempo de carga para ejecución, serán los archivos que usaré para los modelos 3D en el programa.

2.1.3 Programas

Habiendo cercado las extensiones que van a tenerse en cuenta, se ha realizado una pequeña investigación para ver qué tipos de programas permiten la importación de este tipo de archivos y cuál de ellos nos será más útil y beneficioso: Paint 3D, Blender, Bforartistas.

2.1.3.1 Paint 3D

Es una herramienta que viene incluida en el sistema operativo de Windows 10. [3] Dicha herramienta permite la visualización 3D y edición de modelos 3D [4], e incluye la funcionalidad de importar los modelos mencionados.

Sin embargo, no permite el uso de animaciones y otras funcionalidades, requisitos para nuestro proyectos, tal y como se comenta en este foro [5].

2.1.3.2 Blender

Blender es un software libre que está dedicado al modelado, renderización, animación y simulación, entre otras aplicaciones. [6] Permite la importación y exportación de muchos tipos de archivos, entre ellos, FBX y GLB.

2.1.4 Bforartists

Bforartistas es un software de código abierto, dedicado a la creación de contenido 3D, incluido modelado, texturizado, animación, renderizado y pos procesamiento. Tratan de mantener una interfaz gráfica de usuario simple, intuitivo y fácil de usar. [7]

Sin embargo, está orientado más hacia artistas que programadores, y puede dar algunos problemas al usarse en softwares complejos.

2.1.5 Conclusión

Tras ver las características (y distintas opiniones de usuarios que han usado los motores gráficos que se están comparando) se llega a la conclusión de que se usará el programa de Blender.

Este programa nos permitirá exportar estos modelos desde la extensión FBX a GLB, y editar dichos modelados, animaciones y renderizado si así se ve necesario.

2.2 Plataformas para implementación

Para el desarrollo del programa, hemos realizado una investigación de diversas librerías de motores gráficos (y de sus respectivas plataformas). Se debe de tener en cuenta que se han elegido los motores tras restringir la búsqueda, ya que existen muchos más motores (como Three.js) que no se han tenido en cuenta para la comparación. Después de la investigación, se ha llegado a una serie de conclusiones que nos permiten realizar las siguientes comparaciones:

Motor	Playcanvas	Babylon	Unity
WebGL	Motor dedicado expresamente a renderizado en web	Motor dedicado expresamente a renderizado en web	No está plenamente desarrollado para web. [8]
Documentación	Tiene documentación básica y tutoriales, aunque no tiene mucho material en los foros	Buena documentación y mucha información en los foros	Buena documentación y mucha información en los foros
Compatibilidad de modelos	Acepta modelos 3D de las extensiones: FBX, OBJ, DAE, 3DS, DXF, GLB [9]	Acepta modelos 3D de las extensiones: GLB, GLTF, BABYLON, OBJ, STL [10]	Acepta modelos 3D de las extensiones: FBX, OBJ, DAE, DXF [11]
Input	Ratón, teclado, toque*, mando de juego, dispositivos de Realidad Virtual [12]	Ratón, teclado, toque*, mando de juego, dispositivos de Realidad Virtual [13] [14]	Ratón, teclado, toque*, mando de juego [15]
Lenguaje	Javascript (existen adaptaciones para TypeScript) [16]	TypeScript [17]	JavaScript [18]

* Toque: Ejemplo, pantalla táctil.

En base a las comparaciones que se han visto:

- Se observa que Unity WebGL no está suficientemente desarrollado aún para ser considerado para realizar este software, ya que Unity está más centrado en su plataforma offline que online, por lo que no está optimizada al desarrollo en Web.
- Respecto a Babylon, es un motor bastante parecido a Playcanvas. Ambos se especializan en el procesamiento de modelos 3D, y aplicar físicas si fuese necesario. Debido al parecido entre ambos motores, se ha decidido usar ambos y realizar una comparación de los resultados.

2.2.1 Playcanvas

2.2.1.1 Características del Framework

La plataforma de PlayCanvas cuenta con un editor colaborativo, aunque en este proyecto no se le dará uso. Dicho editor colaborativo permite guardar los distintos proyectos en una plataforma en la nube.

Se utilizará su motor, ya que es de código abierto, y compatible con el estándar de WebGL 1.0 y WebGL 2.0. De por sí, su librería de GitHub trabaja con el lenguaje de Javascript, aunque para el caso que nos concierne, nosotros usaremos una plantilla concreta que nos permitirá programar en TypeScript [66]. El motor es capaz de simular la física de cuerpos rígidos, manejar audio tridimensional y animaciones 3D. [20]

PlayCanvas fue liberada en GitHub el 4 de Junio de 2014 bajo la licencia de MIT, principalmente pensado para el desarrollo de videojuegos en web de 3 dimensiones que usasen WebGL. [21]

El uso de esta librería tiene usos similares a otro tipo de librerías gráficas, ya que contiene [22]:

- Funciones de alto nivel, con un sistema compuesto por cámaras, cuerpos geométricos sencillos, juego de luces, distintos tipos de planos (ground, panels...) y edición de los materiales de los distintos cuerpos (para cambios relacionados con el color, la textura, opacidad...)
- Permite cargar un conjunto de modelos 3D con extensión GLB, FBX, OBJ, DAE, 3DS; texturas (incluyendo texturas HDR); ficheros de audios, etc.
- Amplio conjunto de tipos de iluminación, como iluminación ambiente, direccional, por foco, etc.
- Tiene un sistema de “Shader Chunk”, que soporta la personalización parcial y total de los Shaders.
- Soporta WebVR, lo que significa que permite experiencias de inmersión virtual.

2.2.1.2 Jerarquía de Entidades

La parte principal de este proyecto orbita sobre el comportamiento de los modelos y los distintos cuerpos geométricos. Para ello, se explicará cuál es el tipo de Jerarquía que existe en PlayCanvas en referencia a los modelos importados.

Para ello, hay que saber que cuando cargamos un modelo (para la mayoría de extensiones, incluidas GLB y FBX), se carga usando una clase Entity.

Entity es una clase que extiende la clase GraphNode (como su nombre indica, un grafo de nodos). Esta clase Entity se le aplica al modelo completo, y tiene como hijos cada uno de los elementos individuales del propio modelo, que serán de tipo MeshInstance (mallas). [23]

Esto a la hora de llevar a cabo el proyecto dio lugar a varios problemas técnicos, ya que la clase Entity tiene varias propiedades que los elementos hijos (las mallas) no poseen, como la propiedad de animations (para ejecutar animaciones), o RigidBody (para manejar movimientos, cuerpos y colisiones). Por ejemplo, significaba que para realizar animaciones, debían ser ejecutadas desde el nodo superior (aquel que contiene la clase Entity), lo que impide realizar animaciones independientes en los distintos hijos (llamadas en el código).

Tras realizar varias investigaciones, se comprobó que se estaba realizando una nueva clase para solventar este tipo de situaciones, en la que todos los elementos de archivos GLB estarían en una jerarquía compuestos todos por la clase Entity. Sin embargo, dicha clase está en desarrollo (beta), y no podía implementar dicha función. [24]

2.2.2 Babylon JS

2.2.2.1 Características del Framework

Al igual de la librería de PlayCanvas, es de código abierto, y compatible con el estándar de WebGL 1.0 y WebGL 2.0.

El código fuente de Babylon está escrito en TypeScript, y después es compilado en una versión concreta de JavaScript. El motor es capaz de simular la física de cuerpos rígidos, manejar audio tridimensional y animaciones 3D. [25]

El primer commit realizado en GitHub data del 27 de Junio de 2013. Está principalmente pensado para el desarrollo de videojuegos en web de 3 dimensiones que usasen WebGL. [26]

El uso de esta librería tiene usos similares a otro tipo de librerías gráficas, ya que contiene [27]:

- Funciones de alto nivel, con un sistema compuesto por cámaras, cuerpos geométricos sencillos, juego de luces, distintos tipos de planos (ground, panels...) y edición de los materiales de los distintos cuerpos (para cambios relacionados con el color, la textura, opacidad...)
- Permite cargar un conjunto de modelos 3D con extensión GLB, OBJ, STL; texturas; ficheros de audios, etc.
- Tiene un sistema que soporta la personalización parcial y total de los Shaders, llamado NodeMaterial. También tiene un modo PBR (Physical Based Rendering), un renderizado para simular una iluminación realista.
- Amplio conjunto de tipos de iluminación, como iluminación ambiente, direccional, por foco, generada en la propia malla del cuerpo, etc.
- Soporta WebVR y WebXR (nueva versión), lo que significa que permite experiencias de inmersión virtual.

2.2.2.2 Jerarquía de Mallas

Al igual que se ha aclarado anteriormente, principalmente, este proyecto orbita sobre el comportamiento de los modelos y los distintos cuerpos geométricos. Para ello, se explicará cuál es el tipo de Jerarquía que existe en Babylon JS.

La clase BABYLON.Mesh devolverá la figura geométrica del objeto, y todos sus posibles hijos también serán del tipo BABYLON.Mesh. De esta manera, a diferencia de PlayCanvas, todos los cuerpos pueden tener las mismas propiedades. [28]

Además, la clase BABYLON.Mesh dispone de una serie de eventos por defecto que pueden ser usados tras cambios en la escena, permitiendo una mayor personalización.

2.3 TypeScript

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. [29]

2.3.1 Resumen

Como se acaba de ver, Typescript es un lenguaje que se basa en JavaScript.

JavaScript es un lenguaje de programación interpretado, que aporta una orientación a objetos basado en un prototipado de estos, imperativo y débilmente tipado y dinámico; siendo frecuente en la ejecución de programas en el lado del cliente.

El objetivo de Typescript es ser un comprobador de tipos estático para los programas de JavaScript. Esto significa que es una herramienta que se ejecuta antes de la propia ejecución del código, y asegura que los tipos del programa son correctos. [30]

El motivo de que este comprobador sea tan importante, es el motivo de que la complejidad de los programas que usan JavaScript ha aumentado, pero la capacidad de dicho lenguaje para expresar las relaciones entre las distintas unidades del código no lo ha hecho.

2.3.2 Historia

La primera vez que se publicó TypeScript fue en 2012, con la versión 0.8. [31] Este era el resultado de dos años de trabajo de Anders Hejlsberg quien también trabajó en el desarrollo de C#. [32]

Al principio fue criticado Miguel de Ilcaza (programador mexicano que participó en proyectos GNOME, Mono, y Xamarin), quien aunque elogió el lenguaje en sí, criticó la falta de soporte IDE para herramientas más allá de Microsoft Visual Studio. [33]

Por esto mismo, Microsoft afrontó este problema en 2013, dando soporte a otros IDEs, como Eclipse, y varios editores de texto que soportaban TypeScript fueron lanzados ese mismo año, como Vims y Emacs. [34]

La versión de TypeScript 1.0 fue lanzada en la “conferencia de desarrolladores Build de Microsoft” en 2014. El propio equipo de TypeScript anunció que el compilador del lenguaje había ganado 5 veces más rendimiento que la versión anterior. [35]

La versión actual es TypeScript 4.0, que en comparación con la 1.0, se han introducido nuevas características cómo prevenir, de manera opcional, que algunas variables sean asignadas valores null, o la introducción de tuplas. [36]

2.3.3 Tipado estático

Una de las novedades de TypeScript respecto a JavaScript, es que permite anotaciones de tipos en las variables, para permitir la comprobación de tipados durante la compilación.

Los tipos primitivos son: number, boolean y string. También soporta las anotaciones de tipo *Array*, *Enum*, *void*, *Tuple*, *Union*, *never* y *any*. En la Figura 21 podemos ver un ejemplo del uso de tipado para este proyecto.

