



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA
TELECOMUNICACIÓN

Trabajo Fin de Grado

VISUALIZACIÓN DE DATOS EN REALIDAD
VIRTUAL

Autor : Adrián Pizarro Serrano

Tutor : Dr. Jesús M. González Barahona

Curso Académico 2019/2020

Trabajo Fin de Grado

Visualización de Datos en Realidad Virtual

Autor : Adrián Pizarro Serrano

Tutor : Dr. Jesús M. González Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2020, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2020

*Dedicado a
mi familia*

Agradecimientos

Quisiera dar las gracias, en primer lugar, a mi familia tanto a mis padres como mi hermano. Sin su apoyo diario, su constante dedicación y esfuerzo puesto en mi educación y bienestar nunca habría llegado donde estoy. Para mí son una parte fundamental de mi vida y gran parte de lo que soy. Estoy convencido de que sin su paciencia e insistencia nunca habría terminado el proyecto ni la carrera. Gracias por todo.

Otra gran protagonista y responsable de haber terminado este proyecto, es gracias a ti, Elena. Sin tu apoyo y sacrificio no habría podido terminar esta labor. Como bien sabes eres otra gran parte de mi vida y de lo que soy y seré en un futuro.

A mis amigos de toda la vida y a mis compañeros de carrera. Gracias por todas las buenas experiencias y recuerdos que siempre me llevaré y que seguiremos compartiendo. Por supuesto, sin vuestro apoyo y compañerismo hubiera sido un camino totalmente distinto y mucho más difícil de recorrer.

Gracias a mi tutor Jesús M. González Barahona al cual llegué después de renunciar a otro proyecto y me acogió con los brazos abiertos. Siempre teniendo disponibilidad y prestándome ayuda cuando lo necesitaba.

Por último quisiera hacer una mención especial a la comunidad de desarrollo tan amplia que existe y gracias a la cual se comparte conocimiento diariamente y se consiguen avances increíbles.

Resumen

Este proyecto consiste en el desarrollo de un módulo capaz de visualizar datos en tres dimensiones mediante realidad virtual. Este sistema puede ser utilizado en cualquier navegador y plataforma, así como visualizado con distintos dispositivos de realidad virtual.

Se utilizan distintas tecnologías en el *frontend* que permiten visualizaciones clásicas así como otras no tan clásicas en tres dimensiones. Debido a la creciente necesidad de analizar grandes cantidades de información sumado a la inexistencia de un módulo para representar datos en 3D, hacen de este trabajo un proyecto apasionante.

Este módulo es completamente configurable para una mejor experiencia de usuario. Se provee de un API para modificar las distintas visualizaciones e incluir distintos complementos para una mejor representación y comprensión de los datos.

Se ha utilizado el *framework* de A-Frame, el cual, permite construir experiencias de realidad virtual. Gracias al extenso API, entidades y componentes que provee, se ha podido desarrollar con éxito este módulo. Como se explica en esta memoria, dicho *framework* se apoya fundamentalmente en la librería ThreeJS cuyo motor gráfico es WebGL. Por tanto, básicamente se utiliza HTML5 y JavaScript para generar todas las visualizaciones necesarias.

El resultado final ha consistido en publicar este módulo en la comunidad, y que así, el resto de desarrolladores pueda utilizarlo y contribuir a su mejora continua. De hecho, se han atendido varias peticiones de distintos usuarios que ya utilizan este módulo en sus distintos proyectos.

Summary

This project consists in the development of a library able of visualizing data in three dimensions through virtual reality. This system can be used in any browser and platform, as well as visualized with different virtual reality devices.

Different technologies are used in the frontend that allow classic visualizations as well as others not so classic in three dimensions. Due to the growing need to analyze large amounts of information added to the absence of a module to represent 3D data, they make this work an exciting project.

This module is fully configurable for a better user experience. An API is provided in order to modify the different visualizations and include complements for a better representation and understanding of the data.

The A-Frame framework has been used, which allows to build virtual reality experiences. Thanks to the extensive API, entities and components it provides, this module has been successfully developed. As explained in this report, this framework relies primarily on the ThreeJS library whose graphic engine is WebGL. Therefore, HTML5 and JavaScript are basically used to generate all the necessary visualizations.

The final result has been to publish this module in the community, and so, the rest of developers can use it and contribute to its continuous improvement. In fact, several requests from different users that already use this module in their different projects have been committed.

Acrónimos

API: Application Programming Interface

HTML: HyperText Markup Language

HTTP: HyperText Transfer Protocol

JSON: JavaScript Object Notation

XML: eXtensible Markup Language

JS: JavaScript

3D: Tres Dimensiones

VR: Virtual Reality

DOM: Document Object Model

AJAX: Asynchronous JavaScript And XML

ES6: ECMAScript 6

IDE: Integrated Development Environment

SVN: Subversion

RAM: Random Access Memory

GPU: Graphics Processing Unit

FPS: Frame Per Second

Índice general

1. Introducción	1
1.1. Contexto	1
1.2. Motivación	2
1.3. Objetivo general	3
1.4. Objetivos específicos	4
1.5. Estructura de la memoria	5
1.6. Disponibilidad del Software	6
2. Estado del arte	9
2.1. HTML5	9
2.2. JavaScript	10
2.3. JSON	11
2.4. NodeJS	11
2.5. npm	12
2.6. Budo	13
2.7. HighlightJs	13
2.8. Webpack	14
2.9. Git	15
2.10. Realidad Virtual	16
2.11. OpenGL	16
2.12. WebGL	18
2.13. WebVR	18
2.14. Three.js	19
2.15. A-Frame	21

3. Diseño e implementación	25
3.1. Metodología SCRUM	25
3.2. Iteración 0. Estudio previo.	27
3.3. Iteración 1. Primer gráfico.	29
3.4. Iteración 2. Visualizaciones y separación de la capa de datos	39
3.5. Iteración 3. Refactorización y estandarización del módulo.	47
3.6. Iteración 4. API y Resolución de peticiones de la comunidad.	52
3.7. Iteración 5. Leyenda, fuentes de datos y dispositivos VR.	59
4. Resultados	69
4.1. Arquitectura general	69
4.2. Funcionamiento de la Librería	70
4.2.1. Gráfico de burbujas	71
4.2.2. Gráfico de barras	73
4.2.3. Gráfico de barras cilíndricas	75
4.2.4. Gráfico de tarta	77
4.2.5. Gráfico de doughnut	78
4.2.6. Totem	79
4.2.7. Parámetros de los ejes	80
4.2.8. Pop up	81
4.2.9. Leyenda	82
4.2.10. Dashboards	83
4.2.11. LLamadas Ajax	84
4.2.12. API	85
5. Conclusiones	87
5.1. Consecución de objetivos	87
5.2. Aplicación de lo aprendido	88
5.3. Lecciones aprendidas	89
5.4. Trabajos futuros	89
Bibliografía	91

Índice de figuras

2.1. Ejemplo JSON básico	11
2.2. Arquitectura NodeJs	12
2.3. Ejemplo HighlightJs	14
2.4. Qué es Webpack	15
2.5. Dispositivo de Realidad Virtual	16
2.6. Ejemplo de Shader OpenGL	17
2.7. Escena Three.js	20
2.8. Ejemplo de cámara. Forma geométrica ” <i>frustum</i> ”	21
2.9. Ejemplo código HTML de A-Frame	22
2.10. Ejemplo del inspector de A-Frame	23
2.11. Ejemplo escena de A-Frame	24
3.1. Primera escena con A-Frame	28
3.2. Primera arquitectura del proyecto	29
3.3. Iteración 1: Resultado	38
3.4. Iteración 2: Resultado Gráfico de Barras	46
3.5. Iteración 2: Resultado Gráfico de burbujas	46
3.6. Iteración 3: Inicializar proyecto con Angle	51
3.7. Iteración 3: Nueva Arquitectura	51
3.8. Iteración 4: Resultado	58
3.9. Iteración 5: Pop-up	61
3.10. Iteración 5: Leyenda	63
4.1. Resultados: Arquitectura Final	70

4.2. Resultados: Gráfico de burbujas	71
4.3. Resultados: Gráfico de barras	73
4.4. Resultados: Gráfico de barras cilíndricas	75
4.5. Resultados: Gráfico de tarta	77
4.6. Resultados: Gráfico de doughnut	78
4.7. Resultados: Totem	79
4.8. Resultados: Ejemplo de ejes parametrizables	80
4.9. Resultados: Gráfico con Pop-up	81
4.10. Resultados: Gráfico con leyenda	82
4.11. Resultados: Dashboard	83

Capítulo 1

Introducción

Este proyecto trata sobre la visualización de datos mediante realidad virtual en cualquier navegador web y dispositivo VR. Se utiliza como base el *framework* A-Frame¹. Esta biblioteca nace a su vez a partir de la librería Three.js² utilizada para mostrar gráficos animados en 3D.

El objetivo principal es crear un módulo capaz de visualizar datos con distintos tipos de gráficos, fácilmente escalable, mantenible y sencilla de utilizar. Además, como objetivos más específicos, podrá ser usada en cualquier navegador y dispositivo de realidad virtual.

El hecho de que apenas existan librerías para representación de datos con realidad virtual hacen de este proyecto un trabajo estimulante y motivador.

A continuación se describe el contexto, motivación, los objetivos, la estructura de esta memoria y la disponibilidad del software.

1.1. Contexto

Hoy en día, prácticamente todo negocio se lleva a cabo o se promociona a través de Internet. La comunicación y la rentabilidad empresarial han mejorado enormemente gracias a la llegada de este fenómeno y de su mano surge el desarrollo web, el cual es uno de los mayores creadores de nuevos puestos de trabajo.

Para todo este paradigma surgen, casi a diario, nuevos *frameworks* para conseguir una mayor eficiencia y estandarización a la hora de crear páginas webs.

¹<https://aframe.io/>

²<https://threejs.org/>

Cabe destacar que, a nivel empresarial, es necesario un estudio exhaustivo así como una mejor visualización, tratamiento y explotación de los datos. Para ello han surgido infinidad de librerías a nuestra disposición tales como D3.js³, amcharts⁴, highcharts⁵, etc.

Por otro lado, con el nacimiento de Canvas 3D en 2006 y su posterior evolución en WebGL se abre la puerta a la implementación de visualizaciones 3D en páginas webs. Como veremos más adelante, WebGL nos proporciona un API en javascript para renderizar visualizaciones 2D y 3D sobre HTML.

Paralelamente, nace el motor gráfico Unity el cual también ha ayudado a impulsar la fama y utilización de componentes 3D en múltiples plataformas. Ha sentado varias bases y conceptos reutilizados en muchas otras plataformas tales como el concepto de escena, cámaras, materiales, luces, sombras, formas geométricas y parámetros para cada componente.

Más tarde, y concerniente a este proyecto, nace Three.js como librería javascript basada en WebGL para crear y mostrar visualizaciones 3D. Esta librería ya nos proporciona todas las especificaciones necesarias para trabajar con escenas, efectos, animaciones, cámaras, etc. Para un uso más sencillo y estándar de esta librería y para su adaptación a la realidad Virtual nace A-Frame en 2015. Gracias a esta última librería podemos utilizar todas nuestras escenas 3D y representarlas en lo que conocemos como WebVR, la cual es una interfaz que nos provee del soporte necesario para visualizar todo este contenido HTML 3D en un aparato de realidad virtual (Oculus Rift, HTC Vive y Google Cardboard) o un navegador.

Finalmente me gustaría hacer una mención especial a toda la comunidad que hay detrás de todas estas plataformas. Sin su aportación diaria, intercambio de conocimiento y ayuda nada de esto sería posible.

La unión de estos dos mundos, representación de datos y la llegada de visualizaciones 3D en navegadores web, nos lleva a nuestro siguiente apartado la motivación.

1.2. Motivación

Muchas veces se escucha la frase atribuida al filósofo inglés Francis Bacon “La información es poder”, la cual es cierta, pero sin una correcta interpretación de la misma podemos caer en

³<https://d3js.org/>

⁴<https://www.amcharts.com/>

⁵<https://www.highcharts.com/>

el caso de la desinformación. Como se menciona en el apartado anterior la visualización y explotación de datos es un campo en auge del cual se requiere mucha ingeniería para poder sacarle partido. Es un campo apasionante, lleno de retos, con muchas salidas laborales y en constante evolución.

Hasta ahora, tenemos infinidad de librerías para representar datos en 2D pero ¿Qué ocurre con la llegada del 3D a nuestros navegadores? ¿Existen librerías para visualizar datos utilizando realidad virtual o realidad aumentada? Lo cierto es que estas preguntas nos llevan a una de las motivaciones principales de este proyecto. Este mundo es relativamente nuevo y hasta ahora se ha apostado más por realizar visualizaciones complejas en 3D o videojuegos. Por lo que la representación de datos en 3D es un nicho aun por explotar.

Este último punto ha sido muy motivador para realizar una de las primeras librerías capaz de visualizar datos en 3D tanto en un navegador como en dispositivos de realidad virtual. Además, la intención de este proyecto es que sea escalable, mantenible y el resto de la comunidad pueda contribuir a su uso y desarrollo.

Respecto a la motivación personal podemos decir que siempre he querido publicar un módulo en el que la comunidad pueda participar y contribuir. Ha sido muy gratificante ver como distintos usuarios han incluido esta biblioteca en sus proyectos así como realizado peticiones de mejora e inclusión de nuevos gráficos. Por otra parte, las tecnologías que se han utilizado en este proyecto son pioneras en el *frontend* sumado al uso de la realidad virtual como medio para reproducir distintas escenas hacen de este proyecto un trabajo apasionante.

Es así como gracias a todos estos factores se ha podido finalizar este proyecto y escritura de esta memoria.

1.3. Objetivo general

Este proyecto consiste en crear un sistema para la visualización de datos en 3D compatible con cualquier dispositivo que tenga navegador web. El cual además puede ser usado en dispositivos de realidad virtual.

1.4. Objetivos específicos

En este apartado se realiza una breve descripción de los objetivos específicos.

1. Este módulo se debe poder utilizar en cualquier navegador sin necesidad de instalar ningún plugin. Además, se podrá visualizar en distintos dispositivos de realidad virtual.
2. Debe proporcionar varios tipos de visualizaciones tales como gráfico de burbujas, de tarta, de barras, de cilindros o tarta en forma de rosquilla.
3. Se podrá colocar una o varias visualizaciones en cualquier parte del espacio tridimensional también conocido como escena. Además, mediante la escucha de eventos podrán ser actualizados todos ellos de manera simultánea y dinámica.
4. Las visualizaciones podrán ser configurables por el usuario. Por ejemplo, se podrá modificar la posición, color, longitud y separación entre las marcas de medición de los ejes. Permitir que los ejes puedan ser negativos o estén en forma de rejilla para dos o más dimensiones. Poder configurar tamaño y color de la leyenda.
5. El sistema debe ser capaz de añadir una leyenda a un gráfico y/o un pop-up para una mejor interpretación de los datos. Con ello se persigue obtener más información y facilitar la lectura de datos.
6. La biblioteca debe proporcionar una herramienta para filtrar o seleccionar distintas fuentes de datos. Además las visualizaciones se podrán refrescar dinámicamente.
7. Debe ser capaz de interpretar ficheros o datos con formato JSON (JavaScript Object Notation) y lanzar una excepción si no fuera capaz de leer el archivo correctamente.
8. El rendimiento es un factor clave que se desea cuidar en este proyecto. Por ello, esta librería debe mostrar un buen comportamiento cargando grandes volúmenes de datos. Se debe comprobar que es capaz de mostrar visualizaciones con una gran cantidad de datos.
9. La librería se debe distribuir y poner al servicio de la comunidad. Se debe dar fácil acceso y ser subida a un repositorio de código donde se puedan recibir sugerencias para implementar y resolver posibles incidencias.

10. Debe ser fácilmente escalable, es decir, deberá estar desarrollada de tal manera que sea sencillo implementar nuevos desarrollos o modificaciones en el código.
11. Un objetivo importante es que el código deberá ser mantenible tanto por su autor como por la comunidad. Se debe publicar dicho software en una plataforma donde, como se menciona anteriormente, se pueda contribuir a resolver incidencias y mantener actualizado de manera sencilla las versiones de los framework y librerías que utilizará el proyecto.
12. Aprender el framework de A-Frame, el cual viene dado desde el principio como recomendación del tutor. Se debe estudiar en profundidad su documentación y estándares para un uso óptimo de la misma. Esta librería de realidad virtual está basada en Three.js por lo que también se deberá realizar un estudio de la misma.
13. Aprender Webpack para empaquetar y distribuir nuestra librería.
14. Crear el proyecto con Angle. Una herramienta hecha por los creadores de A-Frame para comenzar nuestro componente con un arquetipo que sigue los estándares de A-Frame.
15. Crear una web con ejemplos de uso de la librería.
16. Una vez creado el proyecto siguiendo los estándares y después de realizar los primeros avances se deberá publicar nuestro proyecto en el registry de A-Frame donde aparecen todos los componentes de la comunidad de A-Frame. Esto proporcionará visibilidad al proyecto pudiendo salir en el blog oficial llamado Week of A-Frame
17. Se deberá cuidar y refactorizar el código para una lectura sencilla del mismo siguiendo distintas directrices que podemos encontrar en el libro Clean Code [7].

1.5. Estructura de la memoria

En esta sección se describe la estructura de la memoria para una mejor comprensión de la misma:

- En el primer capítulo se hace una breve introducción al proyecto. Describiremos el marco y contexto del trabajo para un mejor aterrizaje del lector. Acto seguido, se habla de la

motivación del proyecto tanto a nivel personal como profesional. Después se describe el objetivo principal que perseguimos con este trabajo para continuar con los específicos, donde se profundizará y ampliarán los conceptos tratados en el principal. Por último se describe brevemente la estructura de la memoria.

- A continuación se presenta el estado del arte. Aquí haremos un repaso así como una breve descripción de todas las tecnologías que se han utilizado.
- En el cuarto capítulo se desarrolla el diseño e implementación. Se entrará al detalle de como está construida la aplicación, su arquitectura y sus distintos componentes.
- En este quinto capítulo, Resultados, se realiza un análisis del funcionamiento y rendimiento de esta aplicación. Además se podrá ver un amplio abanico de casos de uso donde se verán los resultados y capacidades del proyecto.
- Por último, tendremos las conclusiones. Aquí se presentará un resumen de los distintos conceptos aprendidos y aplicados. Podremos ver si se han alcanzado los objetivos propuestos. Además se expondrán líneas futuras de investigación y mejora de esta aplicación.

1.6. Disponibilidad del Software

Para poder consultar y mantener el proyecto se ha creado un repositorio de software en el portal de GitHub. Este proyecto de software libre cuenta con una licencia de Apache 2.0. Todos los aportes a lo largo del tiempo así como las incidencias se pueden consultar en:

`https://github.com/adrixp/aframe-charts-component`

Además esta librería cuenta con una página web donde podemos ver las *demos* y casos de uso. También se tiene un enlace directo al API para ver todos los parámetros disponibles, y por cada visualización el código utilizado. Se puede acceder a través del siguiente enlace:

`https://adrixp.github.io/aframe-charts-component/`

Por último, se ha publicado la librería en el sistema de gestión de paquetes (npm). Para facilitar su distribución y poder ser importado por un servidor nodejs en cualquier momento:

`https://www.npmjs.com/package/aframe-charts-component/`

Capítulo 2

Estado del arte

En este capítulo se describirán las tecnologías que se utilizan en este trabajo. Se hará un repaso tanto de los lenguajes básicos de programación web utilizados, el servidor web y las herramientas para la paquetización y distribución de la librería.

También se mencionarán las librerías de realidad virtual y los motores en los que estas se apoyan, los cuales facilitan enormemente el trabajo de desarrollo.

2.1. HTML5



En 1993 se lanza el lenguaje de marcado HTML como herramienta principal para construir páginas web. Define una estructura básica a partir de etiquetas que son interpretadas por los navegadores. El encargado de su mantenimiento y estandarización es el Consorcio de la *World Wide Web* (W3C).

Mas adelante, en 2014, se lanza la quinta versión de HTML comúnmente conocida como HTML5 [10]. En esta nueva versión se incluyen nuevos elementos como la cabecera `<header>`, el pie `<footer>` y la sección `<section>` entre otros. Además se añaden elementos gráficos como Canvas, una superficie bidimensional para dibujar mediante Javascript, y svg para gráficos vectoriales.

Por otro lado también se incluyen nuevos atributos y etiquetas para elementos multimedia como audio y video. Además, cabe destacar que el nuevo API nos provee de herramientas para la geolocalización, *drag and drop*, almacenamiento local, caché, etc.

Finalmente subrayar que se proporciona una mayor optimización en el uso del *hardware* así como la inclusión del estándar CSS3 el cual ofrece una nueva gran variedad de opciones para hacer diseños más sofisticados.

2.2. JavaScript



Para dotar de funcionalidad y dinamismo a las páginas web, que como mencionamos anteriormente utilizan el lenguaje de marcado HTML, tenemos JavaScript [5]. En 1995 nace este lenguaje orientado a objetos, el cual tiene tipado dinámico y utiliza prototipos en vez de clases para el uso de herencia.

El estándar que sigue este lenguaje actualmente es ECMAScript. Todos los navegadores incluyen una implementación de dicho estándar. En este proyecto utilizamos la última versión publicada en 2015 ECMAScript 6 (ES6).

Este lenguaje puede ser utilizado tanto en el lado del cliente como en el lado del servidor como se puede ver más adelante. En el lado del cliente se nos ofrece infinidad de posibilidades para manipular el DOM.

