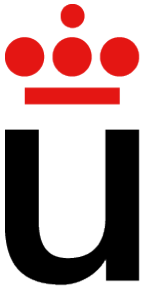


Universidad
Rey Juan Carlos

GRADO EN INGENIERIA EN SISTEMAS AUDIOVISUALES

Y MULTIMEDIA

Curso Académico 2020/2021

Trabajo Fin de Grado

SIMULADOR DE TRAZAS DE COMUNICACIONES EN REALIDAD VIRTUAL

Autor : Alejandro Esteban López

Tutor : Dr. Jesus María González Barahona

Trabajo Fin de Grado

Simulador de Trazas de Comunicaciones en Realidad Virtual

Autor : Alejandro Esteban López

Tutor : Dr. Jesus María González Barahona

La defensa del presente Proyecto Fin de Carrera se realizó el día de de 202X, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Dedicado a
mis padres.

II

Agradecimientos

En primer lugar quería dar las gracias a mis padres, ya que sin ellos nada de esto hubiera sido posible. Ellos han dedicado todo su tiempo, su esfuerzo y sus recursos con tal de educarme y formarme lo mejor posible para afrontar la vida. Siempre me han ayudado en todo, incluso cuando he estado a punto de tirar la toalla y dejarlo, ellos estaban ahí para darme el empujón necesario y la motivación para seguir con ello y finalmente poder conseguirlo. Gracias, gracias y gracias de corazón, por todo lo que habéis hecho y hacéis por mí, nunca estaré lo suficientemente agradecido de todo lo que me habéis dado y enseñado. Os quiero mucho.

Darle las gracias a mi hermano, que aunque a veces tengamos nuestros más y nuestros menos, siempre está ahí para todo lo que necesito ayudándome y dándome los mejores consejos posibles siempre que los necesito.

También quería darle las gracias a mi pareja, que siempre está ahí apoyándome tanto en los buenos momentos como sobre todo en los malos momentos, con todo lo que ello conlleva. Siempre me saca una sonrisa y me hace sentir especial a su lado.

Y por último agradecer a mi tutor Jesús M. González Barahona que desde el principio siempre ha tenido disponibilidad para resolverme cualquier duda y ayudarme en cualquier cosa que necesitase.

Resumen

Este proyecto presenta una herramienta para la construcción de un simulador de red en 3D y Realidad Virtual en la que se produce un intercambio de paquetes. Su función es la de leer la captura que introduzcamos e interpretar los campos que nos interesan, para poder representar la red completa que se refleja en la captura y visualizarla en una escena de Realidad Virtual.

En la escena, podremos ver todos los nodos que hay en la red, tanto en la capa Ethernet como IP, con sus correspondientes carteles identificando que dirección IP o dirección Ethernet tienen. Estos nodos estarán conectados entre sí mediante unas tuberías transparentes y solamente entre los nodos que se intercambien paquetes.

Los paquetes podremos verlos también representados y en una animación, en la que veremos como se desplazan desde su nodo origen hasta su nodo destino. Dentro de cada paquete, al pinchar sobre él, podremos también ver las capas que tiene ese paquete, y los datos más relevantes de cada una de ellas, que se representaran en un cartel encima de ellos.

Además, podremos ver toda esta escena y la animación con unas gafas de realidad virtual, y movernos libremente por la escena para recorrerla en la dirección que queramos y poder ver desde dentro toda la escena y la animación de cada uno de los paquetes.

El proyecto está basado en tecnología A-Frame, un framework de JavaScript, el cual se utiliza para la creación de escenas en realidad virtual. Para ello he construido el código mediante HTML5 y JavaScript.

También he ido incluyendo en un repositorio de GitHub todas las pruebas y avances que he ido realizando durante el proceso de elaboración del proyecto, para tener sincronizado y actualizado a la última versión el código final.

Summary

This project presents a tool for the construction of a 3D network simulator and Virtual Reality in which a packet exchange takes place. Its function is to read the capture that we introduce and interpret the fields that interest us, to be able to represent the complete network that is reflected in the capture and visualize it in a Virtual Reality scene.

In the scene, we will be able to see all the nodes in the network, both in the Ethernet and IP layer, with their corresponding signs identifying which IP address or Ethernet address they have. These nodes will be connected to each other by means of transparent pipes and only between nodes that exchange packets.

The packets will also be represented in an animation, in which we will see how they move from their source node to their destination node. Within each packet, by clicking on it, we can also see the layers that the packet has, and the most relevant data of each one of them, which will be represented in a poster above them.

In addition, we can see this whole scene and the animation with virtual reality glasses, and move freely through the scene to move in the direction we want and to see from inside the whole scene and the animation of each of the packages.

The project is based on A-Frame technology, a JavaScript framework, which is used to create virtual reality scenes. For this I have built the code using HTML5 and JavaScript.

I have also been including in a GitHub repository all the tests and developments that I have been doing during the process of developing the project, to have synchronized and updated to the latest version of the final code.

Índice general

Capítulo 1

Introducción

En este proyecto nos vamos a centrar en la construcción de un simulador de red en 3D, que se creará mediante la introducción de una traza de red capturada anteriormente y se podrá visualizar desde cualquier navegador y dispositivo.
La tecnología en la que está basado el proyecto es A-Frame[1], la cual nos va a permitir también poder utilizarlo y visualizarlo todo en realidad virtual.

1.1. Contexto

La realidad virtual se remonta realmente a los años 50, época en la cual surge como toda una novedad, y en la que los gráficos e ilustraciones apenas simulaban otra realidad alternativa.

Sin embargo, con el pasar de los años, específicamente entre los 80 y 90, el concepto de realidad virtual adquiere más forma. Esto se debió al lanzamiento de dos proyectos relacionados con los videojuegos, como son la Virtual Boy de Nintendo y el Sega VR, los cuales empleaban cascos que permitían al usuario transportarse visualmente a otra dimensión.

Tras estos primeros y mareantes intentos, vino la participación de otras grandes compañías como Google con el lanzamiento del Street View.

Ahora nos encontramos inmersos en una situación social nunca antes vivida, en la que nos hemos tenido que adaptar a una nueva sociedad con mayor distanciamiento social por culpa de la COVID-19.

[1] `https://aframe.io/`

Confinados en nuestras casas, la mayoría de nuestras relaciones sociales se encontraban en nuestra imaginación. Hemos acudido a internet, a las videoconferencias o a las experiencias inmersivas en Realidad Virtual para mandar a nuestro cerebro hasta otro lugar, aunque nuestro cuerpo siga sentado en el sofá de nuestro salón.

Y es que, la pandemia de la COVID-19 inició un cambio forzado en nuestro estilo de vida, medio mundo nos vimos en "cuarentena" y en distanciamiento social. Pero, que la población mundial permaneciera confinada en sus domicilios durante semanas, se ha convertido en una oportunidad para el sector de las tecnologías, como el de la Realidad Virtual.

Y, aunque hasta ahora la adopción de la Realidad Virtual a nivel doméstico había sido modesta, ahora mucha gente utiliza las gafas de Realidad Virtual para jugar, quedar con sus amigos, asistir a conciertos o, incluso, teletrabajar. Esto nos ha llevado a un nuevo resurgir de esta tecnología que vive en constante evolución.

Por ello vemos que cada vez está más presente en nuestras vidas cotidianas el uso de tecnologías cada vez más avanzadas y novedosas, y la realidad virtual es una de esas tecnologías que cada vez está teniendo más importancia y fuerza en el día a día. Por lo tanto hemos querido enfocarnos en este campo de la Realidad Virtual, y para llevar esto a cabo nos hemos basado en el framework A-Frame para crear escenas en realidad virtual.

1.2. Objetivo general

Mi trabajo fin de grado consiste en construir un simulador de redes de comunicación basado en Realidad Virtual, el cual podrá visualizarse en cualquier dispositivo que tenga navegador web y en dispositivos de realidad virtual.

El simulador tiene que ser capaz de leer trazas de redes de comunicación, elaborar la escena de la red y crear una animación de los paquetes.

1.3. Objetivos específicos

En esta parte, pasaremos a realizar una breve descripción de los objetivos específicos del simulador de red:

- El programa se tiene que poder visualizar en cualquier navegador sin la necesidad de tener que instalar ningún programa o Plug-in para su uso. Además, se podrá ejecutar también desde dispositivos móviles y dispositivos de realidad virtual.

- El simulador será creado usando A-Frame.

- El simulador podrá discriminar varios niveles de red y pondrá paquetes en funcionamiento en cada uno de ellos.

- Será un programa que podrá utilizar cualquier captura de red y poder analizar las trazas que contenga.

- Los paquetes de la traza deben poder animarse y congelarse para poder inspeccionarlos.

- El programa debe ser capaz de dibujar todos los paquetes que corresponden a la escena y los representara con distintos colores dependiendo del protocolo más alto que tenga el paquete.Los protocolos que se podrán representar son: Ethernet[2] , IP[3] , TCP[4] , HTTP[5]

- El movimiento de los paquetes en la simulación se animará mediante un elemento de la escena.

- Debe aparecer encima de cada nodo, su dirección IP o Ethernet correspondiente en un cartel, según lo hayamos seleccionado a la hora de poner el documento JSON[6].

- Debe ser capaz de interpretar ficheros o datos con formato JSON (JavaScript Object Notation).

- Al pulsar en un paquete de la escena, deben desplegarse entre 1 y 4 cajas, las cuales corresponden cada una de ellas a un nivel. Deben aparecer una encima de otra y en el siguiente orden:
  Arriba del todo la capa HTTP, debajo la capa TCP, después la capa IP y abajo del todo la capa Ethernet.

[2] `https://es.wikipedia.org/wiki/Ethernet`
[3] `https://es.wikipedia.org/wiki/Protocolo_de_internet`
[4] `https://es.wikipedia.org/wiki/Protocolo_de_control_de_transmisin`
[5] `https://developer.mozilla.org/es/docs/Web/HTTP`
[6] `https://www.json.org/json-en.html`

SOLO aparecerán las capas que tenga cada paquete, es decir si un paquete solo tiene nivel de Ethernet y de IP, solo aparecerán la caja azul y amarilla.

- Dentro de cada paquete, al pinchar en cada uno de los niveles que nos muestre con las cajas de los colores, debe mostrar un cartel con la información correspondiente de ese nivel. Por ejemplo, si pinchamos en el nivel amarillo que corresponde a IP, nos mostrara un cartel indicándonos el nivel de protocolo que es y la dirección origen y destino IP de ese paquete.

- 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 esta basada en Three.js por lo que también se debe realizar un estudio de la misma.

- Crear varias demos para poder ver el resultado final.

1.4. Estructura de la memoria

- En el primer capítulo se hace una introducción al proyecto, donde hablamos sobre el contexto en el que nos hemos encontrado, el objetivo general del proyecto y los objetivos específicos que nos han permitido realizarlo.

- En el capítulo 2 mostraremos las distintas tecnologías que se han utilizado a lo largo del proyecto y veremos algunos ejemplos de su uso.

- El capítulo 3 se centra en el desarrollo del trabajo. En este capítulo podremos observar la metodología que hemos seguido para la realización del proyecto, y las diferentes etapas que hemos tenido que ir pasando hasta llegar al resultado final.

- En el capítulo 4 veremos el resultado final del proyecto, donde podremos ver todo lo que hace falta para que funcione correctamente el programa y explicaremos todos los componentes que tiene incorporados.

- Y por último, en el capítulo 5, escribiremos las conclusiones, donde haremos un resumen de todo el trabajo y aprendizaje realizado y hablaremos de posibles mejoras que se podrían realizar al programa.

Capítulo 2

Tecnologías utilizadas

2.1. A-Frame

A-Frame es un marco web para crear experiencias de realidad virtual (VR). Es un framework web de código abierto de three.js, que a su vez es una librería de JavaScript. A-Frame se basa en HTML, lo que facilita su uso.Fue desarrollado para ser una forma fácil pero poderosa de desarrollar contenido de realidad virtual.

Como proyecto independiente de código abierto, A-Frame se ha convertido en una de las comunidades de realidad virtual más populares en los navegadores. Este framework usa la arquitectura ECS (Entity Component System), usada en el desarrollo de juegos donde cada objeto es una entidad.

La ventaja de este framework es que los objetos ya no están fijos en una jerarquía, por lo que ahora las posibilidades son inmensas, los objetos pueden tener un comportamiento sin límites. A-Frame permite llevar a ECS a otro nivel, A-Frame es declarativo, tienen HTML y está basado en el DOM, lo que permite solucionar muchas de las debilidades de ECS.

A continuación se muestran las capacidades que el DOM proporciona para ECS:

- Referencia a otras entidades con selectores de consulta: El DOM proporciona un potente sistema selector de consultas que nos permite seleccionar una entidad o entidades que coincidan con una condición. Podemos obtener referencias a entidades por ID, clases o atributos de datos. Debido a que A-Frame está basado en HTML, podemos usar selectores de consulta fuera de la caja.

- Comunicación cruzada entre entidades con desacoplamiento de eventos: El DOM proporciona la capacidad de escuchar y emitir eventos. Esto proporciona un sistema de comunicación entre entidades. Los componentes no tienen conocimiento de otros componentes, estos solo emiten eventos y otros componentes pueden escuchar esos eventos.

- APIs para la gestión del ciclo de vida con las API de DOM: El DOM proporciona API para actualizar los elementos HTML y el árbol. Tales como .setAttribute, .removeAttribute, .createElement y .removeChild se pueden utilizar tal y como lo usamos en el desarrollo web normal.

- Filtro de entidad con selectores de atributos: El DOM[1] proporciona selectores de atributos que nos permiten consultar una entidad o entidades que tienen o no ciertos atributos HTML. Esto significa que podemos solicitar entidades que tengan o no un determinado conjunto de componentes.

- Declarativo: Por último, el DOM proporciona HTML. A-Frame es el puente entre ECS y HTML haciendo un patrón ya limpio declarativo, legible y extensible.

En ECS tenemos:

- Entidades: Son objetos contenedores donde los componentes son ligados para otorgarles propiedades. Las entidades son representadas con a-entity y la sintaxis es la siguiente:

```
<a-entity ${componentName}="${propertyName1}: ${propertyValue1}; ${propertyName2:}: ${propertyValue2}">
```

  En un ejemplo lo veremos mejor:

```
<a-entity geometry="primitive: sphere; radius: 1.5"
          light="type: point; color: white; intensity: 2"
          material="color: white; shader: flat; src: glow.jpg"
          position="0 0 -5"></a-entity>
```

  Aquí tenemos un objeto primitivo esfera, con un radio de 1.5, el cuál tiene un punto de luz blanco con intensidad 2, sin sombras. La esfera tendrá una textura que se carga mediante la propiedad src de material.

- Componentes: Son las propiedades que hacen que una entidad sea diferente a otra.Los componentes donan a la entidad de comportamiento, apariencia y funcionalidad.

[1] https://desarrolloweb.com/articulos/que-es-el-dom.html

Los componentes son módulos de JavaScript que se pueden mezclar, combinar y componer en entidades para crear apariencia, comportamiento y funcionalidad. Podemos registrar un componente en JavaScript y usarlo de forma declarativa desde el DOM.
Los componentes son configurables, reutilizables y compartibles. La mayor parte del código en una aplicación A-Frame debería vivir dentro de los componentes.

Podemos utilizar componentes que ya esten creados , o crear nuestros componentes con los parametros y funciones que nos interesen para nuestro programa.

Este es uno de los componentes que he creado en mi programa, cuya funcion es cada vez que le llamamos crea un cartel con los datos que le llegan :

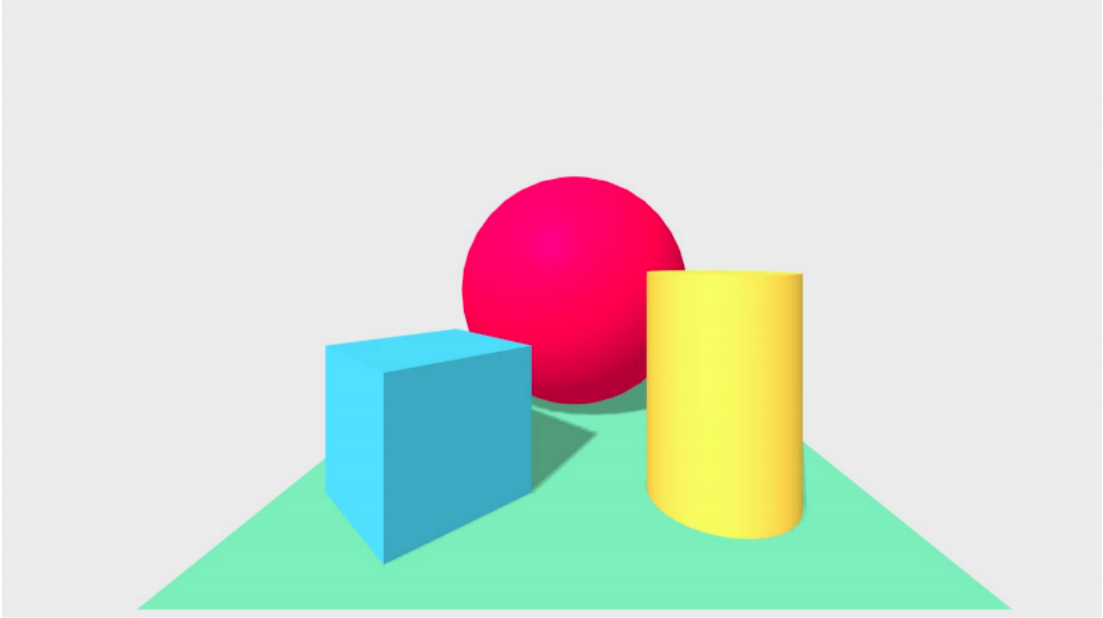
```
AFRAME.registerComponent('poster', {
  schema: {
      color: {type: 'color', default: 'white'},
      position: {type: 'vec3'},
      text: {type: 'string'}

  },
  init: function () {
    let el = this.el
    let data = this.data
    el.setAttribute('geometry', {primitive:'plane',height: 0.5, width: 3});
    el.setAttribute('text', {width:8, color:'red', value: data.text, align:'center'});
    el.setAttribute('material', 'color', data.color);
    el.setAttribute('position', data.position);
  }
});
```

- Sistemas: provienen el entorno donde manejar y desarrollar los componentes. Podemos usar los sistemas para separar la funcionalidad de la información, donde los componentes son los contenedores de la información y el sistema se ocupa de la lógica del uso de estos.