El compilador de TypeScript hace uso de la inferencia de tipados para inferir los tipos cuando no han sido dados. Un ejemplo de esta inferencia sería cuando se declaran dos variables a y b como number, y una nueva variable c que sea declarada como la suma de a y b. No se ha especificado que c sea tipo number, pero por el operador (+) y siendo los tipos de ambos datos a derecha e izquierda tipo number, se infiere que el resultado será tipo number también.

```
addAnimColor(mesh:any,stoppedColor:BABYLON.Color3,
animedColor:BABYLON.Color3){
      const color = mesh.matchesTagsQuery("animar") ?
      animedColor : stoppedColor;
      var mat = <BABYLON.PBRMaterial>mesh.getChildMeshes()[0].material?
      .clone(<string>mesh.getChildMeshes()[0].material?.name);
      mesh.getChildMeshes()[0].material = mat;
      mat.albedoColor = color;
}
```

Figura 21 Código en TypeScript del proyecto

2.4 WebGL

WebGL es una API multiplataforma y estándar, que se utiliza para crear gráficos 3D en un navegador web. Basándose en OpenGL ES 2.0, WebGL utiliza el lenguaje de sombreado de OpenGL, GLSL, y ofrece la familiaridad de la API estándar de OpenGL. Como se ejecuta en el elemento Canvas de HTML5, WebGL se integra plenamente con todas las interfaces del Modelo de Objetos del Documento (DOM). [38]

2.4.1 Representación 3D en el navegador

Además, esta tecnología está ya implementada en los navegadores de uso mayoritario en PC (Internet Explorer, Microsoft Edge, Mozilla Firefox, Safari, Google Chrome...), y también en navegadores para móvil (los anteriores ya mencionados, más otros navegadores específicos de móvil, como Opera Mobile, Firefox for Android...). [39]

La información que está abajo está referenciada a las siguientes fuentes [40] [41] [42]

La API de WebGL está diseñada de manera específica para enviar, desde la CPU, órdenes a la tarjeta gráfica del sistema. Dichas instrucciones están guardadas en ciertos programas, los shaders, quienes son compilados por la API durante la ejecución, y trasmitidos a la GPU.

La GPU reproduce la salida gráfica por medio del elemento de HTML 5 canvas, que permite la renderación de la escena en la página web según las especificaciones que sean descritas en WebGLRenderer asociado.

Todos los objetos descritos anteriormente son nativos en HTML5. Esto significa que no es necesaria ninguna extensión ni plugin para poder poner en funcionamiento la escena que se haya diseñado.

Actualmente existen dos versiones de WebGL, WebGL y WebGL2. Cada una toma base en el lenguaje OpenGL ES 2.0 y OpenGL ES 3.0 respectivamente.

2.4.2 Shaders

Tras comentar de que WebGL usa shaders, así que voy a realizar una pequeña explicación sobre ellos partiendo de la fuente oficial [43].

Como se ha visto anteriormente, los shaders se ejecutan en de las tarjetas gráficas (o hardware asociado a las mismas), para poder representar gráficos en 3 dimensiones.

Los shaders son codificados y ejecutados en una Unidad de Procesamiento Gráfico (GPU). Para ello, los shaders son programados para usar el pipeline gráfico o renderizados. Este pipeline es un proceso por el cual convierte una escena tridimensional en una imagen de dos dimensiones que se pueda representar en una pantalla.

Tanto Direct3D como OpenGL son APIs gráficas que usan pipelines gráficos y shaders para facilitar el trabajo a los programadores y no tener que tratar directamente con los aceleradores gráficos.

Como se ha visto anteriormente, WebGL usa OpenGL para tratar el tema de shaders. Por ello voy a describir brevemente el funcionamiento del pipeline gráfico de OpenGL. Dicho pipeline tiene una serie de etapas que se ejecutan en el siguiente orden:

- Especificación de vértices: es el proceso en el que se preparan los procesos y objetos necesarios para renderizar un programa de shaders en concreto.
- Procesamiento de Vértices: secuencias de vértices son procesados a través de una serie de shaders, y las siguientes etapas de shaders parten de los datos de la anterior. La última etapa proporciona los datos de vértices al post-procesado de vértices. Hay etapas opcionales en este paso:
    - Tessellation: Añaden detalles a las superficies 3D

  - Shaders de Geometría: en base a una única primitiva base (resultado de la interpretación de un flujo de vértices) genera 0 o más primitivas (a diferencia de las etapas normales, que tienen una generación tipo 1:1).
- Post-procesado de Vértices: el resultado de vértices anterior es sometido a distintos tipos de operaciones para preparar el ensamblaje de primitivas y rasterización.
- Ensamblaje de primitivas: proceso por el cual las primitivas se dividen en secuencias de primitivas base individuales. Tras algunos procesamientos menores, pasan a la siguiente etapa.
- Rasterización: etapa en la cual cada primitiva individual se descompone en elementos discretos llamados fragmentos.
- Aplicación de un “Fragment Shader”: se procesa un fragmento generado por la rasterización en un conjunto de colores con un único valor de profundidad.
- Procesamiento Por Muestra: en esta etapa se procesan los fragmentos emitidos por un “Fragment Shader”, y sus datos resultantes se escriben en varios búferes y será nuestra salida.

La Figura 1 muestra el diagrama del pipeline renderizados. Las etapas de esta secuencia que tengan borde discontinuo significan que son etapas opcionales.

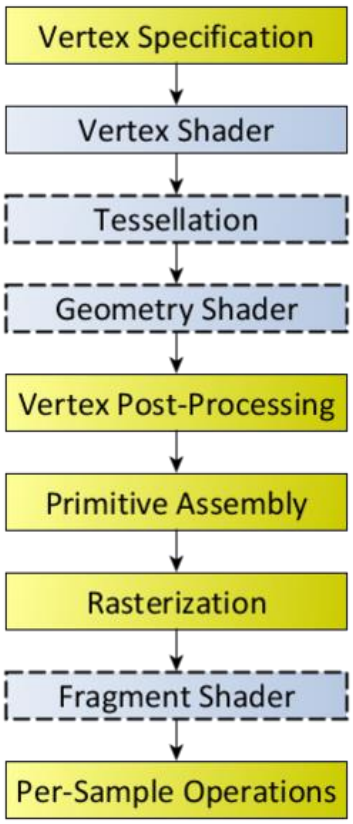

Figura 1 Esquema Shaders [45]

En la Figura 2 se muestra una imagen más fácil de entender sobre las etapas anteriores. El Vertex shader calcula las distintas transformaciones de los vértices, y el Fragment Shader se encarga de otros procesos relacionados con las texturas, colores, iluminaciones, etc. Ambos shaders se compilan y ejecutan en la GPU.

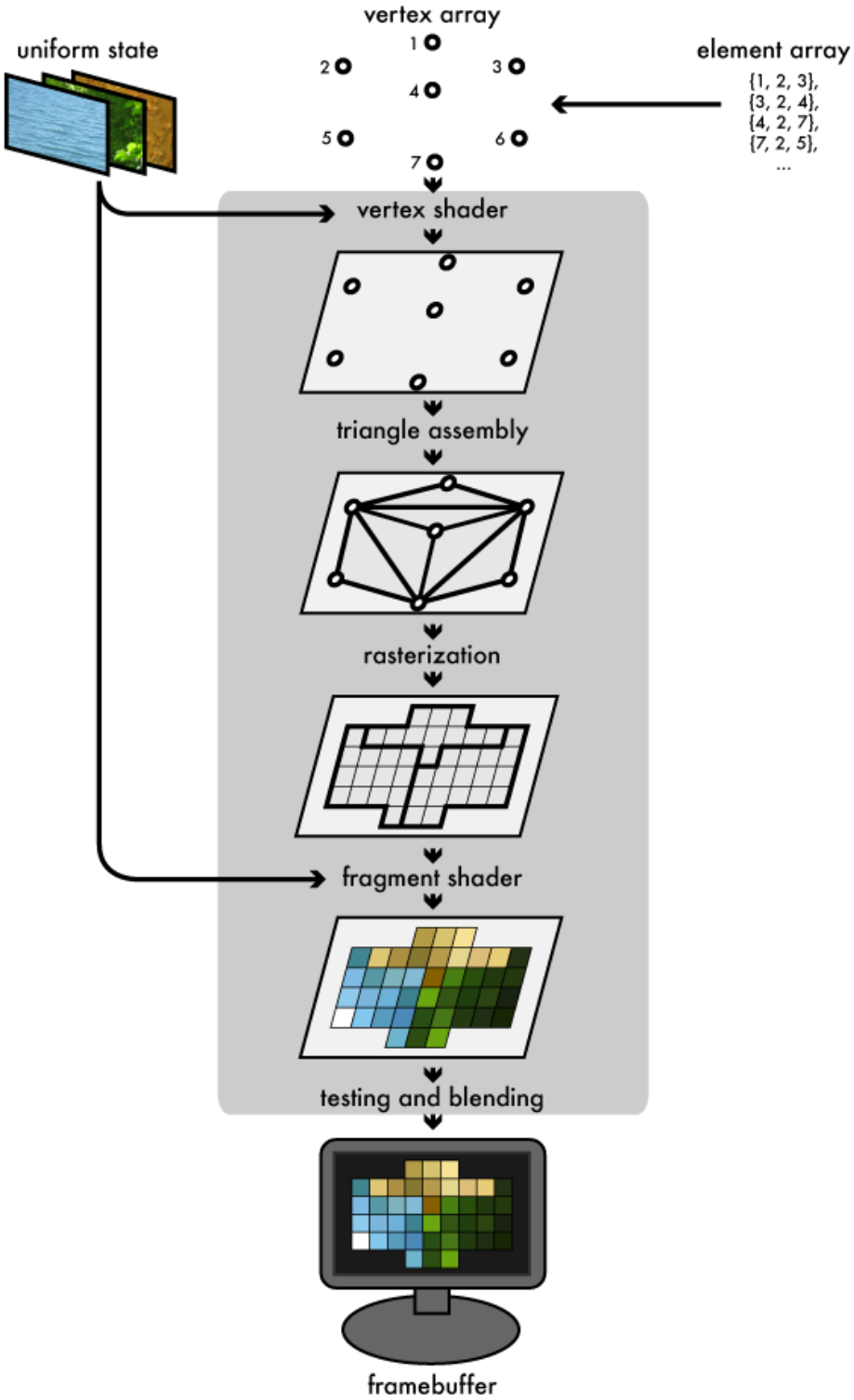

Figura 2 Etapas Shaders [46]

2.5 Librerías

Las librerías que se han usado son:

- Babel:
  - "babel-cli": "^6.26.0", "babel-core": "^6.26.3", "babel-jest": "^25.5.1", "babel-loader": "^8.2.2", "babel-preset-env": "^1.7.0",
  - Estas librerías se utilizarán para el uso el compilador de javascript, Babel. [47]
- Canvas:
  - "canvas": "^2.6.1",
  - Se usarán Canvas para renderizar los modelos 3D a través de WebGL en HTML5 [48]
- ESLint:
  - "eslint": "^6.8.0",
  - Esta librería nos proporciona una herramienta para encontrar y arreglar errores del código fuente en JavaScript. [49]
- JavaScript obfuscator :
  - "javascript-obfuscator": "^2.10.3",
  - JavaScript Obfuscator es un ofuscador gratuito para JavaScript, que contiene una variedad de características que proporcionan protección y seguridad para el código fuente. [50]
- Jest:
  - "jest": "^26.6.3", "@types/jest": "^25.2.3", "ts-jest": "^26.4.4",
  - Jest es un Framework para el testing de código en lenguaje Javascript. [51]
- Babylonjs:
  - "babylonjs": "^4.2.0", "babylonjs-gui": "^4.2.0", "babylonjs-loaders": "^4.2.0", "babylonjs-materials": "^4.2.0"
  - Es una de las librerías más importantes, ya que Babylon-js es uno de los motores que se utilizarán para renderizar los modelos 3D y el resto del software para aplicarlo sobre los modelos.
- Playcanvas:
  - "playcanvas": "^1.42.0",
  - Es una de las librerías más importantes, ya que Playcanvas es el otro de los motores que se usarán para renderizar los modelos 3D y el resto del software para aplicarlo sobre los modelos.
- pre-commit:
  - "pre-commit": "^1.2.2",
  - Esta librería se utilizará para encontrar posibles errores en el código antes de su compilación. [52]
- Typescript:

  - "typescript": "^3.9.7", "@typescript-eslint/eslint-plugin": "^2.34.0", "@typescript-eslint/parser": "^2.34.0", "ts-loader": "^7.0.5", "@types/socket.io-client": "^1.4.35",
  - Son las librerías que se usarán para programar en TypeScript. Ahondare en el motivo por el que se ha elegido este lenguaje más adelante en otro apartado.
- Webpack:
  - "webpack": "^4.46.0", "webpack-cli": "^3.3.12", "webpack-dev-server": "^3.11.2", "script-ext-html-webpack-plugin": "^2.1.5", "clean-webpack-plugin": "^3.0.0", "html-webpack-plugin": "^3.2.0"
  - Webpack es un empaquetador de módulo estático para aplicaciones que funcionen con Javascript. En resumen, al procesar la aplicación, crea un gráfico de dependencia que nos mapea cada módulo que el proyecto necesita y genera uno o más paquetes. [53]

2.6 Herramientas Web

Para poder probar el programa que se ha desarrollado, he usado varias extensiones y navegadores. Estos son: Chrome, Web Server for Chrome, WebXR API Emulator.