Por último cabe destacar la infinidad de *frameworks* que surgen día a día en la actualidad. Un *framework* define el diseño completo de una aplicación web y nos facilita el uso de patrones de diseño [6] como, por ejemplo, el Modelo Vista Controlador (MVC). Hoy en día tenemos a Angular.js¹, React² y Vue.js³ como los *frameworks* más populares

¹<https://angularjs.org/>

²<https://reactjs.org/>

³<https://vuejs.org/>

2.3. JSON

Como alternativa a XML nace JSON. Es un formato de texto para la distribución de datos o puede ser usado como un objeto Javascript que tiene distinta información, como vemos en la Figura 2.1, debido a su gran versatilidad. Es un formato muy sencillo de leer y de *parsear*, es decir, de obtener su información.

```
{
  "colors": [
    {
      "color": "black",
      "category": "hue",
      "type": "primary",
      "code": {
        "rgba": [255,255,255,1],
        "hex": "#000"
      }
    },
    {
      "color": "white",
      "category": "value",
      "code": {
        "rgba": [0,0,0,1],
        "hex": "#FFF"
      }
    }
  ]
}
```

Figura 2.1: Ejemplo JSON básico

Los tipos de datos que soporta JSON son: números, texto, *booleanos*, *arrays* y objetos Javascript formados por tuplas clave-valor. Y en comparación con XML tiene un mayor soporte y herramientas para su uso. En este proyecto se va a utilizar JSON como formato de entrada de datos. Además, se puede ver en los siguientes puntos que para resolver las dependencias de nuestro proyecto y dar una descripción general del mismo se utiliza un fichero de datos llamado *package.json*.

2.4. NodeJS



NodeJs [9] es un servidor web desarrollado en C++ en el año 2009. Esta plataforma nos

permite desarrollar en el lado del servidor con el lenguaje Javascript y el estándar ES6.

Es un servidor cuyas características principales son que es *monohilo* y asíncrono. Como vemos en la Figura 2.2, tenemos un único hilo donde por cada petición se delega el trabajo y se da respuesta de manera asíncrona, es decir, no es bloqueante y se ejecutará de manera concurrente recibiendo dichas respuestas mediante funciones *callback*.

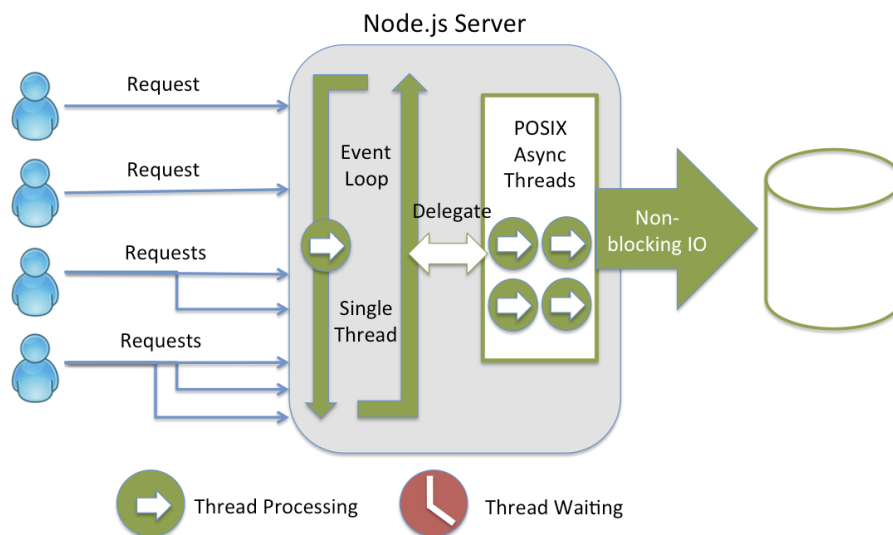


Figura 2.2: Arquitectura NodeJs

Como se menciona anteriormente, el archivo `package.json` es el corazón del sistema Node.js. Es el archivo de manifiesto de cualquier proyecto con Node.js y contiene los metadatos del proyecto. Además, en la siguiente sección, se podrá ver como el gestor de paquetes de node (NPM) utiliza este fichero para obtener información y descargar las dependencias.

Cabe destacar que NodeJs se utiliza como servidor web en infinidad de compañías y tiene una fuerte comunidad detrás apoyando su desarrollo.

2.5. npm



Como gestor de paquetes, módulos y dependencias de NodeJs nace npm en 2014. Es instalado automáticamente con cualquier versión de NodeJs a partir de la 0.6.3. Las instrucciones se

ejecutan por línea de comandos y cualquier usuario puede consumir los paquetes subidos a esta plataforma de manera versionada.

Para este proyecto se ha creado un repositorio de npm para poder distribuir nuestra propia librería como dependencia y poder ser usada de manera ágil⁴. Es necesario registrarse en la plataforma para poder publicar.

Gracias a esta herramienta se facilita el poder crear, compartir, reutilizar y contribuir a los distintos módulos desarrollados por la comunidad.

2.6. Budo

Budo es una herramienta que nos permite desarrollar en el servidor con integración *Live-Reload*. Esto nos permite que cualquier cambio que realicemos en el servidor se vea reflejado instantáneamente sin necesidad de recargar toda la instancia. Para ello, Budo utiliza otro paquete npm llamado *broserfy* que nos permite exportar módulos y dependencias en el cliente.

En este proyecto ha sido muy útil esta herramienta ya que ha facilitado y agilizado el desarrollo. Este paquete está disponible en el repositorio de npm

2.7. HighlightJs

Esta librería es utilizada para representar código de manera legible en una página web. Nos permite mostrar cualquier lenguaje de programación resaltando las palabras clave como si estuviéramos utilizando un IDE (Entorno de desarrollo) o un editor de texto que tuviera esta funcionalidad.

Como parámetros de entrada, esta librería, acepta nombre del lenguaje de programación, un alias o directamente código. Además cuenta con la posibilidad de resaltar texto a pesar de que haya errores de sintaxis en el código. Cuenta con detección automática del lenguaje de programación e incluso se tiene la posibilidad de incluir múltiples lenguajes en una misma instancia.

Los usuarios cuentan con la posibilidad de añadir nuevos lenguajes de programación según la necesidad y proveer soporte de los mismos. Actualmente cuenta con soporte para 185 len-

⁴<https://www.npmjs.com/package/aframe-charts-component>

```
<a-entity position="1 15 10" charts="type: totem; entity_id_list: barId;  
  dataPoints_list: ../data/dataPositive.json,../data/data.json,../data/dataSmall.json">  
</a-entity>
```

Figura 2.3: Ejemplo HighlightJs

guajes de programación y 85 estilos.

En este proyecto, HighlightJs ha sido utilizado para elaborar el manual de usuario⁵ facilitando así la lectura de código y pudiendo reutilizar este código de manera más clara como vemos en la Figura 2.3

2.8. Webpack



Webpack es un empaquetador de módulos estáticos. En la programación modular, los desarrolladores particionan su código en pequeños bloques que cumplen una funcionalidad concreta, y a estos se le llama módulos. Gracias a ellos, es trivial verificar, corregir y probar un programa.

El problema que nos podemos encontrar es que, si dividimos nuestro programa en muchos módulos, podemos tener problemas de escalabilidad debido a la gran cantidad de paquetes o *scripts* que el navegador necesita descargar. Esto puede provocar, lo que comúnmente llamamos e redes, un *cuello de botella* [11].

Por otro lado podríamos tener un único *script* que contenga todo el código. Pero esto nos puede llevar a problemas de alcance, tamaño, legibilidad y mantenibilidad.

Resolviendo esta problemática nace Webpack. El cual nos permite escribir módulos soportando todos los formatos y además gestionar recursos. Como vemos en la Figura 2.4 es una herramienta que nos permite empaquetar aplicaciones *JavaScript* modernas que contengan módulos, imágenes, fuentes, hojas de estilos.

Por último cabe destacar que Webpack se preocupa por el rendimiento y los tiempos de

⁵<https://adrixp.github.io/aframe-charts-component/examples/bubbleChart/code.html>

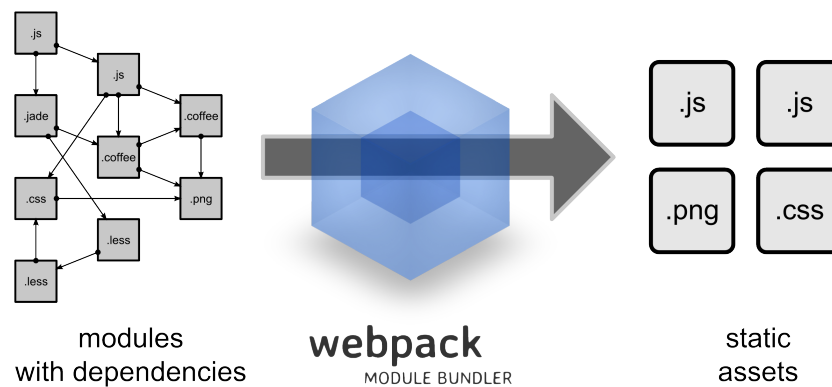


Figura 2.4: Qué es Webpack

carga. Siempre está mejorando o agregando nuevas funciones, como la carga asíncrona, para ofrecer la mejor experiencia posible para el proyecto y los usuarios.

2.9. Git



Como herramienta de control de versiones, en este proyecto se utiliza Git [3]. Este *software* fue diseñado por Linus Torvalds (Creador del *kernel* de *Linux*) en 2005, cuyo propósito era crear un sistema capaz de llevar el registro de cambios en archivos de un ordenador y coordinar el trabajo que realicen varias personas simultáneamente sobre dichos ficheros.

Este software está pensado para ser utilizado de manera distribuida manteniendo un registro tanto en la computadora local como en un servidor. De esta forma cada usuario tiene una copia de los ficheros que valida contra el servidor si ha habido cambios o quisiera registrar alguno.

Gracias a esto, se consigue un gran apoyo al desarrollo mediante la rapidez para gestionar ramas y mezclado de las mismas. De hecho, en la mayoría de proyectos de desarrollo Git ha desbancado a SVN como principal controlador de versiones.



Figura 2.5: Dispositivo de Realidad Virtual

2.10. Realidad Virtual

Entendemos por realidad virtual [2] aquella tecnología que a través de uno o varios dispositivos nos puede generar la sensación de realidad en un escenario artificial creado de manera digital. Este término nace a finales de la década de 1980 y originariamente describe cómo una persona interacciona con un entorno tridimensional (3D) artificial.

En la Figura 2.5 podemos ver un ejemplo de dispositivo de realidad virtual. Por lo general, cuentan con un primer aparato que se coloca en la cabeza a la altura de los ojos para realizar una inmersión en el mundo virtual y abstraerse de la realidad convencional. Además, como vemos en la imagen, estos dispositivos cuentan con unos mandos para moverse o realizar acciones dentro de este mundo virtual.

Desde un inicio, esta tecnología ha estado orientada a la industria de los videojuegos. Pero no ha tardado en expandirse y tener aplicación en el mundo de la medicina, enseñanza, industria y turismo.

2.11. OpenGL



OpenGL [8] es una librería de gráficos de código abierto cuyo lanzamiento inicial fue en el año 1994. Define un API multilenguaje y multiplataforma para producir aplicaciones que rendericen gráficos en 2D y 3D. Esta interfaz cuenta con mas de 250 funciones para dibujar escenas tridimensionales a partir de primitivas más simples bidimensionales.

Sus aplicaciones a día de hoy va desde simuladores de vuelo hasta el desarrollo de videojuegos con una creciente comunidad que impulsa este fenómeno. Además está pensado para conseguir una mayor abstracción sobre la complejidad de interactuar con el hardware de la tarjeta gráfica ofreciendo un API único y uniforme.

En la Figura 2.6 tenemos un ejemplo de como a partir de primitivas simples como vértices podemos ir creando distintas formas como triángulos. Cada uno de estos vértices almacena ciertos atributos como por ejemplo las coordenadas.

Por último la tarjeta gráfica procesa cada uno de estos vértices de manera individual y aplica la transformación necesaria para proyectar estos vértices en el mundo 3D en nuestra pantalla 2D. Una vez transformados lo vértices de entrada se formarán los triángulos, líneas o puntos mencionadas anteriormente como primitivas simples.

Un último paso, introducido recientemente, es el de aplicar el sombreado geométrico o *shader*.

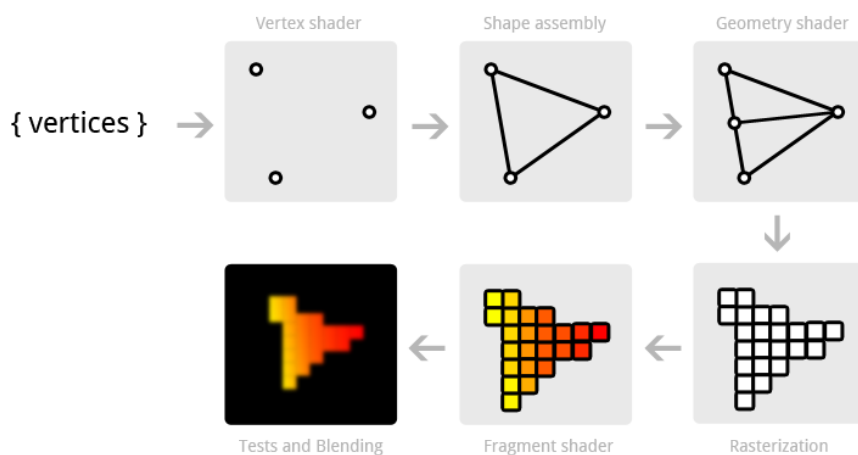


Figura 2.6: Ejemplo de Shader OpenGL

2.12. WebGL



WebGL [4] o *Web Graphics Library* es un estándar que define una API escrito en JavaScript y que implementa OpenGL para la generación de gráficos en 3D en el navegador web. No es necesaria la instalación de un complemento o *plugin* para los navegadores, debido a que actualmente tanto las versiones para móvil como las de escritorio cuentan con soporte OpenGL 2.0.

Esta tecnología nace a raíz del WebVR mencionado en el apartado anterior y es en 2009 cuando se consolida como grupo de trabajo. A raíz de ello en marzo de 2011 se lanza la primera versión oficial.

La primera versión (1.0) de WebGL está basado en OpenGL 2.0 y proporciona una API para gráficos 3D. Esta, utiliza el elemento HTML5 llamado canvas o lienzo al que se accede mediante el Document Object Model (DOM). Para ello se debe definir una región dentro de nuestro HTML donde se representará la escena 3D. Solo es posible crear esta región en los navegadores web que admiten HTML5. Al ser parte del DOM nos permite interactuar de forma dinámica con otros elementos de la página.

WebGL 2.0 está basado en OpenGL 3.0 y garantiza disponibilidad de muchas extensiones opcionales de WebGL 1.0 y presenta nuevas APIs que facilitan el desarrollo.

WebGL está integrado completamente en todos los estándares web del navegador, permitiendo la aceleración de la GPU y el procesamiento de imágenes y efectos. Esta gestión automática de memoria RAM o virtual se proporciona mediante el lenguaje JavaScript.

2.13. WebVR

WebVR es una API experimental escrito en Javascript en 2006. Fue iniciado como prototipo de Canvas3D por Vladimir Vukicevic en Mozilla. Este API proporciona soporte para dispositivos de realidad virtual tales como Oculus Rift, HTC Vive o Google Cardboard, en un navegador de web.

Sus principales objetivos son detectar de manera automática dispositivos de realidad virtual disponibles y obtener sus características principales, como la posición y orientación del dispositivo, para mostrar imágenes con una latencia aceptable.

Esta tecnología tiene una primera versión finalizada en 2017 donde principales compañías como Mozilla, Google y ahora Microsoft dan soporte y mantenimiento.

La API WebVR expone ciertas interfaces que permiten a las aplicaciones web incluir contenido de realidad virtual. Para ello se hace uso de WebGL, descrito en el siguiente apartado, como motor para gráficos 3D. Además permite obtener las configuraciones necesarios para la cámara y los dispositivos controladores (por ejemplo, un mando o el punto de vista).

2.14. Three.js

three.js

Three.js⁶ es una librería escrita en JavaScript para mostrar contenido 3D en páginas web. Nos provee la capacidad de renderizar escenas, modelos, sonido, vídeos, luces, sombras, animaciones, partículas y muchos otros tipos de visualizaciones.

No debemos confundir Three.js con WebGL. WebGL es un sistema de bajo nivel con el que podemos dibujar primitivas simples como puntos, líneas y triángulos. Three.js utiliza este motor a bajo nivel y nos permite abstraernos de su complejidad teniendo ya disponibles ciertas interfaces como escenas, luces, materiales, etc mencionadas anteriormente.

Al igual que WebGL, se utiliza el elemento canvas de HTML5 para comenzar creando una escena. En la Figura 2.7 podemos ver la arquitectura general de como se compone dicha escena. Por un lado tenemos la figura geométrica compuesta por vértices y superficies. Esta figura geométrica puede se le puede añadir un material que está compuesta por una o varias imágenes que a su vez componen lo que conocemos por textura. Si sumamos esta figura geométrica y su material obtenemos lo que denominamos una malla. La malla, las luces y la cámara componen nuestra escena 3D.

⁶<https://threejs.org>

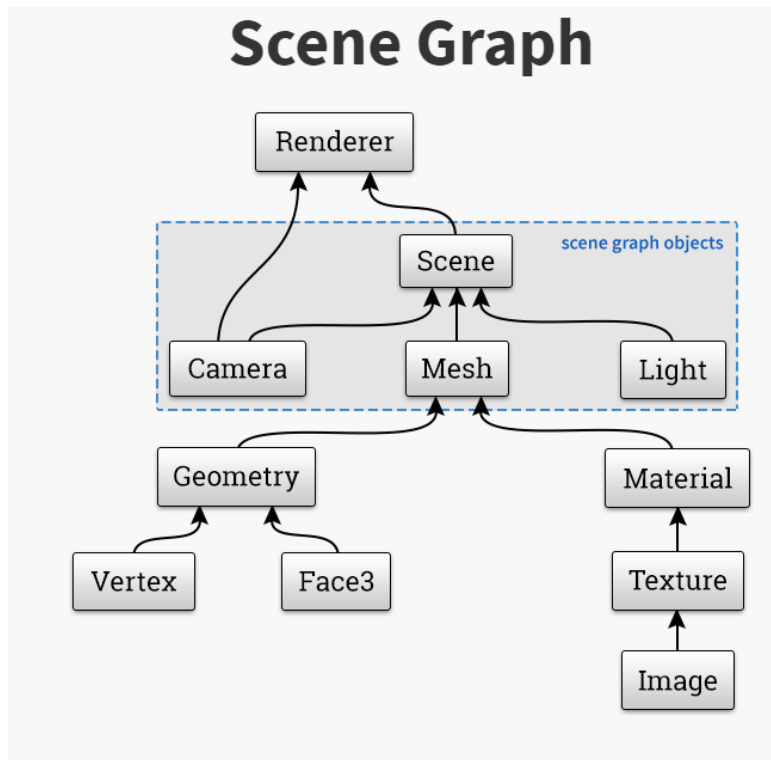


Figura 2.7: Escena Three.js

Por último, de manera transparente para el desarrollador, Three.js llama al renderizador de WebGL para representar nuestra escena que se podrá visualizar en cualquier navegador web.

Para comprender mejor el funcionamiento o para que sirve una cámara en una escena 3D tenemos la Figura 2.8. La cámara es un punto de perspectiva desde donde se visualiza la escena 3D. En Three.js tenemos cuatro parámetros básicos para definir una cámara:

- El "fov" es la abreviatura de *field of view* o campo de visión. Es el ángulo de visión de la cámara y se utilizan grados como medida.
- El segundo parámetro es el "aspect". El cual define el aspecto de visualización del lienzo o el ratio de píxeles. Por ejemplo un canvas de 400 píxeles de ancho por 200 de alto, tendría un "aspect" de dos.
- Por último tenemos los parámetro "far" y "near" que definen la distancia de lo que vamos a poder visualizar en la escena. Si algún objeto quedara fuera de este rango no se visualizará.

Estos cuatro ajustes definen un "*frustum*", es decir, el nombre de una forma 3D en forma de

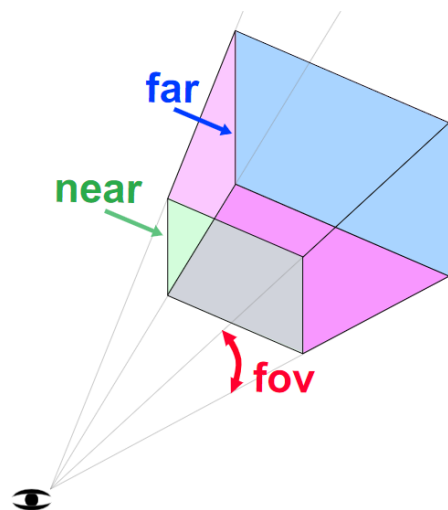
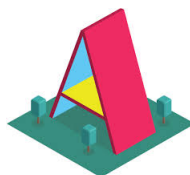


Figura 2.8: Ejemplo de cámara. Forma geométrica "frustum"

pirámide con la punta cortada. Al fin y al cabo esta definición es la que utiliza la cámara para visualizar la escena.

2.15. A-Frame



A-Frame⁷ es un *framework web* que permite construir experiencias de realidad virtual en el navegador. Gracias a su interfaz de entidad-componente podemos visualizar WebVR en cualquier dispositivo de realidad virtual como las HTC Vive, Oculus Rift, Daydream o Samsung GearVR así como en navegadores de escritorio y móvil. Además puede ser incluso usado para realidad aumentada.

A-Frame no es solo un creador de escenas en 3D o un lenguaje de marcado. Su motor está apoyado en el *framework web* entidad-componente, proveyendo una estructura declarativa, extensible y componible de Three.js. A-Frame nos provee un nivel más de abstracción, facilitando incluso más aún el comienzo para desarrollar en 3D que Three.js.

