



GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA
TELECOMUNICACIÓN

Curso Académico 2018/2019

Trabajo Fin de Grado

VISUALIZACIÓN DE DATOS EN REALIDAD
VIRTUAL

Autor : Adrián Pizarro Serrano

Tutor : Dr. Jesús María González Barahona

Trabajo Fin de Grado

Visualización de Datos en Realidad Virtual

Autor : Adrián Pizarro Serrano

Tutor : Dr. Jesús María González Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día de
de 2019, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de 2019

*Dedicado a
mi familia*

Agradecimientos

Aquí vienen los agradecimientos... Aunque está bien acordarse de la pareja, no hay que olvidarse de dar las gracias a tu madre, que aunque a veces no lo parezca disfrutará tanto de tus logros como tú... Además, la pareja quizás no sea para siempre, pero tu madre sí.

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

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. Estructura de la memoria	3
1.4. Disponibilidad del Software	4
2. Objetivos	7
2.1. Objetivo general	7
2.2. Objetivos específicos	7
3. Estado del arte	11
3.1. HTML	11
3.2. JavaScript	12
3.3. JSON	12
3.4. NodeJS	13
3.5. npm	14
3.6. Budo	15
3.7. HighlightJs	15
3.8. Webpack	16
3.9. Git	17
3.10. Realidad Virtual	17
3.11. OpenGL	18
3.12. WebVR	19
3.13. WebGL	20

3.14. Three.js	21
3.15. A-Frame	23
4. Diseño e implementación	27
4.1. Metodología Scrum	27
4.2. Iteración 0. Estudio previo.	28
4.3. Iteración 1. Primer gráfico.	30
4.4. Iteración 2. Enriquecimiento de la librería y separación de la capa de datos. . .	40
4.5. Iteración 3. Refactorización y estandarización de la librería.	48
4.6. Iteración 4. Poblamiento de funcionalidad de la API. Resolución de incidencias de la comunidad.	53
4.7. Iteración 5. Adaptación para dispositivos de VR y representación dinámica de datos.	53
5. Resultados	55
5.1. Arquitectura general	55
5.2. Funcionamiento de la Librería	55
6. Conclusiones	57
6.1. Consecución de objetivos	57
6.2. Aplicación de lo aprendido	57
6.3. Lecciones aprendidas	58
6.4. Trabajos futuros	58
Bibliografía	59

Índice de figuras

3.1. Ejemplo JSON básico	13
3.2. Arquitectura NodeJs	14
3.3. Ejemplo HighlightJs	16
3.4. Qué es Webpack	17
3.5. Dispositivo de Realidad Virtual	18
3.6. Ejemplo de Shader OpenGL	19
3.7. Escena Three.js	22
3.8. Ejemplo de cámara. Forma geométrica " <i>frustum</i> "	22
3.9. Ejemplo código HTML de A-Frame	24
3.10. Ejemplo del inspector de A-Frame	25
3.11. Ejemplo escena de A-Frame	25
4.1. Primera escena con A-Frame	30
4.2. Primera arquitectura del proyecto	31
4.3. Iteración 1: Resultado	39
4.4. Iteración 2: Resultado Gráfico de Barras	47
4.5. Iteración 2: Resultado Gráfico de burbujas	47
4.6. Iteración 3: Inicializar proyecto con Angle	48
4.7. Iteración 3: Nueva Arquitectura	49

Capítulo 1

Introducción

Este proyecto trata sobre la representación de datos en tres dimensiones (a partir de ahora 3D) en un navegador web utilizando como base el *framework* A-Frame¹. Este *framework* nace a su vez a partir de la librería Three.js² utilizada para mostrar gráficos animados en 3D.

El objetivo principal es crear una librería 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.

A continuación se describe el contexto, motivación, 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.

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

¹<https://aframe.io/>

²<https://threejs.org/>

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 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

³<https://d3js.org/>

⁴<https://www.amcharts.com/>

⁵<https://www.highcharts.com/>

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 un dispositivo 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 se ha pasado por distintas fases. En primer lugar, hubo un intento fallido de realizar una aplicación de posible utilidad médica. Siempre se puede interpretar como poner excusas, pero el tiempo, obligaciones y el querer dedicar tu tiempo libre a otros menesteres son factores que impidieron la consecución de este primer proyecto el cual, a pesar de todo, era muy interesante. Todo esto llevó a volver a comenzar con una búsqueda exhaustiva y motivadora donde por suerte pude contactar con Jesús, tutor del proyecto, el cual me ofreció toda la ayuda y un trabajo de fin de grado por el cual no me importara dedicar todo el tiempo libre posible.