2.6.1 Chrome

Google Chrome es un conocido navegador web de código abiertodesarrollado por Google [54], aunque derivado de proyectos de código abierto (como el motor de renderizado Blink) [55].

Para este proyecto ha sido el navegador que ha alojado la mayoría de pruebas de las distintas versiones de este proyecto para comprobar su funcionamiento en la web.

2.6.2 Web Server for Chrome

Es una extensión de código abierto que permite actuar de servidor sobre una carpeta local utilizando HTTP y HTTPS. Dicho servidor permite acceso a través de distintos tipos de redes: local, LAN y WAN. [56]

Se usará esta herramienta para que la carpeta de este proyecto se aloje en un servidor y pueda ser accesible desde distintos dispositivos. Hay que recordar que esta extensión usa el hardware del PC para alojar los archivos, lo que significa que esta extensión no se puede utilizar a gran escala, es decir, para un gran número de dispositivos. Por ello lo usaremos sólo para red LAN (red local, como la que puede generar un router) y local (sólo a través del dispositivo que se ejecuta).

Más adelante, en el siguiente capítulo se explicará su funcionamiento de manera detallada.

2.6.3 WebXR API Emulator

Esta herramienta permite ejecutar y testear el contenido WebXR sin tener que usar un dispositivo XR. Además, permite emular distintos dispositivos y sus componentes más básicos (como los controladores de gafas virtuales). [57]

En este proyecto se utilizará para distintas emulaciones sobre cambios pequeños sin necesidad de usar las gafas virtuales continuamente (que pueden causar cansancio en la vista).

2.7 Otras herramientas

Aparte de las herramientas ya explicadas anteriormente, se van a utilizar otra serie de herramientas que no entran en las categorías anteriores

2.7.1 Oculus Quest 2 y Oculus Developer Hub

Las Oculus Quest 2 son unas gafas de realidad virtual, creadas por la empresa Oculus. [58]

Imagen 3

En comparación con el modelo anterior, estas gafas tienen mayor resolución, incrementan en 2 GB (hasta un total de 6GB) respecto al modelo anterior y es más ligero. [59]

Esta tecnología es algo nuevo que está apareciendo en distintos campos de la informática, como los videojuegos y prácticas virtuales para otros campos, como medicina o militares. [60]

En este caso, se van a usar estas gafas para la visualización por medio del soporte que ofrece la API de JavaScript, WebXR.

Oculus Developer Hub es una aplicación específica para desarrolladores en Oculus, de manera que puedan ejecutar comandos, y duplicar la salida de las Oculus en el ordenador correspondiente. [61]

En este caso, se han usado esta aplicación para poder conectar las Oculus al ordenador y ejecutar una serie de comandos, y para poder ver la salida de la misma y habilitar el modo desarrollador y las funciones que eso trae consigo (facilidades a los desarrolladores a la hora de crear software para este tipo de productos).

2.7.2 Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto,aunque la descarga oficial está bajo software privativo e incluye características personalizadas por Microsoft. [62]

Visual Studio Code se basa en Electron, un framework que se utiliza para implementar Chromium y Node.js como aplicaciones para escritorio, que se ejecuta en el motor de diseño Blink. Aunque utiliza el framework Electron, el software no usa Atom y en su lugar emplea el mismo componente editor (Monaco) utilizado en Visual Studio Team Services (anteriormente llamado Visual Studio Online).

Esta aplicación consta también de extensiones que incluyen multitud de soportes de lenguajes (Javascript y Typescript en este caso) y otro tipo de ayudas al desarrollador, terminales para ejecuciones de los archivos del proyecto y un explorador de archivos.

Además, nos permite usar distintos tipos de terminales para según qué proyectos (PowerShell en nuestro caso), y tiene integrada una funcionalidad para actualizar automáticamente el repositorio de Github que estemos utilizando.

Esta hoja se ha dejado intencionadamente en blanco

3

ANÁLISIS Y DISEÑO

3.1 Estructura del proyecto

La configuración que tiene este proyecto se basa en la estructura básica proporcionada por el Framework de desarrollo que se ha utilizado. Esto quiere decir que la organización de los proyectos de PlayCanvas será distinta a la de los proyectos de Babylon JS.

3.1.1 Arquitectura

La arquitectura de este trabajo no es tan compleja como otros. Aun así, para explicar el funcionamiento del software, vamos a detallarlo a continuación con el siguiente pequeño esquema:

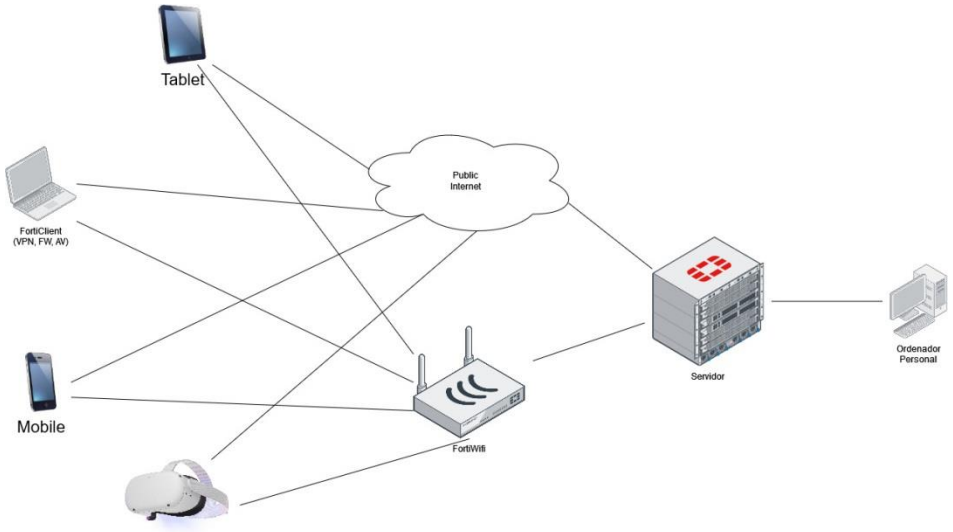

Figura 4 Arquitectura del Servidor

Tras ver el ejemplo anterior, vamos a explicar brevemente su funcionamiento

1. El código diseñado ejecuta una serie de instrucciones para generar un código JavaScript.
2. Dicho código JavaScript se integra en el archivo index.html.
3. Este archivo es cargado en el servidor que usamos de Web Server for Chrome.
4. Este servidor dispone de varias opciones de acceso, entre ellas, acceso personal (sólo el dispositivo donde se esté ejecutando), acceso local (dispositivos conectados a la misma red LAN) y acceso por Internet (sería accesible a todos los dispositivos).

Todo esto se puede llevar a cabo porque se utiliza la extensión de Web Server for Chrome.

La composición de dicha extensión se puede encontrar en su GitHub [63]. Ésta es una herramienta que usa protocolos de WebSockets para establecer un servidor y alojar los archivos que se referencien en un directorio local, y de esta forma, que los dispositivos que se conecten al Socket puedan acceder a las carpetas que se cargan.

El protocolo de transmisión que emplea es TCP. Resumiendo, TCP es Transmission Control Protocol, es decir, un protocolo de transmisión orientado a la conexión y que permite el monitoreo del flujo de datos y su multiplexación (si la información viene de distinto origen, pero en la misma línea, de tal forma que pueda circular concurrentemente).

3.1.2 Estructura de carpetas de los proyectos

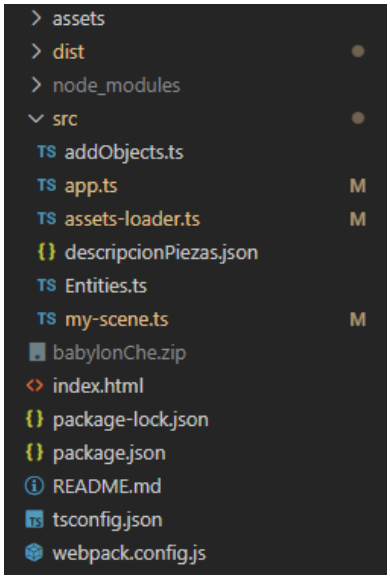

Imagen 5 Estructura de carpetas

A continuación, como se puede apreciar en la Figura 4, la estructura del código de los distintos proyectos está compuesta de la siguiente manera:

- Assets: En esta carpeta se guardan todos los contenidos de tipo estáticos
- Dist: En esta carpeta están los archivos con el código resultante tras compilar el código. En este caso, app.js es el archivo que será llamado por index.html para procesar el programa.
- Node_modules: en esta carpeta auxiliar, se almacenan las distintas librerías utilizadas en el proyecto. Esta carpeta no se sube al GitHub, ya que en este tipo de proyectos en el que se usa Node.js, se puede importar todas las librerías siempre que se sigan los pasos que aparecerán en el manual de usuario.
- Src: En esta carpeta se encuentra la parte principal de los distintos proyectos. Aquí se almacenan las distintas clases que se configuran para este trabajo. A continuación, se va a explicar brevemente cada una de ellas:
    - Assets-loader: en este archivo se importan todos los elementos de tipo estático, de manera que se pueda acceder a ellos más adelante.
    - addObjects: en este archivo se añaden los distintos modelos a escena, incluyendo las características que nosotros definamos.
    - app: En este archivo se ejecuta la parte principal del código desarrollado, que cargará la escena con todas las propiedades que se hayan creado.
    - Entities (proyecto TFG): en este archivo se realizan la mayoría de interacciones que tienen que ver con eventos de entrada, como cuando se hace clic con el ratón, o se usa el botón de la interfaz.
    - My-scene (proyecto TFG): en este archivo se cargan la mayoría de elementos de la escena que se usarán en app.js, para aligerar la carga de los archivos.
    - DescripciónPiezas (proyecto TFG): este archivo contiene las distintas descripciones de los modelos que se van a emplear. Si se quisiera añadir más descripciones, sólo habría que incrementar el array que las almacena.
    - createsCameras (proyecto PlayCanvas): este archivo carga de forma específica las cámaras del proyecto.
    - Lights (proyecto PlayCanvas): este archivo carga, específicamente, las luces del proyecto.
    - Helper (proyecto PlayCanvas): este archivo guarda el enrutamiento para poder acceder más fácilmente a la carpeta de Assets.

3.2 Modelos

Los modelos que se van a usar en este proyecto provienen de 2 fuentes distintas.

Varios de ellos provienen de un banco de datos de la propia universidad de Málaga. En concreto han sido proporcionados por el profesor de Ingenierías Industriales, Miguel Ángel Contreras López.