⁷<https://aframe.io/>

```
HTML
<html>
  <head>
    <script src="https://aframe.io/releases/0.9.2/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 2.9: Ejemplo código HTML de A-Frame

Esta librería, estuvo originalmente concebida por Mozilla y ahora mantenida por los creadores de A-Frame y Supermedium⁸ el cual es un navegador web puramente diseñado para WebVR y usado con dispositivos de realidad virtual. Este proyecto de código libre cuenta con una de las mayores comunidades de Realidad Virtual.

Entre sus mayores características encontramos:

- A-Frame provee de un arquetipo 3D, configuración para VR y los controles predeterminados. Sin necesidad de instalación o compilación, únicamente es necesario importar la librería con la etiqueta `<script>` y crear el componente `<a-scene>`.
- Como vemos en la Figura 2.9, se utiliza un HTML sencillo de entender y reutilizable. Es accesible para todo tipo de público, desde desarrolladores a educadores, artistas, diseñadores y niños.
- Utiliza una arquitectura basada en entidad-componente. Es decir nos provee de una interfaz para reutilizar, declarar y acceder a objetos diseñados y optimizados de Three.js. Además, nos permite el acceso a cualquier elemento del DOM e incluso al motor de bajo nivel WebGL.
- Es una librería VR diseñada para ser multidispositivo y multiplataforma. Como se ha mencionado anteriormente es compatible con Vive, Rift, Windows Mixed Reality, Daydream, GearVR y Cardboard con soporte para todos los controladores respectivos.

⁸<https://supermedium.com/>

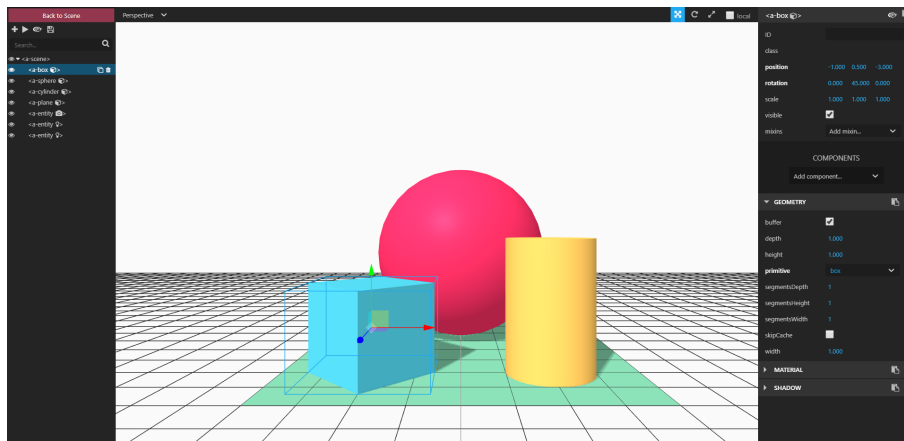


Figura 2.10: Ejemplo del inspector de A-Frame

- El rendimiento es otro punto fuerte de esta librería. A-Frame está optimizada desde cero para WebVR. Los componentes no manipulan el motor de diseño 3D del navegador y las actualizaciones se realizan en memoria llegando a alcanzar los 90 *fps* en escenas con infinidad de elementos.
- La librería de A-Frame nos proporciona un poderoso elemento diseñado para ayudar al desarrollo. Este componente es el inspector de elementos que vemos en la Figura 2.10 . Este inspector recuerda a la manera de crear escenas en otros motores gráficos como el de Unity3D⁹. Pero la principal característica es que, gracias a él, podemos realizar cambios en la escena en vivo. Pudiendo añadir y borrar elementos e incluso actualizar los atributos de cada componente.
- Los componenetes principales que nos proporciona A-Frame son geometrías, materiales, luces, animaciones, modelos, difusores de rayos, sombras, audio posicional, texto y soporte para la mayoría de auriculares. La comunidad además ha aportado otros como medio ambiente , el estado , los sistemas de partículas , la física , los océanos y la realidad aumentada.
- Es una librería de probada escalabilidad llegando a ser utilizada por grandes empresas como Google, Disney, Samsung, Toyota, Ford, Chevrolet, CERN, NPR. Compañías como Google, Microsoft, Oculus y Samsung han realizado contribuciones para mejorar A-Frame.

⁹<https://unity.com/es>



Figura 2.11: Ejemplo escena de A-Frame

En esta potente librería de VR se basa este proyecto fundamentalmente.

Capítulo 3

Diseño e implementación

En este apartado se muestra el diseño e implementación de este proyecto. Se comienza describiendo la metodología utilizada y las distintas etapas por las que ha pasado el proyecto. Más adelante profundizaremos en cada una de dichas etapas, describiendo el desarrollo que se ha realizado de manera incremental para este trabajo.

3.1. Metodología SCRUM

La metodología que se ha seguido en este proyecto es una simplificación de la denominada metodología SCRUM [1]. Este procedimiento se caracteriza principalmente por ser incremental y estar enfocado al desarrollo de software ágil. A diferencia de otros métodos donde se desarrolla de manera secuencial o en cascada (requisitos, diseño, implementación, verificación y mantenimiento), la metodología SCRUM permite solapar estas fases e incluso realizarlas mediante incrementos más pequeños.

El objetivo de SCRUM es aplicar de manera regular buenas prácticas para trabajar en equipo y obtener el mejor resultado y calidad posible para el proyecto. Para esto se definen un conjunto de prácticas y roles como el de SCRUM *máster* encargado del proceso asegurando su correcto uso y guiado por las distintas etapas, el *product owner* el cual es el responsable de mantener el producto y asegurar la continuidad por las líneas de trabajo establecidas. Por último tenemos al equipo de desarrollo que debe ser multidisciplinar y encargado de ir cumpliendo las distintas iteraciones y tareas para sacar adelante el proyecto.

Como se ha mencionado previamente, en este proyecto se ha seguido una simplificación de

la metodología SCRUM. Se han mantenido reuniones periódicas donde el tutor desempeñaba el papel de cliente y el autor el de desarrollador.

A continuación se describe brevemente cada una de las etapas o *sprints* por las que ha pasado este proyecto:

- **Iteración 0:** Esta es la fase previa donde se realizará un estudio de los requisitos del proyecto y las tecnologías a utilizar. La idea principal es familiarizarse con la librería A-Frame utilizando la documentación oficial y practicar mediante ejemplos básicos.
- **Iteración 1:** En esta primera etapa se tiene como objetivo crear las primeras escenas con gráficos en 3D. Para ello crearemos un repositorio de código inicial donde se va a usar nodeJS como servidor y Webpack como empaquetador de nuestra librería. Como objetivo final, se procederá a crear un primer gráfico para nuestra librería con un *dataset* de ejemplo incrustado en el propio código JavaScript.
- **Iteración 2:** El segundo paso será empleado para enriquecer nuestra librería para que soporte un mayor número de gráficas. Se empezará a proporcionar un API para el uso de la librería. Además se procederá a separar la capa de datos de la aplicación para una mejor generalización.
- **Iteración 3:** En la tercera iteración se procederá a una refactorización total del código. Se utilizarán los estándares de A-Frame y crearemos un nuevo proyecto en GitHub, publicaremos nuestra librería en el repositorio de código de NodeJs.
- **Iteración 4:** En este apartado se realizará un enriquecimiento de la parametrización de nuestros gráficos pudiendo ser explotadas gracias al API proporcionado. Además se añadirán nuevos gráficos como el de tarta solicitado por la comunidad cerrando así las primeras incidencias.
- **Iteración 5:** En esta quinta y última iteración se procede a solucionar los distintos errores de funcionamiento de la librería. Se proporciona la posibilidad de una representación dinámica de los datos. Y por último se facilita la integración mediante ejemplos para ser utilizada con dispositivos de realidad virtual.

3.2. Iteración 0. Estudio previo.

En esta fase previa se comienza realizando un estudio previo de las tecnologías que vamos a utilizar en el proyecto. Debido a que se tiene un amplio conocimiento de JavaScript y HTML vamos a cumplir directamente nuestro objetivo específico de estudiar el *framework* A-Frame. Para este menester, se comienza visitando la web oficial de A-Frame¹ donde se puede encontrar toda la documentación² necesaria para comenzar a crear escenas en 3D en nuestro navegador con unos simples pasos.

Para ello se va a comenzar creando un primer proyecto HTML que llamaremos provisionalmente A-Charts. Utilizaremos el IDE WebStorm³ de la compañía JetBrains para comenzar con el desarrollo. Crearemos un proyecto HTML vacío que constará de los siguientes componentes: última versión de la librería de A-Frame⁴ mimificada, y el siguiente fichero HTML con el código que vemos a continuación.

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>My Project</title>
6   <script src="js/aframe.min.js"></script>
7   <script src="js/acharts.js"></script>
8 </head>
9 <body>
10 <a-scene>
11   <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
12   <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
13   <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5" color="#FFC65D"
14     "></a-cylinder>
15   <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4" color=
16     "#7BC8A4"></a-plane>
17   <a-sky color="#ECECEC"></a-sky>
18 </a-scene>
19 </body>
```

¹<https://aframe.io/>

²<https://aframe.io/docs/0.9.0/introduction/>

³<https://www.jetbrains.com/webstorm/>

⁴<https://aframe.io/releases/0.9.2/aframe.min.js>

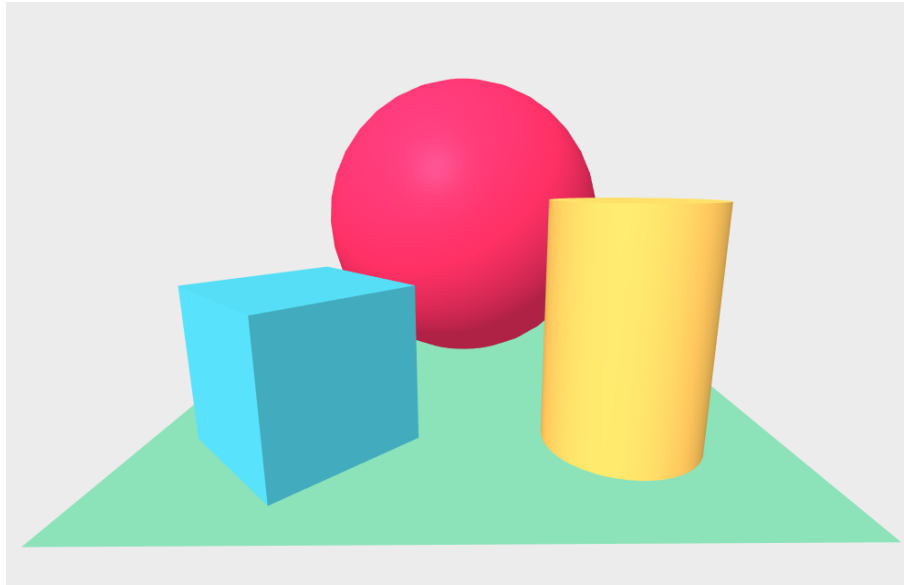


Figura 3.1: Primera escena con A-Frame

18 `</html>`

Como se puede ver en el código, A-Frame interpreta distintas etiquetas de marcado como:

- `<a-scene>`: La escena es el objeto raíz global, y todas las entidades están contenidas dentro de la escena.
- `<a-box>`: Es una de las formas geométricas primitivas de la librería y crea formas como cajas, cubos o paredes.
- `<a-sphere>`: Otra forma primitiva para crear una forma esférica o poliedro
- `<a-cylinder>`: Esta etiqueta crea figuras geométricas en forma de tubo y superficies curvadas.
- `<a-plane>`: Una primitiva más para crear superficies planas.
- `<a-sky>`: El cielo agrega un color de fondo o una imagen de 360 ° a una escena. Un cielo es una esfera grande con un color o textura mapeada hacia adentro.
- `<a-camera>`: Componente que define desde qué perspectiva el usuario ve la escena

Estas etiquetas son las más básicas y el resultado podemos verlo en la siguiente Figura 3.1

```
1 A-Charts:
2 |
3 +---dist
4 |     bundle.js
5 |     bundle.js.map
6 |     index.html
7 |
8 +---src
9 |     acharts.js
10 |    template.html
11 |
12 \---web
13 |     index.html
14 |
15 |    .gitattributes
16 |    .gitignore
17 |    LICENSE
18 |    package-lock.json
19 |    package.json
20 |    README.md
21 |    webpack.config.js
```

Figura 3.2: Primera arquitectura del proyecto

3.3. Iteración 1. Primer gráfico.

En esta primera iteración vamos a proceder a la creación de un repositorio de código en GitHub donde subiremos nuestros primeros desarrollos de la librería. se van a crear dos ramas de desarrollo, develop y master, en nuestro repositorio de git. De esta manera podremos ir desarrollando y probando distinta funcionalidad y solo subiremos a nuestra rama principal cuando tengamos el código probado y listo para producción.

Además, vamos a proceder a nuestra iniciación en el mundo con proyectos NodeJs y Webpack. Para todos estas tecnologías, el tutor de esta memoria ha proporcionado una web⁵ con documentación y ejemplos que también han ayudado en el estudio previo a realizar.

Con esta información, se genera un nuevo proyecto llamado A-Charts con la arquitectura que vemos en la Figura 3.2

Primero se comienza aprendiendo que es Webpack y para qué se utiliza. Como se descri-

⁵<https://jgbarah.github.io/aframe-playground/>

be en el estado de arte, esta tecnología es utilizada para empaquetar módulos estáticos. En la programación modular, los desarrolladores particionan su código en pequeños bloques que cumplen una funcionalidad concreta, y a estos se le llama módulos.

Por tanto, para empezar a usar esta herramienta se crea el fichero de configuración llamado *webpack.config.js*:

```
1 const path = require('path');
2 const webpack = require('webpack');
3 const HtmlWebpackPlugin = require('html-webpack-plugin');
4 module.exports = {
5   entry: './src/acharts.js',
6   output: {
7     filename: 'bundle.js',
8     path: path.resolve(__dirname, 'dist')
9   },
10  module: {
11    rules: [ {
12      test: /\.js$/,
13      include: [path.resolve(__dirname, 'src')],
14      loader: 'babel-loader',
15      query: {
16        presets: ['env']
17      }
18    },
19    {
20      test: /\.html$/,
21      include: [path.resolve(__dirname, 'src')],
22      use: ['html-loader']
23    }
24  ],
25 },
26 stats: {
27   colors: true
28 },
29 devtool: 'source-map',
30 plugins: [ new HtmlWebpackPlugin({
31   template: './src/template.html',
```



```
32     inject: 'head',  
33     filename: 'index.html' })  
34   ]  
35 };
```

En esta configuración se establece que nuestro código, estará en *src/acharts.js* y su salida *bundle.js* en el directorio *dist*. Además se le especifican donde tendremos nuestros módulos JavaScript así como el transpilador a utilizar, en este caso *babel* y el módulo de *HTML*. Por otro lado se utilizará el plugin de *HTML* para *webpack* y definir nuestro *template*.

Por otro lado se comienza con la configuración del servidor web *NodeJs*. Para ello es necesario un fichero llamado *package.json* donde se especifican los metadatos del proyecto como nombre, descripción, versión, licencia, repositorio, etc.

A continuación se describen las líneas del fichero *package.json* que detallan las acciones para las etapas de desarrollo y despliegue.

```
1 {  
2   "scripts": {  
3     "dev": "webpack --mode development",  
4     "build": "webpack --mode production",  
5     "watch": "webpack --mode development --watch",  
6     "test": "echo \"Error: no test specified\" && exit 1",  
7     "start": "webpack-dev-server --mode development --content-  
base dist/",  
8     "clean": "rm -r dist/*",  
9     "cleanbuild": "rm -r dist/* && webpack --mode production"  
10  }  
11 }
```

Y por último las dependencias del proyecto también incluidas en nuestro fichero *package.json*. Node automáticamente buscará estas dependencias en el repositorio de *npm* y las guardará en el directorio *node_modules*.

```
1 {  
2   "devDependencies": {  
3     "babel-core": "^6.26.0",
```

```

4   "babel-loader": "^7.1.4",
5   "babel-preset-env": "^1.6.1",
6   "copy-webpack-plugin": "^4.5.1",
7   "file-loader": "^1.1.11",
8   "html-loader": "^0.5.5",
9   "html-webpack-plugin": "^3.0.6",
10  "webpack": "^4.1.1",
11  "webpack-cli": "^2.0.11",
12  "webpack-dev-server": "^3.1.1"
13 },
14 "dependencies": {
15   "aframe": "^0.8.2"
16 }
17 }

```

Una vez configurado el proyecto comenzaremos a escribir nuestro template HTML.

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 </head>
5 <body>
6   <a-scene>
7     <a-entity id="axis" axis="color: red"></a-entity>
8     <a-sky color="grey"></a-sky>
9     <a-light type="point" intensity="1" position="-2 10 10"></a-light>
10    <a-entity position="2 10 14" rotation="-30 15 0">
11      <a-camera position="3 -1 1" rotation="0 -1 0">
12        <a-cursor></a-cursor>
13      </a-camera>
14    </a-entity>
15  </a-scene>
16 </body>
17 </html>

```

template.html

Como se ve en el código HTML del template, se definen varias etiquetas que el motor de A-Frame interpreta. En el apartado *Iteración 0*, se explican varias de las etiquetas básicas utilizadas en el *template.html* tales como `<a-scene>`, `<a-sky>`, `<a-light>` y `<a-camera>`.

Además se añaden una entidad genéricas de A-Frame con la etiqueta `<a-entity>`. Estas entidades, son componentes genéricos listos para proporcionar una apariencia, comportamiento y funcionalidad determinada. Dichas cualidades se generan mediante código JavaScript completando estos componentes. Por tanto se puede decir que el objetivo de esta librería sería, mediante estas entidades genéricas, dotar de a nuestro HTML de los gráficos que se requieran.

Como se puede ver, la etiqueta `<a-entity>` tiene un identificador para generar un componente llamado *axis*, el cual representará los ejes x,y,z en 3D. Esta definición, entre otras, quedará reflejada en el fichero *src/acharts.js* visto en la Figura 4.2. detallado a continuación por partes.

Primero se comienza importando la librería de A-Frame.

```
1 import aframe from 'aframe';
```

Importación librería A-Frame

Se definen los puntos que tendrá nuestro gráfico de barras en tres dimensiones x,y,z. Además se definen los atributos color y altura.

```
1 var barPlotItems = [  
2   {x: 0, y: 0, z: 0, size: 1, color: "#2dd9d5", height: 1},  
3   {x: 1, y: 0, z: 0, size: 1, color: "#3c90d9", height: 2},  
4   {x: 2, y: 0, z: 0, size: 1, color: "#386eff", height: 3},  
5   {x: 3, y: 0, z: 0, size: 1, color: "#2433ff", height: 4},  
6   {x: 4, y: 0, z: 0, size: 1, color: "#0000ff", height: 5}  
7 ];
```

JSON para representar el gráfico

A continuación se define la función encargada de generar nuestro gráfico de barras una vez el DOM haya sido cargado. Lo primero de todo, almacenamos en la variable *scene* la entidad *a-scene* de A-Frame. Recorremos todos los puntos previamente definidos en la variable *barPlotItems* y por cada punto, creamos una nueva entidad con el atributo *barplot*. Esto quiere decir que cada entidad tomará el comportamiento, funcionalidad y estilo que hayamos registrado en el componente de A-Frame *barplot*. Además a cada nueva entidad se le pasan como argumento

una lista de valores definidos en el esquema del componente que veremos a continuación. Por último, añadimos cada nueva entidad a la escena con la función *appendChild*.

```

1 document.addEventListener("DOMContentLoaded", function(event) {
2     var scene = document.querySelector('a-scene');
3     for (let item of barPlotItems) {
4         var entity = document.createElement('a-entity');
5         entity.setAttribute('barplot', {
6             'color': item['color'],
7             'size': item['size'],
8             'position': {x: item['x'] + item['size']/2,
9                 y: item['height']/2, z: item['z']},
10            'height': item['height'],
11            'type': 'cylinder'
12        });
13        scene.appendChild(entity);
14    };
15 });

```

Función ejecutada al cargarse el DOM

En este siguiente paso vamos a registrar nuestro componente *barplot* mediante la función *registerComponent* de A-Frame. A esta función se le pasa como argumento el nombre, el cual será usado como atributo HTML en la representación del componente en el DOM. Luego se añade la definición del componente, que es un objeto JavaScript de métodos y propiedades. Estos métodos pueden definir el manejo del ciclo de vida del componente.

En este caso, la definición del componente constará de la posición, el color, la altura, el tamaño y el tipo. Este último nos servirá en un futuro para que nuestro componente no sólo sea un gráfico de barras sino que pueda implementar distintos tipos de gráficos en función del esquema que se le pase.

Después, se implementa la función *update* proporcionada por el core de A-Frame. Esta función es ejecutada cuando se modifica alguna propiedad de dicho componente (para actualizar el gráfico dinámicamente sin necesidad de refrescar la página web) o en el inicio del ciclo de vida del componente (después de la función *init*) como es el caso.

Como vemos en el código, se comprueba que el tipo de gráfico sea *cylinder*, es decir una barra cilíndrica, y si se cumple dicha condición, se rellenan todos los parámetros del esquema gracias al método *self.data* donde tenemos almacenados todos los datos introducidos anteriormente.