A-Frame es compatible con la mayoría de los cascos de realidad virtual como Vive, Rift, Windows Mixed Reality, Daydream, GearVR, Cardboard, Oculus Go e incluso se puede utilizar para la realidad aumentada. Aunque A-Frame admite todo el espectro, A-Frame tiene como objetivo definir experiencias de realidad virtual interactivas totalmente inmersivas que van más allá del contenido básico de 360 °, haciendo un uso completo del seguimiento posicional y los controladores.

A continuación podemos ver un ejemplo simple, el tipico (Hola Mundo) en A-Frame:

```
<html>
  <head>
    <title>Hello, WebVR! - A-Frame</title>
    <meta name="description" content="Hello, WebVR! - A-Frame">
    <script src="https://aframe.io/releases/0.7.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box      position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9" shadow></a-box>
      <a-sphere   position="0 1.25 -5" radius="1.25" color="#EF2D5E" shadow></a-sphere>
      <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5"
                  color="#FFC65D" shadow></a-cylinder>
      <a-plane    position="0 0 -4" rotation="-90 0 0" width="4" height="4"
                  color="#7BC8A4" shadow></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```



2.2. Javascript

Como hemos visto anteriormente, el lenguaje que más se utiliza en A-Frame es Javascript y su DOM. Ahora veremos en que se fundamenta este lenguaje, y la importancia que tiene en el proyecto.

JavaScript es un lenguaje poderoso, capaz de aportar soluciones eficaces en la mayoría de los ámbitos de la tecnología. Es especialmente importante porque es el único lenguaje de programación que entienden los navegadores de forma nativa (lenguaje interpretado sin necesidad

de compilación), con el que se desarrolla la parte de la funcionalidad frontend en sitios web y aplicaciones web modernas. Por tanto se utiliza como complemento de HTML y CSS para crear páginas webs. Pero también es fundamental en muchos otros tipos de desarrollos.
JavaScript es el lenguaje de programación encargado de dotar de mayor interactividad y dinamismo a las páginas web. Cuando JavaScript se ejecuta en el navegador, no necesita de un compilador. El navegador lee directamente el código, sin necesidad de terceros. Por tanto, se le reconoce como uno de los tres lenguajes nativos de la web junto a HTML (contenido y su estructura) y a CSS (diseño del contenido y su estructura).

Sus usos más importantes son los siguientes:

- Desarrollo de sitios web del lado del cliente (frontend, en el navegador)
- Desarrollo de todo tipo de aplicaciones gracias a la plataforma NodeJS
- Desarrollo de aplicaciones para dispositivos móviles, híbridas o que compilan a nativo
- Desarrollo de aplicaciones de escritorio para sistemas Windows, Linux y Mac, pudiendo escribir un código compatible con todas las plataformas.

Por tanto, podemos considerar a JavaScript el lenguaje universal, pues es el que más tipos de aplicaciones y usos que puede abarcar en la actualidad. Es por ello que resulta un lenguaje muy recomendable para aprender, ya que nos ofrece capacidades para usarlo en todo tipo de proyectos, siendo que algunos de ellos son parcela exclusiva de JavaScript.

La librería de JavaScript más utilizada en la historia, y que todavía se sigue utilizando es jQuery. Con jQuery podíamos hacer más cosas, escribiendo menos. Con una sintaxis mucho más sencilla, podíamos modificar nuestro sitio web, crear plugins, animar videojuegos y muchas cosas más.

Una de las mayores potencialidades de JavaScript es que puede manipular cualquier elemento de una página web y modificar su contenido, su tamaño, su color, su posición... e incluso hacer aparecer y desaparecer elementos.Para hacer esto posible necesitamos identificar con precisión cada elemento de una página web para poder indicarle a JavaScript sobre qué elemento debe operar.

2.3. DOM

Para permitir que los lenguajes de programación pudieran extraer información o manipular cualquier elemento de una página web era necesario definir de alguna manera qué tipos de elementos conforman una página web, cómo nombrarlos y cómo se relacionan entre sí. Inicialmente esto resultaba bastante problemático, ya que no existía un estándar o especificación oficial sobre cómo debía realizarse esto.

Para evitar los problemas de falta de estandarización, un organismo internacional (el W3C) definió un estándar denominado DOM ó Document Object Model (Modelo de objetos para representar documentos) que define qué elementos se considera que conforman una página web, cómo se nombran, cómo se relacionan entre sí, cómo se puede acceder a ellos, etc...

Las siglas DOM significan Document Object Model, o lo que es lo mismo, la estructura del documento HTML. Una página HTML está formada por múltiples etiquetas HTML, anidadas una dentro de otra, formando un árbol de etiquetas relacionadas entre sí, que se denomina árbol DOM (o simplemente DOM).

En Javascript, cuando nos referimos al DOM nos referimos a esta estructura, que podemos modificar de forma dinámica desde Javascript, añadiendo nuevas etiquetas, modificando o eliminando otras, cambiando sus atributos HTML, añadiendo clases o cambiando el contenido de texto entre otras cosas.

Al estar “amparado” por un lenguaje de programación, todas estas tareas se pueden automatizar, incluso indicando que se realicen cuando el usuario haga acciones determinadas, como por ejemplo: pulsar un botón, mover el ratón, hacer click en una parte del documento, escribir un texto, etc...

2.4. Three.js

ThreeJS es una librería JavaScript para el manejo de escenas 3D dentro del canvas del DOM. Por debajo usa WebGL para ofrecer toda su potencia y rendimiento pero con una API mucho más sencilla de usar para el desarrollador.

ThreeJS es open source, creada en 2010 y es una de las librerías más populares para crear este tipo de escenas 3D. Lo bueno de esta librería es que ofrece un montón de utilidades y abstracciones propias de motores 3D: cámaras, objetos, escenas, animaciones, materiales, luces, texturas, etc.

Dado que three.js se basa en JavaScript, es relativamente fácil agregar interactividad entre objetos 3D e interfaces de usuario como el teclado y el mouse. Entonces three.js es perfectamente adecuado para hacer un juego 3D en una plataforma web.

Vamos a ver como poder crear un pequeño programa para ver su utilización.

Primeramente, para cualquier programa o proyecto que empecemos debemos de crear una escena. La escena es como un universo donde podemos agregar objetos, cámaras y luces, etc.

```
var scene = new THREE.Scene();
```

A continuación tenemos que configurar la cámara, hay cámaras en perspectiva y ortográficas. Durante la mayor parte del tiempo, mantendremos la perspectiva, que es un tipo de cámara normal en el que verá objetos más cercanos, más grandes y más pequeños, cuando estén lejos.

Para la cámara de ortografía, verá una vista isométrica de un ángulo específico sin perspectiva. Entonces el primer parámetro de la cámara es el campo de visión, básicamente es el ancho del ángulo de percepción. Configurémoslo en 75, la siguiente es la relación de aspecto, vamos a usar el ancho actual dividido por la altura.

```
var camera = new THREE.PerspectiveCamera(75,window.innerWidth/
window.innerHeight);
```

Después, vamos a configurar el renderizador. Usaremos el renderizador WebGL.Puede personalizar la opción de representación aquí, como anti-aliasing.

```
var renderer = new THREE.WebGLRenderer({antialias: true});
```

Una vez creada la escena y configurada la cámara, podemos empezar a crear los objetos que queramos introducir en ella.

A continuación vamos a crear un cubo. Lo primero es crear una geometría, pensar en ella como un esqueleto para nuestro objeto. Usaremos BoxGeometry con todos los lados iguales.

```
var geometry = new THREE.BoxGeometry(1,1,1);
```

Ahora vamos a necesitar una textura para caracterizar nuestra geometría. Así que necesitamos material que defina las características de la piel, como la opacidad, el reflejo o la textura. Usemos MeshBasicMaterial (color sólido sin reflexión y cálculo de iluminación) con color rojo por ahora.

```
var material = new THREE.MeshBasicMaterial({color: 0xff0000});
```

Ahora podemos crear nuestro cubo creando un objeto de malla usando nuestra geometría y material y agregando el cubo a la escena.

```
var cube = new THREE.Mesh(geometry,material);
scene.add(cube);
```

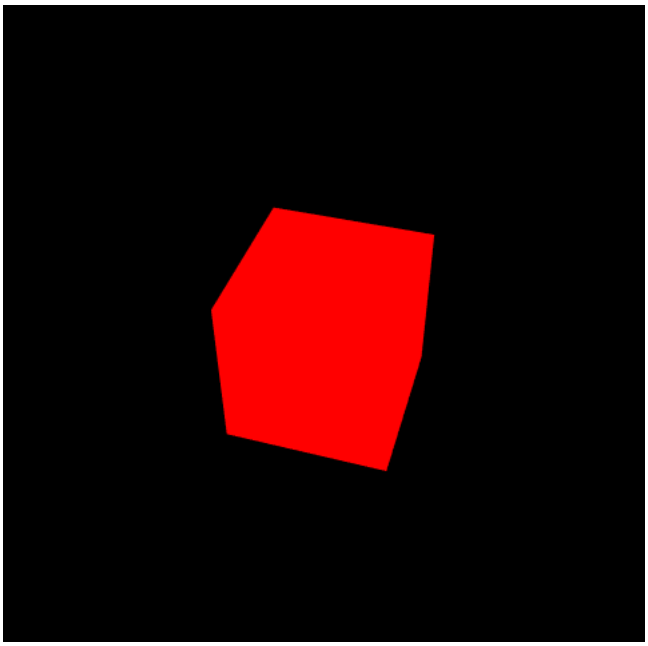
Pero como acabamos de agregar todo a la escena sin especificar la posición, ahora nuestro cubo está en el mismo lugar que la cámara y no se procesará. Lo colocamos en una posición de la escena que se vea y lo rotamos para que se vea como nos interese:

```
cube.position.z = -5;
cube.rotation.x = 10;
cube.rotation.y = 5;
```

Ahora llamamos a un método de renderizado en el objeto renderizador y le pasamos la escena y la cámara.

```
renderer.render(scene,camera);
```

Con esto, ya tendríamos el cubo creado e incluido en la escena y ya estaría renderizado. Podemos incluir también una animación como la rotación a nuestro cubo para que aparezca moviéndose y ya tendríamos nuestro primer pequeño programa.