Es así como gracias a todos estos factores se ha podido finalizar este proyecto y escritura de esta memoria. De la cual procedemos a describir su estructura en la siguiente sección.

1.3. 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. Por último se describe brevemente la estructura de la memoria.

- En el capítulo 2 se muestran los objetivos del proyecto. Se comienza hablando del objetivo principal que perseguimos con este trabajo. Acto seguido, pasaremos a los específicos donde se profundizará y se ampliará los conceptos tratados en el objetivo principal. Además en este apartado se hará una descripción de la planificación temporal que se ha seguido para llevar a cabo este proyecto
- 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.

1.4. 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 todos los 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

Objetivos

2.1. Objetivo general

Mi trabajo fin de grado consiste en crear una librería para la visualización de datos en 3D compatible con cualquier navegador. La cual además pueda ser usada en un dispositivo de realidad virtual.

2.2. Objetivos específicos

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

1. La librería debe proporcionar varios tipos de visualizaciones. Deberá implementar al menos cinco tipos de visualización tales como gráfico de burbujas, de tarta, de barras, de cilindros o tarta en forma de rosquilla.
2. Se debe permitir una gran cantidad de parámetros. Por ejemplo, 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 una leyenda, un pop-up o ambas.
3. La librería 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.

4. La librería debe proporcionar una herramienta para filtrar o seleccionar distintas fuentes de datos. Además podrán refrescar las visualizaciones dinámicamente.
5. Debe ser capaz de interpretar ficheros con formato JSON (JavaScript Object Notation) y lanzar una excepción si no fuera capaz de leer el archivo correctamente.
6. 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
7. Se debe poder utilizar en cualquier navegador sin necesidad de instalar ningún plugin. Además se debe poder utilizar y visualizar en dispositivos de realidad virtual como las Oculus Rift.
8. 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.
9. Debe ser fácilmente escalable, es decir, deberá estar desarrollada de tal manera que sea sencillo implementar nuevos desarrollos o modificaciones en el código.
10. 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.
11. Aprender Webpack para empaquetar y distribuir nuestra librería.
12. Aprender el framework de A-frame. 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. Crear una web con ejemplos de uso de la librería.
14. Crear el proyecto con Angle. Una herramienta hecha por los creadores de Aframe para comenzar nuestro componente con un arquetipo que sigue los estándares de Aframe

15. Una vez creado el proyecto siguiendo los estándares y después de realizar los primeros avances se deberá publicar nuestro proyecto en el registry de Aframe 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
16. 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].

Capítulo 3

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.

3.1. HTML



En 1993 se lanza este lenguaje de marcado 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 [9]. 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. Finalmente cabe destacar que el nuevo API nos provee de herramientas para la geolocalización, *drag and drop*, almacenamiento local, caché, etc.