```

1 AFRAME.registerComponent('barplot', {
2   schema: {
3     position: {type: 'vec3', default: {x:0, y:0, z:0}},
4     color: {type: 'string', default: 'red'},
5     height: {type: 'number', default: 1},
6     size: {type: 'number', default: 1},
7     type: {type: 'string', default: 'cylinder'}
8   },
9   update: function () {
10     var self = this;
11     if(self.data.type === 'cylinder'){
12       self.cylinder = document.createElement('a-cylinder');
13       self.cylinder.setAttribute('position', self.data.position);
14       self.cylinder.setAttribute('color', self.data.color);
15       self.cylinder.setAttribute('radius', self.data.size/2);
16       self.cylinder.setAttribute('height', self.data.height);
17       this.el.appendChild(self.cylinder);
18     }
19   }
20 });

```

Definición del componente barplot

Por último, se registra el componente *axis*. En este caso, se define en el esquema del componente únicamente el color. Además se implementa nuevamente la función *update*.

Dentro de esta función se generan el elemento *line* de A-Frame varias veces. Se crean, en concreto, tres líneas por cada eje de coordenadas (x,y,z) con una longitud de diez unidades. Luego, por cada eje de coordenadas y de una en una unidad, se generan los *ticks* (más líneas de longitud 0.4 unidades) para ayudar en la interpretación de la posición del gráfico final.

El componente *line* dibuja una línea dada una coordenada de inicio y otra final. Como se expone anteriormente en el punto 3. Estado del arte, este componente lo hereda A-Frame de la librería Three.js y por tanto se tiene a disposición para su uso en esta librería.

```

1 AFRAME.registerComponent('axis', {
2   schema: {
3     color: {type: 'string', default: 'red'}
4   },
5   update: function () {

```

```

6      var el = this.el
7      el.setAttribute('line__x', {
8      'start': {x: 0, y: 0, z: 0},
9          'end': {x: 10, y: 0, z: 0},
10         'color': this.data.color});
11      for (var tick = 1; tick < 10; tick++) {
12          el.setAttribute('line__x' + tick, {
13          'start': {x: tick, y: -0.2, z: 0},
14              'end': {x: tick, y: 0.2, z: 0},
15              'color': this.data.color});
16      };
17      el.setAttribute('line__y', {
18      'start': {x: 0, y: 0, z: 0},
19          'end': {x: 0, y: 10, z: 0},
20          'color': this.data.color});
21      for (var tick = 1; tick < 10; tick++) {
22          el.setAttribute('line__y' + tick, {
23          'start': {y: tick, z: -0.2, x: 0},
24              'end': {y: tick, z: 0.2, x: 0},
25              'color': this.data.color});
26      };
27      el.setAttribute('line__z', {
28      'start': {x: 0, y: 0, z: 0},
29          'end': {x: 0, y: 0, z: 10},
30          'color': this.data.color});
31      for (var tick = 1; tick < 10; tick++) {
32          el.setAttribute('line__z' + tick, {
33          'start': {z: tick, x: -0.2, y: 0},
34              'end': {z: tick, x: 0.2, y: 0},
35              'color': this.data.color});
36      };
37  }
38  });

```

Definición del componente barplot

Cuando arrancamos el proyecto con el comando *start* definido en el fichero *package.json* se obtiene como resultado el primer gráfico que se puede ver en la Figura 3.3

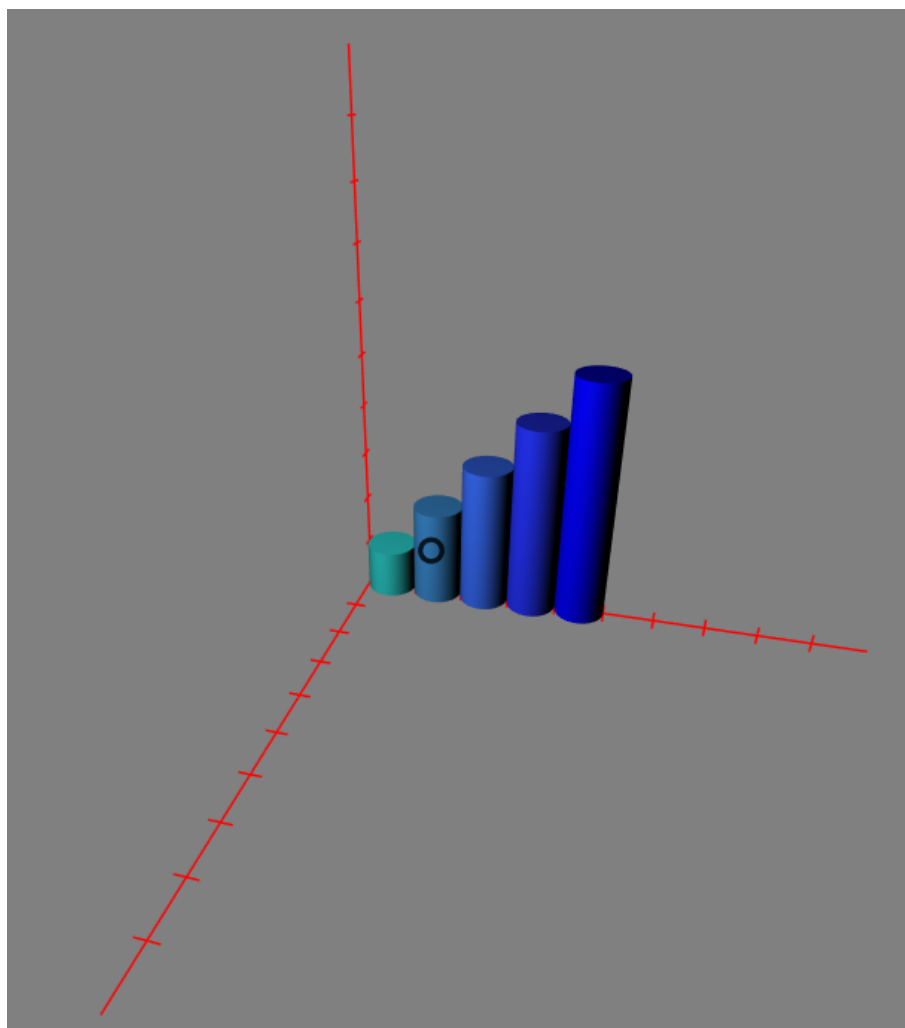


Figura 3.3: Iteración 1: Resultado

3.4. Iteración 2. Enriquecimiento de la librería y separación de la capa de datos.

En esta segunda iteración tenemos como objetivo añadir más tipos de gráficos a nuestra librería. Cada gráfico será un nuevo componente que registraremos en A-Frame como se ha mostrado en la primera iteración.

Por otro lado vamos a separar los datos, los cuales forman puntos en el espacio tridimensional, del código donde se generan las representaciones en sí. Para ello se crea uno o varios ficheros JSON con los datos y se especificará en el componente la ruta del fichero donde obtener estos datos.

Respecto al template HTML mostrado a continuación y al visto en la primera iteración se aprecian varios cambios. Para empezar se ha eliminado la entidad *axis* ya que, como veremos más adelante, cada gráfico llamará al componente *axis*. Por tanto, solo es necesario declarar una entidad de A-Frame teniendo como atributo el nombre del componente.

```

1 <!DOCTYPE html>
2 <html lang="en">
3   <head></head>
4   <body>
5     <a-scene id="my_scene">
6       <a-sky color="grey"></a-sky>
7       <a-light type="point" intensity="1" position="-2 10 10">
8     </a-light>
9     <a-entity bubble_chart="datapoints: data.json; size:1; position
10 : 0 0 0; color: green"></a-entity>
11     <a-entity position="2 10 14" rotation="-30 15 0">
12       <a-camera position="3 -1 1" rotation="0 -1 0">
13     </a-camera>
14   </a-entity>
15 </a-scene>
16 </body>
17 </html>

```

template.html

Como se ve en la imagen *template.html*, dentro del atributo *bubble_chart* añadimos cuatro propiedades: *datapoints*, *size*, *position* y *color*. Estas propiedades están descritas en el esquema del componente que veremos a continuación.

En esta versión de la librería registraremos tres tipos de componentes, uno por cada tipo de gráfico. Solo necesitaremos poner el nombre del componente como atributo para que la librería de A-Frame renderice nuestro gráfico tal y como hemos visto en el código HTML anterior.

Los tres componentes que se desarrollan son *bubble_chart* (gráfico de burbujas), *bar_chart* (gráfico de barras cilíndricas o cúbicas) y *line_chart* (Para el gráfico de líneas). Todos ellos siguen la siguiente estructura de código:

- Registro del componente y definición del esquema.

```
1 AFRAME.registerComponent('bubble_chart', {  
2   schema: {  
3     position: {type: 'vec3', default: {x:0, y:0, z:0}},  
4     color: {type: 'string', default: 'red'},  
5     size: {type: 'number', default: 1},  
6     datapoints: {type: 'asset'}  
7   },
```

Registro del componente y esquema

En este esquema se puede ver la concordancia del HTML visto anteriormente y el componente. Vemos que se definen las propiedades y el tipo de dato. La propiedad más importante y novedosa en esta iteración es la de *datapoints*, a la cual se le especifica que sea de tipo *asset* (activo o recurso). Esto quiere decir que vamos a poder pasar al componente una ruta donde tengamos nuestro fichero JSON con los datos del gráfico.

La única diferencia de esquema entre los tres componentes es el atributo *type*. Este atributo se usa únicamente en el componente *bar_chart* para especificar si queremos un gráfico con barras cilíndricas o cúbicas. En posteriores iteraciones se usará este concepto para unificar los componentes en uno siguiendo los estándares de A-Frame.

- Definición de la función *init* y *update*.

```
1   init: function () {  
2     var self = this;  
3     this.loader = new THREE.FileLoader();
```

```

4      },
5
6      update: function (oldData) {
7          var self = this;
8          const data = this.data;
9          if (AFRAME.utils.deepEqual(oldData, data))
10             return;
11          if (data.datapoints && data.datapoints !== oldData.datapoints)
12             this.loader.load(data.datapoints, this.onDataLoaded.bind(
13                 this));
14      },

```

Funciones init y update

La función *init* es llamada al inicio del ciclo de vida del componente. Sirve para inicializar ciertas variables del componentes o funciones *listener*. En este caso, se inicializa la función *FileLoader* que nos provee Threejs encargada de cargar en memoria cualquier tipo de fichero.

Por otro lado, la función *update* es llamada después de la función *init* en el ciclo de vida del componente. Además tiene la peculiaridad de ser invocada cada vez que se cambie algún atributo, por ello realizamos ciertas comprobaciones en el código. La primera de ellas, es comparar si el fichero de datos que se ha modificado es igual que el que teníamos cargado con la función *deepEqual* de A-Frame. Si son iguales se ignora el fichero y no se realiza ningún trabajo. En la segunda comprobación volvemos a comprobar que los datos en JSON son efectivamente distintos y si el fichero contiene datos válidos. Si es así, invocamos a la función *onDataLoaded* que hemos definido en nuestro componente. Esta función no es nativa de A-Frame, si no que ha sido implementada en este proyecto y es donde se desarrolla toda la lógica de nuestro componente.

■ Definición de la función *onDataLoaded*

```

1 onDataLoaded: function (file) {
2     const data = this.data;
3     const pos = data.position;
4
5     var entity_axis = document.createElement('a-entity');

```

```

6     entity_axis.setAttribute('axis',
7         {
8             'color': data.color,
9             'position': data.position
10        }
11    );
12    this.el.appendChild(entity_axis);
13    this.el.setAttribute('id', 'bubble_chart');
14
15    var datapoints = JSON.parse(file);
16
17    for (let point of datapoints) {
18        var entity = document.createElement('a-sphere');
19        entity.setAttribute('position', {x: pos.x + point['x'],
20            y: pos.y + point['y'],
21            z: pos.z + point['z']});
22        entity.setAttribute('color', point['color']);
23        entity.setAttribute('radius', point['size']);
24        entity.addEventListener('mouseenter', function () {
25            this.setAttribute('scale', {x: 1.3, y: 1.3, z: 1.3});
26        });
27        entity.addEventListener('mouseleave', function () {
28            this.setAttribute('scale', {x: 1, y: 1, z: 1});
29        });
30        this.el.appendChild(entity);
31    }
32    }
33    });

```

Función onDataLoaded

Finalmente, se define la función *onDataLoaded* cuyo argumento *file* es el fichero de datos. Primero se crea el componente genérico *axis* que utilizan todos los gráficos pasándole como esquema el color y la posición de los ejes. Seguidamente parseamos el fichero con la función JavaScript `JSON.parse`. Gracias a ella podemos recorrer cada punto del gráfico en un bucle. Por cada elemento de los datos del JSON se crea una entidad de A-Frame, en este caso `<a-sphere>`. Los atributos vienen dados por el propio fichero de datos y

el esquema del componente. Además se le añade una pequeña función para que cuando pasemos el ratón por encima de cada elemento, este se haga mayor.

Cabe destacar que el componente *axis* ha sido refactorizado respecto a la primera iteración. En este caso se ha subdividido en dos componentes, uno de ellos genérico donde se recorren los ejes x, y, z y el otro donde se crean las líneas y ticks de los propios ejes quedando de la siguiente manera:

```

1 AFRAME.registerComponent('axis', {
2   schema: {
3     position: {type: 'vec3', default: {x:0, y:0, z:0}},
4     color: {type: 'string', default: 'red'}
5   },
6   update: function () {
7     const data = this.data;
8     for (let axis of ['x', 'y', 'z']) {
9       var axis_line = document.createElement('a-entity');
10      axis_line.setAttribute('axis-line', {
11        'axis': axis,
12        'color': data.color,
13        'position': data.position
14      });
15      this.el.appendChild(axis_line);
16    }
17  }
18 });
19 AFRAME.registerComponent('axis-line', {
20   ...
21   update: function () {
22     ...
23     el.setAttribute('line', {
24       'start': {x: pos.x, y: pos.y, z: pos.z},
25       'end': line_end,
26       'color': data.color
27     });
28     for (var tick = 1; tick < 10; tick++) {
29       ...
30       el.setAttribute('line__' + tick, {

```

```

31         'start': tick_start,
32         'end': tick_end,
33         'color': this.data.color}
34     );
35 }
36 }
37 });

```

Componente axis

Por último, se ha añadido a modo de ejemplo para uso de esta librería un menú de opciones. Si se selecciona alguna de estas opciones se sustituye el gráfico anterior por el seleccionado.

Para esto se han seguido los siguientes pasos:

- Se añade al fichero *webpack.config.js* las siguiente configuración para empaquetar CSS y JSON.

```

1 {
2   test: /\.css$/,
3   use: [ 'style-loader', 'css-loader' ]
4 },
5 {
6   type: 'javascript/auto',
7   test: /\.json$/,
8   include: [path.resolve(__dirname, 'src')],
9   use: [
10     {
11       loader: 'file-loader',
12       options: {name: '[name].[ext]'}
13     }
14   ]
15 }

```

webpack.config.js CSS y JSON

- Añadimos la dependencia *css-loader* al fichero *package.json*.

```

1 "css-loader": "^1.0.0"

```

Dependencia para package.json

- Código HTML para el menú de opciones

```

1 <div class="div-opt">
2   <select id="selectChart" class="max-width">
3     <option value="bubble_chart">Bubble Chart</option>
4     <option value="box_bar_chart">Bar Chart</option>
5     <option value="cylinder_bar_chart">Cylinder Bar Chart</option>
6     <option value="line_chart">Line Chart</option>
7   </select>
8 </div>

```

HTML menú de opciones

- El CSS para el menú de opciones simplemente hace que el elemento con la clase *div-opt* se posicione arriba a la izquierda de la web.
- Código JavaScript para el menú de opciones

```

1 function changeChart() {
2   ...
3   var entity = document.createElement('a-entity');
4   if(selectedChart === "bubble_chart")
5     entity.setAttribute(selectedChart, bubble_chart);
6   if(selectedChart === "box_bar_chart")
7     entity.setAttribute("bar_chart", box_bar_chart);
8   if(selectedChart === "cylinder_bar_chart")
9     entity.setAttribute("bar_chart", cylinder_bar_chart);
10  if(selectedChart === "line_chart")
11    entity.setAttribute(selectedChart, line_chart);
12  parent.appendChild(entity);
13 }
14 document.getElementById("selectChart")
15   .addEventListener("change", changeChart);

```

JS menú de opciones

Cuando arrancamos el proyecto, nuevamente con el comando *start* definido en el fichero *package.json*, se obtiene como resultado las Figuras 3.4 y 3.5.

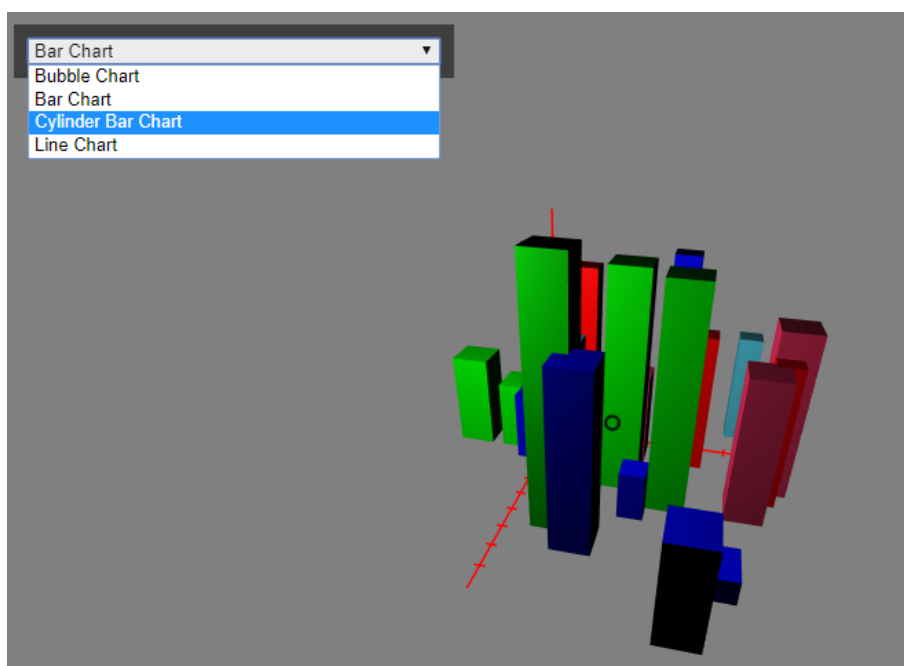


Figura 3.4: Iteración 2: Resultado Gráfico de Barras

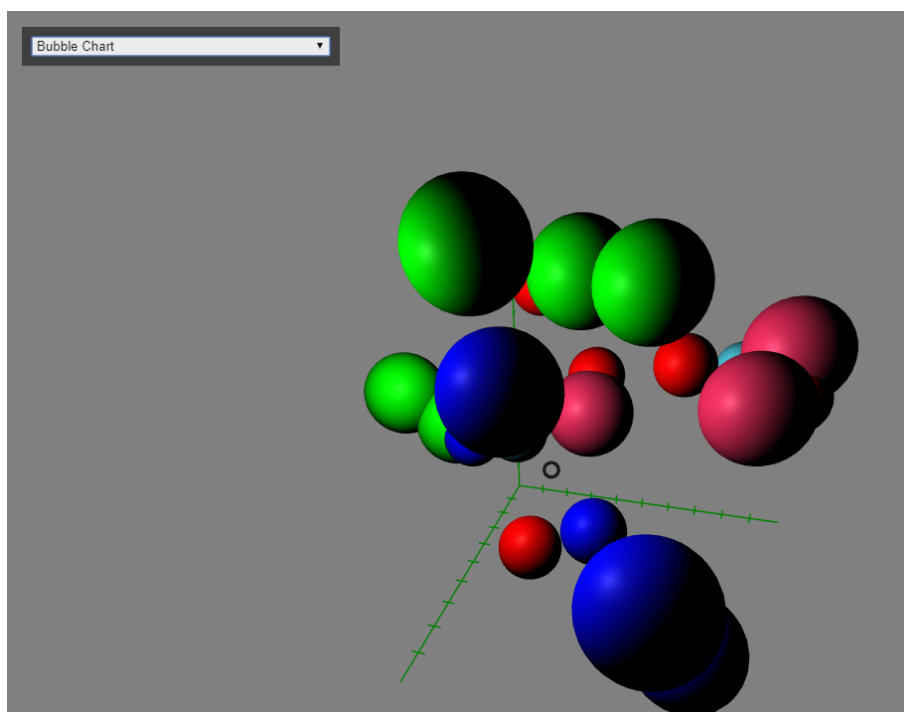


Figura 3.5: Iteración 2: Resultado Gráfico de burbujas

3.5. Iteración 3. Refactorización y estandarización del módulo.

En esta iteración, el objetivo es publicar nuestro componente en la comunidad de A-Frame. Para ello seguimos las instrucciones de A-Frame Registry⁶ donde debemos seguir una serie de pasos para cumplir con las *best practices*.

Gracias a estas medidas tendremos nuestro componente correctamente versionado, publicado en varios repositorios de código listo para ser utilizado por terceros e incluso permitiendo la contribución de otros desarrolladores. Se creará un proyecto de cero donde integraremos nuestro antiguo componente siguiendo los siguientes pasos:

- En primer lugar se utiliza la herramienta *angle*⁷, la cual permite crear un proyecto de cero por línea de comando (Figura 3.6). Se genera un componente con una arquitectura genérica (Figura 3.7).
- Se publica el componente en npm, el repositorio de paquetes de nodeJs, mediante el comando *npm publish*. Para ello es necesario tener instalado npm en nuestro sistema y crearse una cuenta gratuita en la web oficial ⁸
- Se publica el componente en GitHub, un repositorio online de código. Para ello es necesario tener instalado git en nuestro sistema y crearse una cuenta gratuita en la web oficial ⁹. Una vez creado el proyecto en la web, debemos ir mediante línea de comandos donde tenemos nuestro proyecto y ejecutar *git init*, *git commit* y *git push*.
- Se añade documentación sobre las propiedades del componente y una página de ejemplo de uso de la librería. Todo ello se puede consultar en la sección de Resultados.
- Se debe publicar una página web con ejemplos. Se propone *GitHub Pages* la cual utilizamos para ver nuestro resultado¹⁰.
- Se debe añadir un enlace desde dicha página web al repositorio de GitHub.

⁶<https://github.com/aframevr/aframe-registry>

⁷<https://github.com/ngokevin/angle>

⁸<https://www.npmjs.com/>

⁹<https://github.com/>

¹⁰<https://adrixp.github.io/aframe-charts-component/>

- Debe tener sentido en el contexto de una aplicación WebVR.
- Debe incluir una imagen de vista previa o GIF en el archivo README.
- Deberá seguir semver¹¹ en su esquema de versión de componente, reflejando la última versión estable de A-Frame.
- Se debe auto registrar el componente con la función *AFRAME.registerComponent*. En este último punto aprovechamos para realizar una refactorización del código. Nuestro template HTML queda de la siguiente manera:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     ...
5   </head>
6   <body>
7     <a-scene>
8       <a-sky color="grey"></a-sky>
9       <a-light type="point" intensity="1" position="-2 10 10"></
a-light>
10      <a-entity charts="dataPoints: ../data/data.json; type: bar
"></a-entity>
11      <a-entity position="2 10 14" rotation="-30 15 0">
12        <a-camera position="3 -1 4" rotation="0 -1 0">
13          <a-cursor></a-cursor>
14        </a-camera>
15      </a-entity>
16    </a-scene>
17  </body>
18 </html>
```

Iteración 3: Template HTML

Como se puede ver únicamente se instancia una entidad de A-Frame con el atributo *charts*. Además, se utilizan dos propiedades, una donde se le indica la ruta del fichero de datos y el tipo de gráfico a representar.