Dichos modelos serán empleados para cargarlos en el software y que se puedan realizar varias animaciones e interacciones entre ellos, así como mostrar distintas descripciones y sus respectivos nombres. El formato en el que se presentan es FBX, como se ha mencionado anteriormente, y se convertirá a GLB utilizando la herramienta de Blender.

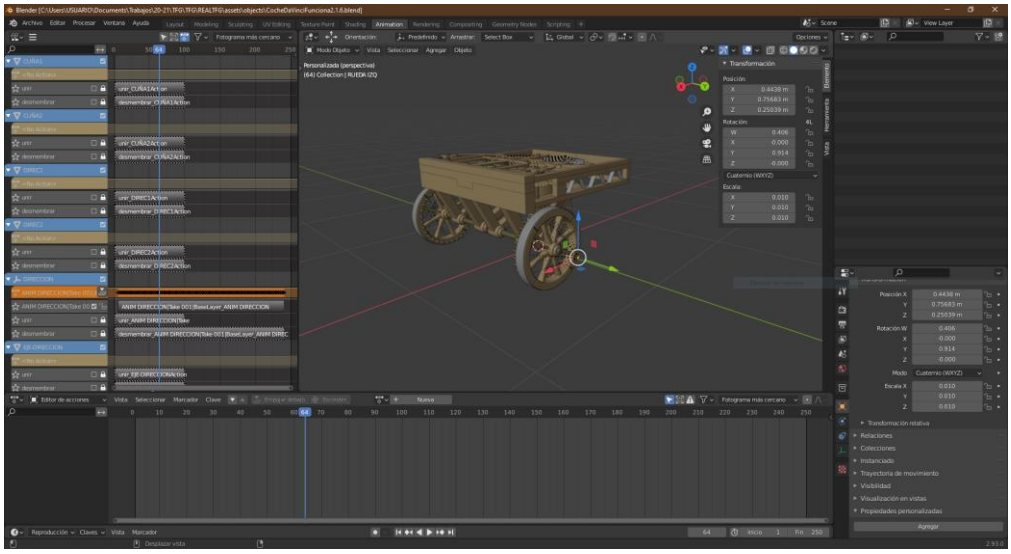

Imagen 6 Animando modelos en Blender

También se ha usado Blender para reducir el número de polígonos que se generaban en distintos modelos muy pesados. Para ello se empleaba una herramienta de colapso de polígonos según una razón dada.

Por último, también se ha utilizado la herramienta para el desarrollo de algunos modelos a través de distintas funciones geométricas.

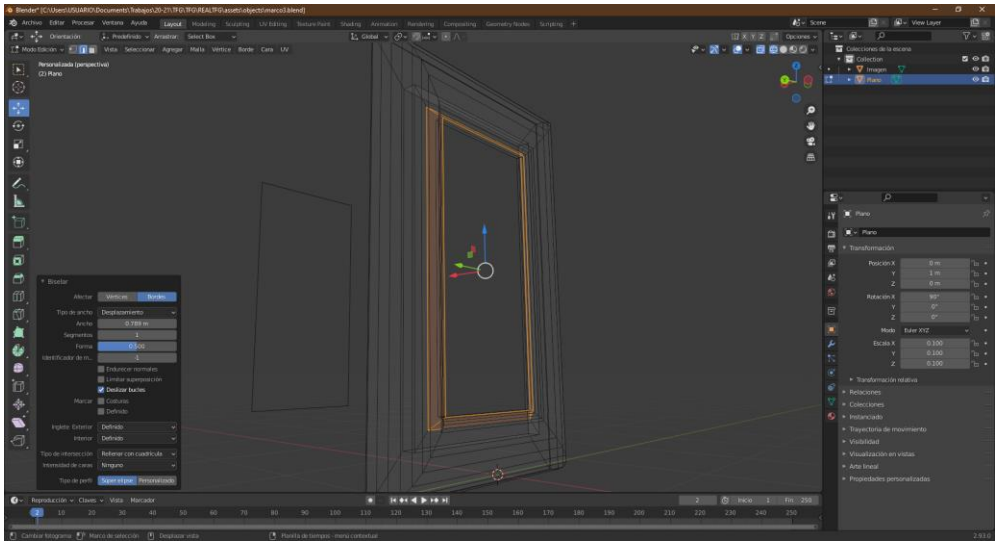

Imagen 7 Modelando con funciones geométricas

3.3 Metodología del desarrollo

En este proyecto se ha usado la metodología Ágil, en concreto, la metodología Scrum, ya que nos permite planificar este trabajo en distintas iteraciones y permitiendo un alto grado de rapidez y flexibilidad, en caso de que hubiera que hacer frente a posibles cambios del proyecto.

Esta metodología se compone de un número de etapas denominadas sprints (de un tiempo cercano a dos semanas), en los cuales se desarrollan los requisitos acordados con el cliente, y en la que, al finalizar, se hace una evaluación de los resultados con el mismo.

Para poder realizar las reuniones que no podían ser presenciales, se ha utilizado google Meet, así como un repositorio github con las instrucciones para la descarga e inicialización de los proyectos, con el fin de que el cliente (tutor), pudiese estar al tanto de los cambios que se producían.

Además de las reuniones que se llevaban a cabo y las indicaciones de los requisitos que se debían tener listos para el próximo encuentro, se fijaba la siguiente fecha en el Google Calendar. De esta forma, las reuniones y los objetivos quedaban claramente especificados.

Por último, sobre el diseño a seguir, éste era un proyecto que partía desde cero. Por esto mismo la arquitectura del software no tenía ninguna restricción ni convención. Uno de los objetivos era investigar los distintos tipos de motores mencionados anteriormente (Playcanvas, Babylon JS y Unity WebGL) los cuales se habían discutido previamente con el tutor para ver cuáles de ellos eran viables para realizar este tipo de programa. El código del software se ha ido creando siguiendo un criterio personal, empleando las librerías que se piensan más convenientes, o actualizando mejoras que se consideraban que incrementarían el valor de este proyecto.

Esta hoja se ha dejado intencionadamente en blanco

4

Desarrollo de la aplicación

En este capítulo se explicará todo el desarrollo que ha seguido el proyecto. Se incluye el desarrollo de software y cualquier otro tipo de tarea que afecte al desarrollo del trabajo.

Para ello, iremos viendo cada uno del sprint que se han seguido.

4.1 Proyecto Playcanvas

En este proyecto se trataba de hacer una serie de pasos para realizar un trabajo de prueba en el Framework de Playcanvas y, así, poder observar su compatibilidad.

4.1.1 Creación del Entorno

Como primer paso, se crea una estructura de carpetas y archivos básica para realizar el primer programa en Playcanvas.

Como requisitos previos, se debe tener instalado Node.js para poder cargar varios de los archivos y de los módulos, usando ***npm install***.

Tras hacer una búsqueda en diversos foros, se localizó una jerarquía parecida a la de los proyectos web, donde existía una carpeta public con un archivo index.html, otra carpeta static, con el contenido estático necesario para la ejecución del programa (cámaras, fuente de texto, modelos, imágenes...) y una última carpeta src, donde se encuentra el código principal con los distintos archivos y clases para poder ejecutar el programa.

En esta última carpeta están definidos los archivos de *assets-loader.ts, app.ts, helper.ts, index.ts* y *lights.ts*.

4.1.2 Implementación de elementos básico

Después se trató de añadir elementos sencillos, ya que se estaba empezando a programar en Playcanvas. Por elementos sencillos, se hace referencia a luz ambiente, suelo, fondo, cámara, sombras y entidades básicas (figuras geométricas).

Todos estos elementos se añaden en el archivo de app.ts para ser llamados por la función principal.

Tras añadir estos elementos se intentó cargar distintos los modelos que se poseían del banco de datos. Los modelos cargados son los archivos de *uploads_files_2792345_Koenigsegg.glb, ejemplo_2.glb* y *ejemplo_2_mod.glb* (este segundo tras hacer varias modificaciones en Blender).

Imagen 8 uploads_files_2792345_Koenigsegg.glb

Imagen 9 ejemplo_2_mod.glb

4.1.3 Avance en cámaras y Meshes (mallas)

A continuación, se estuvo probando distintas configuraciones de cámaras respecto a la que tenía. La cámara que había desarrollado anteriormente era una cámara orbital, y estas nuevas cámaras eran en primera persona y libre, lo cual permite el movimiento alrededor del escenario

Mientras que para la cámara libre (llamada *flyCamera*) se permite el desplazamiento rápido. La cámara en primera persona se centra en el uso de cuerpos y colisiones, además de simular efecto de gravedad.

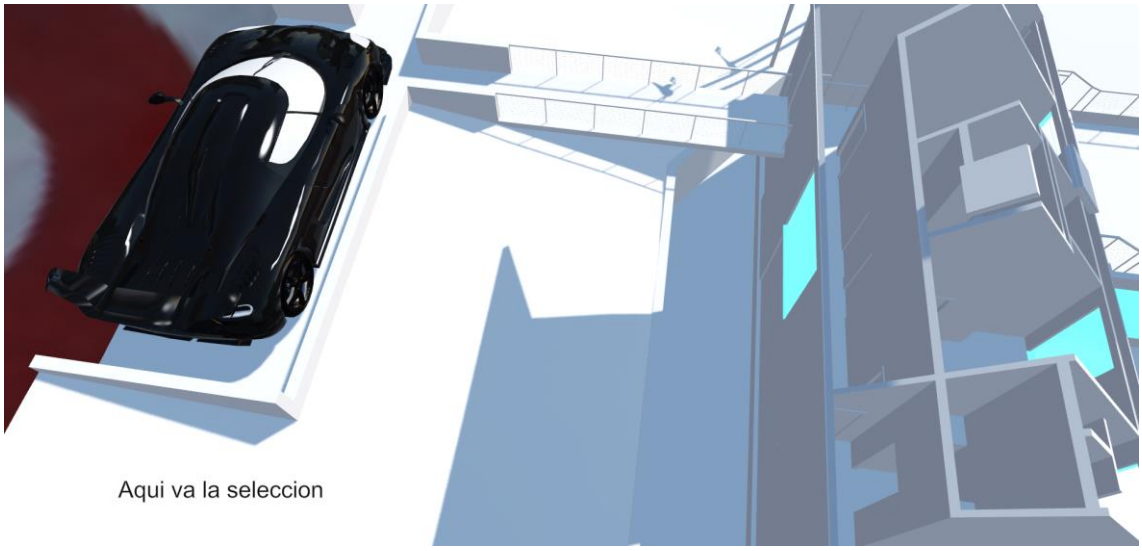

Imagen 10 PlayCanvas Camara flyCamera

También se probó la jerarquía de mallas de los distintos modelos, para probar si se podía acceder a distintas mallas de un mismo objeto y poder realizar, así, animaciones y cambios sobre ella de manera individual.

En este caso, se consiguió realizar una serie de animaciones simples para lograr girar las ruedas del modelo del coche al mismo tiempo que el coche rotaba sobre sí mismo (Matización: las ruedas del coche no formaban parte de la entidad del coche, ya que al probarlas siendo parte del mismo no funcionaba bien la animación). Aparte, se implementó el modelo de una casa con la que se probó a transformar distintos subelementos de la misma.

4.1.4 Reorganización de elementos e implementación de selector

Seguidamente, se reorganizó en distintos archivos la parte del programa que hacía referencia a la carga y modificación de los objetos, y a las cámaras, para no saturar de código el archivo principal. Para ello se crearon los archivos de *addObjects.ts* y *createCameras.ts*, donde se alojaron estas funcionalidades y son importados en la clase principal, *app.ts*.

Además, se creó un selector de mallas para que se mostrara el nombre de malla que fuese seleccionada. Esto era un primer paso de toma de contacto con los *Raycast* (objetos que proyectan un rayo y devuelven el objeto, con el que han chocado, si es que lo han hecho) para poder seleccionar las mallas y, más tarde, poder aplicarle efectos y animaciones.