```
<html>
        <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
        <script src="jquery-2.1.4.js"></script>
        <link rel="stylesheet" type="text/css" href="styles.css">
        </head>
        <body>
        <div id="navbar"><span>Three.js Tutorial</span></div>
        <script src="three.min.js"></script>
        <script>
          var scene = new THREE.Scene();
          var camera = new THREE.PerspectiveCamera(75,window.innerWidth/window.innerHeight);
          var renderer = new THREE.WebGLRenderer({antialias: true});
          renderer.setSize(window.innerWidth,window.innerHeight);
          $('body').append(renderer.domElement);
          var geometry = new THREE.BoxGeometry(1,1,1);
          var material = new THREE.MeshBasicMaterial({color: 0xff0000});
          var cube = new THREE.Mesh(geometry,material);
          scene.add(cube);
          cube.position.z = -5;
          cube.rotation.x = 10;
          cube.rotation.y = 5;
          renderer.render(scene,camera);
          var animate = function(){
            cube.rotation.x += 0.01;
            renderer.render(scene,camera);
            requestAnimationFrame(animate);
          }
          animate();
        </script>
        </body>
</html>
```

2.5. WebGL

WebGL es una especificación estándar que está siendo desarrollada actualmente para desplegar gráficos en 3D en navegadores web. El WebGL permite activar gráficos en 3D acelerados por hardware en páginas web, sin la necesidad de plug-ins en cualquier plataforma.

A grandes rasgos, WebGL es un mecanismo que tienen los navegadores para usar toda la potencia de la tarjeta gráfica de los usuarios que visitan las páginas web para poder pintar escenas en 3D. Antiguamente, antes de que existiera WebGL se usaba la propia API del canvas del DOM y Flash lo que hacía que las escenas 3D fueran muy simples.

Hoy en día, con WebGL puedes crear escenas 3D complejas, ya que todo se va a renderizar en la tarjeta gráfica del usuario que entre en la web (queda en manos del programador ajustar la calidad de los gráficos dependiendo de la potencia de la tarjeta gráfica del usuario).

WebGL creció desde los experimentos del canvas 3D comenzados por Mozilla. Mozilla primero demostró un prototipo de Canvas 3D en 2006. A finales de 2007, tanto Mozilla como Opera habían hecho sus propias implementaciones separadas. A principio de 2009 Mozilla y Khronos comenzaron el WebGL Working Group (Grupo de Trabajo del WebGL). El Grupo de Trabajo del WebGL incluye Apple, Google, Mozilla, y Opera, y WebGL.

Para usar WebGL no necesitas ningún framework, lo puedes hacer con Javascript sin instalar nada, pero programar en WebGL es tan complejo que lo mejor es usar alguna librería tipo ThreeJS o PixiJS para facilitar la tarea de programación.

2.6. WebXR

WebXR es un grupo de estándares que se utilizan juntos para admitir la representación de escenas 3D en hardware diseñado para presentar mundos virtuales (realidad virtual o VR) o para agregar imágenes gráficas al mundo real (realidad aumentada o AR). Las API de Dispositivos WebXR implementa el núcleo del conjunto de características WebXR, la gestión de la selección de los dispositivos de salida, hacen que la escena 3D en el dispositivo elegido a la tasa de trama adecuado y administrar vectores de movimiento creados utilizando controladores de entrada.

Los dispositivos compatibles con WebXR incluyen auriculares 3D totalmente inmersivos con seguimiento de movimiento y orientación, anteojos que superponen gráficos sobre la escena del mundo real que pasa a través de los marcos y teléfonos móviles de mano que aumentan la realidad al capturar el mundo con una cámara y aumentar esa escena con una computadora.

Para lograr estas cosas, la API del dispositivo WebXR proporciona las siguientes capacidades clave:

- Encuentra dispositivos de salida de realidad virtual o realidad aumentada compatibles.
- Renderice una escena 3D en el dispositivo a una velocidad de fotogramas adecuada.
- (Opcionalmente) refleje la salida en una pantalla 2D.
- Crear vectores que representen los movimientos de los controles de entrada.

En el nivel más básico, una escena se presenta en 3D calculando la perspectiva a aplicar a la escena con el fin de renderizarla desde el punto de vista de cada uno de los ojos del usuario calculando la posición de cada ojo y renderizando la escena desde esa posición, mirando en la dirección en la que se encuentra el usuario. Cada una de estas dos imágenes se renderiza en un solo framebuffer, con la imagen renderizada del ojo izquierdo a la izquierda y el punto de vista del ojo derecho renderizado en la mitad derecha del buffer. Una vez que se han representado las perspectivas de ambos ojos sobre la escena, el framebuffer resultante se envía al dispositivo WebXR para que se lo presente al usuario a través de sus auriculares u otro dispositivo de visualización apropiado.

2.7. HTML5

HTML5 es la quinta versión del lenguaje de programación HTML (HyperText Markup Language) que mejora y aprovecha todo el potencial de la Word Wide Web (WWW). El término representa dos conceptos diferentes:

- Se trata de una nueva versión de HTML, con nuevos elementos, atributos y comportamientos.
- Contiene un conjunto más amplio de tecnologías que permite a los sitios Web y a las aplicaciones ser más diversas y de gran alcance. A este conjunto se le llama HTML5 y amigos, a menudo reducido a HTML5 .

Diseñado para ser utilizable por todos los desarrolladores de Open Web, esta página referencía numerosos recursos sobre las tecnologías de HTML5, clasificados en varios grupos según su función.

Para entender de forma sencilla en qué consiste HTML5 debemos explicar que ofrece funcionalidades que mejoran la experiencia de navegación, es decir, los diferentes navegadores (Chrome, Explorer, Safari o Firefox) entienden rápidamente como está estructurada una web y cuáles son los elementos que la componen, aunque esto ya lo incorporaba la versión anterior, ahora se pueden incluir nuevos elementos, reduciendo el uso de plugins y mejorando la velocidad de carga.

Se trata de una nueva versión de HTML, con nuevos elementos, atributos y comportamientos. Contiene un conjunto más amplio de tecnologías que permite a los sitios Web y a las aplicaciones ser más diversas y de gran alcance.

Algunas de las nuevas características de HTML5 serían:

- Semántica: Permite describir con mayor precisión cuál es su contenido.
- Conectividad: Permite comunicarse con el servidor de formas nuevas e innovadoras.
- Sin conexión y almacenamiento: Permite a las páginas web almacenar datos localmente en el lado del cliente y operar sin conexión de manera más eficiente.
- Multimedia: Nos otorga un excelente soporte para utilizar contenido multimedia como lo son audio y video nativamente.

- Gráficos y efectos 2D/3D: Proporciona una amplia gama de nuevas características que se ocupan de los gráficos en la web como lo son canvas 2D, WebGL, SVG, etc.
- Rendimiento e Integración: Proporciona una mayor optimización de la velocidad y un mejor uso del hardware.
- Acceso al dispositivo: Proporciona APIs para el uso de varios componentes internos de entrada y salida de nuestro dispositivo.
- CSS3: Nos ofrece una nueva gran variedad de opciones para hacer diseños más sofisticados.
- Nuevas etiquetas semánticas para estructurar los documentos HTML, destinadas a remplazar la necesidad de tener una etiqueta que identifique cada bloque de la página.
- Los nuevos elementos multimedia como<audio> y <video>.
- La integración de gráficos vectoriales escalables (SVG) en sustitución de los genéricos <object>, y un nuevo elemento <canvas>; que nos permite dibujar en él.
- El cambio, redefinición o estandarización de algunos elementos, como <a>, <cite> o <menu>.
- MathML para fórmulas matemáticas.
- Almacenamiento local en el lado del cliente.

El funcionamiento HTML5 consiste en un proceso que solicita una página HTML a través del navegador.

Desde el navegador se realiza una petición a un servidor, lo que se hace a través de una dirección del tipo http://..../index.html. Después el servidor recupera de su disco duro esa página, la devuelve al navegador y la página se muestra.

2.8. GitHub

GitHub es un servicio basado en la nube que aloja un sistema de control de versiones (VCS) llamado Git. Este permite a los desarrolladores colaborar y realizar cambios en proyectos compartidos, a la vez que mantienen un seguimiento detallado de su progreso. Fue comprada por Microsoft en junio del 2018.

La plataforma está creada para que los desarrolladores suban el código de sus aplicaciones y herramientas, y que como usuario no solo puedas descargarte la aplicación, sino también entrar a su perfil para leer sobre ella o colaborar con su desarrollo.

Como su nombre indica, la web utiliza el sistema de control de versiones Git diseñado por Linus Torvalds. Un sistema de gestión de versiones es ese con el que los desarrolladores pueden administrar su proyecto, ordenando el código de cada una de las nuevas versiones que sacan de sus aplicaciones para evitar confusiones. Así, al tener copias de cada una de las versiones de su aplicación, no se perderán los estados anteriores cuando se va a actualizar.

Esto significa que cualquier desarrollador del equipo que tenga acceso puede gestionar el código fuente y su historial de cambios utilizando las herramientas de línea de comandos de Git.

A diferencia de los sistemas de control de versiones centralizados, Git ofrece ramas de características. Con lo que cada ingeniero de software en el equipo puede dividir una rama de características que proporcionará un repositorio local aislado para hacer cambios en el código.

Las ramas de características no afectan a la rama maestra, que es donde se encuentra el código original del proyecto. Una vez que se hayan realizado los cambios y el código actualizado esté listo, la rama de características puede fusionarse de nuevo con la rama maestra, que es la forma en que se harán efectivos los cambios en el proyecto.

Además, la interfaz de usuario de GitHub es más fácil de usar que la de Git, lo que la hace accesible para personas con pocos o ningún conocimiento técnico. Esto significa que se puede incluir a más miembros del equipo en el progreso y la gestión de un proyecto, haciendo que el proceso de desarrollo sea más fluido. Las principales características de la plataforma es que ofrece las mejores características de este tipo de servicios sin perder la simplicidad, y es una de las más utilizadas del mundo por los desarrolladores. Es multiplataforma, y tiene multitud de interfaces de usuario.

2.9. Atom

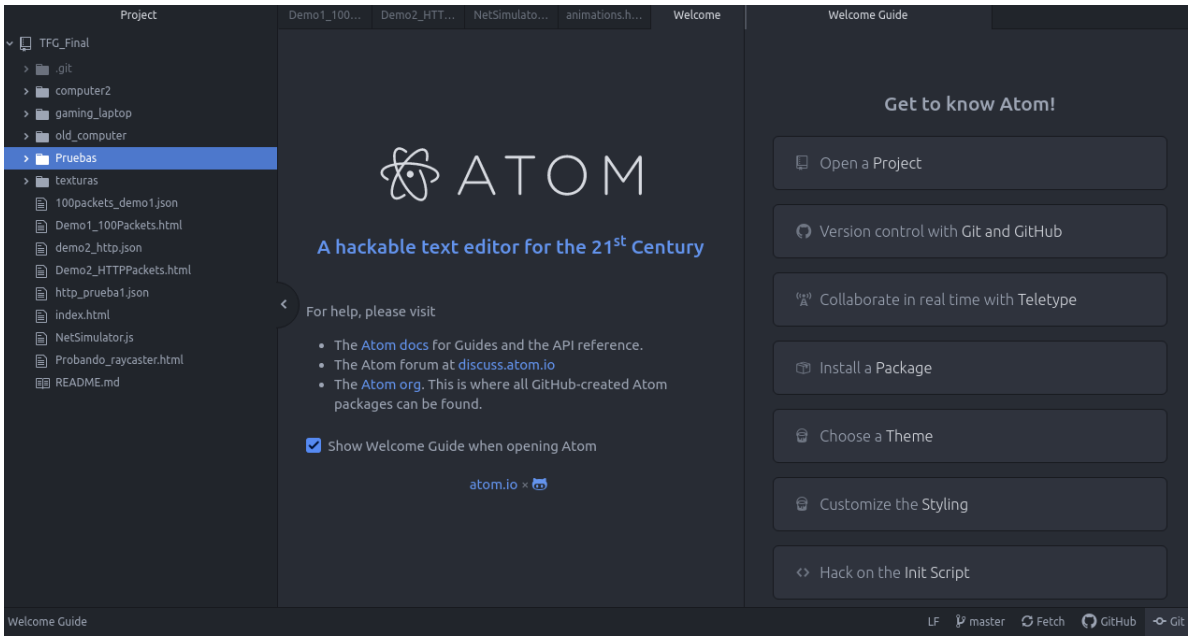
Atom es un editor de código de fuente abierta para macOS, Linux, y Windows con soporte para plug-ins escrito en Node.js, Incrustando Git Control, desarrollado por GitHub. Es una aplicación de escritorio construida utilizando tecnologías web.

Está basado en Electrón (Anteriormente conocido como Atom Shell), Un framework que permite aplicaciones de escritorio multiplataforma usando Chromium y Node.js. También puede ser utilizado como un entorno de desarrollo integrado (IDE).

Sus características principales son:

- Sirve para trabajar en cualquier sistema operativo (Windows, OS X o Linux).
- Puedes instalarle paquetes o crear los tuyos propios.
- Autocompletado inteligente que permite escribir código más rápido.
- Buscar y reemplazar texto de una forma sencilla.

Este editor de código nos facilita la manera de programar ayudándonos a autocompletar algunos campos, y también nos facilita sobre todo a incluir nuestro código en nuestro repositorio de GitHub e ir actualizandolo de manera muy sencilla.



2.10. LaTeX

LaTeX, es un sistema que ayuda al usuario a preparar un documento. Con él puedes preparar cualquier tipo de documento para presentarlo tanto en papel como en pantalla tales como manuscritos, cartas, artículos de revistas y tesis.

La historia de LaTeX empieza con TeX, un sistema de tipografía creado por Donald E. Knuth y publicado por primera vez en 1978. Su propósito era ayudar a matemáticos, físicos e informáticos en la composición de sus textos, que habitualmente incluían nomenclatura compleja.

LaTeX es una capa superior de TeX, creado por Leslie Lamport en 1984, y que emplea macros de TeX para emplear composición tipográfica de una manera más simple que con el TeX original. Además de para emplear notación matemática o científica de manera más cómoda que con un editor de texto genérico, también permite aplicar elementos de formato como notas al pie, índices o capítulos.

La calidad de imprenta de LaTeX pueden ser usados en áreas como química, física, computación, biología, leyes, literatura, música y en cualquier otro tema el cual usen simbologías. Otra característica es que te permite separar el contenido y el formato del documento. Así tener la oportunidad de concentrarte en generar y escribir ideas en una parte y plasmar esas ideas en otra.

Otra ventaja es que los textos creados con LaTeX son compatibles con cualquier sistema operativo o plataforma, y que pueden exportarse a formatos populares como PDF, HTML, RTF o Postscript, entre otros.

2.11. Realidad Virtual

La realidad virtual (VR) consiste en la inmersión sensorial en un nuevo mundo, basado en entornos reales o no, que ha sido generado de forma artificial, y que podemos percibir gracias a unas gafas de realidad virtual y sus accesorios (cascos de audio, guantes, etc...).

El objetivo de esta tecnología es crear un mundo ficticio del que puedes formar parte e incluso ser el protagonista: viendo un coche en un concesionario virtual, siendo protagonista de un videojuego o bien practicando como hacer una operación a corazón abierto.

En estas experiencias de realidad virtual puedes interactuar y explorar. Es el usuario quien controla la escena y puede interactuar con los objetos de la misma.

A diferencia de la realidad aumentada, que mejora la realidad sensorialmente, pero que es una realidad mixta entre el mundo real y el virtual, la realidad virtual debe ser inmersiva, permitir la interacción y tener una historia que respalde esa realidad.

La medicina, la cultura, la educación y la arquitectura son algunos de los ámbitos que ya han sucumbido a las ventajas que ofrece esta tecnología. Desde visitas guiadas a museos hasta la disección de un músculo, la RV nos permite cruzar unos límites que de otra forma no serían imaginables.

Capítulo 3

Diseño e implementación

En este capítulo hablaré del diseño e implementación del proyecto. Trataré el tema de la metodología que he utilizado e iré viendo todas las etapas hasta llegar al resultado final.

3.1. Metodología ágil SCRUM

En este proyecto he utilizado una versión muy simplificada de SCRUM para desarrollar el trabajo a través de sprint y facilitar con ello la construcción del programa.

La metodología SCRUM es un marco de trabajo o framework que se utiliza dentro de equipos que manejan proyectos complejos. Es decir, se trata de una metodología de trabajo ágil que tiene como finalidad la entrega de valor en períodos cortos de tiempo y para ello se basa en tres pilares: la transparencia, inspección y adaptación. Esto permite al cliente, junto con su equipo comercial, insertar el producto en el mercado pronto, rápido y empezar a obtener ventas.

Con la metodología SCRUM, el equipo tiene como foco entregar valor y ofrecer resultados de calidad que permitan cumplir los objetivos de negocio del cliente.

Para ello, los equipos de SCRUM son autoorganizados y multifuncionales. Es decir, cada uno es responsable de unas tareas determinadas y de terminarlas en los tiempos acordados. Esto garantiza la entrega de valor del equipo completo, sin necesidad de ayuda o la supervisión minuciosa de otros miembros de la organización.

En SCRUM existen 3 roles muy importantes: Product Owner, SCRUM Master y Equipo de desarrollo.

- Product owner: Es el responsable de maximizar el valor del trabajo del equipo de desarrollo. La maximización del valor del trabajo viene de la mano de una buena gestión del Product Backlog. El Product owner es el único perfil que habla constantemente con el cliente, lo que le obliga a tener muchos conocimientos sobre negocio. Para finalizar, un equipo SCRUM debe tener solo un Product Owner y este puede ser parte del equipo de desarrollo.

- SCRUM Master: Es el responsable de que las técnicas SCRUM sean comprendidas y aplicadas en la organización. Es el manager de SCRUM, un líder que se encarga de eliminar impedimentos o inconvenientes que tenga el equipo dentro de un, aplicando las mejores técnicas para fortalecer el equipo de marketing digital. Dentro de la organización, el SCRUM Master tiene la labor de ayudar en la adopción de esta metodología en todos los equipos.

- Equipo de desarrollo: Son los encargados de realizar las tareas priorizadas por el Product Owner. Es un equipo multifuncional y autoorganizado. Son los únicos que estiman las tareas del product backlog, sin dejarse influenciar por nadie. Los equipos de desarrollo no tienen sub-equipos o especialistas. La finalidad de esto es transmitir la responsabilidad compartida si no se llegan a realizar todas las tareas de un sprint.

En este proyecto,voy a actuar como desarrollador, y el rol de SCRUM Master y Product owner será los que tome mi tutor Jesus María González Barahona.

El núcleo central de la metodología de trabajo ‘SCRUM’ es el ‘sprint’. Se trata de un miniproyecto, cuyo objetivo es conseguir un incremento de valor en el producto que estamos construyendo. Todo ‘sprint’ cuenta con una definición y una planificación que ayudará a lograr las metas marcadas.

El primer paso para alcanzar este objetivo -o hito del proyecto- es la reunión de planificación, una sesión en la que debe participar todo el equipo ‘SCRUM’ y que supone el pistoletazo de salida del ‘sprint’. Esta reunión se divide en dos partes que tratan de dar respuesta a dos preguntas fundamentales: ¿Qué se va a entregar?, y ¿cómo se va a realizar el trabajo?

Cuando el ‘sprint’ ha comenzado, cada uno de los miembros del equipo ejerce su rol, asegurándose de que se cumplan siempre las siguientes condiciones:

- No se realizan cambios que pongan en peligro el objetivo.
- Los estándares de calidad no disminuyen.
- El 'Product owner' y el equipo de desarrollo trabajan conjuntamente ajustando el detalle de las funcionalidades planificadas para el 'sprint'.
- Además de construir el producto, todo equipo trabaja conjuntamente en la redefinición del proyecto. El 'Product owner' y los miembros del equipo aclaran y negocian entre ellos a medida que se aprende más.

Al final de cada 'sprint' se lleva a cabo una labor de inspección y revisión del trabajo realizado, en la que el 'Product owner' (o incluso el propio cliente) da 'feedback' al equipo. En esta sesión, el propietario del producto decide si se acepta o no como válida la funcionalidad o entregable desarrollado. La reunión debe cumplir una serie de condiciones:

- Los asistentes somos, mi tutor del proyecto y yo.
- Se identifica lo que se ha hecho durante la semana anterior y los avances que ha habido.
- Se detectan los problemas y se analiza cómo se resolvieron.
- Vemos hacia donde podemos orientar el proyecto, y cuáles son los siguientes pasos a seguir.

Todas las metodologías ágiles buscan mejorar de manera continua la forma en la que el equipo se relaciona durante el proceso de desarrollo.

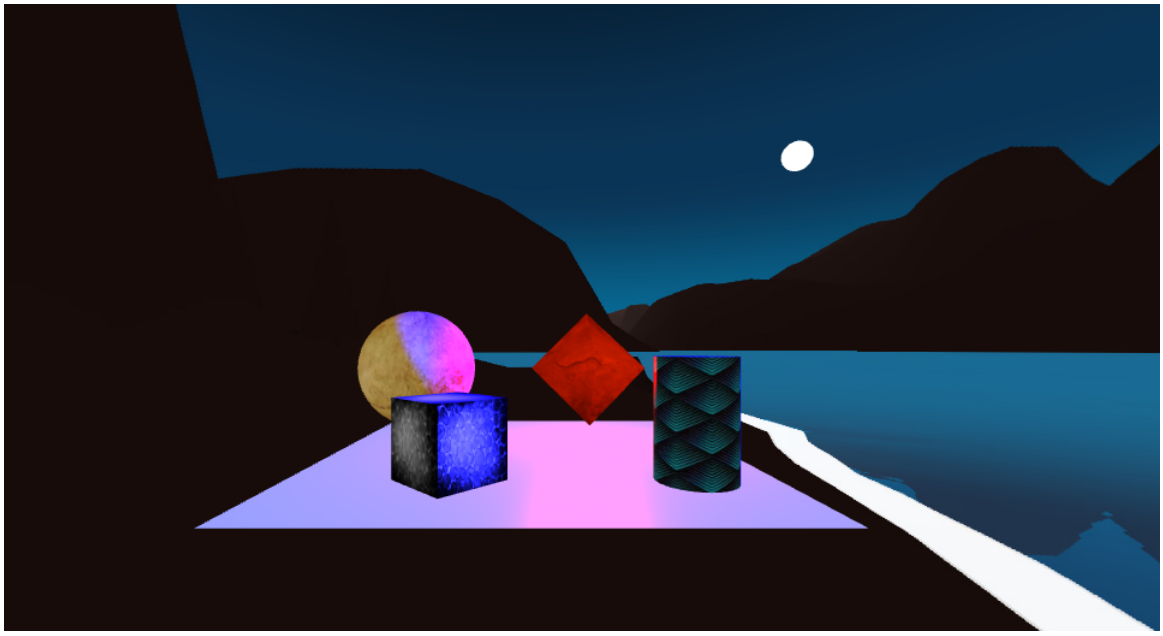
3.2. ETAPAS

3.2.1. Etapa 0: Aprendizaje

Esta etapa es fundamental y una de las más importantes en el desarrollo del proyecto. En esta etapa tuve que aprender desde 0 todo sobre la tecnología A-Frame, ya que no tenía ninguna noción de esta tecnología, y gracias a la documentación[1], ejemplos[2] y tutoriales conseguí aprender y poder construir escenas en realidad virtual.

[1] `https://aframe.io/docs/1.1.0/introduction/`
[2] `https://aframe.io/`

Para poner todo esto en práctica e inicializarme en el mundo de A-Frame, construí una escena con varias animaciones, texturas, luces y una componente que tiene varias propiedades, que es la que se muestra a continuación:



Después de todas estas pruebas llegamos a la creación de nuestra primera escena[3] del proyecto, que está compuesta por 2 ordenadores que se intercambian los 7 paquetes que indicábamos anteriormente(3 paquetes de señalización(SYN), 2 paquetes de datos y otros 2 paquetes de Fin de conexión) al pulsar el botón azul.

En esta escena podemos ver varios elementos que la componen, y algunos de ellos tienen asociada una animación. Si visitáis la escena[4], podréis ver las siguientes animaciones:

- Una esfera blanca que se comportara como punto de luz y se irá moviendo de un lado a otro.
- Un octaedro, que está girando constantemente, y al pinchar sobre él, se irá alejando y acercando.
- La esfera colocada en el plano, al pinchar sobre ella, empezará a rotar y se pondrá de color verde.

[3] `https://alejandroeslo.github.io/TFG/Pruebas/Pasos_seguidos/P1.html`

[4] `https://alejandroeslo.github.io/TFG/Pruebas/Prueba_inicial/Escena_prueba.html`

3.2.2. Etapa 1: Comienzo del proyecto

Después de haber estado practicando con la escena inicial tras haberme leído y estudiado la documentación de A-Frame y las otras tecnologías necesarias para su elaboración, pasaremos a comenzar a crear nuestro primer programa relacionado con el proyecto. Este programa será una aplicación sencilla [Cliente – Servidor], y avanzaremos en dos direcciones:

- A-Frame:

  Comenzaremos construyendo una escena en la que tengamos dos cajas iniciales, una será el cliente y otra será el servidor.

  Incluiremos 7 paquetes que serán enviados desde el cliente (emisor) hasta el servidor (receptor), estos paquetes los empezaremos introduciendo en una lista en la que distinguiremos 3 paquetes de señalización (SYN), 2 paquetes de datos y otros 2 paquetes de Fin de conexión.

  Estos paquetes se enviarán mediante diferentes acciones que definiremos como por ejemplo pulsar una tecla, y haremos que los paquetes se envíen de manera automática o paso a paso(break point).

  Resumen:

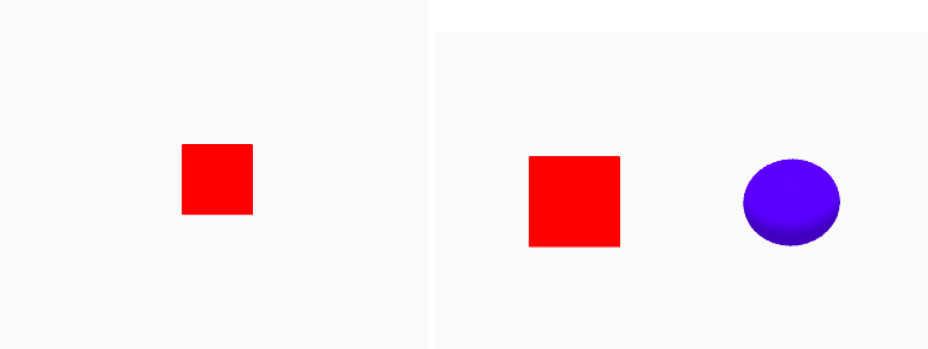
  - 2 Cajas (cliente y servidor).
  - 7 paquetes diferentes que vayan de una caja a otra.
  - Funcionalidad para movimiento de los paquetes.

- Wireshark:

  Sacar una traza de TCP en formato JSON para poder luego analizarla y ver que campos contiene y poder extraer estos datos mediante Javascript.

Para empezar con ello, primeramente tuve que realizar unas demos para ver como controlar las entidades y gestionar llamadas a los eventos para ver como podía manejar correctamente el movimiento de los paquetes de un ordenador a otro al darle a un botón.

Primero comencé por crear una escena[5] básica donde únicamente tenía un cubo y al pinchar sobre el cubo apareciese una esfera a su lado:

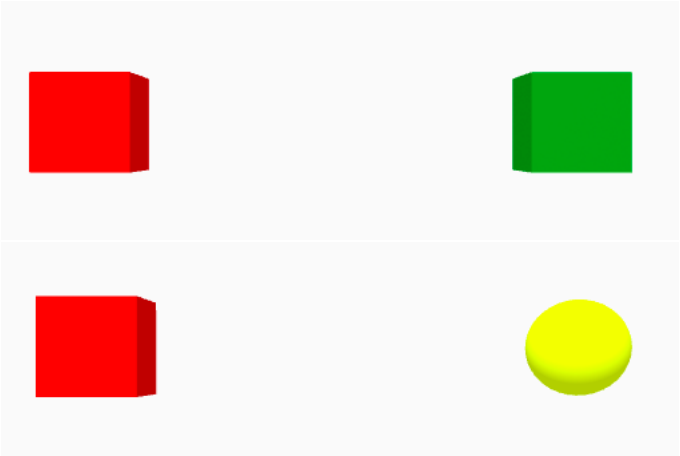


Con ello comenzamos a ver como se controlan los eventos, y en este ejemplo podemos apreciar el evento creado con la función "created_sphere" que nos creara esa esfera azul al pinchar en el cubo.