¹¹<https://semver.org/>

En cuanto al código JavaScript, únicamente registramos un componente llamado *charts*.

```

1 AFRAME.registerComponent('charts', {
2   schema: {
3     type: {type: 'string', default: 'bubble'},
4     dataPoints: {type: 'asset'}
5   },
6   ....
7   onDataLoaded: function (file) {
8     let dataPoints = JSON.parse(file);
9     const data = this.data;
10    generateAxis(this.el);
11    for (let point of dataPoints) {
12      let entity;
13      if(data.type === "bar"){
14        entity = generateBar(point);
15      }else if(data.type === "cylinder"){
16        entity = generateCylinder(point);
17      }else{
18        entity = generateBubble(point);
19      }
20      ...
21      this.el.appendChild(entity);
22    }
23  }
24 });

```

Iteración 3: Componente Charts

Como vemos en su esquema, únicamente tendremos por ahora dos propiedades. La propiedad *type* nos indica el tipo de gráfico que se va a generar, y en base a dicho tipo se selecciona una función u otra como la que vemos a continuación:

```

1 function generateBubble(point) {
2   let entity = document.createElement('a-sphere');
3   entity.setAttribute('position', {x: point['x'], y: point['y'], z:
4     point['z']});
5   entity.setAttribute('color', point['color']);
6   entity.setAttribute('radius', point['size']);

```

```

6   return entity;
7 }

```

Iteración 3: Función para generar Gráfico Burbujas

Finalmente, el eje del gráfico también a recibido una refactorización importante de código quedando de la siguiente manera:

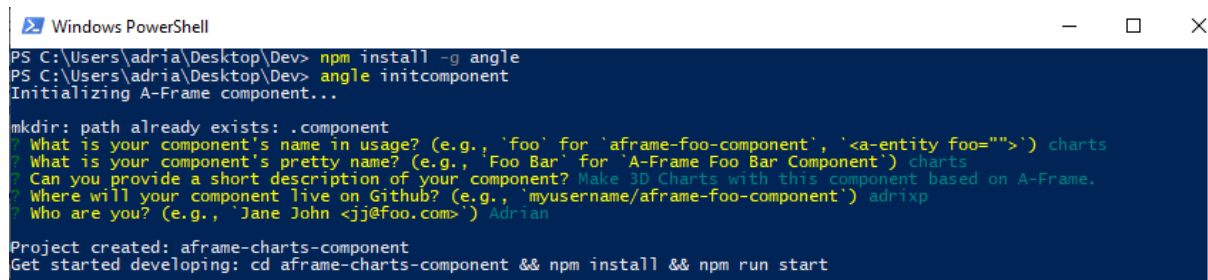
```

1 function generateAxis(element) {
2   for (let axis of ['x', 'y', 'z']) {
3     ...
4     let axis_line = document.createElement('a-entity');
5     axis_line.setAttribute('line', {
6       'start': {x: 0, y: 0, z: 0},
7       'end':   line_end,
8       'color': 'red'
9     });
10
11    for (let tick = 1; tick <= 10; tick++) {
12      ...
13      axis_line.setAttribute('line__' + tick, {
14        'start': tick_start,
15        'end':   tick_end,
16        'color': 'red'
17      });
18    }
19    element.appendChild(axis_line);
20  }
21 }

```

Iteración 3: Función para generar ejes

El resultado final es un único componente, cuyo código es mucho más limpio y nos vale para generar distintos tipos de gráficos. Además, cumple con todas las recomendaciones de A-Frame. Cabe destacar que se ha eliminado el gráfico de líneas debido a que visualmente no aporta información sino confusión.



```

Windows PowerShell
PS C:\Users\adria\Desktop\Dev> npm install -g angle
PS C:\Users\adria\Desktop\Dev> angle initcomponent
Initializing A-Frame component...
mkdir: path already exists: .component
? What is your component's name in usage? (e.g., 'foo' for 'aframe-foo-component', '<a-entity foo="">') charts
? What is your component's pretty name? (e.g., 'Foo Bar' for 'A-Frame Foo Bar Component') charts
? Can you provide a short description of your component? Make 3D Charts with this component based on A-Frame.
? Where will your component live on Github? (e.g., 'myusername/aframe-foo-component') adrixp
? Who are you? (e.g., 'Jane John <jj@foo.com>') Adrian
Project created: aframe-charts-component
Get started developing: cd aframe-charts-component && npm install && npm run start

```

Figura 3.6: Iteración 3: Inicializar proyecto con Angle

```

1  +---dist
2  |
3  +---examples
4  |   +---basic
5  |   |       index.html
6  |   |
7  |   |
8  +---tests
9  |   helpers.js
10 |   index.test.js
11 |   karma.conf.js
12 |   __init.test.js
13 |
14 |   .gitattributes
15 |   .gitignore
16 |   .travis.yml
17 |   index.html
18 |   index.js
19 |   LICENSE
20 |   package-lock.json
21 |   package.json
22 |   README.md

```

Figura 3.7: Iteración 3: Nueva Arquitectura

3.6. Iteración 4. Poblamiento de funcionalidad de la API. Resolución de peticiones de la comunidad.

En esta cuarta iteración se tiene como objetivo enriquecer el API de la librería. Para ello se aumentará el número de propiedades del esquema de nuestro componente y se adaptarán las distintas funciones para que todos los gráficos disfruten de dicha operatividad.

Por otro lado, se irá actualizando la documentación del componente así como su página de ejemplo y casos de uso.

Finalmente, gracias a la publicación del componente en la comunidad se recibe la petición para añadir nuevos gráficos como el gráfico de tarta quedando implementado al final de la iteración.

Para aumentar dicha funcionalidad, se comienza actualizando el esquema del componente. En esta iteración se quiere dotar a los ejes x, y, z de un mayor protagonismo ya que ayuda a la hora de obtener información del gráfico. Por ello el esquema del componente queda de la siguiente manera:

```
1 AFRAME.registerComponent('charts', {
2   schema: {
3     type: {type: 'string', default: 'bubble'},
4     dataPoints: {type: 'asset'},
5     axis_position: {type: 'vec3', default: {x:0, y:0, z:0}},
6     axis_color: {type: 'string', default: 'red'},
7     axis_length: {type: 'number', default: 0},
8     axis_tick_separation: {type: 'number', default: 1},
9     axis_tick_length: {type: 'number', default: 0.2},
10    axis_tick_color: {type: 'string', default: 'red'},
11    axis_negative: {type: 'boolean', default: true},
12    axis_grid: {type: 'boolean', default: false},
13    axis_grid_3D: {type: 'boolean', default: false}
14  }
```

Iteración 4: Esquema del componente

La descripción de la funcionalidad de estos nuevos parámetros es la siguiente:

- **axis_position:** Posición de los ejes en las coordenadas tridimensionales. Es posible que en función de los datos el usuario prefiera tener los ejes en una u otra posición. Por defecto están en (0,0,0).
- **axis_color:** Color de los ejes. El valor que se le pasa es un *string* con el valor que se le daría al CSS de cualquier elemento HTML. El valor puede ser el color en inglés o códigos de color Hexadecimal, RGB y HSL.
- **axis_length:** Longitud de los ejes. Por defecto, si no se le pasa ningún valor los ejes serán adaptativos. Es decir, los ejes por defecto tienen la longitud x,y,z del punto que mayor valor tenga en cada eje de coordenada.
- **axis_negative:** Opción para mostrar los ejes en las coordenadas x,y,z negativos.

Para implementar estas dos últimas propiedades se utiliza la siguiente comprobación en el código y una función que recorre los puntos para obtener el mayor de todos en cada eje.

```

1 if(properties.axis_length === 0){
2   let adaptive_props = getAdaptiveAxisProperties(dataPoints);
3   properties.axis_length = adaptive_props.max;
4   if(properties.axis_negative)
5     properties.axis_negative = adaptive_props.has_negative;
6 }
7 function getAdaptiveAxisProperties(dataPoints) {
8   let max = 0;
9   let has_negative = false;
10  for (let point of dataPoints) {
11    if(point.x < 0 || point.y < 0 || point.z < 0)
12      has_negative = true;
13    let point_x = Math.abs(point.x);
14    let point_y = Math.abs(point.y);
15    let point_z = Math.abs(point.z);
16    if( point_x > max)
17      max = point_x;
18    if( point_y > max)
19      max = point_y;
20    if( point_z > max)

```

```

21         max = point_z;
22     }
23     return {max: max, has_negative: has_negative};
24 }

```

Iteración 4: Ejes adaptativos y negativos

Como vemos se comprueba si la longitud de los ejes es cero o si se quieren ejes negativos. Después recorremos todos los datos para devolver un objeto que posteriormente se utiliza en la generación de los ejes de coordenadas.

- **axis_tick_separation:** Separación de los *ticks* de los ejes. Por defecto será de una unidad.
- **axis_tick_length:** Longitud de los *ticks*.
- **axis_tick_color:** Color de los ticks
- **axis_grid:** Esta opción permite generar una rejilla por cada plano que forman los ejes x,y,z. Esta rejilla ayuda a la hora de interpretar los datos representados
- **axis_grid_3D** En este caso, se genera una rejilla por cada punto de cada eje tridimensional. Añade mayor precisión que el punto anterior. Por contra genera muchas líneas y puede generar, en algún caso, confusión.

La función que genera nuestros ejes quedaría de la siguiente manera:

```

1 function generateAxis(element, properties) {
2     let axis_length = properties.axis_length;
3     let axis_position = properties.axis_position;
4     ...
5     for (let axis of ['x', 'y', 'z']) {
6
7         let line_end = {x: axis_position.x, y: axis_position.y, z:
            axis_position.z};
8         line_end[axis] = axis_length + axis_position[axis];
9         let line_start = {x: axis_position.x, y: axis_position.y, z:
            axis_position.z};
10
11         if (axis_negative) {
12             axis_negative_offset = axis_length + 1;

```



```

13         line_start[axis] = -axis_length + axis_position[axis];
14     }
15     ...
16     for (let tick = tick_separation - axis_negative_offset; tick
17         <= axis_length; tick += tick_separation) {
18         ...
19         if (axis === 'x') {
20             tick_start = {x: axis_position.x + tick,
21                         y: axis_position.y - tick_length,
22                         z: axis_position.z};
23             tick_end   = {x: axis_position.x + tick,
24                         y: axis_position.y + tick_length,
25                         z: axis_position.z};
26             ...
27         }
28         element.appendChild(axis_line);
29     }

```

Iteración 4: Nueva generación de ejes

Como se puede ver, primero se inicializan todas las propiedades tales como color, longitud etc. tanto de los ejes como de los *ticks*. Seguidamente, se generan dichos ejes en función de la posición y de si se requiere que tomen valores negativos o no. Por último, se utiliza esto mismo para generar cada *tick* de cada eje. Teniendo en cuenta su longitud e intervalos.

Una vez llegado a este punto se recibe la petición por parte de un miembro de la comunidad de A-Frame de incluir en la librería un gráfico de tarta¹². Por ello se decide aplazar otras propiedades definidas en el *roadmap* de esta iteración y se pospone al final de la misma. Estas nuevas funcionalidades serían las de incluir numeración en los ejes y color y tamaño de los mismos.

En un estudio previo se observa que la implementación del gráfico de tartas y el gráfico con forma de *donut* siguen un desarrollo muy parecido, por lo que se decide añadir ambos.

Primero se comienza añadiendo dos nuevas propiedades al esquema del componente

```

1 pie_radius: {type: 'number', default: 1},

```

¹²<https://github.com/adrixp/aframe-charts-component/issues/1>

```
2 pie_doughnut: {type: 'boolean', default: false},
```

Iteración 4: Nuevas propiedades: gráfico de tarta

En estas propiedades se define el radio del gráfico de tartas, es decir, el tamaño que tendrá. Además se permite elegir si se quiere en forma de *donut* o no.

En cuanto al código, antes de llamar a la función que genera las partes del gráfico se realizan ciertas operaciones y comprobaciones.

```
1 ...
2 if(properties.type === "pie"){
3   for (let point of dataPoints)
4     pie_total_value += point['size'];
5 }
6
7 pie_angle_end = 360 * point['size'] / pie_total_value;
8 if(properties.pie_doughnut){
9   entity = generateDoughnutSlice(...);
10 }else{
11   entity = generateSlice(...);
12 }
13 pie_angle_start += pie_angle_end;
14 ...
```

Iteración 4: Gráfico de tarta: Inicialización

Primero se necesita calcular el valor total de todos los puntos a representar. Para ello se recorre en un bucle y se añade a la variable *pie_total_value*. Acto seguido se calcula el ángulo que tendrá cada valor a representar. Para ello se utiliza la siguiente fórmula:

$$angulo = \frac{360 * valorpunto}{valortotal}$$

Por último incrementamos el ángulo inicial para el siguiente punto y llamamos a la función que se ve a continuación:

```
1 function generateSlice(point, theta_start, theta_length, radius) {
2   let entity = document.createElement('a-cylinder');
3   entity.setAttribute('color', point['color']);
4   entity.setAttribute('theta-start', theta_start);
5   entity.setAttribute('theta-length', theta_length);
```

```

6   entity.setAttribute('side', 'double');
7   entity.setAttribute('radius', radius);
8   return entity;
9 }

```

Iteración 4: Función gráfico de tarta

Como se puede observar, esta función recibe cuatro argumentos. El punto de donde se toma el color, ángulo de inicio, ángulo de fin y el radio. Se utiliza la entidad pura <a-cylinder> de A-Frame y gracias a los atributos *theta-start* y *theta-end* se genera nuestro trozo de tarta. Para el gráfico en forma de *donut* se sigue el mismo procedimiento. Lo único que cambia, es que se utiliza el elemento toroide de A-Frame llamado <a-torus> con las propiedades *rotation* y *arc* para los ángulos de inicio y fin. Podemos ver el resultado en la Figura 3.8.

Finalmente, se decide implementar las últimas propiedades para esta iteración. Nuevamente se quiere aumentar la funcionalidad y legibilidad de los ejes ya que aportará bastante información del gráfico. Por ello se decide añadir numeración en los ejes tridimensionales y que se permita elegir el color y tamaño.

Para este menester, se utilizará el elemento <a-text> de A-Frame como veremos a continuación.

Como siempre, se comienza añadiendo al esquema las nuevas propiedades que hacen que el componente sea totalmente configurable para el usuario y pueda utilizar las herramientas que necesite para cada caso de uso.

```

1 axis_text:           {type: 'boolean', default: true},
2 axis_text_color:     {type: 'string', default: 'white'},
3 axis_text_size:      {type: 'number', default: 10},

```

Iteración 4: Esquema texto de los ejes

Como se observa, simplemente se añade si se quiere incluir texto, su color y el tamaño. En cuanto al código, se modifica la función *generateAxis* para incluir el texto de la siguiente manera:

```

1 ...
2 if(axis_text){
3   let axis_text = document.createElement('a-text');
4   axis_text.setAttribute('position', tick_start);
5   axis_text.setAttribute('text__' + tick, {

```

```
6      'value': Math.round(tick * 100) / 100,  
7      'width': axis_text_size,  
8      'color': axis_text_color,  
9      'xOffset': 4.5  
10    });  
11    element.appendChild(axis_text);  
12  }  
13  ...
```

Iteración 4: Código para generar el texto de los ejes.

Finalmente, por cada tick de cada eje tridimensional se añade el valor correspondiente redondeado con la función *Math.round*. Se puede ver el resultado en la Figura 3.8.

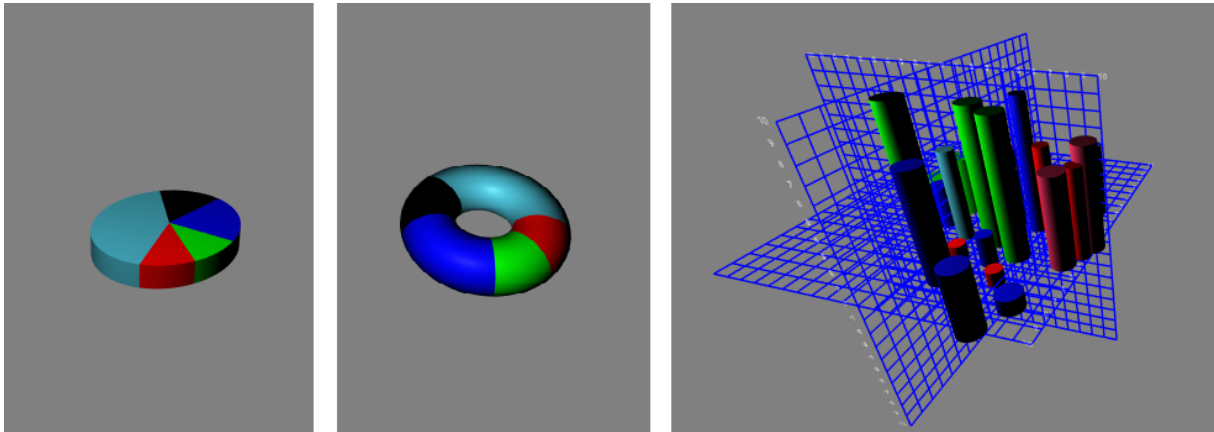


Figura 3.8: Iteración 4: Resultado

3.7. Iteración 5. Leyenda y pop-up, consumo de otras fuentes de datos y adaptación para dispositivos de VR.

En esta última iteración, se tiene como objetivo seguir aumentando la información proporcionada por los gráficos mediante la implementación de una leyenda y de un panel informativo o *pop-up*.

Por otro lado, se desarrollará un nuevo componente que nos permitirá cambiar los datos del gráfico de manera dinámica con un solo *click*. Además, se habilitará en la librería la opción de consumir de otras fuentes de datos. Hasta ahora se tenía la opción de incluirlos mediante ficheros JSON, pero para poder utilizar la librería de manera más versátil se considera importante este desarrollo.

Por último, cumpliendo con otro de los objetivos más importantes, se dotará de un módulo de ejemplo para el uso de la librería con dispositivos de realidad virtual.

Primero se comienza con el desarrollo del *pop-up*. Esta funcionalidad se encarga de mostrar un pequeño recuadro encima de cada punto tridimensional. Gracias a ello, pasando el cursor por cada punto se mostrará dicha información. Para ello añadimos al esquema de nuestro componente la siguiente propiedad:

```
1 show_popup_info:      {type: 'boolean', default: false}
```

Iteración 5: Propiedad del esquema para el pop-up

Esta propiedad únicamente nos indica si queremos que se muestre o no el *pop-up*. Además solo se podrá mostrar para gráficos que no sean del tipo tarta. Por cada punto, utilizamos una función que escucha eventos. Es decir, si se detecta que el cursor está sobre un punto en cuestión se llamará a la función, y si sale del espacio que ocupa dicho punto llamará a otra para dejar de mostrar el *pop-up*.

```
1 entity.addEventListener('mouseenter', function () {  
2   this.setAttribute('scale', {x: 1.3, y: 1.3, z: 1.3});  
3   if(show_popup_condition){  
4     popUp = generatePopUp(point, properties);  
5     element.appendChild(popUp);  
6   }
```

```

7    ...
8  });
9
10 entity.addEventListener('mouseleave', function () {
11   this.setAttribute('scale', {x: 1, y: 1, z: 1});
12   if(show_popup_condition){
13     element.removeChild(popUp);
14   }
15 });

```

Iteración 5: Funciones que escuchan eventos para mostrar u ocultar el pop-up

A continuación se puede ver la función que genera el elemento *pop-up*. Simplemente se crea la entidad plano (nativa de A-Frame) con un tamaño predefinido y un texto con la información.

```

1 function generatePopUp(point, properties) {
2   ...
3   let entity = document.createElement('a-plane');
4   entity.setAttribute('position', {x: point['x'], y: point['y'], z: point
5     ['z']});
6   entity.setAttribute('height', '2');
7   entity.setAttribute('width', width);
8   entity.setAttribute('color', 'white');
9   entity.setAttribute('text', {
10     'value': 'DataPoint:\n\n' + text,
11     'align': 'center',
12     'width': 6,
13     'color': 'black'
14   });
15 }

```

Iteración 5: Código para generar el pop-up

Por último, si se cumple la condición de mostrar el *pop-up* llamamos a la función *generatePopUp* la cual simplemente genera un plano con la información del punto. Se puede ver el resultado en la Figura 3.9.