3.2. 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

3.3. 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

¹<https://angularjs.org/>

²<https://reactjs.org/>

³<https://vuejs.org/>

```
{
  "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 3.1: Ejemplo JSON básico

Figura 3.1, debido a su gran versatilidad. Es un formato muy sencillo de leer y de *parsear*, es decir, de obtener su información.

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*.

3.4. NodeJS



NodeJs [8] 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 3.2, tenemos un único hilo donde por cada petición se delega el trabajo

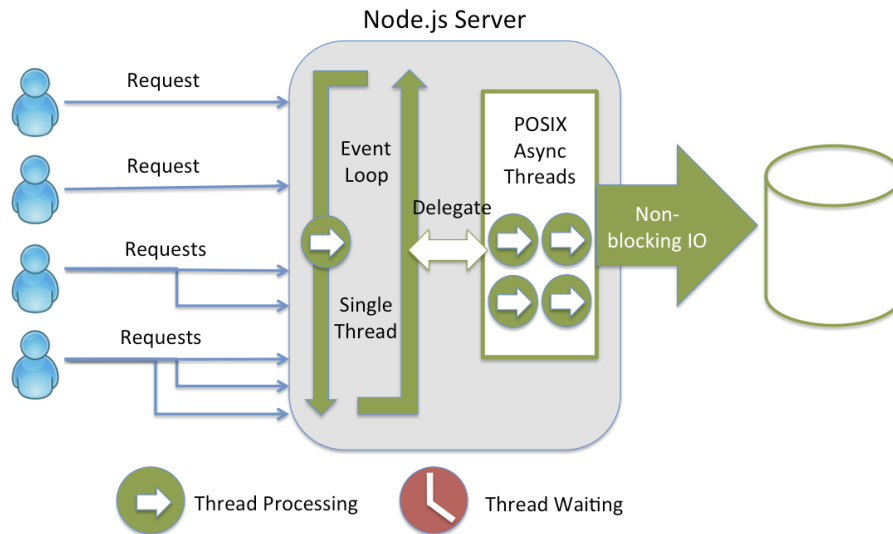


Figura 3.2: Arquitectura NodeJs

y se da respuesta de manera asíncrona, es decir, no es bloqueante y se ejecutará de manera concurrente recibiendo dichas respuestas mediante funciones *callback*.

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.

3.5. npm



Como gestor de paquetes, módulos y dependencias de NodeJs nace en 2014 NPM. 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.

3.6. 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

3.7. 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 lenguajes de programación y 85 estilos.

En este proyecto, HighlightJs ha sido utilizado para elaborar el manual de usuario⁵ facilitan-

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

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

```
<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 3.3: Ejemplo HighlightJs

do así la lectura de código y pudiendo reutilizar este código de manera más clara como vemos en la Figura 3.3

3.8. 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* [10].

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

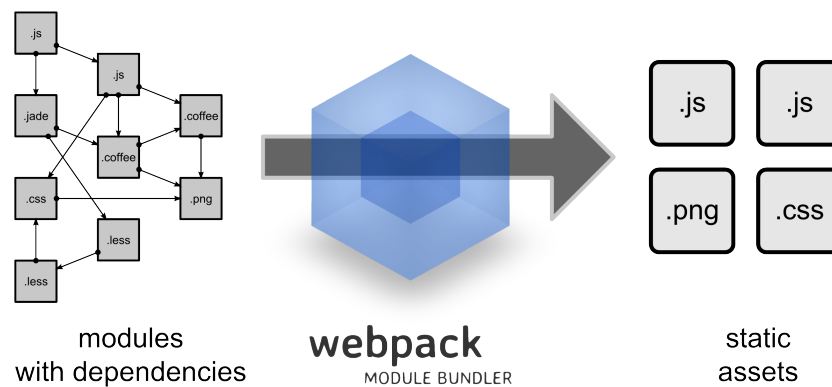


Figura 3.4: Qué es Webpack

3.9. 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.

3.10. Realidad Virtual

Entendemos por realidad virtual [2] aquella tecnología que a través de uno o varios dispositivos nos puede general 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.



Figura 3.5: Dispositivo de Realidad Virtual

En la Figura 3.5 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.

3.11. OpenGL



OpenGL 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 video-

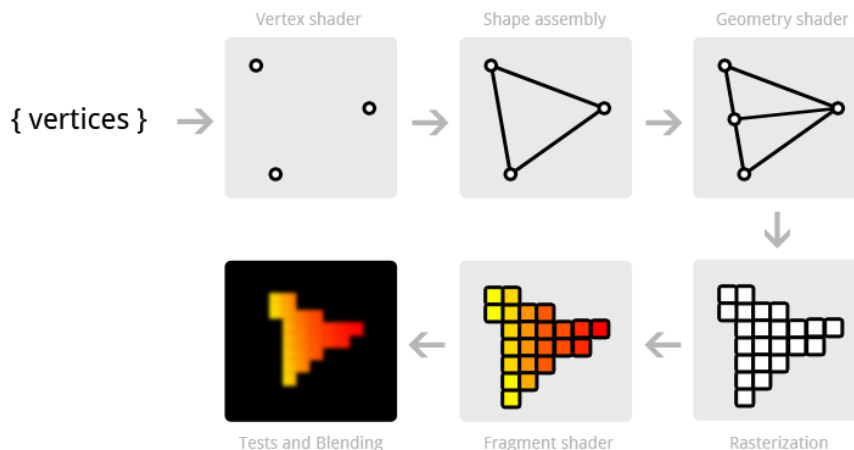


Figura 3.6: Ejemplo de Shader OpenGL

juegos 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 3.6 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 los 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*.