```
var scene = document.querySelector('a-scene');
var entity = document.createElement('a-entity');
// Creamos la caja y agregamos la caja a nuestra escena.
scene.appendChild(entity);

entity.setAttribute ( 'geometry', {primitive: 'box', height: 2, width: 2});
entity.setAttribute('material', 'color', 'red');
entity.setAttribute ( 'position' , { x : 0 , y : 2 , z : -10 });

entity.addEventListener('click', created_sphere);
  function created_sphere()
  {
    console.log('Dentro de eventlistener')
    var new_sphere = document.createElement('a-sphere');
    // Creamos la caja y agregamos la caja a nuestra escena.
    scene.appendChild(new_sphere);
    new_sphere.setAttribute('material', 'color', 'blue');
    new_sphere.setAttribute ( 'position' , { x : 5 , y : 2 , z : -10});

  };
```

[5] https://alejandroeslo.github.io/TFG/Pruebas/Pruebas_eventos/e2.html

A continuación pasamos a crear 2 cajas, y al hacer click en una, se cree un evento en la otra. Por lo tanto, ahora en esta nueva escena[6], al hacer click en la caja roja, la caja verde se convertirá en una esfera amarilla:

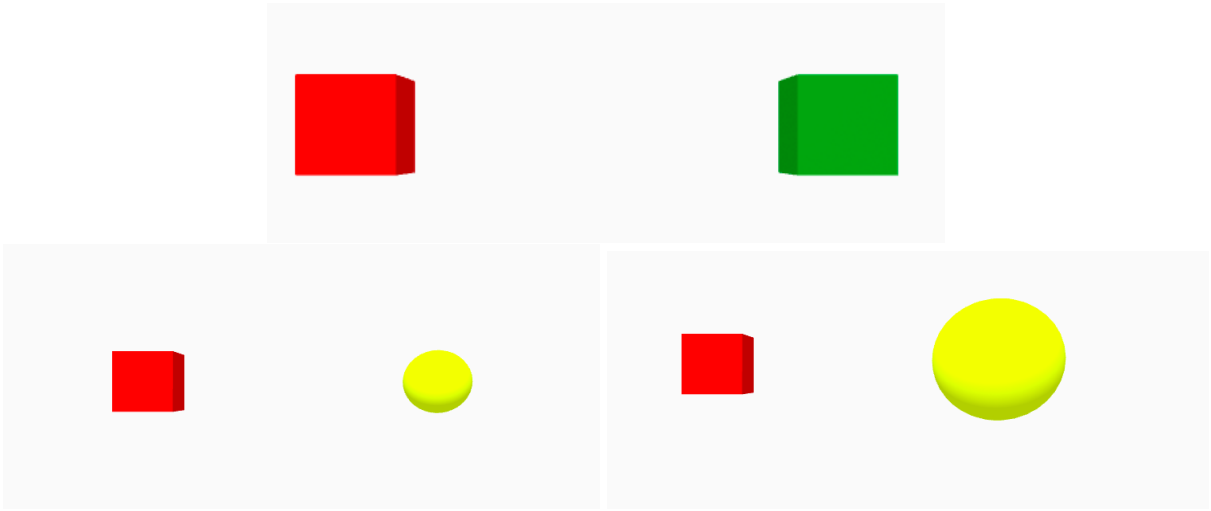


Ahora podemos ver el evento creado con la función " change_box2" que al hacer click en el cubo verde, que es la entidad 2(entity2), se cambiara por una esfera amarilla.

```
// Creamos la caja y agregamos la caja a nuestra escena.
scene.appendChild(entity1);
scene.appendChild(entity2);

entity1.setAttribute ( 'geometry', {primitive: 'box', height: 2, width: 2});
entity1.setAttribute('material', 'color', 'red');
entity1.setAttribute ( 'position' , { x : -5 , y : 2 , z : -10 });

entity2.setAttribute ( 'geometry', {primitive: 'box', height: 2, width: 2});
entity2.setAttribute('material', 'color', 'green');
entity2.setAttribute ( 'position' , { x : 5 , y : 2 , z : -10 });

entity1.addEventListener('click', change_box2);
  function change_box2()
  {
    console.log('Dentro de eventlistener')
    entity2.setAttribute('geometry',{primitive:'sphere'});
    entity2.setAttribute('material', 'color', 'yellow');
  };
```

[6] https://alejandroeslo.github.io/TFG/Pruebas/Pruebas_eventos/e3.html

Y en la escena[7] final para controlar los eventos pondremos crearemos un componente para usarlo como manejador sobre la escena anterior, que lo que nos permitira sera que al hacer click en la esfera amarilla, esta comenzara una animacion en loop que creara un efecto de que la esfera se aleja y se acerca :

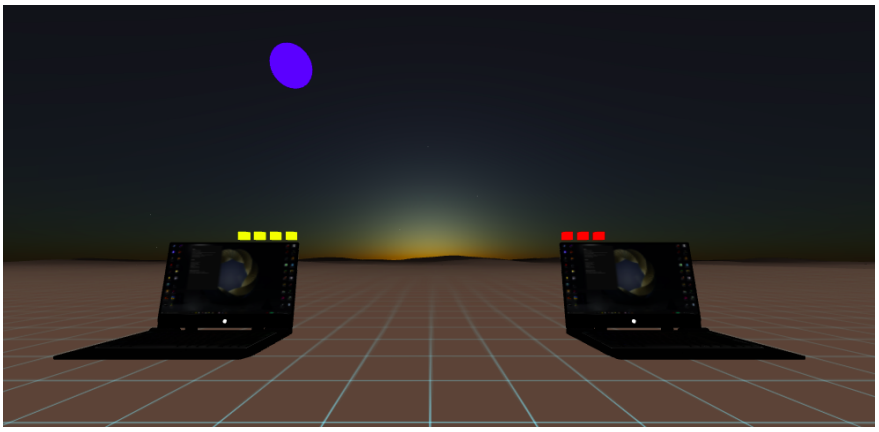


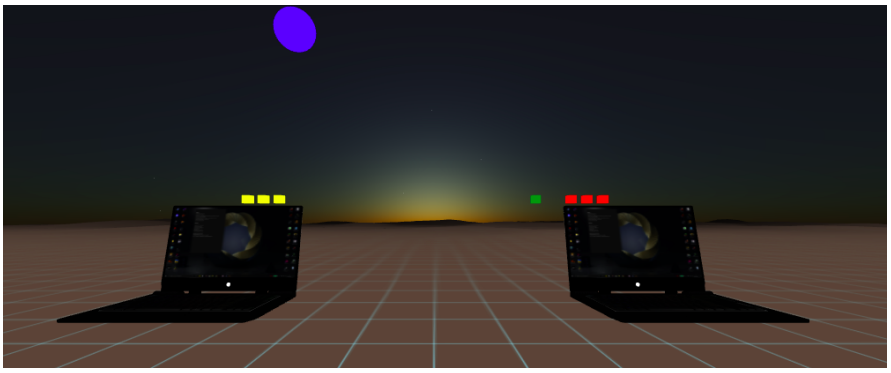
Ahora podemos ver el evento creado con la función “change_box2” ahora incluimos el manejador, que es un componente que hemos creado anteriormente para darle la animación a la esfera amarilla.

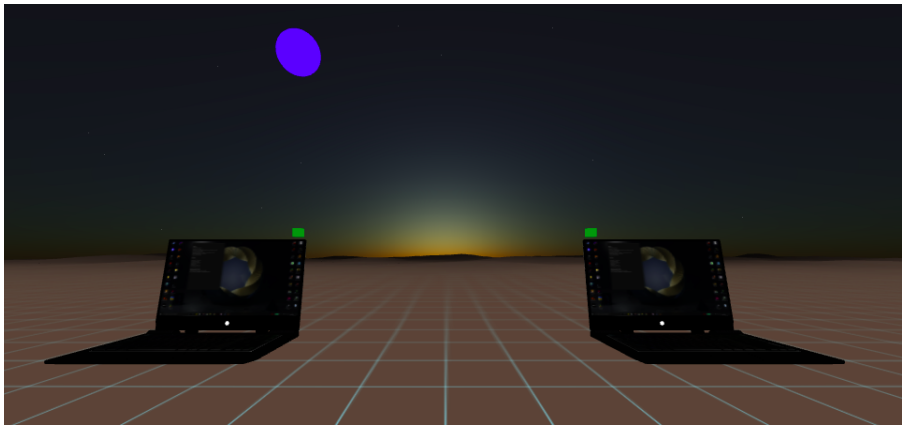
```
entity1.addEventListener('click', change_box2);
  function change_box2()
  {
    console.log('Dentro de eventlistener')
    entity2.setAttribute('geometry',{primitive:'sphere'});
    entity2.setAttribute('material', 'color', 'yellow');
    entity2.setAttribute('manejador','');
    //Prueba:
    //entity2.setAttribute('animation', 'property:position; to:1.75 1.85 -4; dur:4000; dir:alternate; loop:true')

  };


AFRAME.registerComponent('manejador', {
  init: function () {
    console.log('Dentro de MANEJADOR');
    this.el.addEventListener('click', () => {
        this.el.setAttribute('animation', 'property:position; to:1.75 1.85 -4; dur:4000; dir:alternate; loop:true')
      })
  }
});
```

[7] https://alejandroeslo.github.io/TFG/Pruebas/Pruebas_eventos/e4.html

Después de todas estas pruebas llegamos a la creación de nuestra primera escena[8] del proyecto, que está compuesta por 2 ordenadores que se intercambian los 7 paquetes que indicábamos anteriormente (3 paquetes de señalización (SYN), 2 paquetes de datos y otros 2 paquetes de Fin de conexión) al pulsar el botón azul. Los paquetes empezarán con un color, y una vez que lleguen al destino se volverán verdes.







[8] https://alejandroeslo.github.io/TFG/Pruebas/Pasos_seguidos/P1.html

3.2.3. Etapa 2: Primer prototipo

Después de haber creado nuestra primera escena con animaciones de los paquetes entre cliente y servidor, ahora vamos a aumentar el número de nodos(ordenadores) y conectarlos entre sí mediante tubos transparentes. Avanzaremos en 2 direcciones diferentes:

- Animación:

  Crearemos varios ordenadores mas distribuidos en la escena e interconectados todos a través de unos tubos transparentes por los cuales se enviaran y recibirán los paquetes.

- Wireshark:

  Trataremos de leer las capturas exportadas en JSON desde JavaScript. Para ello lo que haremos sera recibir el fichero JSON de la red mediante http y pedir este fichero mediante GET para convertirlo después en un objeto y poderlo mostrar por la linea de comandos del navegador.

  Crearemos un bucle para leer(por ejemplo: las direcciones Ethernet de todos los paquetes de la captura), y después mostrarlos a través de la linea de comandos del navegador.

En esta etapa nos centraremos en crear una escena[9] más grande, con mayor número de ordenadores interconectados entre sí a través de tuberías transparentes por las cuales se enviaran los paquetes después de pulsar el botón que iniciara la animación.

Comenzaremos con la creación y colocación de los nodos(ordenadores) con unas posiciones fijas que he calculado para que quede visualmente bien y se puedan ver todos los nodos intercambiar los paquetes. Aqui podemos ver un par de nodos como ejemplo:

```
<!-- Link Computer 2 -->
<a-gltf-model id = "Computer_link2"
  src="gaming_laptop/scene.gltf"
  position="8 0.5 -20"
  scale="0.5 0.5 0.5"
  rotation="0 -90 0">
</a-gltf-model>

<!-- Link Computer Center -->
<a-gltf-model id = "Computer_linkCenter"
  src="gaming_laptop/scene.gltf"
  position="0 0.5 -12"
  scale="0.5 0.5 0.5"
  rotation="0 -90 0">
</a-gltf-model>
```

[9] https://alejandroeslo.github.io/TFG/Pruebas/Pasos_seguidos/P2.html

A continuación, pasaremos a crear los 7 paquetes que queremos intercambiar, de manera que cada uno de ellos empiece desde uno de estos nodos que hemos creado anteriormente y llegue hasta otro diferente. Los crearemos de manera manual, creando las entidades a mano, sin automatizar aun el proceso, y serían los siguientes:

```
var p_syn = document.createElement('a-entity');
var p_ack = document.createElement('a-entity');
var p_ack2 = document.createElement('a-entity');
var p_datos = document.createElement('a-entity');
var p_ack_data = document.createElement('a-entity');
var p_fin = document.createElement('a-entity');
var p_ack_fin = document.createElement('a-entity');
```

Y un ejemplo de la composición de uno de ellos sería así:

```
p_syn.setAttribute( 'geometry', {primitive: 'box'});
p_syn.setAttribute( 'id', 'syn');
p_syn.setAttribute( 'scale', {x : 0.1 , y : 0.1 , z : 0.1});
p_syn.setAttribute('material', 'color', 'yellow');
p_syn.setAttribute( 'position' , { x : -8 , y : 2 , z : -4.8 });
```

Para la conexión de los nodos, crearemos las tuberías transparentes, que crearán la conexión entre los nodos por las que irán los paquetes. Para ello las crearemos de manera manual, y las rotaremos para que nos encajen con las posiciones de los nodos que hemos establecido.