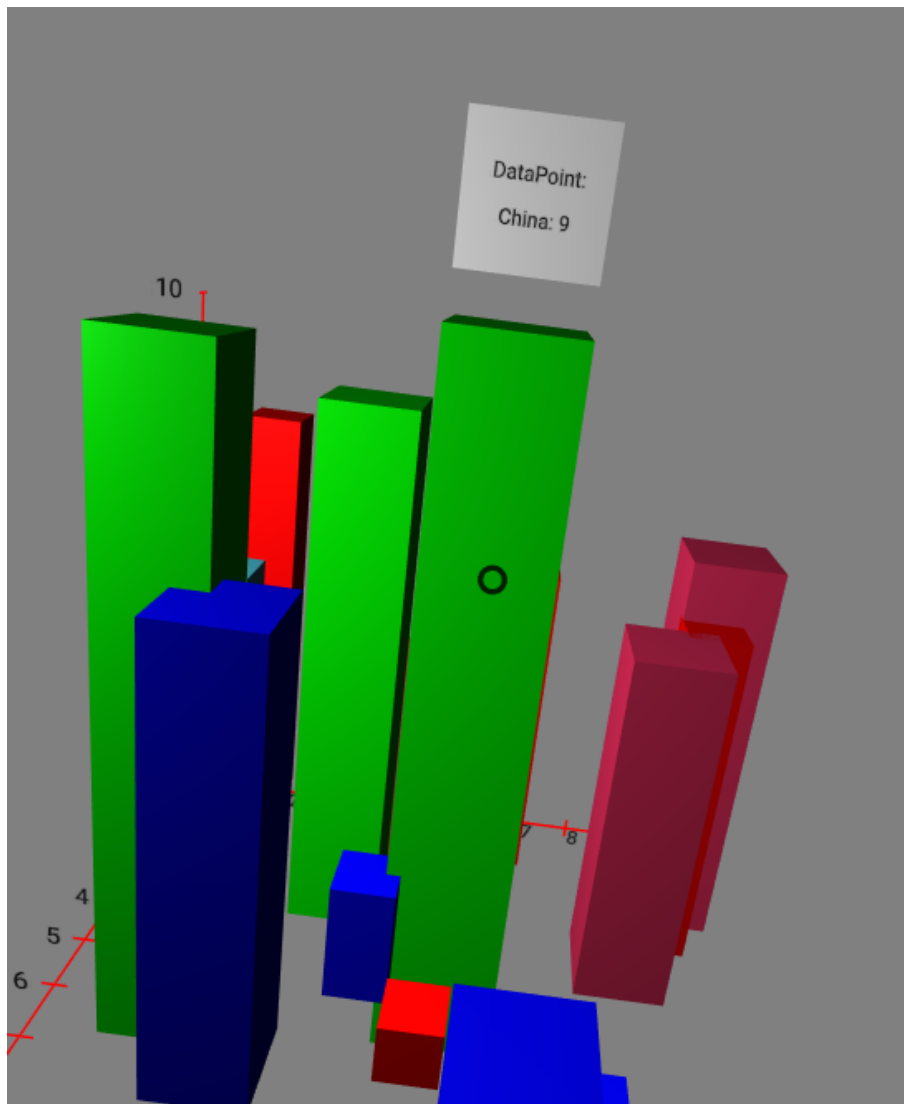


Figura 3.9: Iteración 5: Pop-up

El siguiente objetivo para esta iteración es incluir una leyenda para gráficos de tipo tarta. Para ello se sigue una estrategia similar que para el *pop-up*. Se comienza añadiendo varias propiedades al esquema del componente.

```

1 show_legend_info:      {type: 'boolean', default: false},
2 show_legend_position: {type: 'vec3', default: {x:0, y:0, z:0}},
3 show_legend_rotation: {type: 'vec3', default: {x:0, y:0, z:0}},
4 show_legend_title:    {type: 'string', default: 'Legend'},

```

Iteración 5: Propiedad del esquema para la leyenda

En este caso se usan cuatro propiedades. La primera, *show_legend_info*, se utiliza para mostrar o no la leyenda. Las otras propiedades son para establecer la posición y rotación. Al ser

un plano en el espacio, se facilita al usuario el poder colocarla donde necesite. Además se da la opción de utilizar un título para la leyenda, que por defecto es 'Leyenda'. En contraposición con el *pop-up* se da una mayor parametrización debido a que el *pop-up* hereda estos parámetros de sus puntos mientras que la leyenda no.

En cuanto al código, se utilizan cuatro funciones para generarla:

```

1 if(show_legend_condition && dataPoints.length > 0){
2   legend_properties = getLegendProperties(dataPoints, properties, element);
3   legend_title = generateLegendTitle(legend_properties);
4   legend_sel_text = generateLegendSelText(legend_properties, dataPoints[0],
      properties);
5   legend_all_text = generateLegendAllText(legend_properties, getLegendText(
      dataPoints, dataPoints[0], properties));
6   element.appendChild(legend_title);
7   element.appendChild(legend_sel_text);
8   element.appendChild(legend_all_text);
9 }

```

Iteración 5: Funciones para generar la leyenda

Primero obtenemos una recopilación de las propiedades que hemos establecido en el esquema. Seguidamente se genera el título de la leyenda y una entrada de texto para cada dato del gráfico. Por otro lado, se quiere mostrar de manera destacada el punto donde se tenga el cursor. Para ello, de la misma manera que se hizo para el *pop-up* se utilizan funciones que escuchan eventos. De esta manera si se selecciona un punto se destaca dicho texto para una mejor interpretación de los datos por parte del usuario.

A continuación vemos el código utilizado para generar lo comentado anteriormente:

```

1 function generateLegendTitle(legendProperties) {
2   let entity = document.createElement('a-plane');
3   entity.setAttribute('position', legendProperties.position_tit);
4   entity.setAttribute('rotation', legendProperties.rotation);
5   entity.setAttribute('height', '0.5');
6   entity.setAttribute('width', legendProperties.width);
7   entity.setAttribute('color', 'white');
8   entity.setAttribute('text__title', {
9     'value': legendProperties.title,
10    'align': 'center',

```



```

11     'width': '8',
12     'color': 'black'
13   });
14   return entity;
15 }

```

Iteración 5: Código para generar la leyenda

En la Figura 3.10 se observa el resultado final.

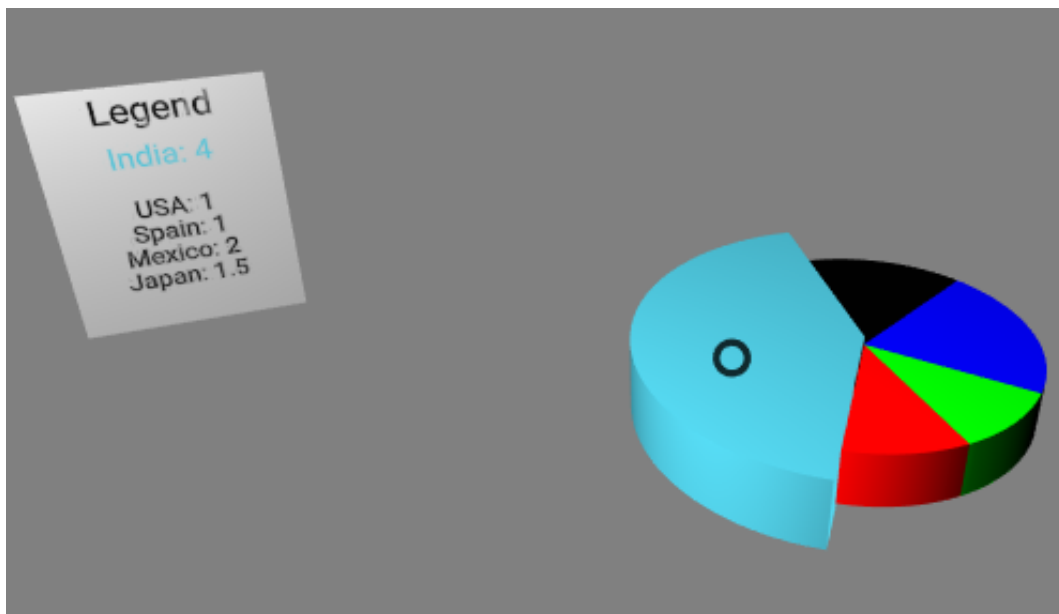


Figura 3.10: Iteración 5: Leyenda

Por otro lado, dado que A-Frame permite cambiar los componentes de manera dinámica mediante la función *update*, se quiere añadir una nueva funcionalidad para facilitar al usuario la tarea de cambiar los datos de los gráficos de manera activa.

Para este menester, se añade otro tipo de gráfico llamado *totem* cuya función no es otra que mostrar distintos ficheros de datos en un plano y si el usuario los selecciona cambien los gráficos de manera dinámica.

Se comienza añadiendo al esquema del componente dos nuevas propiedades que se ven a continuación:

```

1  entity_id_list:      {type: 'string', default: ''}
2  dataPoints_list:    {type: 'string', default: ''}

```

Iteración 5: Función para generar el totem

La propiedad *entity_id_list* será una lista de identificadores proporcionada por el usuario para cada *entity* de A-Frame que se añada en el HTML. Por otro lado, la propiedad *dataPoints_list* provee de un listado de ficheros. Por tanto, cuando el usuario seleccione un fichero de la lista de ficheros proporcionada y gracias a la lista de identificadores, podremos llamar a la función *update* de cada entidad pasándole los nuevos datos y que se refresque de manera dinámica.

Podemos observar en el siguiente fragmento de código, como en la función *update* se llama a la función *generateTotem* solo si el tipo de gráfico es *totem* y además no se le han pasado un fichero de datos debido a que no es un gráfico normal y, como se ha mencionado anteriormente, recibe un listado de ficheros y un listado de identificadores.

```

1 update: function (oldData) {
2   if(data.dataPoints) {
3     ...
4   }else if(data.type === "totem"){
5     generateTotem(data, this.el);
6   }
7 }

```

Iteración 5: Propiedades del esquema para el totem

Por último, se procede a explicar como se construye el *totem* mediante la función *generateTotem* y la subfunción *generateTotemSlice*.

```

1 function generateTotemSlice(properties, entity_id_list, dataPoints_path) {
2   let entity = document.createElement('a-plane');
3   ...
4   entity.addEventListener('click', function () {
5     let entity_list = entity_id_list.split(',');
6     for(let id of entity_list){
7       let myChart = document.getElementById(id);
8       let data = myChart.getAttribute("charts");
9       data.dataPoints = dataPoints_path;
10      myChart.setAttribute('charts', data);
11    }
12  });
13  return entity;
14 }
15

```

```

16 function generateTotem(properties, element) {
17     ...
18     for(let myDataPoints in dataPoints_list){
19         ...
20         generateTotemSlice(dataProperties, properties.entity_id_list,
21             dataPoints_list[myDataPoints]);
22     }
23
24 }

```

Iteración 5: Funciones generateTotem y generateTotemSlice

Como se ve en la función *generateTotem*, por cada fichero de datos de la lista de archivos se genera lo que se denomina un *totem slice*. Esta porción de totem, no es más que un plano con el nombre del fichero al que se le asigna una función que escucha el evento *click*. Cada vez que se haga un *click*, se recopilan todos los identificadores de los gráficos y se le asigna el nuevo fichero de datos seleccionado. A-Frame es el encargado de llamar a la función *update* de cada gráfico cambiando así sus datos de manera dinámica.

A continuación y relacionado con el punto anterior para la carga dinámica de datos, uno de los últimos cambios que se han realizado en la librería es la posibilidad de obtener los datos con llamadas AJAX y no únicamente con ficheros JSON. Hoy en día, en cualquier entorno o aplicación web, lo común es obtener los datos mediante llamadas AJAX en incluso ir actualizando los mismos con dichas llamadas. Por ello, se ha dotado a la librería de esta funcionalidad. Gracias a podrá ser utilizada con mayor versatilidad en cualquier aplicación web.

A continuación se muestra la parte de código encargada de gestionar ficheros u objetos de datos provenientes de llamadas AJAX o variables:

```

1 update: function (oldData) {
2     ...
3     if(data.dataPoints)
4         if (data.dataPoints.constructor === ([{}]).constructor) {
5             this.onDataLoaded(this, data.dataPoints, true);
6         }else if(data.dataPoints.constructor === "".constructor){
7             try{
8                 this.onDataLoaded(this, JSON.parse(data.dataPoints), true);

```

```

9      }catch(e) {
10         this.loader.load(data.dataPoints, this.onDataLoaded.bind(this,
11         false));
12     }
13 }

```

Iteración 5: Código gestión de Objetos o ficheros

Como se puede ver, simplemente se comprueba el tipo de objeto que se le pasa a la función update de A-Frame. Si es un JSON no recibe tratamiento, si es un String se parsea a JSON y si es un fichero se obtienen los datos con la función loader de ThreeJS descrita anteriormente.

Finalmente y como último punto, se provee de un ejemplo para la visualización de la librería con dispositivos de Realidad Virtual. Para ello simplemente es necesario añadir a nuestro código HTML las siguientes propiedades que nos ofrece la librería de A-Frame.

```

1 <a-scene background="color: grey" cursor="rayOrigin:mouse"> <!-- Cursor
   mouse clickable. -->
2 <a-light type="point" intensity="1" position="-2 10 10"></a-light>
3
4 <a-entity charts="type: bar;      dataPoints: ../data/dataPositive.json"
   position="-50 0 10" rotation="0 30 -10"></a-entity>
5
6 <a-entity movement-controls="fly: true" position="0 30 30" rotation="-24
   15 0"> <!-- Camera fly movement. -->
7 <a-camera position="4.2 -1 1.5" rotation="0 -1 0">
8 </a-camera>
9 <a-entity laser-controls="hand: right"></a-entity> <!-- Control for
   Oculus Rift. -->
10 </a-entity>
11 </a-scene>

```

Iteración 5: Código para integración con dispositivos VR

Como se ha podido ver, añadiendo la entidad *laser-control* de A-Frame dotamos al dispositivo VR del control necesario para poder interactuar con la librería. Además para facilitar la visualización, se le añade la entidad *movement-control* con el parámetro *fly* y valor *true*. Esto

nos permite que la cámara se mueva a la par del movimiento del usuario que utilice el dispositivo de realidad virtual.

Capítulo 4

Resultados

4.1. Arquitectura general

En esta sección se explica la foto final de la arquitectura con la que se ha desarrollado la librería. Como se ha mencionado previamente, gracias a la herramienta *angle* de A-Frame se genera un arquetipo con los ficheros necesarios para desarrollar un componente. En la Figura 4.1 se observa el resultado que se describirá a continuación.

- **package.json:** Este fichero, tal y como se ha mencionado anteriormente en esta memoria, contiene los metadatos del proyecto. Además, se utiliza como gestor de dependencias y es el encargado de definir las etapas de desarrollo, despliegue, tests, etc.
- **index.js:** Este archivo contiene el código JavaScript que se ha desarrollado en este proyecto. Como se ha visto en el punto anterior, aquí se registra el componente de A-Frame así como todas sus funcionalidades.
- **index.html:** Este fichero es utilizado como página de ejemplo¹ para mostrar la funcionalidad del componente.
- **node_modules:** En esta carpeta se almacenan todas las dependencias del proyecto reflejadas en el archivo *package.json* o aquellas que se utilicen de manera indirecta.
- **tests:** Esta ubicación contiene los tests del proyecto. Su propósito es el de probar la funcionalidad desarrollada, mejorar la calidad del código y evitar posibles errores.

¹<https://adrixp.github.io/aframe-charts-component/>

- **examples:** En esta carpeta se almacenan todos los ejemplos que se ven en el *index.html*. En el siguiente apartado se describirá uno a uno cada ejemplo.
- **README.md:** Este archivo sirve para describir el funcionamiento y puesta en marcha del componente. En el se realiza una breve descripción y se pone algún ejemplo de uso para mostrar el software de un vistazo rápido.
- **.gitignore y .npmignore:** Estos ficheros se utilizan para ignorar ciertos directorios o archivos que no deben subir al repositorio de código.

```
1 +---dist
2   |   aframe-charts-component.js
3 +---examples
4   |
5 +---img
6   |
7 +---node_modules
8   |
9 +---tests
10  |   helpers.js
11  |   index.test.js
12  |   karma.conf.js
13  |   __init.test.js
14  |
15  |   .gitattributes
16  |   .gitignore
17  |   .npmignore
18  |   .travis.yml
19  |   .index.html
20  |   .index.js
21  |   LICENSE
22  |   package.json
23  |   package-lock.json
24  |   README.md
```

Figura 4.1: Resultados: Arquitectura Final

4.2. Funcionamiento de la Librería

En este apartado se describe el funcionamiento de la librería. Se pondrá un ejemplo de uso y una imagen por cada gráfico y funcionalidad que soporta la librería.

4.2.1. Gráfico de burbujas

Este tipo de gráfico consiste en una variación de un gráfico de dispersión. En el cual, los datos representados por puntos se reemplazan por burbujas o esferas. El tamaño de dichas burbujas puede representar una dimensión adicional.

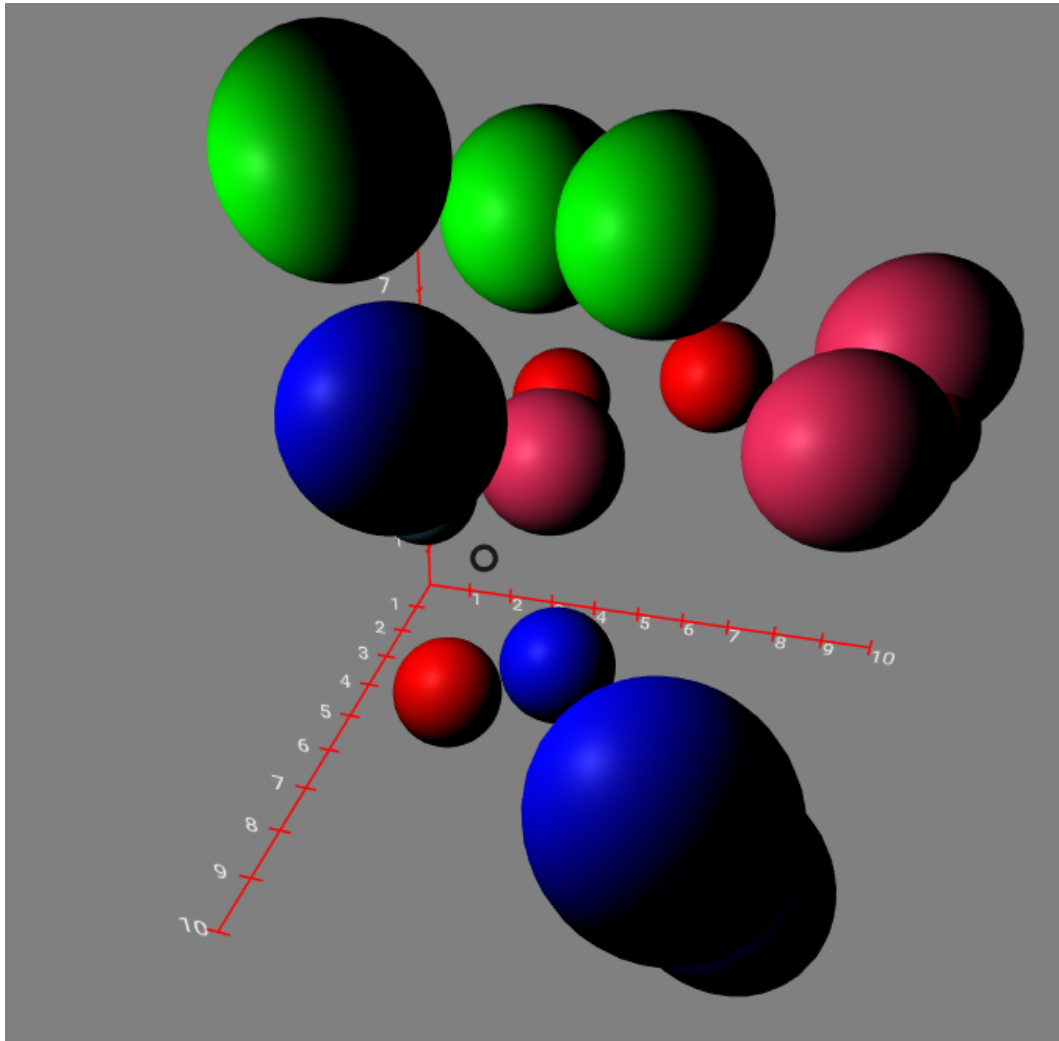


Figura 4.2: Resultados: Gráfico de burbujas

Gracias a este proyecto, únicamente son necesarias las siguientes líneas de código HTML para ver el gráfico de burbujas.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
```