3.12. WebVR

WebVR es una API experimental escrito en Javascript en 2006. Fue iniciado como prototipo de Canvas3D por Vladimir Vukićević 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).

3.13. 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.

3.14. 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 3.7 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.

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 3.8. 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.

⁶<https://threejs.org>

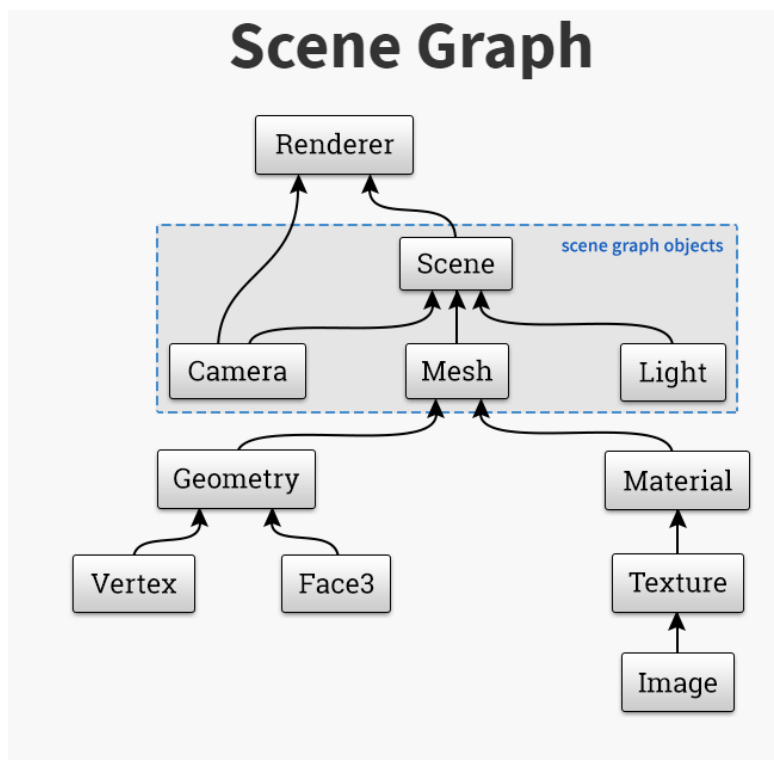


Figura 3.7: Escena Three.js

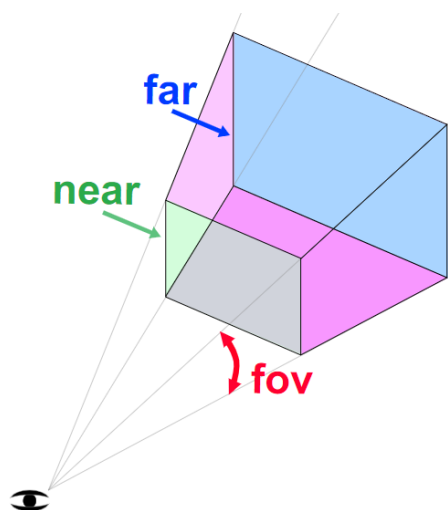
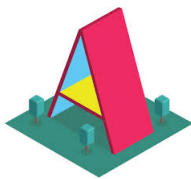


Figura 3.8: Ejemplo de cámara. Forma geométrica "frustum"

- Por último tenemos los parámetros "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 pirámide con la punta cortada. Al fin y al cabo esta definición es la que utiliza la cámara para visualizar la escena.

3.15. 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.

Esta librería, estuvo originalmente concebida por Mozilla y ahora mantenida por los co-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>`.