```
var pipeline_ahead = document.createElement('a-entity');
var pipeline_behind = document.createElement('a-entity');
var pipeline_ClientToCenter = document.createElement('a-entity');
var pipeline_ServerToCenter = document.createElement('a-entity');
var pipeline_Computer_link1ToCenter = document.createElement('a-entity');
var pipeline_Computer_link2ToCenter = document.createElement('a-entity');
var pipeline_right = document.createElement('a-entity');
var pipeline_left = document.createElement('a-entity');
```

Y un ejemplo de una de ellas sería la siguiente:

```
pipeline_ahead.setAttribute( 'geometry', {primitive: 'cylinder'});
pipeline_ahead.setAttribute( 'id', 'pipeline_ahead');
pipeline_ahead.setAttribute( 'scale', {x : 0.1 , y : 15 , z : 0.1});
pipeline_ahead.setAttribute('material', 'color', 'cyan');
pipeline_ahead.setAttribute('material', 'opacity', '0.15');
pipeline_ahead.setAttribute('rotation', {x: 90, y: 90, z: 0});
pipeline_ahead.setAttribute( 'position' , { x : 0 , y : 2 , z : -4.8 });
```

Y, para terminar, añadiremos a la escena un botón, el cual nos permitirá iniciar la animación de los paquetes de un ordenador a otro:

```
var button_action = document.createElement('a-entity');
```

Después añadiremos todos los elementos a la escena y nos pondremos con la creación de los componentes que crearan las animaciones de los paquetes. El primer componente que he creado es “clicker”, el que gestionara cada click que hagamos en el botón, comprobara que paquete es y lo enviara a la función “moving_packages” que incorporara la animación a dicho paquete.

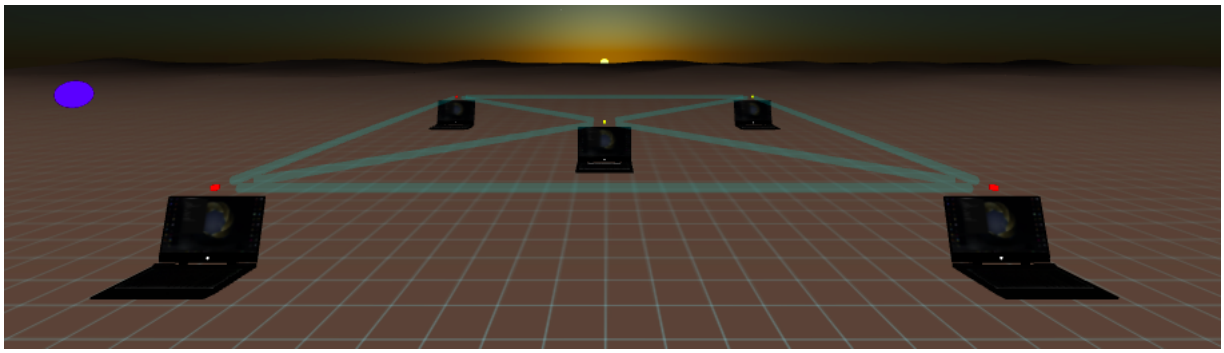
```
AFRAME.registerComponent('clicker', {
  schema: {
    to:{type: 'vec3'}
  },
  init: function () {
    console.log('In to the clicker');
    let packet = 'syn';

    this.el.addEventListener('click', () => {
      packet_el = document.getElementById(packet);
      packet_el.emit('moving_packages');
      console.log('packages active = ' + packet);

      if (packet == 'syn') {
        packet = 'ack';
      } else if (packet == 'ack') {
        packet = 'ack2';
      }else if (packet == 'ack2') {
        packet = 'datos';
      }else if (packet == 'datos') {
        packet = 'ack_data';
      }else if (packet == 'ack_data') {
        packet = 'fin';
      }else if (packet == 'fin') {
        packet = 'ack_fin';
      }else {
        packet = 'syn';
      }
    })
  }
```

Y por último añadiremos el componente “move” que incluye la función “moving_packages”. Esta componente vera que paquete es el que le ha llegado y le añadirá la animación, con la posición destino que le hemos metido manualmente antes, como en este ejemplo:

`p_syn.setAttribute('move','to','0 2 -12.8');` , y lo pondrá de color verde.

```
AFRAME.registerComponent('move', {
  schema: {
    to:{type: 'vec3'}
  },
  init: function () {
    console.log('In to the MOVE');
    let el = this.el
    let data = this.data
    let dir = Object.values(data.to)

    this.el.addEventListener('moving_packages', () => {
        console.log('First value to DATA.to = ' , data.to);
        el.setAttribute('animation', {property:'position', to:data.to , dur:3000, dir:'alternate'});
        el.setAttribute('material', 'color', 'green')
        // Make the package disappear upon arrival at the destination
    //      if (el.position == data.to) {
    //      el.setAttribute('material', 'opacity:0');
    //    }
        console.log('el.position = ' + el.position)
      })
  }
});
```

Ahora cada vez que se pulsa el botón, el paquete correspondiente cambia de color a verde, y se moverá desde el ordenador origen al destino a través de las tuberías.Y la escena final quedaria así:



En la parte de Wireshark, empezaremos a sacar algunos datos que nos interesan de la captura. Mediante XMLHttpRequest(), empezaremos a obtener estos datos, realizando al principio una petición GET para ir obteniendo los valores que nos interesen.

```
const request = new XMLHttpRequest();
const requestURL = '/http_ejemplo.json';

request.open('GET', requestURL);
request.responseType = 'text';
request.send();
```

Crearemos un for para recorrer todos los paquetes de la captura y sacar los datos que nos interesan:

```
for (let i = 0; i < num_packets; i++)
```

Y obtendremos las direcciones IP y Ethernet recorriendo estos campos:

```
for(key in data[i]._source){
  //console.log(data[i]._source[key]);
  let acces1 = data[i]._source[key].eth
  //console.log(acces1);
  console.log("--------------------Paquete " + i + "------------------
  //Destination Address
  let eth_dst = Object.values(acces1)[0]
  console.log("Ethernet Destination Address " + i + " => " + eth_dst);
  //Source Address
  let eth_src = Object.values(acces1)[2]
  console.log("Ethernet Source Address " + i + " => " + eth_src);
}

for(key in data[i]._source){
  let acces2 = data[i]._source[key].ip
  //Destination Address
  let ip_dst = Object.values(acces2)[14]
  console.log("Ip Address " + i + " => " + ip_dst);
  //Source Address
  let ip_src = Object.values(acces2)[13]
  console.log("Ip Source Address " + i + " => " + ip_src);
```

3.2.4. Etapa 3: Componentes

Ahora comenzaremos con la creación de los componentes para crear de una manera más automatizada los elementos que van a componer nuestra escena.

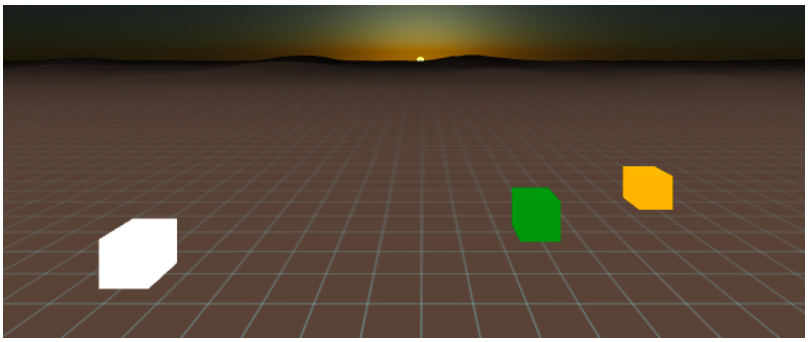
Empezamos con la construcción del componente para los nodos[10]. Este componente creará cada uno los nodos con su correspondiente color y posición, que le será enviada al componente como una variable. Aquí podemos ver el componente:

```
AFRAME.registerComponent('node', {
  schema: {
    obj_type: {type: 'string', default: 'box'},
    color: {type: 'color', default: 'blue'}
  },
  init: function () {
    console.log('In to the NODE Component');
    let el = this.el
    let data = this.data
    //let geo = Object.values(data.obj_type)
    //console.log('Value to data.geometry = ' , data.obj_type);
    el.setAttribute('geometry', {primitive: data.obj_type});
    el.setAttribute('material', 'color', data.color);

  }
});
```

Y crearemos las entidades con los parámetros que nos interesa pasarle luego a nuestro componente `node`:

```
<a-entity id="node1" node="color:green; obj_type:box" position="3 0.5 -6"></a-entity>
<a-entity id="node2" node="color:white; obj_type:box" position="-6 0.5 -4"></a-entity>
<a-entity id="node3" node="color:orange; obj_type:box" position="7 0.5 -8"></a-entity>
```

El resultado que obtendremos será el siguiente:



[10] `https://alejandroeslo.github.io/TFG/Pruebas/Pasos_seguidos/Demos_componentes/P3_1b.html`

A continuación crearemos el componente `connection`, que será el que conectara nuestros nodos mediante tuberías transparentes.

Crearemos las entidades para cada tubería, añadiéndoles el componente `connection` con los atributos TO y FROM que le introduciremos manualmente de momento.

```
<a-entity connection="from: node1; to: node2"></a-entity>
<a-entity connection="from: node1; to: node3"></a-entity>
```

Más tarde, estos campos los rellenaremos con los datos que extraigamos del archivo JSON.

Ahora vamos a crear el componente `connection`, cuya función va a ser la de crear las tuberías transparentes que conectan los nodos.

```
AFRAME.registerComponent('connection', {
  schema: {
    from:{type: 'string'},
    to:{type: 'string'},
  },
```

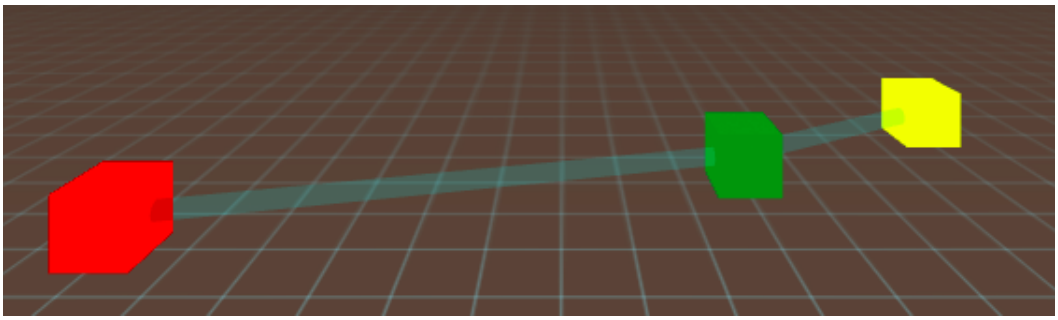
Cogeremos la posición `to` y `from` del nodo correspondiente a su Id, y pasaremos estas coordenas a String para luego poderlas pasar como atributo al objeto y crear el elemento. Aquí podemos ver el ejemplo de las coordenadas `from`:

```
let el = this.el
let data = this.data
let node_from = data.from
let node_to = data.to

// We obtain position Node From
let node_from_el = document.getElementById(node_from);
let pos1 = node_from_el.getAttribute('position');
let pos_from = Object.values(pos1);

// We pass each component of position FROM to string
var fromX = pos_from[0];
var fx = fromX.toString();

var fromY = pos_from[1];
var fy = fromY.toString();

var fromZ = pos_from[2];
var fz = fromZ.toString();

// We concatenate all of them into a single string
var coorFrom = fx.concat(' ', fy.concat(' ',fz))
console.log('Strings Position from = ' + coorFrom );
```

Y en esta parte añadiriamos las coordenadas al elemento `tube`, para que nos cree esas conexiones.

```
// Draw the connection
el.setAttribute('tube', {'path': [coorFrom, coorTo] , 'radius': '0.15' });
el.setAttribute('material', 'color', 'cyan');
el.setAttribute('material', 'opacity', '0.15');
```

Y la escena final se vería de esta forma:



Por último, crearemos el componente `packet`, que será el encargado de ir creando todos los paquetes que vaya encontrando e incluyéndolos en la escena[11].

La entidad creada en este caso estará compuesta por el componente `packet`, que tendrá los siguientes argumentos:color, duración, nodo from, nodo to, tiempo de comienzo:

```
<a-entity packet="color:blue; duration: 2000; from: node1; to: node3; start: 3000"></a-entity>
<a-entity packet="color:black; duration: 4000; from: node2; to: node1; start: 6000"></a-entity>
```

Y en el componente `packet` obtendremos también las posiciones `to` y `from` de los nodos correspondientes, para saber desde donde va el paquet y hasta donde irá:

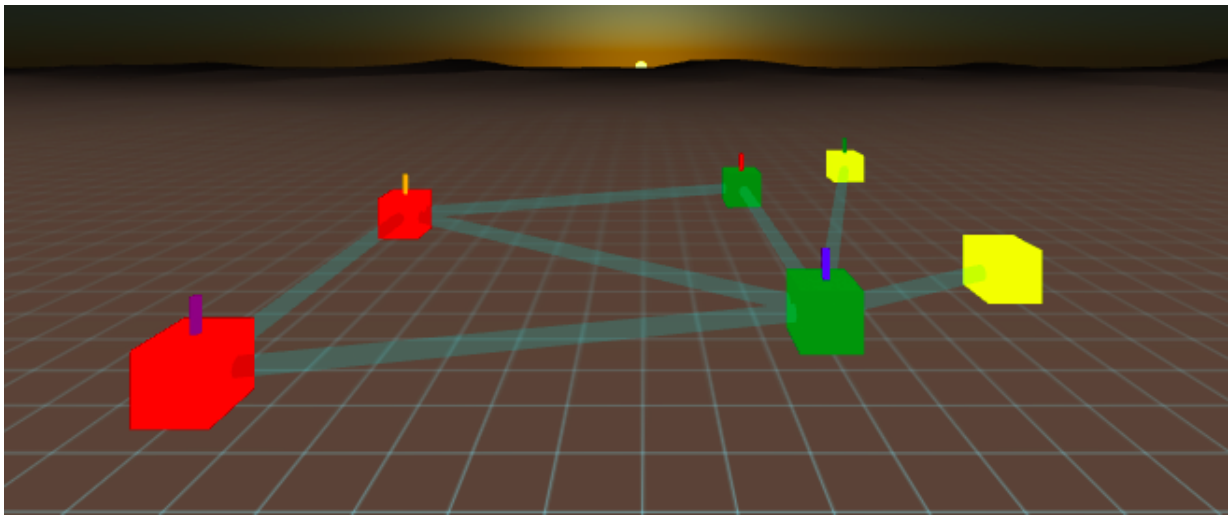
```
let node_from_el = document.getElementById(node_from);
let pos1 = node_from_el.getAttribute('position');
let pos_from = Object.values(pos1);
 data.pos_ini.x = pos_from[0];
 data.pos_ini.y = pos_from[1];
 data.pos_ini.z = pos_from[2];
```

[11] `https://alejandroeslo.github.io/TFG/Pruebas/Pasos_seguidos/Demos_componentes/P3_3a.html`

Añadiremos todos estos datos como atributo a la entidad y crearemos también la animación que será la que hará que se desplacen los paquetes de un nodo al otro:

```
el.setAttribute('geometry', {primitive: 'box'});
el.setAttribute('material', 'color', data.color);
el.setAttribute( 'scale', {x : 0.1 , y : 2 , z : 0.1});
el.setAttribute('position', data.pos_ini);
el.setAttribute('animation', {property:'position', to:data.pos_end , delay:data.start, dur:data.duration});
```

Y la escena final sería así:



3.2.5. Etapa 4: Componente Net Simulator

En este apartado crearemos un nuevo componente, que será el más importante, se trata del componente `net-simulator`. Este componente se encargará de sacar todos los datos necesarios para la creación de la escena, y luego irlos pasando a los componentes correspondientes (`node`, `packet`, `connection` y `poster`) para crear todos los elementos de la escena, nodos, conexiones y paquetes.

Primeramente este componente lo que creará serán todos los nodos que encuentre en la captura, ira iterando por todas las direcciones IP o Ethernet y con ellas creara los nodos de la escena. Aquí comenzaremos creando los nodos de forma lineal para ver como será el funcionamiento, y finalmente acabaremos colocándolos en estructura circular, para que sean visibles todos los nodos.

A continuación creara las conexiones entre estos nodos, solamente creara las conexiones entre los nodos que intercambien paquetes, entre los que no haya ningún intercambio de paquetes no habrá ninguna conexión.

Y por último, creara los paquetes que vengan en la captura y los colocara en el nodo de salida, los cuales se iniciaran la animación después de pulsar el botón.

3.2.6. Etapa 5: Manejador botón y puesta de carteles

Ahora nos ocuparemos de la funcionalidad del botón, para poder controlar la animación correctamente de los paquetes.

Para ello utilizaremos el manejador de tiempos de set interval, para poder iniciar la animación, parar la animación y también reanudar la animación.