Para realizar este paso, se empleó una propiedad en específico de la clase de *MeshInstance*, ***aabb***, en la cual se crea una entidad que es una caja invisible, la cual se activa cada vez que el cursor hace clic sobre ella. El motivo por el que se tuvo que usar una propiedad tan específica, es porque los elementos generados del modelo eran una clase de tipo *MeshInstance*, que no tenía superficie de colisión como tal, es decir, que los rayos atraviesan dicho elemento sin detectar colisión. Y si se trata de usar la posición, esta no hace referencia al centro del objeto, cosa que sí hace la propiedad aabb.

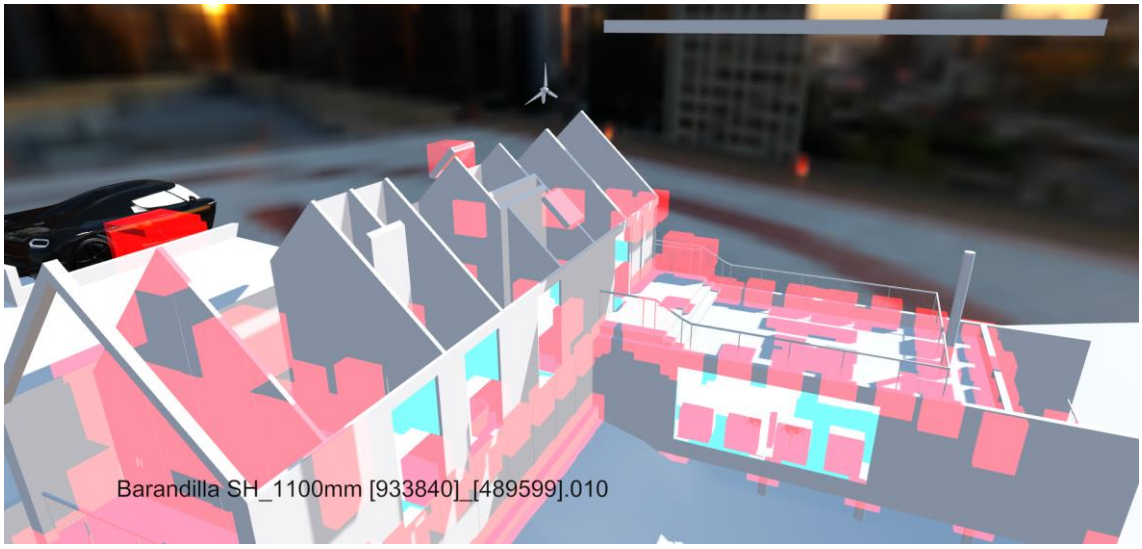

Imagen 11 PlayCanvas Representación aabb

En la Imagen 11 podemos observar una caja roja. Dicha caja hace referencia a dónde estaría la “caja invisible” aabb.

Para poder usar la propiedad *aabb*, se modificaron las clases de las cámaras en la carpeta *statics/cameras*. Allí se importó la clase de *Raycast*, y se utilizó un evento que, al hacer clic, llamaba a una función para comprobar si se había encontrado algún tipo de colisión con la propiedad mencionada en la parte de arriba.

Dicha función realizaba un *Raycast* desde la antigua posición del cursor hasta la nueva, utilizando una función para portar la posición del cursor hasta una nueva posición, pero en la escena, no en la pantalla (la función se denomina *screenToWorld*).

Tras ello, el *RayCast* devolvía la malla concreta, si es que había colisionado con alguna. Así conseguía obtener el objeto, y en este caso, mostrar el nombre por pantalla.

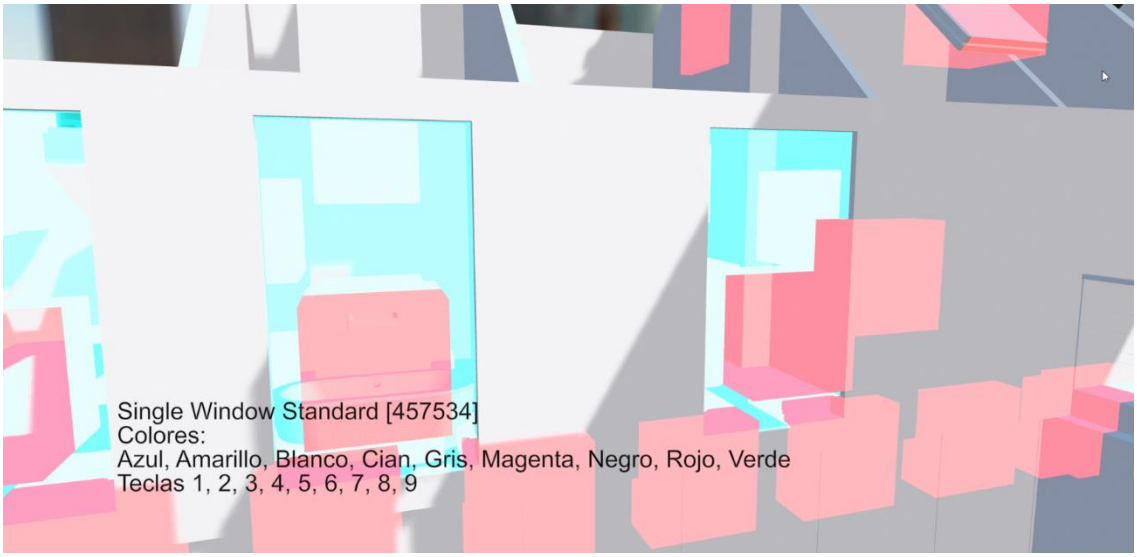

Imagen 12 PlayCanvas aabb ejemplo

En la Imagen 12 se ha tratado de hacer clic en la ventana en sí, pero el Raycast no lo reconoce. Sin embargo, al hacer clic en la parte roja, sí lo detecta.

4.1.5 Animaciones, materiales y movimientos de mallas

Por último, tras crear el selector, se trabajó para crear distintas animaciones para mallas con nombre en específico, de tal forma que se activasen al hacer clic sobre ellas, a la vez que también podía cambiar el color de los materiales de otro tipo de mallas con órdenes por teclado, y podía mover distintas mallas por botones que ofrecía el motor *Playcanvas*.

Para ello, se ha creado un atributo en la clase *addObjects.ts,* ***interactiveParts***, que recoge los distintos elementos que tienen una etiqueta específica en el nombre en un array por cada tipo junto con otras propiedades, como las instrucciones para ejecutar el efecto asociado a los mismos elementos.

Dichos efectos se encuentran en *app.ts*, donde se ha realizado un *Switch* con las distintas entradas que son válidas, para aplicar distintos efectos según la tecla de entrada y el elemento que se haya seleccionado.

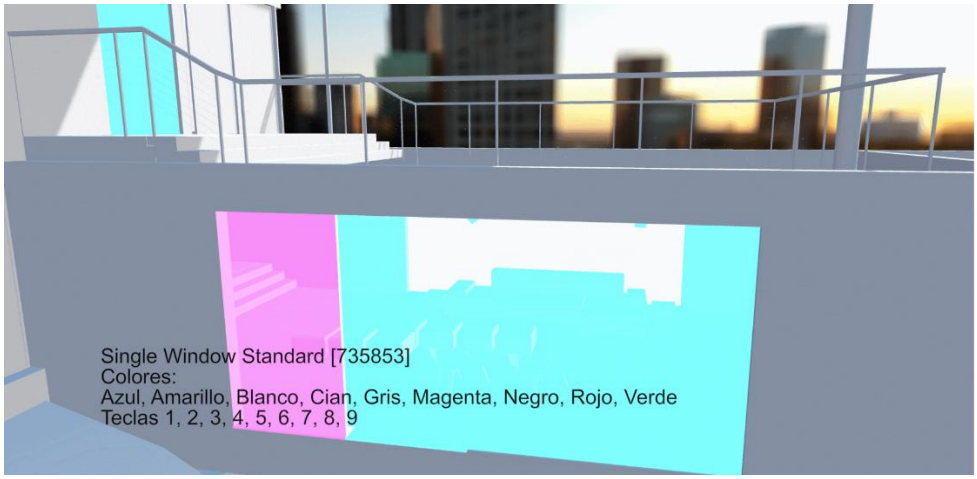

Imagen 13 PlayCanvas Interacción

En la Imagen 13 se puede observar cómo resulta la función para cambiar colores en las ventanas.

Sin embargo, tras todo este trabajo, se llega a la conclusión de que la estructura de mallas de Playcanvas no es la más adecuada para el tipo de proyecto que se pretendía hacer. Tras la información recopilada en los foros, parece que se estaba desarrollando un nuevo tipo de clase que solucionaría el tema de jerarquía (*RenderComponent*), pero todavía está en fase de desarrollo [64] [65]. Ya sólo quedaba ver que tal serían los resultados con el Framework de Babylon JS.

4.2 Proyecto Babylon JS

En este proyecto se trataba de hacer una serie de pasos para probar el Framework de *Babylon JS* y ver su compatibilidad con el proyecto a realizar.

4.2.1 Creación del Entorno

Como primer paso, se decide crear una estructura de carpetas y archivos básica, para poder realizar el primer programa en Babylon JS.

Tras investigar en diversos foros, y debido a la experiencia adquirida en el desarrollo del proyecto de Playcanvas, se lleva a cabo una estructura similar.

4.2.2 Implementación de las funcionalidades de Playcanvas

Después de comenzar el proyecto en Babylon JS, se percibió enseguida que las funciones estaban mucho más completas, más documentadas, y que estaba mucho más enfocado en el soporte de Typescript y tipado. Además, ya se tenía experiencia previa habiendo programando en PlayCanvas, y ya se sabía cómo afrontar varios de los problemas.

Dejando clara esta parte, se avanzó rápidamente en la implementación de las funcionalidades y en un solo sprint se hizo lo que me costaba realizar en Playcanvas en tres sprints.

A pesar de que las funcionalidades eran las mismas, la manera de llevarlas a cabo es distinta.

En el caso de la selección, por ejemplo, los *Raycast* ya están integrados en una de las funciones del propio objeto que representa a la escena, y las *meshes* (mallas) funcionaban con colisiones, animaciones, y con todas las propiedades a pesar de no estar en lo alto de la jerarquía.

Para aplicar efectos a varios elementos con mismas etiquetas, podía simplemente añadir a cada uno un *Tag* (etiqueta), y pedir que el propio objeto de la escena me devolviese todos los elementos con cierto *Tag*.

Para resumir, muchos de los requisitos y funcionalidades que fueron difíciles implementar en *PlayCanvas* existen en *Babylon JS*, o son cuasi inmediatos, y están muy bien documentados, lo que permitía llevar a cabo investigaciones más ambiciosas.

4.2.3 Desarrollo de paneles complejos y selectores

Como última parte de Babylon JS, se realizó una investigación para tratar de hacer paneles de botones, paletas de colores, selectores y efectos especiales más concretos para ahondar el conocimiento, ya que se discutió con el tutor cuál era el mejor Framework para realizar el proyecto, y al final se decidió era Babylon JS. El resultado final de este proyecto lo podemos apreciar en la Imagen 14.