⁷<https://aframe.io/>

⁸<https://supermedium.com/>

```

<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 3.9: Ejemplo código HTML de A-Frame

- Como vemos en la Figura 3.9, 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.
- 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 3.10 . 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.

⁹<https://unity.com/es>

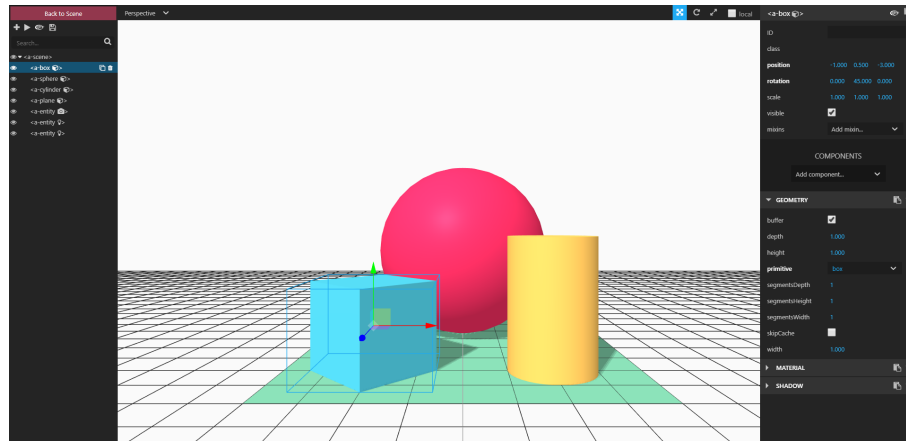


Figura 3.10: Ejemplo del inspector de A-Frame



Figura 3.11: Ejemplo escena de A-Frame

- Los componentes 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.

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

Capítulo 4

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.

4.1. Metodología Scrum

La metodología que se ha seguido en este proyecto es 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.

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.

4.2. 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>
20 </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

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>

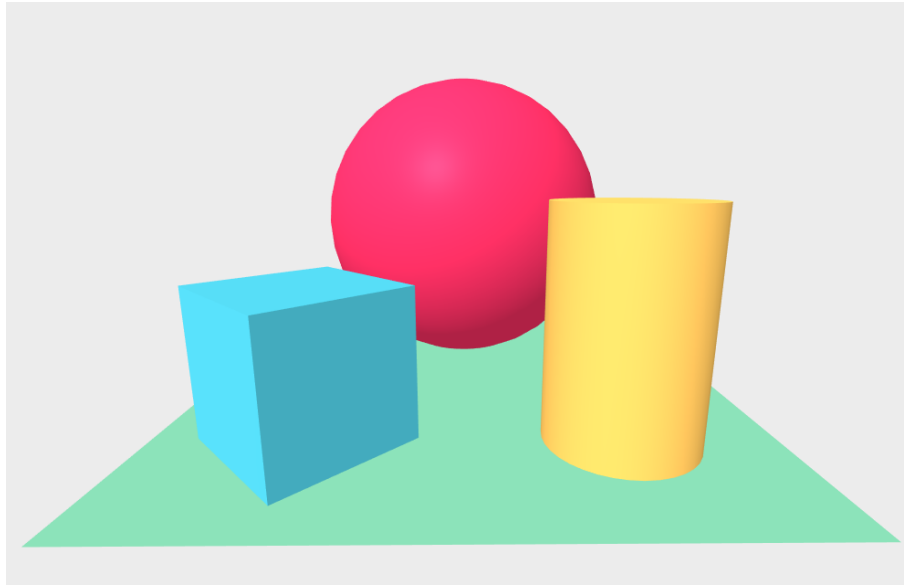


Figura 4.1: Primera escena con A-Frame

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 4.1

4.3. 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


```
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 4.2: Primera arquitectura del proyecto

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 todas 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 4.2