```

5      <script src="../../dist/aframe-charts-component.min.js"></script>
6  </head>
7  <body>
8      <a-scene background="color: grey">
9          <a-light type="point" intensity="1" position="-2 10 10"></a-
light>
10 <!--Important Line-->
11     <a-entity charts="type: bubble; dataPoints: ../data/dataPositive.json
"></a-entity>
12 <!--End Important Line-->
13         <a-entity position="2 10 14" rotation="-30 15 0">
14             <a-camera position="3 -1 4" rotation="0 -1 0">
15                 <a-cursor></a-cursor>
16             </a-camera>
17         </a-entity>
18     </a-scene>
19 </body>
20 </html>

```

Como vemos en el código, se utiliza únicamente la última versión de la librería de A-Frame y la del componente desarrollado en este proyecto. Se construye una escena de realidad virtual con el elemento `<a-scene>` y dentro de ella se utiliza el elemento `<a-light>` para dar luminosidad a la escena y `<a-camera>` el cual indica el punto de vista inicial del usuario. Por último se añade el elemento `<a-entity>` con los atributos *type* y *dataPoints*. Este elemento es interpretado por el módulo desarrollado en este proyecto y genera un gráfico de tipo burbuja con la representación de los puntos descritos en el fichero *dataPositive.json* en el espacio tridimensional.

En adelante, y para el resto de resultados que veremos a continuación en esta memoria, nos centraremos en la siguiente línea de código. El resto de código HTML no va a cambiar, únicamente se sustituirá la siguiente línea de código por las que veremos a partir del punto 4.2.2:

```

1 <a-entity charts="type: bubble; dataPoints: ../data/data.json"></a-entity>

```

4.2.2. Gráfico de barras

La representación clásica del gráfico de barras se utiliza para resumir un conjunto de datos mediante categorías. Se muestran los datos usando distintas barras, las cuales representan una categoría concreta. Mediante la altura se puede cuantificar la frecuencia de cada categoría. Además gracias a la representación en 3D se tiene una dimensión extra donde podemos enriquecer nuestro gráfico dotándole de mayor información.

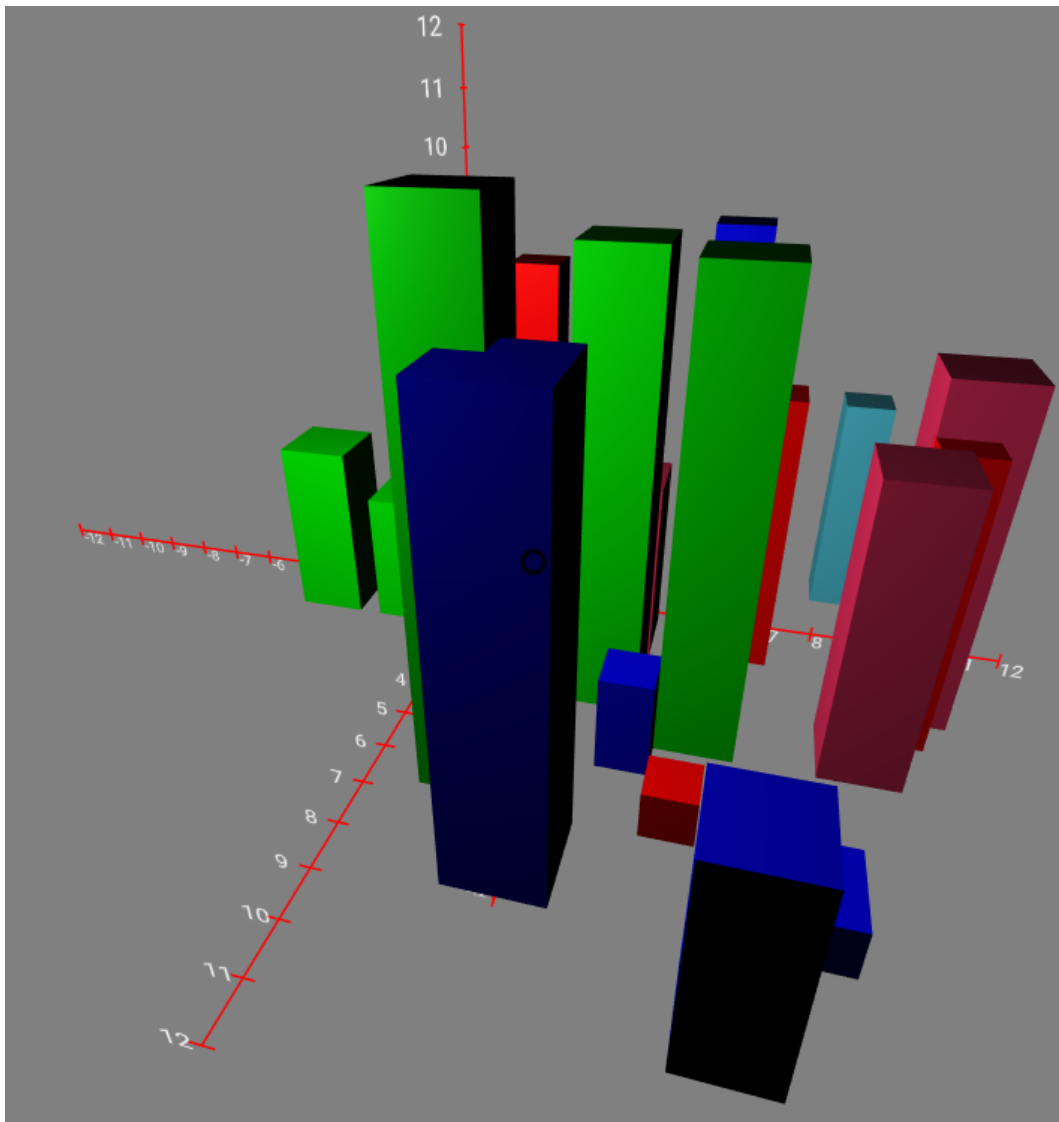


Figura 4.3: Resultados: Gráfico de barras

Como se menciona anteriormente, únicamente hay que cambiar la siguiente línea de código respecto al apartado 4.2.1. Únicamente se añade el elemento `<a-entity>` con los atributos *type* y *dataPoints*. Este elemento es interpretado por el módulo desarrollado en este proyecto y genera un gráfico de barras con la representación de los puntos descritos en el fichero *data.json* en el espacio tridimensional. Además se añade otro atributo dispuesto por la librería el cual es la longitud de los ejes que fijamos al valor 12 mediante la propiedad *axis_length*.

```
1 <a-entity charts="type: bar; dataPoints: ../data/data.json; axis_length: 12  
  "></a-entity>
```

4.2.3. Gráfico de barras cilíndricas

Al igual que el apartado anterior, la representación clásica del gráfico de cilindros se utiliza para resumir un conjunto de datos mediante categorías. Además gracias a la representación en 3D se tiene una dimensión extra donde podemos enriquecer nuestro gráfico dotándole de mayor información. La única diferencia es que la representación de las barras de manera cilíndrica.

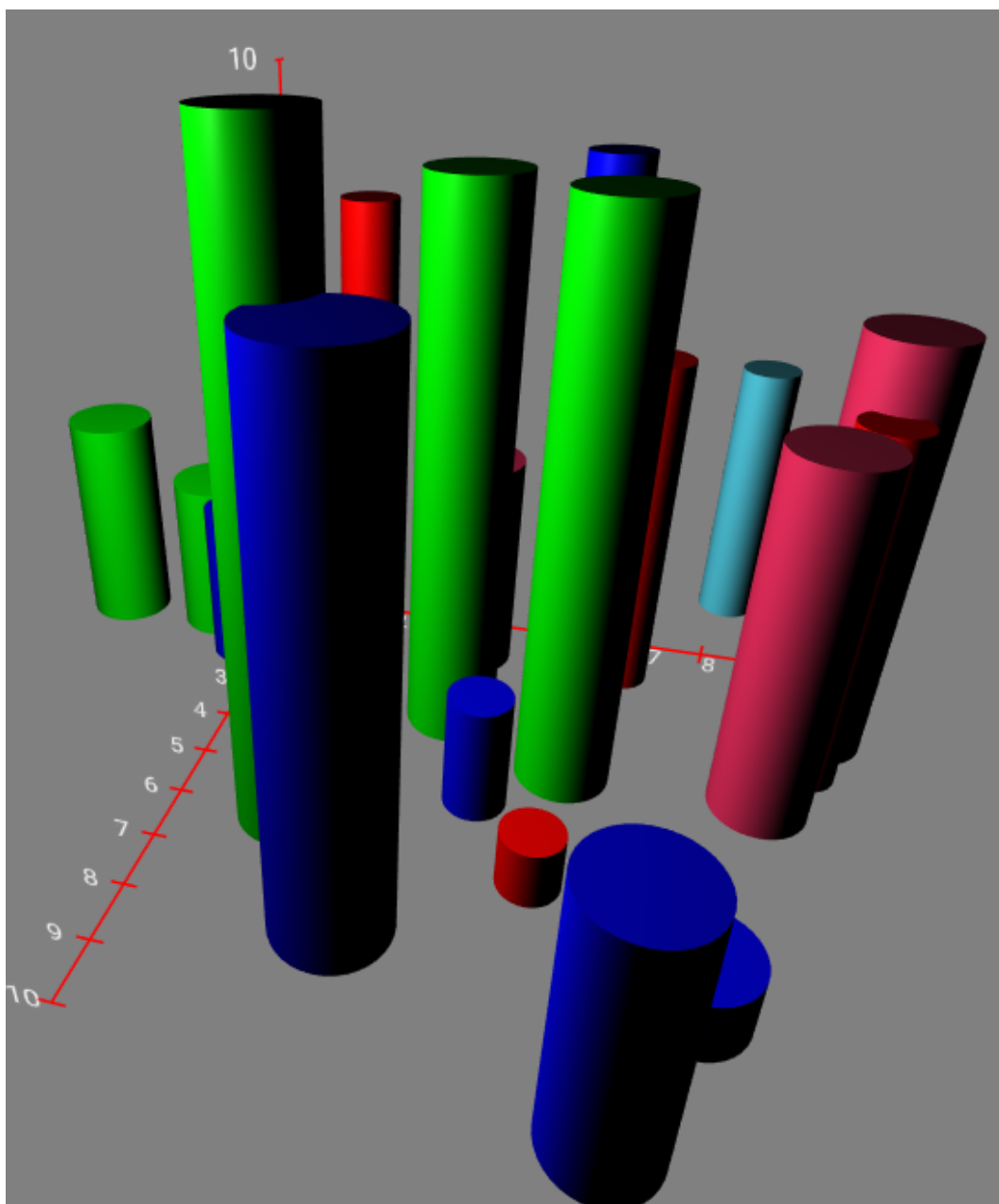


Figura 4.4: Resultados: Gráfico de barras cilíndricas

Como se menciona en el apartado 4.2.1, únicamente hay que cambiar la siguiente línea de

código para visualizar este gráfico. Se añade el elemento `<a-entity>` con los atributos *type* y *dataPoints*. Este elemento es interpretado por el módulo desarrollado en este proyecto y genera un gráfico barras de tipo cilíndrico con la representación de los puntos descritos en el fichero *data.json* en el espacio tridimensional. Además se añade otro atributo dispuesto por la librería que no permite que los ejes sean negativos *axis_negative*.

```
<a-entity charts="type: bubble; dataPoints: ../data/data.json;  
  axis_negative: false"></a-entity>
```

4.2.4. Gráfico de tarta

El gráfico circular también conocido como "gráfico de tarta"^{es} un tipo de visualización clásica en el mundo de la estadística utilizado para representar porcentajes y proporciones. Gracias a la dimensión extra que nos proporciona la representación en 3D tenemos una nueva dimensión donde podemos enriquecer nuestro gráfico dotándole de mayor información.

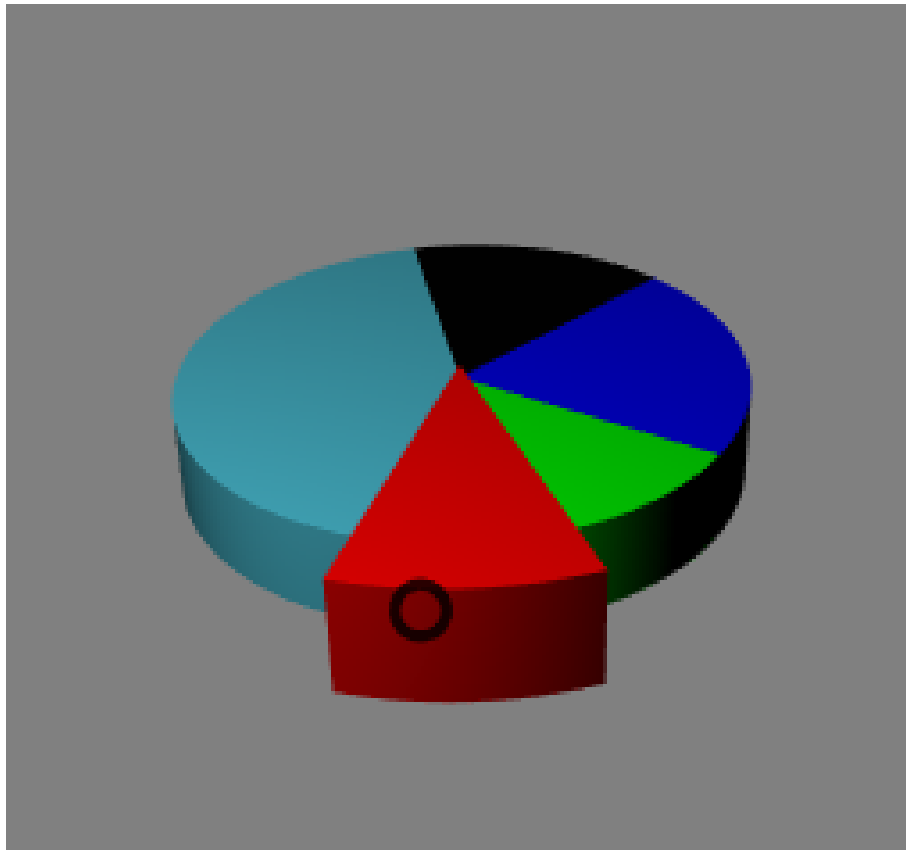


Figura 4.5: Resultados: Gráfico de tarta

Como se menciona anteriormente, únicamente hay que cambiar la siguiente línea de código para visualizar este gráfico. Se añade el elemento `<a-entity>` con los atributos `type` y `dataPoints`. Este elemento es interpretado por el módulo desarrollado en este proyecto y genera un gráfico de tarta. Además se añade otro atributo dispuesto por la librería que determina el radio de la tarta `pie_radius`.

```
<a-entity charts="type: pie; dataPoints: ../data/dataPie.json; pie_radius:  
3"></a-entity>
```

4.2.5. Gráfico de doughnut

Este tipo de gráfico es similar al del apartado 4.2.4. La diferencia es simplemente visual ya que se representa de manera cilíndrica. Gracias a la dimensión extra que nos proporciona la representación en 3D tenemos una nueva dimensión donde podemos enriquecer nuestro gráfico dotándole de mayor información.

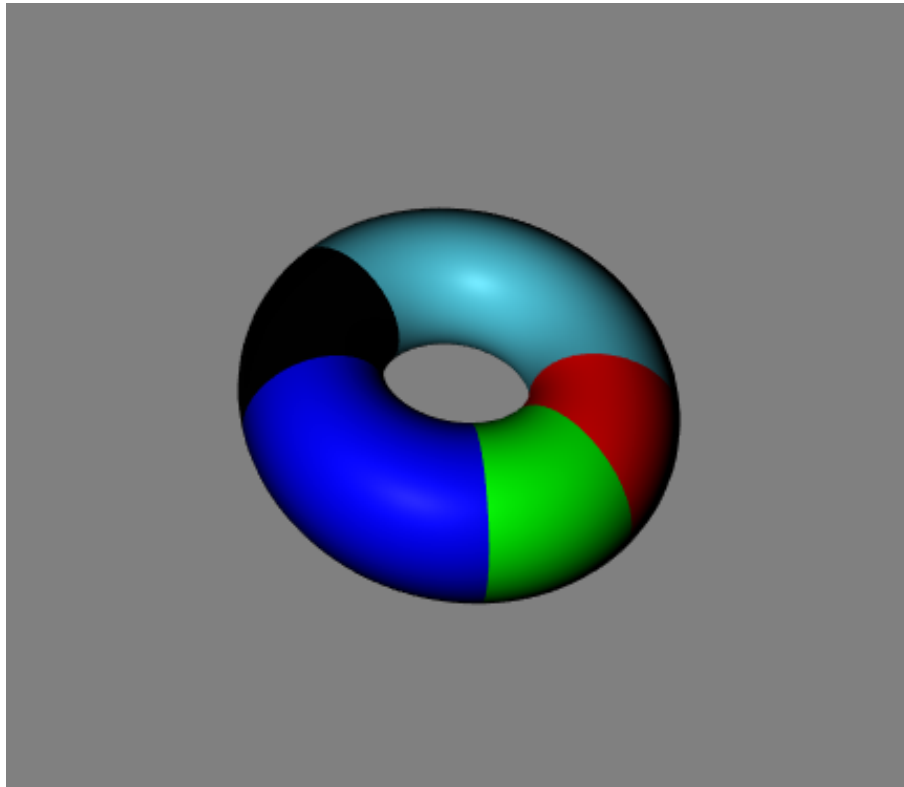


Figura 4.6: Resultados: Gráfico de doughnut

Como se menciona anteriormente, únicamente hay que cambiar la siguiente línea de código para visualizar este gráfico. Se añade el elemento `<a-entity>` con los atributos `type` y `dataPoints`. Además se añade un nuevo atributo `pie_doughnut` respecto al apartado anterior, que determina que sea de tipo doughnut.

```
1 <a-entity charts="type: pie; dataPoints: ../data/dataPie.json; pie_radius:  
    3; pie_doughnut: true">  
2 </a-entity>
```


4.2.6. Totem

Este elemento es proporcionado por el módulo desarrollado en este proyecto. Su objetivo es cambiar los datos del gráfico de manera dinámica utilizando distintas fuentes de datos. Simplemente con un *click* si estamos en un navegador, o con la interacción *touch* si estamos en un dispositivo de realidad virtual cambiaría nuestro gráfico.

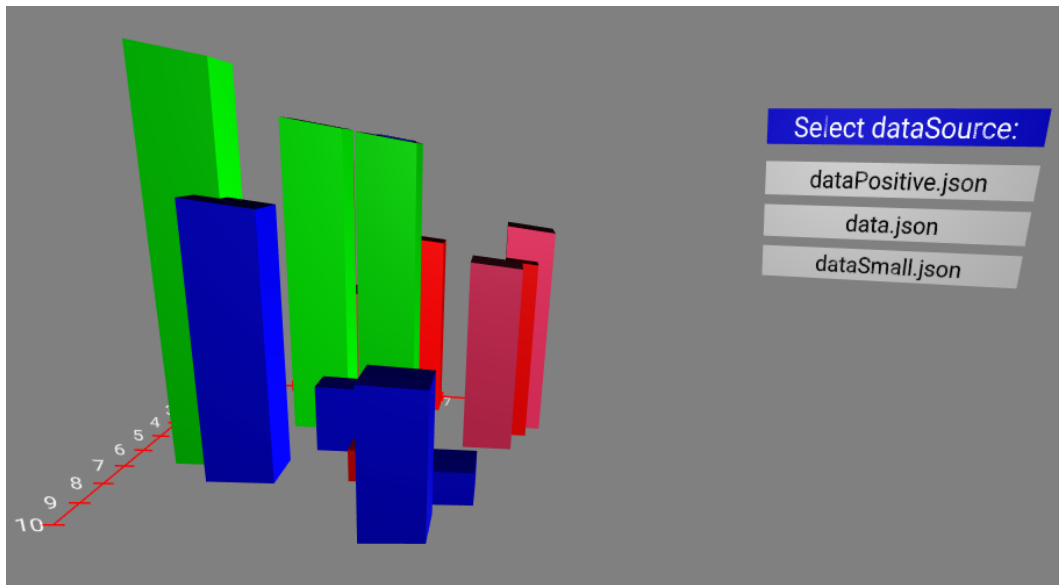


Figura 4.7: Resultados: Totem

Para añadir este elemento únicamente hay que cambiar la siguiente línea de código. Se añade el elemento `<a-entity>` con los atributos `type`, `entity_id_list` y `dataPoints_list`. Es decir, una lista de identificadores de los gráficos que van a cambiar y un listado de ficheros o variables que contengan los nuevos datos a modificar dinámicamente.

```
1 <a-entity position="1 15 10" charts='type: totem; entity_id_list: barId;  
2   dataPoints_list: {"dataPositive.json": "../data/dataPositive.json",  
3   "data.json": "../data/data.json", "dataSmall.json": "../data/dataSmall.  
   json"}'>  
4 </a-entity>
```

4.2.7. Parámetros de los ejes

Como se menciona en la Iteración 4 dentro del capítulo de diseño e implementación, el módulo desarrollado en este proyecto nos ofrece una gran cantidad de parámetros para poder configurar los gráficos al gusto del usuario. Además ofrece herramientas para obtener una mayor información de los mismos. Finalmente tal y como se ve en la Figura 4.8 podemos añadir a nuestra escena N visualizaciones.

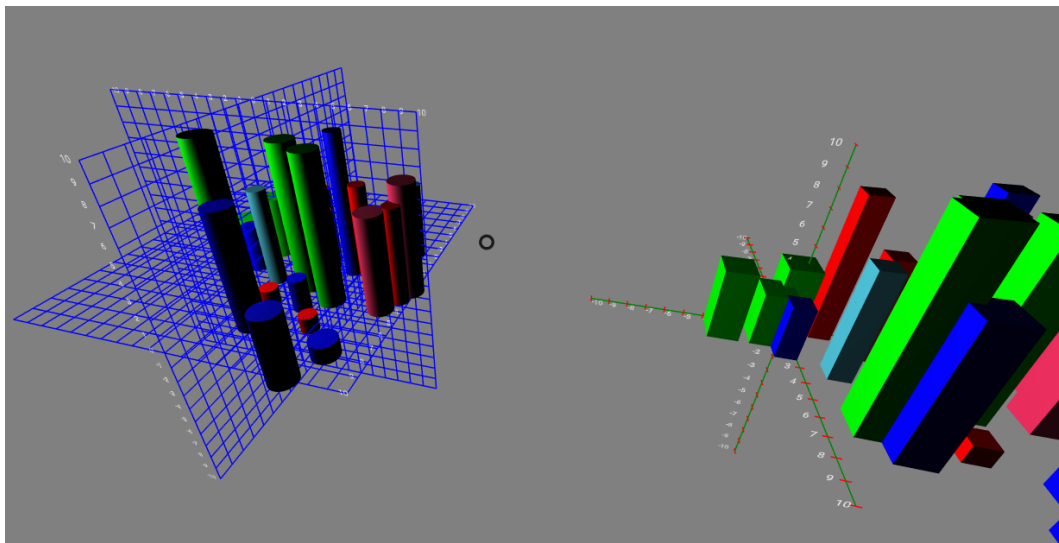


Figura 4.8: Resultados: Ejemplo de ejes parametrizables

Como se menciona en el punto 4.2.1, únicamente hay que añadir las siguientes líneas de código para visualizar los dos gráficos. Se añaden dos elementos `<a-entity>` con los atributos `type` y `dataPoints`. Además se añaden los atributos `axis_color`, `axis_length`, `axis_grid`, `axis_negative` y `axis_position`. Estos atributos permiten modificar el color y la longitud de los ejes, permitir que los ejes sean de tipo rejilla, que sean negativos así como su posición respectivamente.