```
let i = 1;

function run() {
  if (currentEvent == 'move-pause') {
    //console.log("currentEvent = move-pause");
      if (i == Math.ceil(data.start/1000)) {
        //console.log("DATA %O:",data)
        var packet_move = document.getElementById(data.id);
        //console.log("poner a mover "+ data.id);
        packet_move.setAttribute('animation', {property:'position', to:data.pos_end ,dur:data.duration,
                        pauseEvents:'move-pause', resumeEvents:'move-resume'});
      }
    //console.log("time = "+i);
    i++;
  }
}
```

Controlaremos el click sobre el botón con un evento que se quede escuchando junto al setinterval y dependiendo del estado en el que se encuentre la variable currentEvent, el botón parara o reanudara la animación:

```
var currentEvent = 'move-begin'
mybutton.addEventListener('click', function () {

  //Scroll through the list of selected items
  for (var a=0; a< elements_packets.length; a++) {
    //You add an event to each element
    elements_packets[a].emit(currentEvent,null,false)
  }

  switch(currentEvent) {
    case 'move-begin':
    var refreshInterval = setInterval(run, 1000);
    currentEvent = 'move-pause'
    break

    case 'move-resume':
    currentEvent = 'move-pause'
    break

    case 'move-pause':
    currentEvent = 'move-resume'
    break
    //console.log("currentEvent = " + currentEvent)
  }
});
```

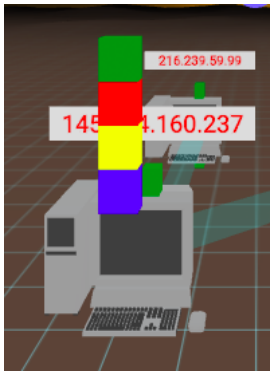
Por último en esta etapa, cogeremos las direcciones IP o Ethernet de cada nodo y las colocaremos en un cartel encima de todos los nodos para identificarlos de manera más sencilla.

3.2.7. Etapa 6: Creación de capas y escena final

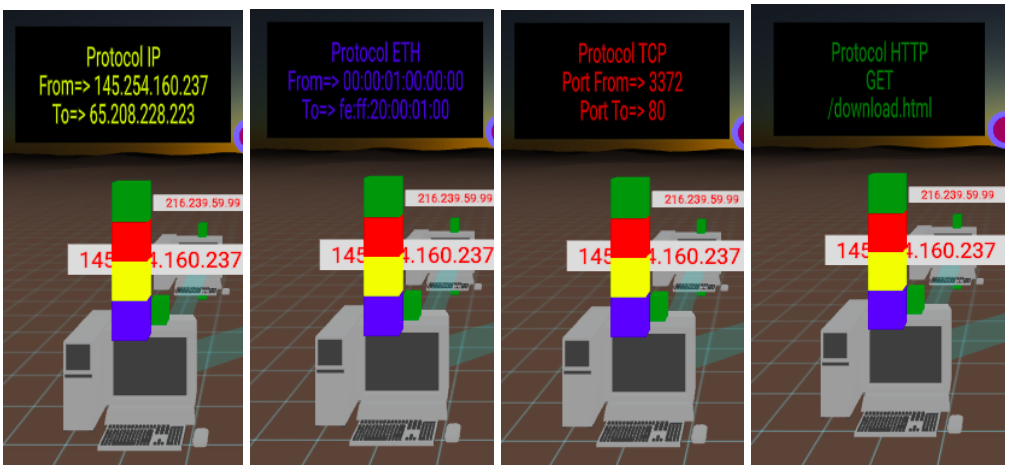
En esta etapa veremos la creación de las capas en los paquetes y las demos finales.

Para terminar la escena hemos introducido en cada uno de los paquetes, las capas de internet que tienen con los datos más relevantes de cada una de ellas. Estas capas no estarán incluidas en todos los paquetes, cada paquete solo mostrará aquellas capas que traiga incorporadas en la captura realizada, es decir, si un paquete solo tiene capa IP y Ethernet, solamente se mostraran estas capas, en cambio si también tiene TCP y HTTP, mostrara las 4 capas.

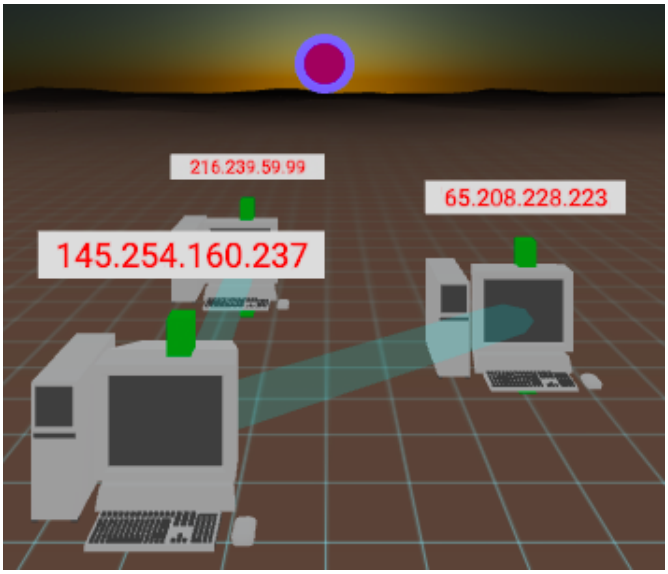
Estas capas aparecerán al pinchar sobre cada paquete, y aparecerán en forma de cubos, cada cubo representará una capa, y están ordenadas de menor a mayor importancia, la de abajo será la de Ethernet, luego IP, después TCP y la última HTTP.

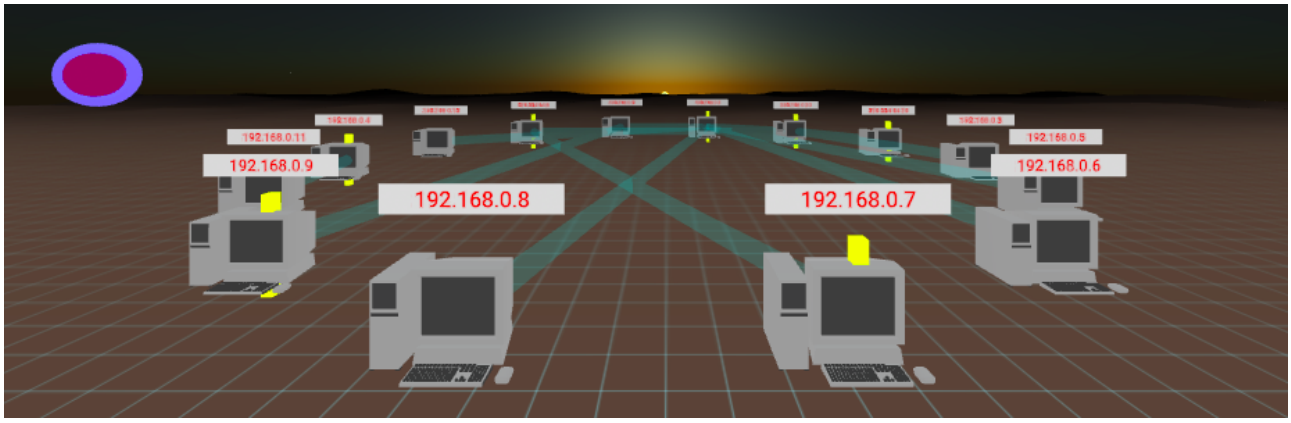


Una vez sean mostradas estas capas, pincharemos sobre la que nos interese para que nos muestre la información que contiene esa capa.



Y para finalizar, crearemos las escenas finales, donde se verán todos los elementos y funcionalidades finales en una misma escena.





3.2.8. Etapa 7:Aplicación en realidad virtual

Esta última etapa ha consistido en la familiarización con las gafas y mandos de realidad virtual, que en mi caso he utilizado las oculus quest, y con la modificación de las escenas para poder visualizarlas y desplazarnos por ellas en Realidad Virtual.

Gracias a que nuestros componentes están hechos con A-Frame, y cada uno de ellos se podría utilizar de manera independiente en otros proyectos, únicamente con pocas líneas de código somos capaces de controlar el movimiento y las animaciones en la escena.

Para poder interactuar con los objetos y botones de la escena y podernos mover por ella con los mandos de las gafas de realidad virtual, he tenido que incluir las siguientes líneas de comandos:

```
<a-entity movement-controls="fly: true" position="0 0 0">
  <a-entity camera position="0 5 10" look-controls></a-entity>
  <a-entity cursor="rayOrigin:mouse"></a-entity>
  <a-entity laser-controls="hand: right"></a-entity>
</a-entity>
```

- El componente `movement-controls` actúa como una plataforma para la cámara.
- El parámetro `fly: true` es para permitir que la cámara se mueva por encima y por debajo del plano horizontal (Y = 0).
- Incluiremos también la entidad cursor que ya habíamos utilizado anteriormente y la entidad camera con la modificación de añadirle el parámetro `look-controls`
- `Laser-controls` se ocupa del trazado de rayos láser en la escen para seleccionar un objeto con el mando.

Capítulo 4

Resultado Final

4.1. Manual de usuario

4.1.1. Usuario de una escena ya constriuida

En este apartado explicaremos como usar la aplicación en un usuario a nivel alto, es decir, como un usuario con la aplicación ya desplegada en el navegador (una página HTML que incluye ya el simulador y la traza para dibujar la escena completa) debería usarla. Diferenciaremos entre usuarios de escritorio, que abrirán la escena desde un ordenador, tablet o cualquier dispositivo móvil y los usuarios de realidad que utilizarán las gafas de realidad virtual para visualizar la escena.

- Usuarios de Escritorio.

  Primeramente accederemos al repositorio GitHub[1], dónde tendremos disponibles dos demos para visualizar, Demo1_100Packets.html[2] y Demo2_HTTPPackets.html[3], elegiremos la que deseemos y la abriremos en un navegador.

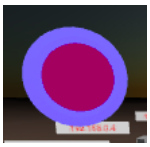
  Una vez nos encontremos en la escena, podremos movernos con las flechas del teclado hacia adelante, hacia atrás y hacia ambos lados. Para movernos en diferentes ángulos, podremos ayudarnos del ratón, pulsaremos con el click izquierdo del ratón y sin soltarlo moveremos el ratón para girar sobre la escena y desplazarnos por toda la escena.

[1] https://github.com/AlejandroEsLo/TFG

[2] https://alejandroeslo.github.io/TFG/Demo1_100Packets.html

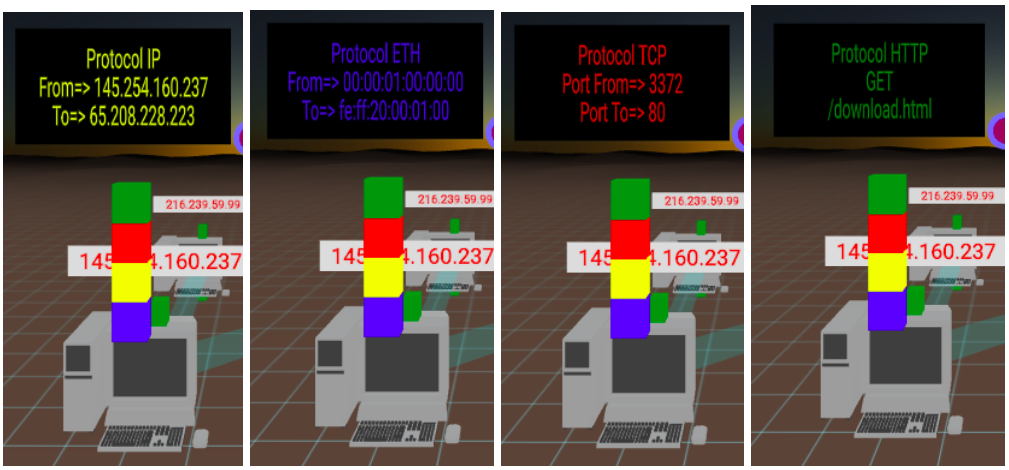
[3] https://alejandroeslo.github.io/TFG/Demo2_HTTPPackets.html

Para poner en funcionamiento la animación de los paquetes, deberemos de colocarnos enfrente del botón y pulsarlo con un click del ratón, esto nos permitirá empezar la animación de los paquetes. De esta misma forma, podremos detener los paquetes para analizarlos, y volverlos a poner en marcha cuando queramos de nuevo.



Si ahora queremos analizar los paquetes, lo único que debemos de hacer es clicar sobre un paquete (será más fácil si paramos la animación), y una vez hagamos esto, se nos desplegaran unos subniveles con las capas que contiene cada paquete.

Para ver la información que trae cada capa, volvemos a pinchar en cada uno de estos subniveles y se nos mostrará un cartel con los datos de esa capa.



Para cerrar cada subnivel, volvemos a pinchar en el subnivel seleccionado y si queremos salir de la visión de los subniveles, volvemos a pinchar sobre el paquete y se volverán a ocultar.

- Usuarios de Realidad Virtual.

  Para este tipo de usuario, el acceso a la escena es similar que el explicado en el usuario de escritorio, hay que acceder a un navegador y del repositorio anteriormente mencionado elegir la escena que queramos visualizar.

Una vez se nos abra la escena en el navegador de las gafas de realidad virtual, con uno de los mandos apuntaremos hacia el botón que nos da acceso a la realidad virtual, que está colocado en la parte inferior izquierda, es el siguiente:

Pulsaremos al gatillo de atrás del mando cuando estemos sobre este botón y nos meteremos en la escena de realidad virtual. Ahora ya estamos inmersos en la escena y podremos empezar a desplazarnos por ella.

Una vez estemos en la escena con el joystick del mando izquierdo podremos desplazarnos por la escena. Para movernos tendremos que mirar hacia donde queremos ir y con el joystick apuntar hacia esa dirección, por ejemplo, si queremos subir en la escena y situarnos más arriba, lo que podemos hacer es mirar hacia abajo y darle al joystick hacia atrás y con eso subiremos en la escena hacia arriba.

El mando derecho, nos aparecerá en la escena con un láser azul. Este nos permitirá apuntar sobre cualquier botón u objeto con el que queramos interaccionar, y presionando el gatillo de atrás accionaremos el botón u objeto al que estemos apuntando con el láser.

Para poner en marcha y ver las capas de cada paquete, lo que haremos es apuntar con el láser del mando derecho al botón o a los paquetes y darle al gatillo para poder visualizar tanto la animación de los paquetes con el botón, como las capas desplegadas en cada uno de los paquetes.

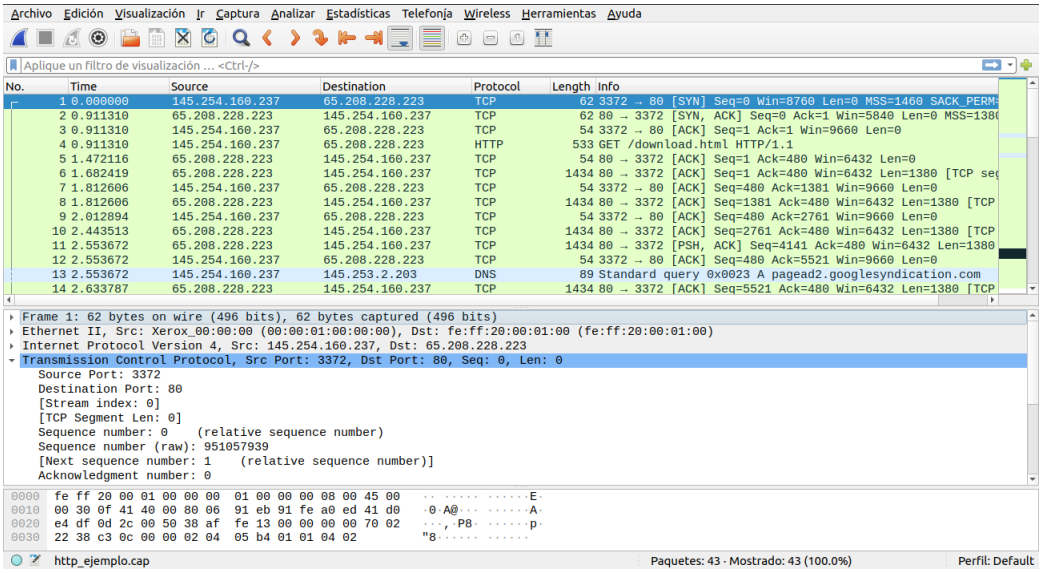
4.1.2. Usuario que construye una escena

En este apartado veremos como ya un usuario avanzado puede crear él mismo una escena.

Primeramente realizaremos la captura de red en Wireshark de la red que nos interese analizar,y la exportaremos como un archivo JSON, para ello:

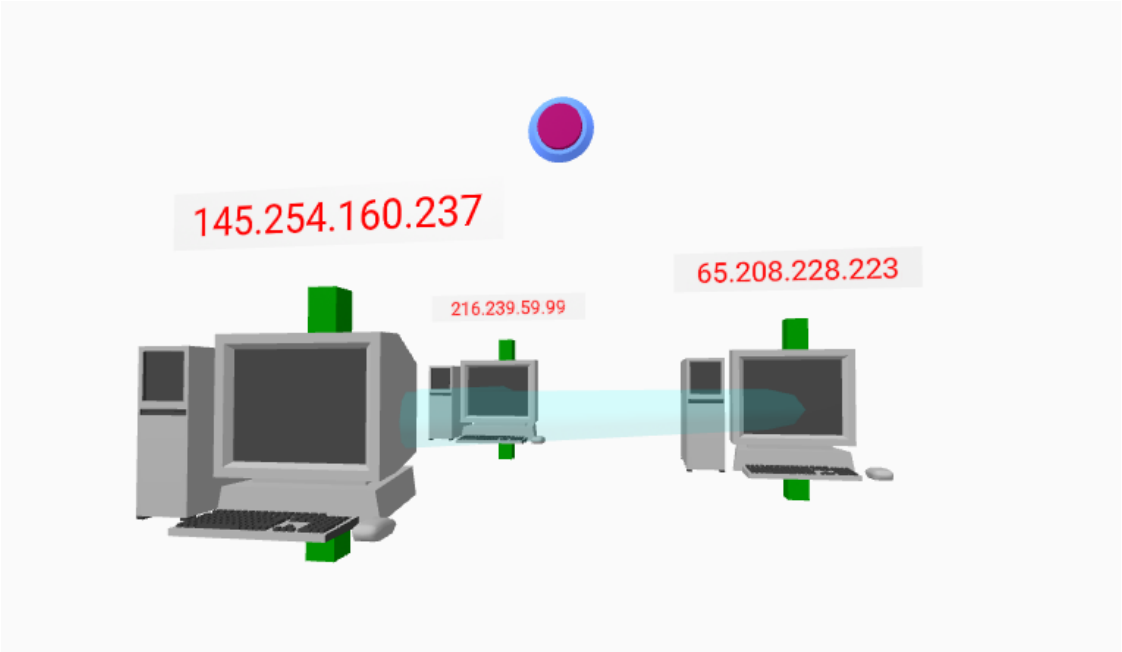
1. Pincharemos sobre la pestaña Archivo.
2. Buscaremos la opción Exportar análisis de paquetes.
3. Y elegiremos la opción exportar como JSON.

Un ejemplo de captura en Wireshark sería la siguiente:



Con esto ya tendríamos exportada nuestra captura para poderla introducir en el programa y crear la escena de dicha captura.

A continuación, lo que hay que hacer es descargarse la demo básica HTML `demo_pruebas.html`[4] de mi repositorio de GitHub. Se trata de una escena vacía en la que se incluye una escena de referencia, pero sin ningún entorno gráfico ni botones adicionales, únicamente se incluyen los repositorios básicos para el funcionamiento del programa.



[4] `https://github.com/AlejandroEsLo/TFG/blob/master/demo_pruebas.html`

Si nos interesasen también por ejemplo, el environment que utilizamos, la posición de la cámara, o el botón de instrucciones, se pueden obtener de las demos que se encuentran alojadas en el mismo repositorio y son las siguientes:

`Demo1_100Packets.html`[5] y `Demo2_HTTPPackets.html`[6]

En este archivo podemos ver los parámetros que tiene el componente `net-simulator`, y son los siguientes:

```
<a-entity id="simulator" net-simulator="file: 100packets_demo1.json; proto: ip"></a-entity>
```

- File: es el parámetro en donde tendremos que introducir el nombre completo de nuestra captura realizada en JSON.
- Proto: es el protocolo que nosotros querremos representar en la escena. Este parámetro puede tener dos valores:
  - ip : Nos representará todos los nodos de la capa IP.
  - eth :Nos representará todos los nodos de la capa Ethernet.

Una vez realizados todos estos pasos, e introducida correctamente la captura en el archivo, tendremos varias opciones para visualizar la escena:

- La primera opcion es ejecutar un servidor local con Python para visualizarlo en cualquier navegador.

  Para ello, nos situaremos en la carpeta que tengamos los anteriores archivos y abriremos un terminal, y escribiremos la siguiente línea de comandos:

  ```
  python3 -m http.server 8000
  ```

  Después nos dirigimos a cualquier navegador y escribimos la siguiente dirección(teniendo en cuenta el nombre de nuestro archivo si lo hemos cambiado):

  http://localhost:8000/demo_pruebas.html

[5] `https://github.com/AlejandroEsLo/TFG/blob/master/Demo1_100Packets.html`
[6] `https://github.com/AlejandroEsLo/TFG/blob/master/Demo2_HTTPPackets.html`

Y en esa página ya podremos visualizar nuestra escena completa con todos los cambios o complementos que le hayan sido añadidos.

- Podemos crearnos un repositorio en GitHub donde alojemos el archivo y lo vayamos actualizando, y esta plataforma nos servirá como servidor, y desde ahí podremos acceder a nuestra escena.

- También podemos ejecutar las demos que hay directamente desde el repositorio de GitHub[7], y en el README, al pulsar sobre cualquier dirección, te ejecutara la demos para poder visualizarla.

4.2. Manual técnico

En este apartado vamos a ver los detalles de la estructura del programa y que elementos lo componen.

Los elementos principales del programa son los componentes. Los componentes son módulos de JavaScript que se pueden mezclar, combinar y componer en entidades para crear apariencia, comportamiento y funcionalidad.

Los componentes que componen el simulador son 5: componente poster, componente `node`, componente `connection`, componente `packet` y el componente `net-simulator`. Este último componente será el más importante, ya que será el que cree todos los componentes para su funcionamiento y relacionara todos los componentes para crear la escena final.

Cada uno de estos componentes es una pieza individual del programa y se podrían utilizar en cualquier otro programa, ya que cualquiera de estos componentes puede funcionar por separado.

4.2.1. Componente Node

Este componente será el encargado de crear todos los nodos que sean necesarios en la escena. Tendrá cuatro argumentos de entrada, que le podremos pasar cuando queramos utilizar el componente, que son los siguientes:

[7] `https://github.com/AlejandroEsLo/TFG`

```
obj_type: {type: 'string', default: 'box'},
color: {type: 'color', default: 'blue'},
position: {type: 'vec3'},
id: {type: 'string'}
```

- obj_type: Este argumento nos indicará el tipo de nodo que queremos que nos dibuje en la escena. Puede ser de dos tipos:
  - computer : Nos representará los nodos con un gltf de un ordenador(que es el que nos aparece en las demos).

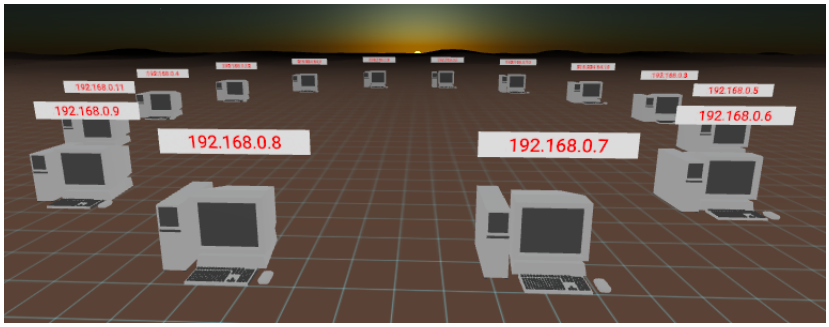
```
if (data.obj_type == 'computer') {
  el.setAttribute('gltf-model', "old_computer/scene.gltf");
  el.setAttribute('scale', '0.015 0.015 0.015');
  el.setAttribute('rotation', '0 90 0');
  el.setAttribute('position', data.position);
  el.setAttribute('id', data.id);
```

  - Si no ponemos nada, nos dibujara una caja como nodo, con el color que le hayamos indicado en la entrada.

```
else {
  el.setAttribute('geometry', {primitive: data.obj_type});
  el.setAttribute('material', 'color', data.color);
  el.setAttribute('position', data.position);
  el.setAttribute('id', data.id)
```

- color: Nos permitirá elegir el color que le queramos poner a cada uno de nuestros nodos.
- position: En este argumento introduciremos la posición en la que dibujaremos y colocaremos nuestro nodo.
- id: Será el identificador correspondiente al nodo.

En la escena final se verían los nodos de la siguiente forma:



4.2.2. Componente Connection

Este componente será el encargado de conectar los nodos creados en el componente anterior mediante tuberías transparentes por las que más adelante pasaran los paquetes de la escena. Los argumentos que tendremos de entrada en este componente son 2:

```
from:{type: 'string'},
to:{type: 'string'},
```

- from: Tendremos las coordenadas del nodo from, desde el nodo en el que debe empezar la tubería de la conexión.
- to: Tendrá las coordenadas del nodo to, desde el nodo en el que acabara la tubería de la conexión.

La funcionalidad de este componente es recibir las coordenadas "from" y "to" y ponerlas en la tubería que dibujemos para ver desde donde hasta donde va a ir esa tubería. Para ello cogeremos por ejemplo la variable from que nos llega y con un getElementById del argumento from que recibimos obtenemos el nodo desde donde tiene que salir la tubería, y después con el getAttribute del atributo position del nodo que hemos obtenido anteriormente, obtendremos las coordenadas en las que empezará la tubería.

```
let node_from_el = document.getElementById(node_from);
let pos1 = node_from_el.getAttribute('position');
let pos_from = Object.values(pos1);
```

Cogeremos estas coordenadas y las pasamos a string una a una (x,y,z).

```
var fromX = pos_from[0];
var fx = fromX.toString();
```

A continuación las concatenamos todas para poderla dibujar después.

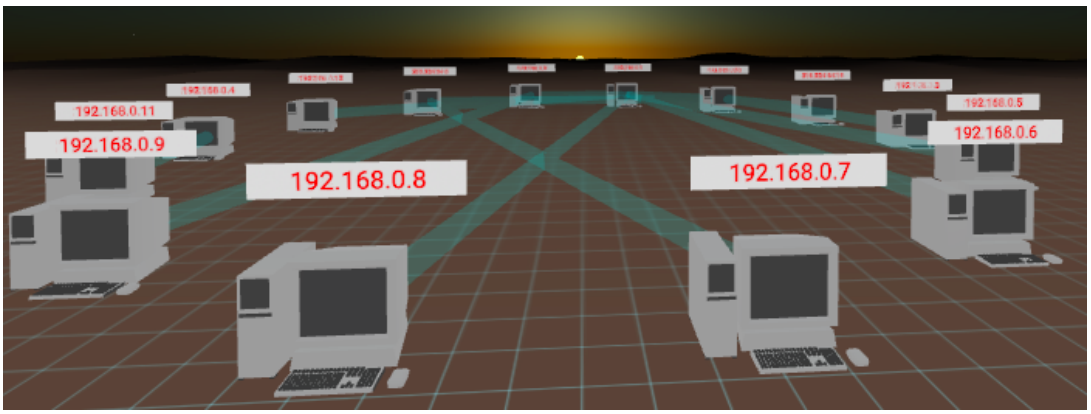
```
var coorFrom = fx.concat(' ', fy.concat(' ',fz))
```

Y por último las dibujaremos poniendo los atributos correspondientes usando la primitiva "tube"[8], que nos permitirá dibujar las conexiones.

[8] https://www.npmjs.com/package/aframe-extras.tube

```
el.setAttribute('tube', {'path': [coorFrom, coorTo] , 'radius': '0.25' });
el.setAttribute('material', 'color', 'cyan');
el.setAttribute('material', 'opacity', '0.15');
```

En la escena final se verían las conexiones de la siguiente forma:



4.2.3. Componente Packet

Este componente tendrá la función de darle forma a los paquetes que contiene la captura, y de controlar el movimiento que estos tendrán al pulsar sobre el botón.

Tendrá varios argumentos de entrada, que le podremos pasar cuando queramos utilizar el componente, y también tenemos otros argumentos que están vacíos y solo los utilizaremos dentro del componente.

Los argumentos de entrada que tenemos son:

- id: Será el identificador correspondiente a cada paquete.
- start: Es el tiempo que tiene que pasar hasta que comience a animarse ese paquete. Se corresponde al time_relative correspondiente en la captura a cada paquete, que es el tiempo correspondiente a cada paquete.
- color: El color que pondremos a cada uno de los paquetes, que dependiendo del protocolo más alto que tenga ese paquete, el paquete tendrá un color u otro.
- duration: Se corresponde al tiempo que transcurre desde que el paquete sale del nodo origen hasta que llega al nodo destino.
- class: Será la clase a la que pertenece cada paquete.

- proto: Es el protocolo que hemos elegido al comenzar, para dibujar los nodos con el nivel IP o Ethernet.
- LastValue: Es la variable en la que introduciremos el nivel más alto de capa que tenga ese paquete, en nuestro caso, el valor más alto que se puede introducir es HTTP, pero también podrá ser TCP, IP o Ethernet.
- info_from_ip: Aquí introduciremos la dirección "from" de la capa IP del paquete correspondiente, para luego mostrarlo en el cartel correspondiente a su capa.
- info_to_ip: Aquí introduciremos la dirección "to" de la capa IP del paquete correspondiente, para luego mostrarlo en el cartel correspondiente a su capa.
- info_from_eth: Aquí introduciremos la dirección "from" de la capa Ethernet del paquete correspondiente, para luego mostrarlo en el cartel correspondiente a su capa.
- info_to_eth: Aquí introduciremos la dirección "to" de la capa Ethernet del paquete correspondiente, para luego mostrarlo en el cartel correspondiente a su capa.
- port_tcp_dst: Aquí introduciremos el puerto destino de la capa TCP del paquete correspondiente, para luego mostrarlo en el cartel correspondiente a su capa.
- port_tcp_src: Aquí introduciremos el puerto origen de la capa TCP del paquete correspondiente, para luego mostrarlo en el cartel correspondiente a su capa.
- info_http1: En esta variable introduciremos el campo correspondiente al método HTTP utilizado en la petición o a la respuesta recibida, por ejemplo la petición podría ser un GET, y la respuesta un OK, esta información la usaremos luego para mostrarla en el cartel correspondiente a su capa.
- info_http2: En esta variable introduciremos el campo correspondiente a la URL utilizado en la petición HTTP o al código de respuesta recibido, por ejemplo la URL podría ser www.download.html, y el código de respuesta un 200, esta información la usaremos luego para mostrarla en el cartel correspondiente a su capa.

Este componente será uno de los más importantes, ya que tendrá varias funcionalidades, las cuales pasaremos a explicar a continuación.

La primera funcionalidad de este componente será la de crear los paquetes primero y situarlos sobre la escena. Esto lo hará cogiendo los datos de las variables anteriormente nombradas, y también obteniendo las posiciones de cada uno de los nodos origen y destino que tendrán que tomar para saber desde qué nodo tendrá que salir el paquete y hasta cuál tendrá que desplazarse. Esto lo haremos igual que en el componente `connection`, utilizando getElementById del argumento para saber las coordenadas de origen y destino del paquete. Con todos estos datos, crearemos los paquetes ya en la escena:

```
el.setAttribute('geometry', {primitive: 'box'});
el.setAttribute('material', 'color', data.color);
el.setAttribute('scale', {x : 0.25 , y : 2.25 , z : 0.25});
el.setAttribute('position', data.pos_ini);
el.setAttribute('class', data.class);
el.setAttribute('id', data.id);
```

A continuación colocaremos también el póster de cada paquete. En este proceso hemos optado por colocar dos pósters, uno por delante y otro girado 180º en las coordenadas x y z, para poder visualizar los pósters tanto si lo queremos ver por delante o por detrás.

```
poster2.setAttribute('poster','position',data.pos_poster);
poster2.setAttribute('scale', {x : 5 , y : 1 , z : 5});
poster2.setAttribute('poster','color','black');
poster2.setAttribute('poster','text',' ' );
poster2.setAttribute('visible', false);
```

```
poster_behind2.setAttribute('poster','position',data.pos_poster);
poster_behind2.setAttribute('scale', {x : 5 , y : 1 , z : 5});
poster_behind2.setAttribute('poster','color','black');
poster_behind2.setAttribute('poster','text',' ');
poster_behind2.setAttribute('rotation', {x: 180, y: 0, z: 180});
poster_behind2.setAttribute('visible', false);
```

Por último, en este componente, controlaremos el tiempo de los paquetes con el setinterval y dependiendo de su estado iremos empezando la animación, congelando los paquetes o reanudando la animación de los paquetes.