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

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

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 en-

tidades, 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 4.3

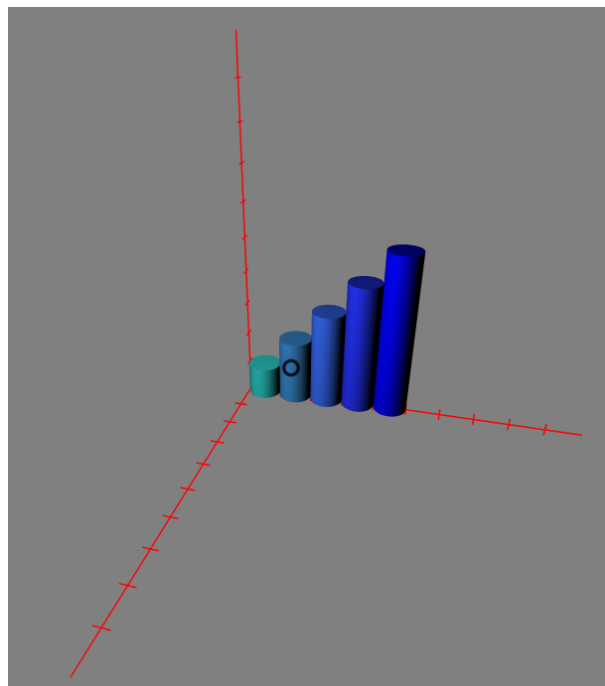


Figura 4.3: Iteración 1: Resultado

4.4. 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

4.4. ITERACIÓN 2. ENRIQUECIMIENTO DE LA LIBRERÍA Y SEPARACIÓN DE LA CAPA DE DATOS.

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');

```

4.4. ITERACIÓN 2. ENRIQUECIMIENTO DE LA LIBRERÍA Y SEPARACIÓN DE LA CAPA DE DATOS.

```
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, {

```

4.4. ITERACIÓN 2. ENRIQUECIMIENTO DE LA LIBRERÍA Y SEPARACIÓN DE LA CAPA DE DATOS.

```
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 4.4 y 4.5.