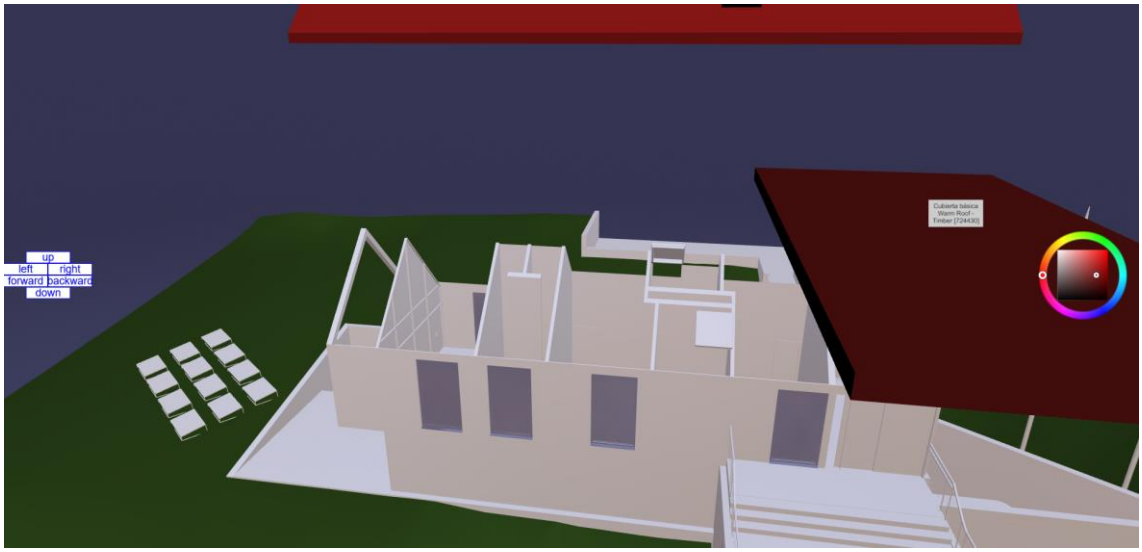

Imagen 14 Babylon

4.3 Aplicación final

Para el proyecto final se trató de implementar lo aprendido de Babylon JS, junto con el uso de modelos y animaciones a través de Blender.

4.3.1 Modelados con Blender e investigación de servidores

En esta última parte del proyecto, se estuvo tratando con distintos modelos (banco de modelos del profesor Miguel Ángel Contreras) para concretar el tema de animaciones, jerarquía de objetos y modelado de distintos cuerpos en Blender.

Para ello, se utilizaron varios de los modelos que se iban a importar en el trabajo final, y se llevaron a cabo distintas jerarquías de animaciones. Esto en sí es importante, ya que más adelante, en Babylon JS, se usará esta misma jerarquía para crear grupos de animaciones que serán llamados para actuar al mismo tiempo.

También se estuvieron creando distintos modelos para ser usados en el proyecto, mediante el uso de formas geométricas y texturas.

Por último, se realizaron búsquedas en distintos foros para encontrar algún tipo de programa o extensión para poder usar el software en un servidor accesible desde otros dispositivos (uno de los objetivos de este proyecto), y se encontró la extensión mencionada anteriormente de Web Server for Chrome. Dicha extensión permite crear un servidor que tenga distintos tipos de conexiones, como servidor que use exclusivamente una red local (es decir, solo el ordenador que alojaba al servidor tenía acceso a su contenido), redes locales (todos los dispositivos que estén conectados a una misma red local, como una conexión a un router o switch), y por Internet.

En el caso de este proyecto, se ha usado un servidor que funcione en una red local para comprobar que la aplicación se ejecutaba correctamente en el navegador de otros dispositivos.

4.3.2 Cámara, paneles y visión 3D

Después se implementó una de las partes importantes de este proyecto, que era la adaptación de la escena a distintos dispositivos, entre ellos, las gafas virtuales.

Para ello se ha tenido que realizar varios cambios en los paneles y en las cámaras para poder adecuarlas a la especificación de WebXR (inmersión virtual) que se usa en distintos dispositivos de realidad aumentada, como las gafas virtuales, como vemos en la Imagen 15.

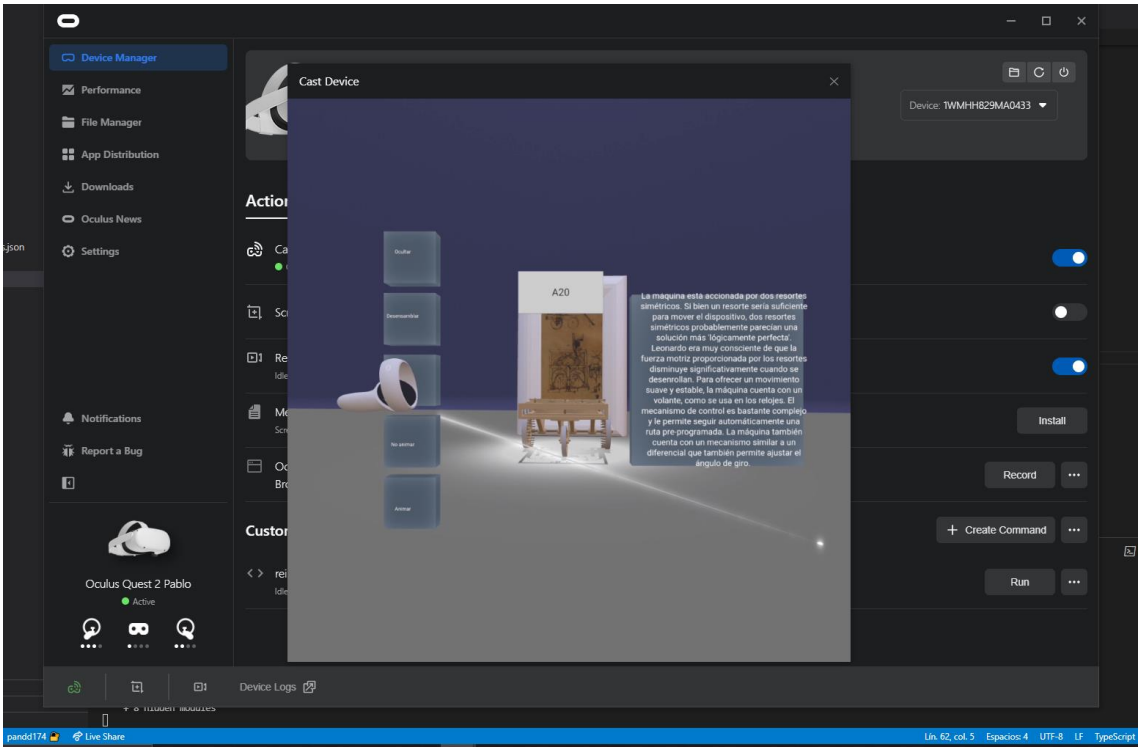

Imagen 15 Ejecución en Oculus Quest 2

Principalmente, adaptar los distintos controles y superficies al escenario de realidad virtual, y al mismo tiempo que se utilizan eventos para las posibles entradas de los controladores de las gafas virtuales.

Al principio se encontraron varias dificultades, entre ellas, por ejemplo, que cargaba la malla del suelo después de ser llamada por la cámara virtual, lo que generaba problemas de desplazamiento con las gafas; o la superposición de la cámara normal con la cámara de realidad virtual. Sin embargo, al final estos problemas fueron superados.

4.3.3 Escenario final

Por último, se ha estado implementando el resto de modelos y funcionalidades restantes que faltaban, entre ellas, juego de luces, descripciones para los objetos seleccionados con un JSON, arreglos en distintas animaciones, mejoras en la interfaz, y en el renderizado de la escena.

Entre ellas, uno de los problemas solucionados era el hecho de poder mover la interfaz junto a la cámara. La solución consistió en hacer la interfaz hija de los nodos de las distintas cámaras, y con distintas cámaras, hacer referencia tanto a la cámara sin inversión, como a la cámara de realidad virtual.

Uno de los puntos era tener un evento para detectar cuando se cambiaba de cámara, y cambiar el padre de la interfaz. Para ello, se ha usado la escena XR, y la enviamos, junto a la primera cámara activa al archivo de Entities.ts. Allí, a la hora de crear la interfaz, creamos dicho evento usando la escena XR para que nos avise cuando se crea la sesión, y cuando nos salimos de ella.

Al final, tras corregir estos problemas, quedó la parte de renderizado visual, para que los archivos pudiesen ser cargados más rápidos, ya que tenían muchos polígonos (debido a que los modelos importados están muy detallados), para poder reducirle el tamaño de archivo.

El resultado final se puede apreciar en la Imagen 16

Imagen 16 Aplicación Final

5

Conclusiones

Al final se ha realizado un proyecto que permite la visualización de una escena que se ha desarrollado para distintos dispositivos, con posibilidad de usar una versión de realidad virtual.

5.1 Mejoras y trabajo futuro

Tras realizar las distintas comparaciones, se podía apreciar de manera casi inmediata que, a la hora de programar, Babylon JS ofrece muchas más ventajas y facilidades que Unity WebGL y Playcanvas. El material disponible que hay en foros, en la API, y en el propio soporte de las funciones al programar es superior en el caso de Babylon JS, ofreciendo mucha más información y opciones.

La principal mejora sería realizar un proyecto enfocándose únicamente en el Framework de Babylon JS, ya que, como se comentó anteriormente, lo que se tardó en realizar en Playcanvas en 3 sprints, se llevó a cabo en Babylon JS en un único sprint.

Además, el hecho de centrarse demasiado en los motores, impidió aprovechar el máximo de las herramientas de modelados como Blender. Es posible que si el proyecto hubiese tenido más tiempo, se hubiera podido haber creado a mano modelos complejos con animaciones más específicas y completas, dando más diversidad al proyecto, ya que, a pesar de que las animaciones fueron creadas en Blender, muchos de los modelos son de un banco de datos proporcionado por la UMA.

Otra de las partes de las que habría sido positivo ahondar más es en el desarrollo de la parte de realidad virtual con las gafas virtuales. Es muy probable que se hubiese podido haber desarrollado un tipo de movimiento con el joystick de los controladores de las gafas que fuese suave, en 3 direcciones (en el proyecto, está ligado al suelo) y complementase la experiencia; ya que, aunque se pudo crear movimiento con el controlador, era muy brusco, y se decidió cambiarlo por un teletransporte del usuario con la posibilidad de cambiar de dirección.

Por último, habría sido positivo haber podido acortar los tiempos de carga, ya que varios de los modelos son muy pesados, y seguramente se podrían haber hecho más modificaciones en Blender para aligerar el peso que suponen al arrancar el programa.

5.2 Aprendizaje personal

Para empezar, es cierto que estaba interesado en la parte gráfica de la programación, pero todos los proyectos que había hecho utilizaban Unity, o en el caso de ser en un navegador, simplemente eran elementos simples (gráficas de datos en mi caso) partiendo de un caso base.

Es más, fue el reto por parte de mi tutor lo que hizo que me interesase en el mundo de los Frameworks para renderizado de modelos tridimensionales en los distintos navegadores.

Esto me ha permitido aprender a desarrollarme con este tipo de Frameworks y a realizar modelos en 3 dimensiones con herramientas que nunca antes había usado.

Asimismo, el proyecto en sí era un continuo aprendizaje para mí, ya que, quitando cierto conocimiento previo de TypeScript y JavaScript, realmente no estaba especializado en estas áreas.

Además, he aprendido a desenvolverme bastante bien buscando información a través de los foros de las distintas herramientas. Es más, desde aquí querría hacer un agradecimiento especial a todos los usuarios que han estado debatiendo e informando sobre los temas que tenía que tratar, ya que sin ellos me habría costado avanzar en ciertos puntos del proyecto.