```

1 <a-entity charts="dataPoints: ../data/data.json; type: cylinder; axis_color
  : blue; axis_length: 10;
2   axis_grid: true"></a-entity>
3 <a-entity charts="dataPoints: ../data/dataAxis.json; type: bar; axis_color:
  green; axis_length: 10;
4   axis_negative: true; axis_position: 30 0 0"></a-entity>

```

4.2.8. Pop up

El módulo desarrollado en este proyecto permite el uso de un panel informativo o *pop up* para una mejor comprensión de los datos.

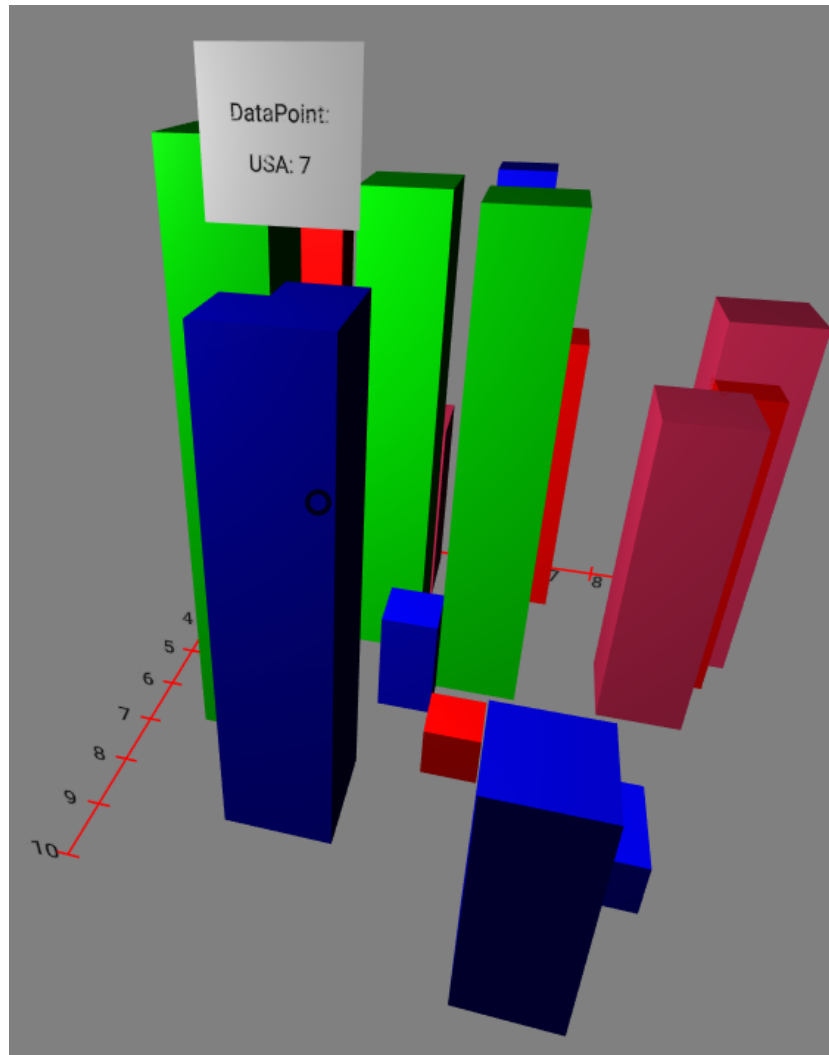


Figura 4.9: Resultados: Gráfico con Pop-up

Como se menciona anteriormente, únicamente hay que añadir la siguiente línea de código para visualizar el pop up. Se añade al elemento `<a-entity>` el atributo `show_popup_info` que permite visualizar dicha ventana emergente por cada elemento del gráfico, enriqueciendo así la información mostrada por el gráfico.

```
<a-entity charts="type: bar; dataPoints: ../data/dataPopUp.json;  
  axis_text_color: black; show_popup_info: true"></a-entity>
```

4.2.9. Leyenda

El módulo desarrollado en este proyecto permite el uso de una leyenda para identificar con mayor claridad los distintos elementos en un gráfico, tales como colores y su valor.

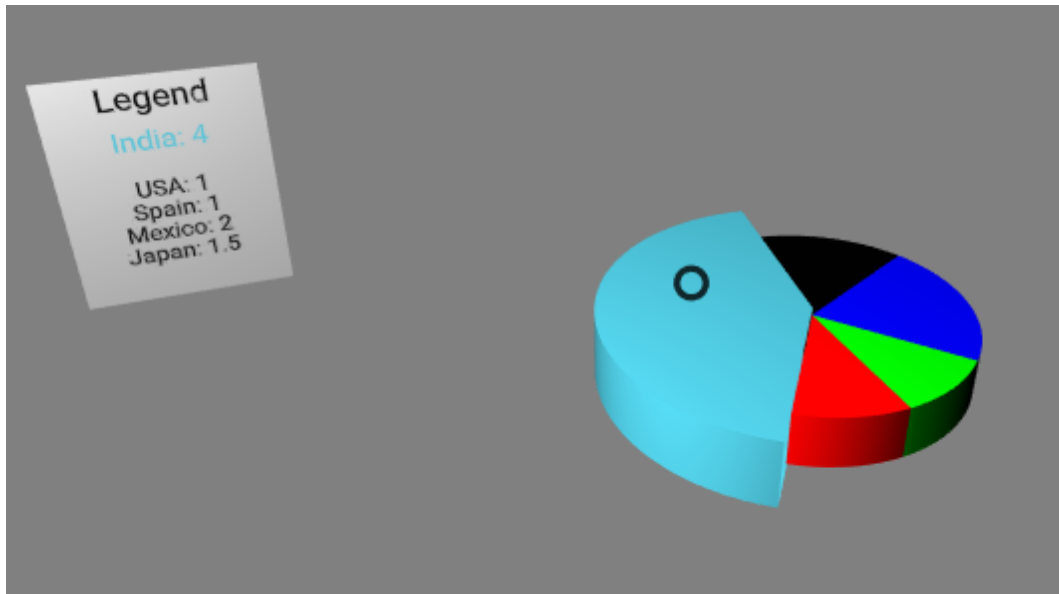


Figura 4.10: Resultados: Gráfico con leyenda

Como se menciona en el punto 4.2.1, únicamente hay que añadir la siguiente línea de código para visualizar la leyenda. Se añade al elemento `<a-entity>` los atributos `show_legend_info` que permite visualizar dicha ventana emergente, enriqueciendo así la información mostrada por el gráfico. `Show_legend_position` y `show_legend_rotation` se utilizan para posicionar la leyenda en el espacio tridimensional deseado por el usuario

```
1 <a-entity charts="type: pie; dataPoints: ../data/dataLegend.json;
  pie_radius: 3; show_legend_info: true; show_legend_position: -2 5 7;
2 show_legend_rotation: 0 35 0" position="5 0 0"></a-entity>
```

4.2.10. Dashboards

Este módulo ofrece además la posibilidad de colocar distintas visualizaciones en una misma escena. Cada una de ellas representando los mismos o diferentes datos. Además pueden trabajar de manera simultánea, es decir, están preparados para escuchar eventos y actualizar su información de manera dinámica.

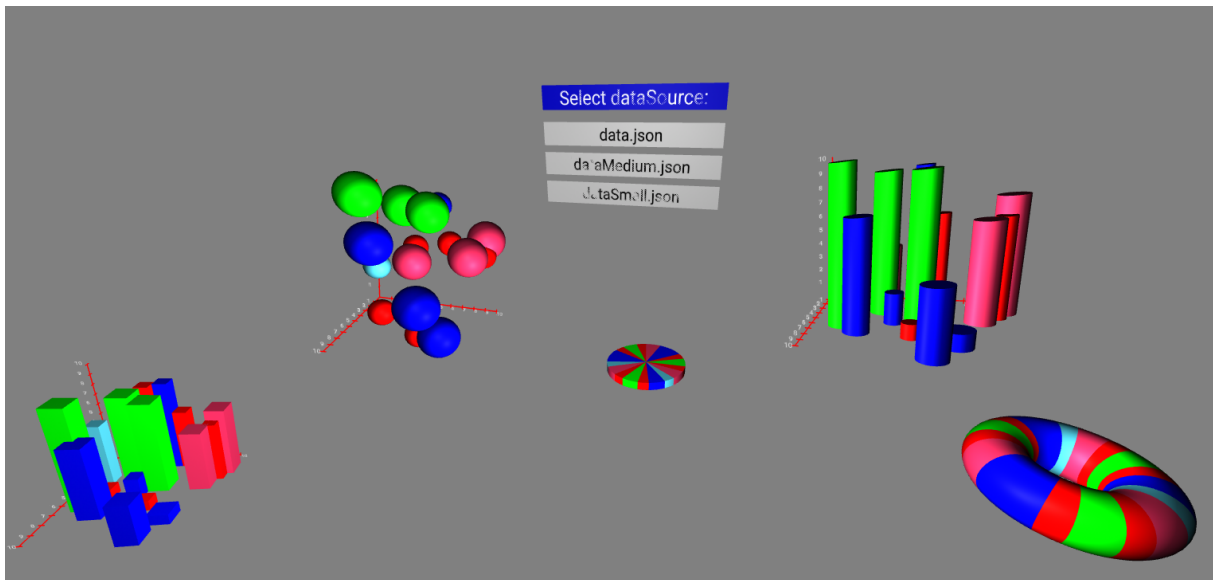


Figura 4.11: Resultados: Dashboard

```

1 <a-scene background="color: grey" cursor="rayOrigin:mouse">
2   <a-entity id="barId"          charts="type: bar;      dataPoints: ../
    data/dataPositive.json" position="-50 0 10" rotation="0 30 -10"></a-
    entity>
3   <a-entity id="bubbleId"       charts="type: bubble;   dataPoints: ../
    data/dataPositive.json" position="-25 15 0" rotation="0 20 -10"></a-
    entity>
4   <a-entity id="pieId"          charts="type: pie;      dataPoints: ../
    data/dataPositive.json; pie_radius: 5" position="-5 0 -20"></a-entity>
5   <a-entity id="cylinderId"     charts="type: cylinder; dataPoints: ../
    data/dataPositive.json" position="12 17 -5" rotation="0 -20 10"></a-
    entity>
6   <a-entity id="pieDoughnutId"  charts="type: pie;      dataPoints: ../
    data/dataPositive.json; pie_radius: 5; pie_doughnut: true" position="35
    0 -10"></a-entity>
7

```

```

8  <a-entity position="1 15 10" charts='type: totem; entity_id_list: barId,
    bubbleId, pieId, cylinderId, pieDoughnutId;
9  dataPoints_list: {"dataPositive.json": "../data/dataPositive.json", "data
    .json": "../data/data.json", "dataSmall.json": "../data/dataSmall.json"}'
    >
10 </a-entity>
11 </a-scene>

```

Código HTML de ejemplo para Dashboard

Como se puede ver en el código, solo son necesarias estas líneas de código para representar un Dashboard, en 3 dimensiones y visionado mediante realidad virtual, que trabaje de manera dinámica y conjuntamente para mostrar los datos.

4.2.11. LLamadas Ajax

Además de la representación de datos a partir de ficheros JSON. El módulo creado en este proyecto permite la representación de datos de otras fuentes externas mediante llamadas AJAX.

A continuación tenemos un ejemplo de uso. Se mantiene toda la estructura HTML que se viene utilizando hasta ahora y se deben añadir las siguientes líneas HTML y JavaScript.

```

1 <a-entity id="myChart"></a-entity>

```

Código HTML de ejemplo

```

1 $.get("/examples/data/data.json", function( data ) {
2   let myChart = document.getElementById("myChart");
3   myChart.setAttribute('charts', {type: "bar", dataPoints: data});
4 });

```

Código Ajax de ejemplo

Este apartado es muy útil ya que permite una integración con distintas bases de datos o llamadas externas mediante servidor. En el apartado líneas futuras de esta memoria, se deberá utilizar este ejemplo para poder desarrollarlo con éxito.

4.2.12. API

Propiedad	Descripción	Valor por defecto
type	Tipo de gráfico	bubble
dataPoints	Ruta al fichero JSON de entrada, asset o array de datos	../data/data.json
axis_position	Posición de los ejes	{x:0, y:0, z:0}
axis_visible	Si es falso, no se muestran los ejes	true
axis_color	Color de los ejes	red
axis_length	Longitud de los ejes. Por defecto son adaptativos.	0
axis_tick_separation	Separación de los ticks de los ejes	1
axis_tick_length	Longitud de los ticks de los ejes	0.2
axis_tick_color	Color de los ticks de los ejes	red
axis_negative	Habilita ejes negativos.	true
axis_grid	Habilita rejilla en los ejes	false
axis_grid_3D	Habilita rejilla tridimensional en los ejes	false
axis_text	Numeración de los ejes	true
axis_text_color	Color de la numeración de los ejes	white
axis_text_size	Tamaño de texto de la numeración	10
pie_radius	Radio del gráfico de tarta	1
pie_doughnut	Si en vez de gráfico con forma de tarta se quiere con forma de doughnut	false
show_popup_info	Mostrar pop up	false
show_legend_info	Mostrar leyenda	false
show_legend_position	Posición de la leyenda	{x:0, y:0, z:0}
show_legend_rotation	Rotación de la leyenda	{x:0, y:0, z:0}
show_legend_title	Título de la leyenda	Legend
entity_id_list	Lista de identificadores separada por comas.	barId,pieId
dataPoints_list	Lista de JSON de datos.	{ "data1": "data.json", "data2": "data2.json"}

Capítulo 5

Conclusiones

5.1. Consecución de objetivos

El objetivo principal, alcanzado en este proyecto, ha sido el de crear una librería para la visualización de datos en 3D en cualquier dispositivo con un navegador web y dispositivos de realidad virtual. Por otro lado, se ha conseguido la participación de la comunidad, la cual ha pedido nuevas funcionalidades, resolución de errores e incluso utilizado esta librería en otros proyectos.

Por otra parte y de manera más específica, se ha logrado concluir con éxito varios de los objetivos previstos. Se ha conseguido proporcionar distintos tipos de visualizaciones en tres dimensiones. También se ha dotado de gran versatilidad al usuario a la hora de configurar los ejes o gráficos gracias a la gran cantidad de parámetros proporcionados.

Se ha dotado de distintas herramientas para mejorar la lectura, filtrado y comprensión de los datos. Además, se ha facilitado la manera de consumir dichos datos por la librería, ya sea mediante ficheros o variables que contengan un formato JSON o mediante llamadas AJAX. También se han realizado pruebas de rendimiento y control de calidad del código cumpliendo con el objetivo de tener una librería escalable y fácilmente mantenible. Se ha dotado de una página web con ejemplos y casos de uso. Además se han proporcionado ejemplos para el uso de esta librería en dispositivos de realidad virtual.

Se ha podido dotar a la herramienta del potencial suficiente para representar en una misma escena varios gráficos y que pudieran ser actualizados de manera dinámica mediante la escucha de eventos.

Por último y personalmente considerado como objetivo más importante. Se han aprendido conceptos y nuevas tecnologías en un campo apasionante y en pleno crecimiento. En los siguientes apartados se procede a describir los conocimientos aplicados y aquellos que se han aprendido con este proyecto. Todo ello siguiendo los estándares y las *best practices* propuestas por la comunidad.

5.2. Aplicación de lo aprendido

A lo largo de la carrera, me vienen a la mente varias asignaturas protagonistas que han facilitado mucho la consecución de este proyecto. La mayoría de ellas, por razones obvias, están relacionadas con el aprendizaje y desarrollo de *software*. Entre ellas, se destacan las siguientes:

- **Fundamentos de la Programación:** Asignatura base donde se aprenden los conceptos básicos y los pilares fundamentales de la programación. Se aprende desde qué es un programa, hasta lo que es un algoritmo, un diagrama de flujo, variables, estructuras de control y un largo etc. Todo ello, obviamente, aplicado en la base de este proyecto.
- **Programación de Sistemas de Telecomunicación:** De esta asignatura se aplican los conceptos aprendidos sobre modularidad y encapsulación. Además se aplican los conocimientos sobre programar en diferentes niveles de una jerarquía de protocolos.
- **Sistemas Operativos:** En esta asignatura se comprende el funcionamiento de un sistema operativo así como su uso y diseño. Es otra de las asignaturas base donde se apoyan muchas de las herramientas de desarrollo utilizadas en este proyecto.
- **Procesamiento Digital de la Información:** Aquí se han obtenido conocimientos fundamentales sobre el tratamiento de la información y de los datos. Considero que es una de las asignaturas más completas de la carrera donde además se aprenden técnicas de procesamiento de datos y de *machine learning* perfectamente aplicables a cualquier librería de tratamiento de datos.
- **Servicios y Aplicaciones Telemáticas:** Asignatura clave para el proyecto donde se adquiere conocimiento de arquitecturas y protocolos de comunicación para servicios y aplicaciones Web. Se han aplicado innumerables conocimientos como uso de ficheros JSON,

Gestión de HTML en el navegador, motor DOM, tecnologías AJAX, frameworks y construcción de aplicaciones.

- **Ingeniería de Sistemas de Información:** De esta asignatura se aplican conceptos como toma de requisitos, pruebas de software, mantenimiento y metodología.

5.3. Lecciones aprendidas

Además de los conocimientos aplicados y adquiridos durante la carrera que se mencionan en el apartado anterior, en este proyecto se han reforzado y aprendido una gran cantidad de tecnologías que hasta ahora desconocía. Entre ellas se destacan las siguientes.

1. Aprender a utilizar *WebPack* como empaquetador de módulos estáticos.
2. Uso de *NodeJS* como servidor web y *npm* como gestor de paquetes y dependencias.
3. Se ha aprendido el funcionamiento básico de WebGL y OpenGL. Tecnologías en las cuales se basan todas las herramientas de desarrollo 3D en el navegador.
4. Descubrir y formar parte de la comunidad desarrollo de aplicaciones en realidad virtual.
5. Se ha aprendido el uso del *framework* de A-Frame de manera exhaustiva, así como ThreeJS (librería en la que se basa A-Frame) de manera básica.

5.4. Trabajos futuros

Siguiendo con las líneas principales marcadas en este proyecto donde se pretendía crear una librería fácilmente escalable, mantenible y sencilla de utilizar. Por tanto, siguiendo con esta filosofía se describen a continuación las líneas futuras que se propone abordar.

- Integración con bases de datos no relacionales como Elasticsearch. Gracias a ello, la librería podría ser utilizada en proyectos para visualizar datos en *near real time* y con grandes volúmenes de datos.
- Integración en proyectos con bases de datos relacionales tales como PostgreSQL o MySQL.

- Aumentar la customización y parametrización de los gráficos. Por ejemplo, se podrían incluir animaciones, eventos, zoom, estilos, filtrado y manipulación de los datos, etc.
- Incrementar el número de visualizaciones incluyendo alguna como de mapas de calor, gráficos de radar, gráficos de velas, etc.
- Permitir la creación de *plugins* o *addons*. De esta manera se podría crear una comunidad que usase y contribuyese a la expansión de la librería.
- Facilitar la creación de *dashboards*.
- Mayor empuje hacia la integración con proyectos de realidad virtual. Por ejemplo mostrar estadísticas en forma de gráfico mientras se utiliza una aplicación o se juega a algún juego en 3D.

Bibliografía

- [1] T. Blokehead. *Scrum : Ultimate Guide to Scrum Agile Essential Practices!* Booktango, 2015.
- [2] G. C. Burdea and P. Coiffet. *Virtual Reality Technology*. John Wiley and Sons, 2017.
- [3] S. Chacon. *Pro Git*. Apress, 2009.
- [4] B. Danchilla. *Beginning WebGL for HTML5*. Apress, 2012.
- [5] D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, 2011.
- [6] E. Gamma, J. Vlissides, R. Helm, and R. Johnson. *Design Patterns*. Addison-Wesley, 1994.
- [7] R. C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.
- [8] P. Martz. *OpenGL Distilled*. Addison-Wesley Professional, 2006.
- [9] S. Pasquali. *Mastering Node.js*. Packt Publishing Ltd, 2013.
- [10] M. Pilgrim. *HTML5: Up and Running*. O'Reilly Media, 2010.
- [11] D. L. Shinder. *Computer Networking Essentials*. Cisco Press, 2001.