4.4. ITERACIÓN 2. ENRIQUECIMIENTO DE LA LIBRERÍA Y SEPARACIÓN DE LA CAPA DE DATOS.

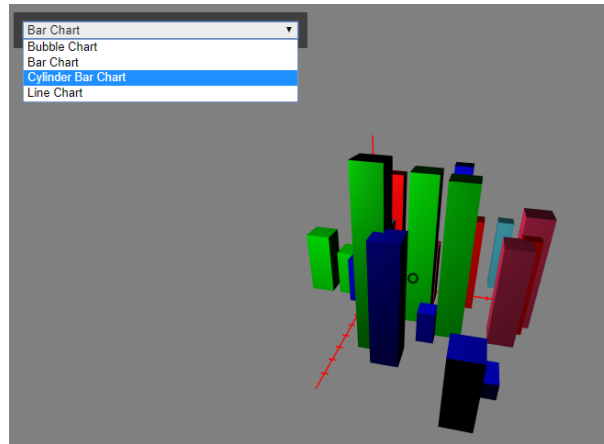


Figura 4.4: Iteración 2: Resultado Gráfico de Barras

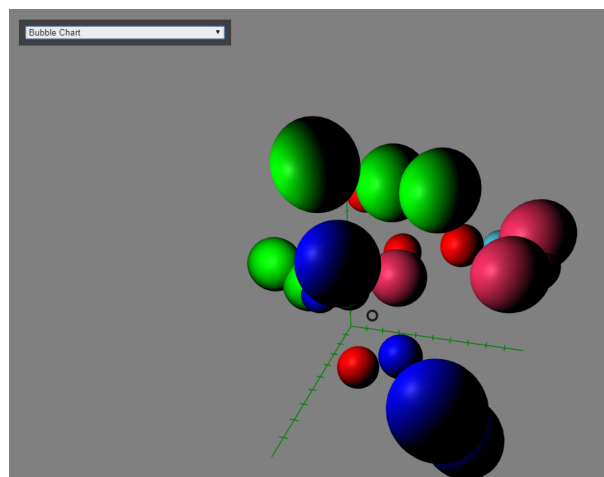


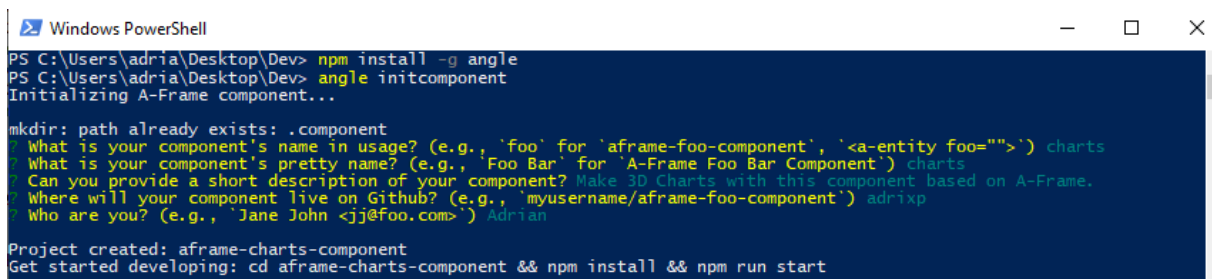
Figura 4.5: Iteración 2: Resultado Gráfico de burbujas

4.5. Iteración 3. Refactorización y estandarización de la librería.

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 4.6). Se genera un componente con una arquitectura genérica (Figura 4.7).



```
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 4.6: Iteración 3: Inicializar proyecto con Angle

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

⁷<https://github.com/ngokevin/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 4.7: Iteración 3: Nueva Arquitectura

- 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.
- Debe tener sentido en el contexto de una aplicación WebVR.

⁸<https://www.npmjs.com/>

⁹<https://github.com/>

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

- 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"
11      "></a-entity>
12        <a-entity position="2 10 14" rotation="-30 15 0">
13          <a-camera position="3 -1 4" rotation="0 -1 0">
14            <a-cursor></a-cursor>
15          </a-camera>
16        </a-entity>
17      </a-scene>
18    </body>
19  </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.

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

¹¹<https://semver.org/>

```

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']);
7   return entity;
8 }

```

```
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.

4.6. Iteración 4. Poblamiento de funcionalidad de la API. Resolución de incidencias de la comunidad.

En esta cuarta iteración se tiene como objetivo poblar de funcionalidad 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 a la par 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 reciben peticiones para añadir nuevos gráficos como el gráfico de tarta.

- Se añaden parametrización en los ejes. separacion de los ticks, grosor, posicion de los ejes y los ticks.

- Ejes parametrizables si hay datos negativos

- Grid axis

- Texto y numeración de los ejes, color y tamaño del mismo

- Se añade gráficos de cilindros

- Se añaden los gráficos de tarta y donut Gracias al Change Request de la comunidad! cerrando el issue

- Json de Datos, Formato para tarta y donut.

4.7. Iteración 5. Adaptación para dispositivos de VR y representación dinámica de datos.

- Añadimos pop up

- Añadimos leyenda

- Pruebas de rendimiento (Creación de programa para generar random json con los datos que necesitamos)

- Añadimos la funcionalidad del totem. Cambiar datos de manera dinámica.

- Adaptación para Oculus Rift

Capítulo 5

Resultados

5.1. Arquitectura general

Hablo de la arquitectura con nodejs.
package json, index.js node modules.
Ejes.js
Leyenda.js
Totem.js
JSON para datos
Foto general de la arquitectura

5.2. Funcionamiento de la Librería

Manual de usuario con lo que hay en la web.

Capítulo 6

Conclusiones

6.1. Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Todos conseguidos of course.

6.2. Aplicación de lo aprendido

Aquí viene lo que has aprendido durante el Grado/Máster y que has aplicado en el TF-G/TFM. Una buena idea es poner las asignaturas más relacionadas y comentar en un párrafo los conocimientos y habilidades puestos en práctica.

La de Meteor, La de Barahona de Js etc, alguna con Node

1. a

2. b

6.3. Lecciones aprendidas

Aquí viene lo que has aprendido en el Trabajo Fin de Grado/Máster. webpack y todo lo que tenga en objetivos A frame, realidad virtual

1. a

2. b

6.4. Trabajos futuros

Ningún software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Integracion con Elasticsearch (Que pueda ser Near Real Time) Otras bases de datos relacionales Mayor customización y mas tipos de gráficos Animaciones Permitir añadir plugins y addons a la libreria Crear comunidad Dashboards Mas Realidad Virtual

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] S. Pasquali. *Mastering Node.js*. Packt Publishing Ltd, 2013.
- [9] M. Pilgrim. *HTML5: Up and Running*. O'Reilly Media, 2010.
- [10] D. L. Shinder. *Computer Networking Essentials*. Cisco Press, 2001.