6

Referencias

[1] [Autodesk] Información Extensión FBX (Accedido en Junio 2020) Disponible en: https://www.autodesk.com/products/fbx/overview

[2] Inforación Extensión GLB (Accedido en Junio 2020) Disponible en: https://www.file-extension.info/es/format/glb

[3] [Microsoft] Paint 3D (Accedido en Junio 2020) Disponible en: https://www.microsoft.com/en-us/p/paint-3d/9nblggh5fv99#activetab=pivot:overviewtab

[4] Entenciones de Paint 3D (Accedido en Junio 2020) Disponible en: https://www.file-extensions.org/paint-3d-file-extensions

[5] [Microsoft] Crear animación 3D con Paint 3D (Accedido en Junio 2020) Disponible en: https://answers.microsoft.com/en-us/windows/forum/all/how-can-i-create-animation-using-paint-3d-and-3d/f5293c55-0a42-46e8-bcb1-bc8a084c083b

[6] [Blender] Página principal Blender (Accedido en Junio 2020) Disponible en: https://www.blender.org/

[7] [Bforartists] Características Bforartists (Accedido en Junio 2020) Disponible en: https://www.bforartists.de/

[8] [Unity] Publicación de proyectos WebGL (limitaciones) (Accedido en Julio 2020) Disponible en: https://learn.unity.com/tutorial/how-to-publish-for-webgl#5d925085edbc2a0c00bd4608

[9] [PlayCanvas] Formatos 3D PlayCanvas (Accedido en Julio 2020) Disponible en: https://developer.playcanvas.com/en/user-manual/assets/

[10] [Babylon] Formatos 3D Babylon (Accedido en Julio 2020) Disponible en: https://doc.babylonjs.com/divingDeeper/importers/loadingFileTypes

[11] [Unity] Formatos 3D Unity (Accedido en Julio 2020) Disponible en: https://docs.unity3d.com/Manual/3D-formats.html

[12] [PlayCanvas] Entradas soportadas PlayCanvas (Accedido en Julio 2020) Disponible en: https://developer.playcanvas.com/en/user-manual/user-interface/input/

[13] [PlayCanvas] Entradas XR soportadas PlayCanvas (Accedido en Julio 2020) Disponible en: https://developer.playcanvas.com/en/user-manual/xr/input-sources/

[14] [Babylon] Entradas soportadas Babylon Disponible en: https://doc.babylonjs.com/divingDeeper/input

[15] [Unity] Entradas soportadas Unity (Accedido en Julio 2020) Disponible en: https://docs.unity3d.com/Manual/Input.html

[16] [PlayCanvas] Características PlayCanvas (Accedido en Julio 2020) Disponible en: https://playcanvas.com/features

[17] [Babylon] Características Babylon (Accedido en Julio 2020) Disponible en: https://doc.babylonjs.com/divingDeeper/developWithBjs/npmSupport#typescript-support

[18] [Untiy] Características Untiy (Accedido en Julio 2020) Disponible en: https://docs.unity3d.com/es/2018.4/Manual/webgl-gettingstarted.html

[19] [PlayCanvas] Página principal PlayCanvas (Accedido en Agosto 2020) Disponible en: https://playcanvas.com/

[20] Navegadores soportados por PlayCanvas (Accedido en Julio 2020) Disponible en: https://www.phoronix.com/scan.php?page=news_item&px=MTcxMDA

[21] [GamingOnLinux] Apertura Github (Accedido en Julio 2020) Disponible en: https://www.gamingonlinux.com/articles/playcanvas-3d-webgl-game-engine-now-open-source.3843?module=articles_full&title=playcanvas-3d-webgl-game-engine-now-open-source&aid=3843

[22] [PlayCanvas] Características PlayCanvas (Accedido en Julio 2020) Disponible en: https://playcanvas.com/features#!

[23] [PlayCanvas] Clase Entidad PlayCanvas (Accedido en Agosto 2020) Disponible en: https://developer.playcanvas.com/en/api/pc.Entity.html

[24] [PlayCanvas] Chat sobre problema Entidad (Accedido en Agosto 2020) Disponible en: https://github.com/playcanvas/engine/issues/2063

[25] [Babylon] Página Principal (Accedido en Julio 2020) Disponible en: https://www.babylonjs.com/

[26] [Babylon] Primer Commit Github (Accedido en Julio 2020) Disponible en: https://github.com/BabylonJS/Babylon.js/commit/4b747f4c71fd479cd75a213f3c0de27efdc69738

[27] [Babylon] Especificaciones Babylon (Accedido en Julio 2020) Disponible en: https://www.babylonjs.com/specifications/

[28] [Babylon] Clase Mesh Babylon (Accedido en Agosto 2020) Disponible en: https://doc.babylonjs.com/typedoc/classes/babylon.mesh

[29] [Microsoft] Página principal TypeScript (Accedido en Septiembre 2020) Disponible en: https://www.typescriptlang.org/

[30] [Microsoft] Documentación TypeScript (Accedido en Septiembre 2020) Disponible en: https://www.typescriptlang.org/docs/handbook/intro.html

[31] [InfoWorld] Desarrollo de Microsoft sobre JavaScript (Accedido en Septiembre 2020) Disponible en: https://www.infoworld.com/article/2614863/microsoft-augments-javascript-for-large-scale-development.html

[32] [Channel9] Anders Hejlsberg introduciendo TypeScript (Accedido en Septiembre 2020) Disponible en: https://channel9.msdn.com/posts/Anders-Hejlsberg-Introducing-TypeScript

[33] [Blog de Miguel de Icaza] Entrada del 1 Octubre 2012 (Accedido en Septiembre 2020) Disponible en: https://tirania.org/blog/archive/2012/Oct-01.html

[34] [Microsoft] Soporte de editores para TypeScript (Accedido en Septiembre 2020) Disponible en: https://github.com/Microsoft/TypeScript/wiki/TypeScript-Editor-Support#visual-studio-20132015

[35] [Blog de Microsoft] Anuncio Compilador TypeScript (Accedido en Septiembre 2020) Disponible en: https://devblogs.microsoft.com/typescript/new-compiler-and-moving-to-github.aspx no acceso, citado por:

http://www.ktorides.com/2014/07/typescript-compiler-got-re-write-now-5-times-faster/

https://i-programmer.info/news/136-open-source/7571-typescript-goes-light-moves-to-github.html

https://www.spantip.com/wiki/TypeScript

[36] [Blog de Microsoft] Anuncio TypeScript 4.0 (Accedido en Septiembre 2020) Disponible en: https://devblogs.microsoft.com/typescript/announcing-typescript-4-0/

[37] [Microsoft] Tipado TypeScript (Accedido en Septiembre 2020) Disponible en: https://www.typescriptlang.org/docs/handbook/basic-types.html

[38] [Khronos] Introducción a WebGL (Accedido en Septiembre 2020) Disponible en: https://www.khronos.org/webgl/wiki/Getting_Started

[39] Navegadores que soportan WebGL (Accedido en Agosto 2020) Disponible en: https://caniuse.com/webgl

[40] [Khronos] Índice WebGL (Accedido en Septiembre 2020) Disponible en: https://www.khronos.org/webgl/wiki/Main_Page

[41] [Khronos] Especificaciones WebGL (Accedido en Septiembre 2020) Disponible en: https://www.khronos.org/registry/webgl/specs/1.0/

[42] [Mozilla, Robert Nyman] Tutorial sobre WebGL (Accedido en Septiembre 2020) Disponible en: https://hacks.mozilla.org/2013/04/the-concepts-of-webgl/

[43] [Khronos] Información sobre Shaders (Accedido en Septiembre 2020) Disponible en: https://www.khronos.org/opengl/wiki/Shader

[44] [Khronos] Imagen descriptiva del proceso de Pipeline (Accedido en Septiembre 2020) Disponible en: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

[45] [OpenGL] Página principal OpenGL (Accedido en Septiembre 2020) Disponible en: https://www.opengl.org//

[46] [Blog de Joe Groff, Entrada 2010] Introducción a Pipeline (Accedido en Septiembre 2020) Disponible en: https://duriansoftware.com/joe/an-intro-to-modern-opengl.-chapter-1:-the-graphics-pipeline

[47] [Babel] Página principal Babel (Accedido en Septiembre 2020) Disponible en: https://babeljs.io/

[48] [NPM] Dependencias react-canvas (Accedido en Septiembre 2020) Disponible en: https://www.npmjs.com/package/react-canvas-js?activeTab=dependencies

[49] [Eslint] Página principal Eslint (Accedido en Septiembre 2020) Disponible en: https://eslint.org/

[50] [GitHub] GitHub Java Obfuscator (Accedido en Septiembre 2020) Disponible en: https://github.com/javascript-obfuscator/javascript-obfuscator

[51] [Jest] Página principal Jest (Accedido en Septiembre 2020) Disponible en: https://jestjs.io/es-ES/

[52] [Pre-Commit] Página principal Pre-Commit (Accedido en Septiembre 2020) Disponible en: https://pre-commit.com/

[53] [webpack] Guía webpack (Accedido en Septiembre 2020) Disponible en: https://webpack.js.org/guides/author-libraries/

[54] [Google] Página principal Chromium (Accedido en Agosto 2020) Disponible en: https://www.chromium.org/

[55] [Google, Entrada Blog Enero 2017] Código abierto Chromium (Accedido en Agosto 2020) Disponible en: https://blog.chromium.org/2017/01/open-sourcing-chrome-on-ios.html

[56] [Google] Extensión Web Server For Chrome (Accedido en Septiembre 2020) Disponible en: https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb?hl=es

[57] [Google] Extensión WebXR API Emulator (Accedido en Septiembre 2020) Disponible en: https://chrome.google.com/webstore/detail/webxr-api-emulator/mjddjgeghkdijejnciaefnkjmkafnnje

[58] [Oculus] Oculus Quest 2 (Accedido en Septiembre 2020) Disponible en: https://www.oculus.com/quest-2/?locale=es_ES

[59] [Polygon, artículo del 16 de Septiembre de 2020 por Ben Kuchera] Artículo sobre Oculust Quest 2 (Accedido en Septiembre 2020) Disponible en: https://www.polygon.com/reviews/2020/9/16/21437762/oculus-quest-2-review-virtual-reality-vr-facebook-oculus-power-resolution-tracking

[60] [OSSO] Página principal OSSO (Accedido en Agosto 2020) Disponible en: https://www.ossovr.com/

[61] [Oculus] Documentación sobre Oculus Developer Hub (Accedido en Septiembre 2020) Disponible en: https://developer.oculus.com/documentation/unity/ts-odh/

[62] [Wikipedia Visual Studio 2021] Introducción y Características de Visual Studio (Accedido en Septiembre 2020) Disponible en: https://en.wikipedia.org/wiki/Visual_Studio_Code

[63] [GitHub Web Server for Chrome] (Accedido en Septiembre 2020) Disponible en: https://github.com/kzahel/web-server-chrome

[64] [GitHub PlayCanvas] Implementación Clase RenderComponent (Accedido en Agosto 2020) Disponible en: https://github.com/playcanvas/engine/pull/2345

[65] [GitHub PlayCanvas] Cargar modelos GLTF como jerarquía de Entidades (Accedido en Agosto 2020) Disponible en: https://github.com/playcanvas/engine/issues/2063

[66] [GitHub PlayCanvas] (Accedido en Agosto 2020) Disponible en: https://github.com/playcanvas/playcanvas-viewer

[API PlayCanvas] https://playcanvas.github.io/#/misc/hello-world

[API Babylon] https://doc.babylonjs.com/typedoc

Nota: Todas las referencias a URL se han accedido por última vez correctamente el día 25 de spetiembre de 2021.

Apéndice A

Manual de Instalación

7.1 Instalación

Para poder desplegar el software, es necesario tener instalado previamente:

- Instalar la versión de Node.js v14.16.0.
- Tener un compilador de Typescript.
- Clonar y descargar un Zip del repositorio que se puede encontrar en https://github.com/pandd174/TFG
- Extraer el archivo Zip clonado del repositorio en una carpeta
- Tener instalado el navegador Chrome en la versión 93.0.4577.63 (a día 31 de Agosto de 2021, es la última).
- Tener instalada la extensión Web Server for Chrome en Google Chrome.
- Si se quiere apreciar la inmersión virtual, instalar la extensión en Chrome de WebXR API Emulator.

A continuación, se instalan los módulos de cada uno de los distintos softwares.

Se puede usar la Consola de Comandos de Windows para la ejecución de la siguiente serie de comandos, o la consola de PowerShell.

    - Aclaratorio, cuando hablamos de PATH, se hace referencia a la ruta de la carpeta en la cual se ha extraído el Zip del repositorio clonado.
- cd PATH
- cd Babylon
- npm install
- cd ..
- cd PlayCanvas
- npm install
- cd ..
- cd TFG
- npm install

7.2 Ejecución de las aplicaciones

A continuación, se procede a explicar cómo se pueden ejecutar cada una de las aplicaciones.

7.2.1 PlayCanvas

Para poder hacer funcionar el apartado de PlayCanvas, se debe, tras realizar la instalación:

- Acceder a la carpeta PlayCanvas con la Consola de Comandos
- Preparar los archivos ejecutando: **npm run-script build**

- Ejecutar el programa usando: **npm start**
- Abrir un navegador con la dirección de http://localhost:8080/

7.2.2 Babylon

Para poder hacer funcionar el apartado de Babylon, se debe, tras realizar la instalación:

- Acceder a la carpeta Babylon con la Consola de Comandos
- Preparar los archivos ejecutando: **npm run-script build**
- Ejecutar el programa usando: **npm start**
- Abrir un navegador con la dirección de http://localhost:8080/

7.2.3 Proyecto Final

Para poder hacer funcionar el último apartado, se debe, tras realizar la instalación:

- Acceder a la carpeta TFG (cámbiale el nombre) con la Consola de Comandos
- Ejecutar el programa usando: **npm run-script watch**
- Abrir Google Chrome sin modo incógnito, y acceder a las Aplicaciones del mismo.
- Abrir la extensión Web Server for Chrome
- Pulsar el botón de CHOOSE FOLDER, y acceder a la carpeta TFG
- Tener habilitadas las opciones de
    - Run in background
    - Accessible on local network
    - Prevent computer from sleeping
    - Automatically show index.html
    - Enter Port: 8887
- En opciones avanzadas
    - Use https://
- Web Server: STARTED (darle al botón tras cambiar todas las opciones anteriores)
- Abrir un navegador con la dirección de http://127.0.0.1:8887/.
- Aceptar acceder a la web (aunque aparezca un mensaje que diga que existe riesgo, ya que el hecho de que detecte riesgo se debe a que el certificado y la clave privada no han sido debidamente certificadas)

7.3 Guía de la Aplicación

7.3.1 PlayCanvas

Al entrar en la aplicación, nos encontraremos en la siguiente posición como vemos en la Imagen 17:

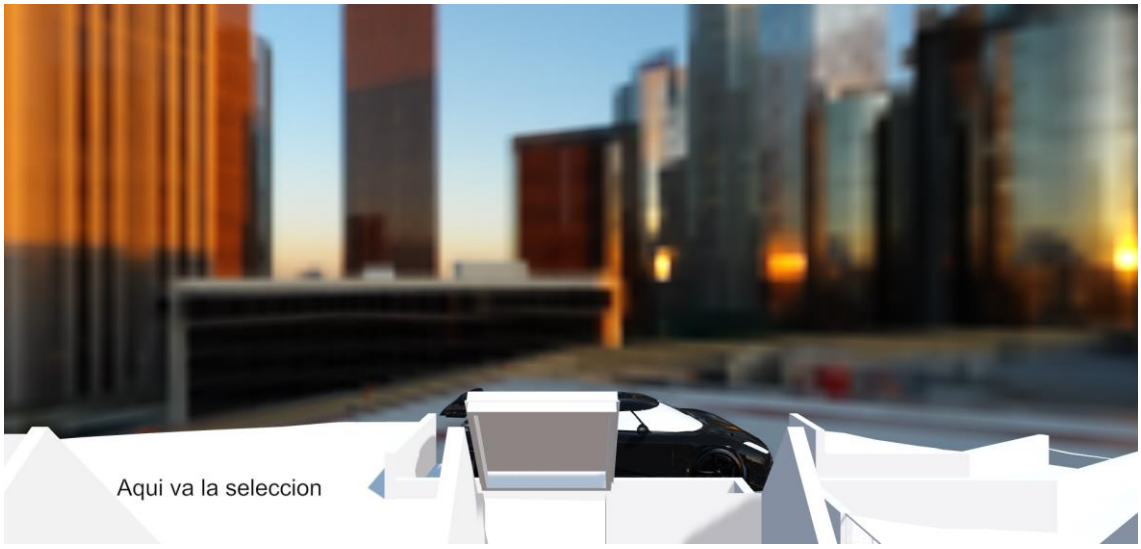

Imagen 17 PlayCanvas Inicio

Acciones que se pueden realizar:

- Moverse con las flechas del teclado. También podemos usar el ratón para mover la cámara. Para mover la cámara con el ratón, hemos de hacer clic en la pantalla, y arrastrar hasta estar satisfechos con la nueva dirección.
- Seleccionar los distintos objetos.
- Al seleccionar los distintos objetos, interactuar con algunos de ellos, como las ventanas y puertas, activando animaciones básicas (rotación de cuerpo) o cambiando colores. Para ello, habría que seguir las instrucciones que aparecerán en el texto por pantalla.

7.3.2 Babylon

Al entrar en la aplicación, nos encontraremos en la siguiente posición como vemos en la Imagen 18.

Acciones que se pueden realizar:

- Moverse con las flechas del teclado. Asimismo, se puede usar el ratón para mover la cámara. Para ello, se ha de hacer clic en la pantalla, y arrastrar hasta estar satisfechos con la nueva dirección.
- Seleccionar los distintos objetos.

- Al seleccionar los distintos objetos, se puede interactuar con todos ellos, cambiando colores (usando la paleta de colores que aparece a la derecha en la pantalla al seleccionar un objeto), moviéndolos (utilizando los botones de la interfaz a nuestra izquierda), hacer clic en el panel que nos indica el nombre, y activar animaciones en alguno de ellos al hacer clic.

Imagen 18 Babylon Inicio

7.3.3 Proyecto final

Al entrar en la aplicación, nos encontraremos en la siguiente posición como vemos en la Imagen 19:

Imagen 19 Aplicación Final Inicio

Acciones que se pueden realizar:

- Moverse con las flechas del teclado. También se puede usar el ratón para mover la cámara. Para ello, se ha de hacer clic en la pantalla, y arrastrar hasta estar satisfechos con la nueva dirección.

- Seleccionar los distintos objetos.

- Al seleccionar los distintos objetos, nos saldrá la descripción de los mismos en un panel a la derecha. Aparte de ello, si hay algún tipo de animación, podemos hacer clic derecho sobre ella, para pararla o volverla a activar.

- Usar los botones de la interfaz de la izquierda para dar las siguientes órdenes:
    - Ocultar: Permite ocultar (y hacer aparecer) todos los botones de la interfaz.
    - Desensamblar: Los modelos preparados para ello se despiezarán.
    - Ensamblar: Los modelos desensamblados se unirán.
    - No animar: parar las animaciones en loop, como las ruedas.
    - Animar: parar o ejecutar las animaciones en loop, aparte de activar y desactivar animaciones en específico.

- El botón de la derecha sirve para mostrar las distintas descripciones de los distintos modelos que aparecen. Para ver una descripción, sólo hay que hacer clic en el objeto del que queramos una descripción.

- Se puede acceder a la inmersión virtual haciendo clic en el botón de las gafas en la parte de abajo a la derecha de la web. Recuerda que debes de tener un emulador o dispositivo de realidad virtual para poder interactuar.

- Para moverse, puedes moverte físicamente (usando las gafas en el emulador), o se puede teletransportar usando uno de los controladores, apuntando a la nueva localización (en el suelo), y usando el joystick para indicar la dirección desde la que se quiere mirar (por ejemplo, si se quiere mirar atrás nuestra en una posición, se apunta con el controlador derecho y se mueve el joystick hacia nuestra dirección).

- En la inmersión virtual se puede usar los botones del controlador para activar los efectos de los botones de la interfaz, como se ve en la Imagen 20
    - A botón ejecuta la animación de desensamblado

- B botón ejecuta la animación de ensamblado
- X botón ejecuta las animaciones que no son de ensamblado y desensamblado
- Y botón para todas las animaciones.

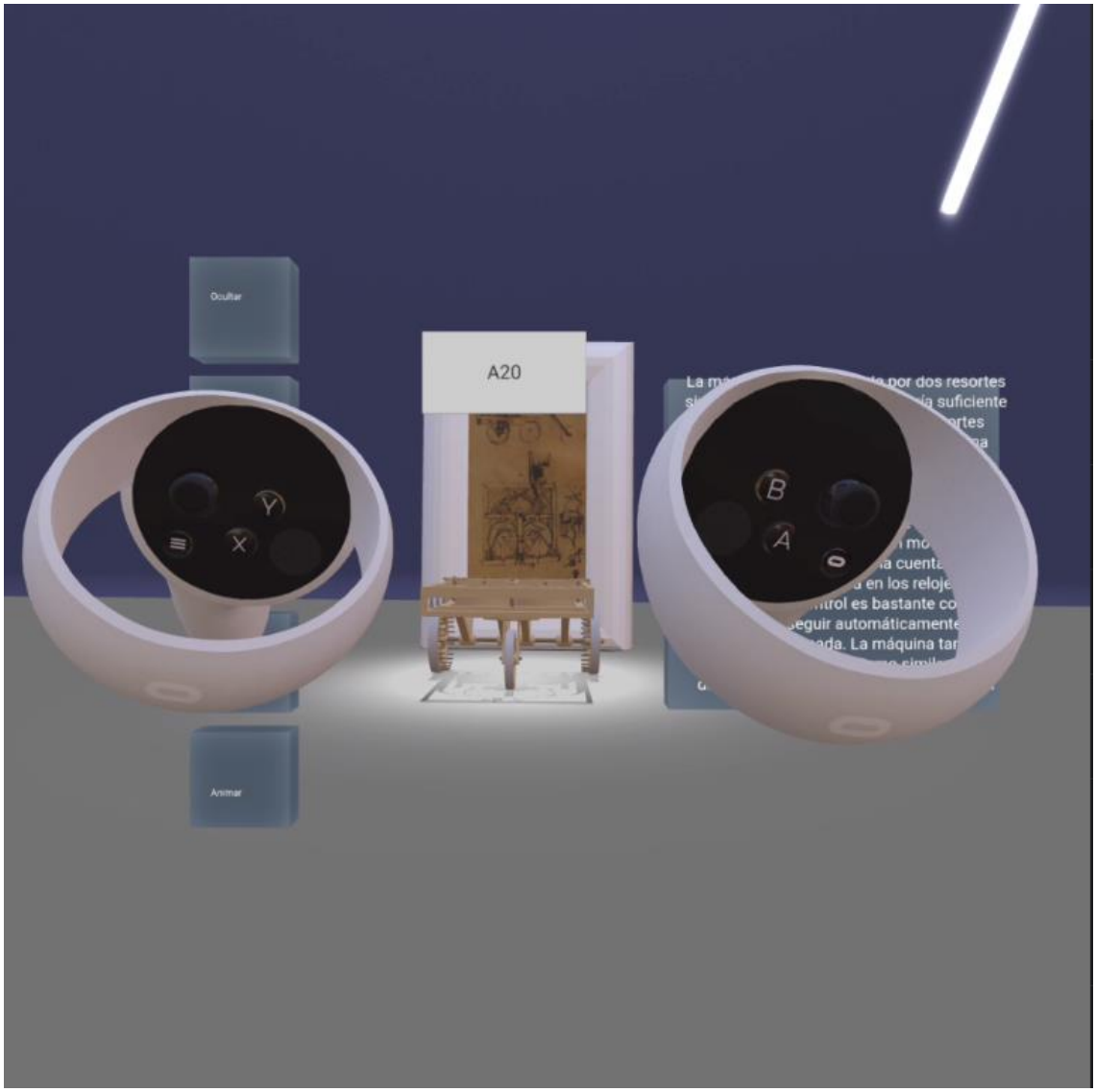

Imagen 20 Controladores Oculus Quest 2