```
var mybutton = document.querySelector('#mybutton');
var elements_packets = document.getElementsByClassName('all_packets');
//console.log("paquetes %O:",elements_packets)

var currentEvent = 'move-begin'
mybutton.addEventListener('click', function () {

  //Scroll through the list of selected items
  for (var a=0; a< elements_packets.length; a++) {
    //You add an event to each element
    elements_packets[a].emit(currentEvent,null,false)
  }

  switch(currentEvent) {
    case 'move-begin':
    var refreshInterval = setInterval(run, 1000);
    currentEvent = 'move-pause'
    break

    case 'move-resume':
    currentEvent = 'move-pause'
    break

    case 'move-pause':
    currentEvent = 'move-resume'
    break
    //console.log("currentEvent = " + currentEvent)
  }
});
```

Podemos ver como, cada vez que hacemos click en el botón, cambiara el estado de nuestra variable “currentEvent”, y dependiendo de ese estado, la funcionalidad de los paquetes será distinta.

Después crearemos dos variables(“active” y “view_info”), que utilizaremos para saber cuando mostrar los carteles de cada paquete o cuando mostrar las capas de los paquetes. Para ello crearemos otro evento y dependiendo del estado de estas variables realizaremos una tarea u otra.

El primero de los estados que vamos a ver es cuando las dos variables estén activas (“on”):

```
el.addEventListener('click', function () {
  if (active == 'on' & view_info == 'on') {
```

En este estado, querremos que se vean tanto las capas como los carteles, por lo tanto los pondremos en visible, y añadiremos los textos correspondientes a cada uno de las capas de los

paquetes. Aquí podemos ver un ejemplo del caso de la capa IP:

```
if (info_ip_from != "" || info_ip_to != "") {
  packet_ip.setAttribute('visible', true);
  packet_ip.addEventListener('click', function () {
    //console.log("active ON view ON IP" );
    poster2.setAttribute('visible', true);
    poster2.setAttribute('text', {width:4, color:'yellow', value: 'Protocol IP\nFrom=> '+info_ip_from+ '\nTo=> '+info_ip_to ,align:'center'});
    poster2.setAttribute('geometry', {primitive:'plane',height: 0.75, width: 2.5});
    poster_behind2.setAttribute('visible', true);
    poster_behind2.setAttribute('text', {width:4, color:'yellow', value: 'Protocol IP\nFrom=> '+info_ip_from+ '\nTo=> '+info_ip_to ,align:'center'
    poster_behind2.setAttribute('geometry', {primitive:'plane',height: 0.75, width: 2.5});

    active = 'on'
    view_info = 'off'
    //console.log("Change to => active ON view OFF " );

  });
};
```

Esto lo haremos para cada una de las capas que tengamos , pero con su correspondiente información.

El segundo estado que nos encontraremos será cuando “active” esté en “on” y “view_info” en “off”:

```
}else if (active == 'on' & view_info == 'off') {
```

En este estado, querremos que se vean las capas del paquete, pero no los carteles porque llegaran sin información, ya que no tendrán en este caso. Aquí podemos ver un ejemplo del caso de la capa IP:

```
if (info_ip_from != "" || info_ip_to != "") {
  packet_ip.setAttribute('visible', true);
  packet_ip.addEventListener('click', function () {
    //console.log("active ON view  OFF IP" );
    poster2.setAttribute('visible', false);
    poster_behind2.setAttribute('visible', false);
    active = 'on'
    view_info = 'on'
    //console.log("Change to => active OFF view ON " );
  });
};
```

Esto lo haremos para cada una de las capas que tengamos , pero con su correspondiente información.

El siguiente estado que nos encontraremos será cuando “active” esté en “off” y “view_info” en “off”:

```
}else if (active == 'off'& view_info == 'off') {
```

En este estado, lo que queremos es que no nos muestre ni la capa y no los carteles, por ello pondremos el atributo “visible” en “false”. Aquí podemos ver un ejemplo del caso de la capa IP:

```
if (info_ip_from != "" || info_ip_to != "") {
    packet_ip.setAttribute('visible', false);
}
```

Esto lo haremos para cada una de las capas que tengamos , pero con su correspondiente información.

El último estado que nos encontraremos será cuando “active” esté en “off” y “view_info” en “on”:

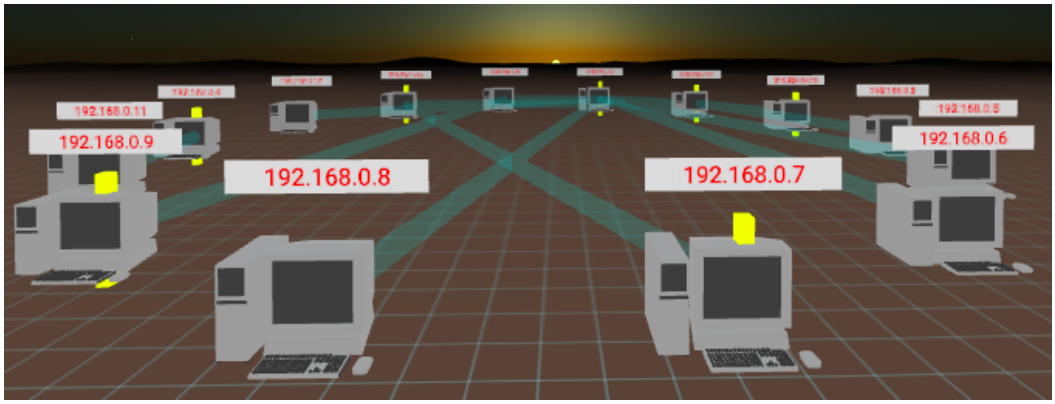
```
}else if (active == 'off'& view_info == 'on') {
```

En este estado, como en el anterior, lo que queremos es que no nos muestre ni la capa y no los carteles, por ello pondremos el atributo “visible” en “false”. Aquí podemos ver un ejemplo del caso de la capa IP:

```
if (info_ip_from != "" || info_ip_to != "") {
    packet_ip.setAttribute('visible', false);
}
```

Esto lo haremos para cada una de las capas que tengamos , pero con su correspondiente información.

En la escena final se verían los paquetes de la siguiente forma:



4.2.4. Componente Poster

Este componente será el encargado de crear los carteles o póster que se ponen en cada una de las capas o encima de los nodos que sean necesarios en la escena. Tendrá tres argumentos de entrada, que le podremos pasar cuando queramos utilizar el componente, que son los siguientes:

```
color: {type: 'color', default: 'white'},
position: {type: 'vec3'},
text: {type: 'string'}
```

- color: Este argumento nos indicará el color de fondo que tendrá el póster.
- postion: Este argumento nos indicará la posición en la que ira colocado el póster.
- text: Este argumento nos indicará el texto que pondremos dentro del póster correspondiente.

Después procederemos a dibujarlos poniendo los atributos correspondientes usando la primitiva “plane”, que nos permitirá dibujar estos carteles.

```
el.setAttribute('geometry', {primitive:'plane',height: 0.5, width: 3});
el.setAttribute('text', {width:8, color:'red', value: data.text, align:'center'});
el.setAttribute('material', 'color', data.color);
el.setAttribute('position', data.position);
```

4.2.5. Componente Net-simulator

Este componente será el más importante de todo el programa. Se trata de una arquitectura modular, es decir, cada uno de los componentes puede funcionar por separado, y este componente tiene la función de unir todos los componentes anteriormente mencionados y ponerlos en funcionamiento.

Tiene dos argumentos de entrada, que son los siguientes:

```
file: {type: 'string'},
proto: {type: 'string',default: 'eth'},
```

- file: Será el archivo JSON qué queramos analizar en nuestro programa. EN nuestra demo, utilizamos por ejemplo este archivo (file: 100packets_demo1.json).

- proto: Será el protocolo que utilizaremos para dibujar la capa de los nodos. Puede ser ip o eth, dependiendo la variable que elijamos, nos dibujara los nodos con las direcciones Ethernet o los nodos con las direcciones IP.

Este componente de lo primero que se va a encargar será de ir sacando los datos necesarios de la captura que le introduzcamos en formato JSON. Para ello lo que haremos será crear un XMLHttpRequest() e iniciarlo con un GET, después iremos sacando los datos que veamos que nos interesa del archivo JSON.

```
const request = new XMLHttpRequest();
const requestURL = data.file;

request.open('GET', requestURL);
request.responseType = 'text';
request.send();
```

Primero sacaremos el time relative de cada paquete, que será el que utilizaremos par ver cuando tiene que salir cada uno de los paquetes del nodo origen. Lo multiplicaremos por 5000 ms para que tenga un tiempo que podamos apreciar en la animación de manera correcta.

```
let acces3 = data[i]._source[key].frame
let time_relative = 5000*Object.values(acces3)[6]
```

Después nos centraremos en sacar los direcciones origen y destino de cada uno de los paquetes, tanto las direcciones Ethernet:

```
let acces1 = data[i]._source[key].eth

let eth_dst = Object.values(acces1)[0]//Destination Address
//console.log("Ethernet Destination Address " + i + " => "

let eth_src = Object.values(acces1)[2]//Source Address
```

como las direcciones IP:

```
let ip_dst = Object.values(acces2)[14]
//console.log("Ip Address Destination '

let ip_src = Object.values(acces2)[13]
```

Ahora pasaremos a ir creando todos los elementos que queremos incluir en la escena. Primeramente comenzaremos por colocar los nodos de la escena.

Comenzaremos creando una pequeña condición que dependerá del argumento que hayamos puesto al principio en el componente `net-simulator`, y dependiendo del protocolo de entrada que hayamos metido, cogeremos el array con los datos de la capa Ethernet o la capa IP.

```
if (proto == "ip") {
  nodes = ip_fin //
}else {
  nodes = eth_fin
}
```

Después iniciaremos unas variables que nos servirán para ir colocando tanto los nodos como los póster de cada uno de ellos. Tendremos una variable para la posición inicial de los nodos(node_pos), otra variable con la posición inicial de los póster de su correspondiente nodo (poster_pos), y luego otras dos variables que usaremos para colocar los nodos en una red tipo circular e iremos aumentando estas variables según vayamos introduciendo más nodos (variables increase y angle).

```
var node_pos = '0 0.5 0' // Position initial
var poster_pos = '0 2 0'
var increase = Math.PI * 2 / nodes.length ;
angle = 0
```

Ahora crearemos un “for” que recorrerá todo el array de nodos que hayamos elegido.

```
for (let i = 0; i < nodes.length; i++) {
```

Cada vez que pasemos por este bucle, cambiaremos la posición X y Z del nodo para formar la red circular como anteriormente habíamos comentado.

```
// Change the position X
var pos_x = parseInt(change_pos[0])
var pos_x = nodes.length*Math.cos(angle);

// Change the position Z
var pos_z = parseInt(change_pos[2])
var pos_z = nodes.length*Math.sin(angle);
pos_z = Math.round(pos_z);
```

Incrementaremos el ángulo también para colocar los nodos correctamente.

```
//Increase the angle
angle += increase;
```

Después añadiremos los atributos necesarios al componente `node` para que nos cree todos los nodos.

```
entity.setAttribute('node','obj_type','computer');
entity.setAttribute('node','color',generarColor());
entity.setAttribute('node','id',nodes[i]);

 // Add the position to the node
 entity.setAttribute('node','position',node_pos);
```

Y finalmente dibujaremos el nodo en la escena final.

```
// Add to the scene
scene.appendChild(entity);
```

Para los carteles de los nodos, la única posición que cambiaremos será la posición Y para que se pongan encima del nodo final.

```
var posterY_node = change_pos[1] * 6;
var posterY_node_behind = change_pos[1] * 6;
```

Añadiremos los atributos correspondientes de cada cartel.

```
poster.setAttribute('poster','position',poster_pos_node);
poster.setAttribute('poster','color','lightgrey');
poster.setAttribute('poster','text',nodes[i]);

poster_behind.setAttribute('poster','position',poster_pos_node_behind);
poster_behind.setAttribute('poster','color','lightgrey');
poster_behind.setAttribute('poster','text',nodes[i]);
poster_behind.setAttribute('rotation', {x: 180, y: 0, z: 180});
```

Y finalmente colocaremos tanto el póster delantero como el trasero ,para verlo por los dos lados en la escena final.

```
// Add to the scene
scene.appendChild(poster);
scene.appendChild(poster_behind);
```

Ahora pasaremos a dibujar las tuberías de conexión entre los nodos. Para ello, primeramente crearemos también un pequeño “if” en el cual elegiremos un array u otro dependiendo del protocolo de entrada como anteriormente hemos visto.

```
if (proto == "ip") {
  connections = connections_ip
}else {
  connections = connections_eth
}
```

Después crearemos un “for” que recorrerá todo este array para coger los datos necesarios:

```
for (let i = 0; i < Object.keys(connections).length; i++) {
```

Después añadiremos los atributos necesarios al componente `connection` para que nos cree estas tuberías.

```
entity2.setAttribute('connection','from',prop.from);
entity2.setAttribute('connection','to',prop.to);
```

Y por último añadiremos la tubería a la escena final.

```
scene.appendChild(entity2);
```

Finalmente crearemos los paquetes y los iremos añadiendo a la escena final. Primeramente comprobaremos el protocolo elegido para la escena, como en los casos anteriores y dependiendo de eso elegiremos unos arrays u otros para obtener los valores necesarios.

```
if (proto == "ip") {
   all_packet = all_packet_ip // cas
   all_packet_info = all_packet_eth
 }else {
   all_packet = all_packet_eth
   all_packet_info = all_packet_ip
 }
```

A continuación iremos sacando los datos de cada capa, los cuales utilizaremos para los carteles de cada paquete. Sacaremos los datos de la capa Ethernet, IP, TCP y HTTP. En este caso veremos el ejemplo de los datos de la capa IP:

```
//----------<<<<< IP Address >>>>>----------
let acces_ip = data[i]._source[key].ip

if (acces_ip) {
  ip_dst_data = Object.values(acces_ip)[14]//Destination Add
  //console.log("Ip Address Destination " + i + " => " + ip_

  ip_src_data = Object.values(acces_ip)[13]//Source Address
  //console.log("Ip Source Address " + i + " => " + ip_src_d
  entity3.setAttribute('packet','info_from_ip',ip_src_data);
  entity3.setAttribute('packet','info_to_ip',ip_dst_data);
```

Por último veremos cuál es el protocolo más alto que contiene cada paquete, y dependiendo de eso, el paquete lo pintaremos de un color u otro:

```
for (let i = 0; i < arrayProtocols.length; i++) {
  if (arrayProtocols[i] == "eth") {
    arrayFinal.push("eth")
    packet_color = "blue"
  }else if (arrayProtocols[i] == "ip") {
    arrayFinal.push("ip")
    packet_color = "yellow"
  }else if (arrayProtocols[i] == "tcp") {
    arrayFinal.push("tcp")
    packet_color = "red"
  }else if (arrayProtocols[i] == "http") {
    arrayFinal.push("http")
    packet_color = "green"
  }
}
```

Para acabar enviaremos todos estos valores obtenidos al componente `packet`, para que con ellos elabore todos los paquetes correspondientes:

```
entity3.setAttribute('packet','id',id_packet);
entity3.setAttribute('packet','start',time_relative_data);
entity3.setAttribute('packet','color',packet_color);
entity3.setAttribute('packet','duration','5000');
entity3.setAttribute('packet','LastValue',LastValue);
entity3.setAttribute('packet','class', 'all_packets')
entity3.setAttribute('packet','proto',proto);

entity3.setAttribute('packet','info_from_eth',eth_src_data);
entity3.setAttribute('packet','info_to_eth',eth_dst_data);
entity3.setAttribute('packet','port_tcp_dst',tcp_dst);
entity3.setAttribute('packet','port_tcp_src',tcp_src);

entity3.setAttribute('packet','info_http1',http_method);
entity3.setAttribute('packet','info_http2',http_url);
```

Y después lo añadiremos todo a la escena:

```
scene.appendChild(entity3);
```

Capítulo 5

Conclusiones

5.1. 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.

Hemos logrado superar el objetivo general que era construir un simulador de redes de comunicación basado en Realidad Virtual, el cual podrá visualizarse en cualquier dispositivo que tenga navegador web y en dispositivos de realidad virtual. Sin duda, este objetivo esta más que superado, ya que finalmente hemos podido construir este simulador, que es capaz de leer una captura de trazas de red de Wireshark, exportándolos en un JSON, y con los datos obtenidos crear una escena en realidad virtual en la que se dibujan todos los elementos que la componen.

Hemos construido un simulador de paquetes en realidad virtual, en el que utilizamos una arquitectura modular, esto nos permitirá utilizar cualquiera de los componentes creados para interaccionar con cualquier otro elemento o programa de A-Frame. Se trata del primero o uno de los primeros simuladores de redes en Realidad Virtual, ya que no se han hecho aun ninguno de este tipo, y esto da más valor a este proyecto, ya que no teníamos ninguna referencia con la que orientarnos.

Centrándonos un poco más en los objetivos específicos, se han logrado concluir con éxito todos los objetivos previstos. Primeramente queríamos que el programa se pudiera visualizar de manera sencilla en cualquier navegador y sin tener que instalar ningún programa externo, y además de ello, poder verlo en cualquier dispositivo móvil o dispositivos de realidad virtual, esto lo hemos conseguido gracias a que hemos ido almacenando todo el proyecto en un repositorio

de GitHub y con una simple dirección URL, puedes acceder a cualquiera de las demos que se encuentran alojadas en ese repositorio.

Otro gran objetivo era el de familiarizarnos y entender el framework de A-Frame. Esta librería es fundamental en nuestro proyecto, por lo que desde el principio nos hemos metido en profundidad con toda su documentación y ejemplos que hemos ido creando para ir entendiendo mejor esta tecnología y poder progresar correctamente en nuestro proyecto.

Hemos sido capaces de dar animación a nuestros paquetes, que era otro de los objetivos específicos que teníamos por hacer. Mediante un botón que se encuentra en la escena, somos capaces de comenzar la animación, congelar la animación en cualquier momento y reanudarla nuevamente cuando deseemos. Esta función nos facilitará la inspección de los paquetes, porque podremos congelar el paquete que queramos y pinchar sobre el paquete que nos interese, en el que se nos desplegaran las capas que lleva el paquete y poder ver la información que transporta en cada una de ellas. Pueden traer información de los protocolos que hemos incluido en este proyecto que son:Ethernet , IP , TCP y HTTP.

5.2. Aplicación de lo aprendido

A lo largo del grado he podido aprender conocimientos de muchas ramas diferentes. En este caso nos centraremos sobre todo en la rama de programación y conocimientos de protocolos de internet.

Empezamos el grado con la asignatura de Informática I, en la cual te introducen de una manera sencilla, en el mundo de la programación y te enseñan los conceptos básicos y fundamentales para ello.

Después vamos avanzando en otras asignaturas como por ejemplo Informática II, Gráficos y Visualización en 3D, Laboratorio de Tecnologías Audiovisuales en la Web, Construcción de Servicios y Aplicaciones Audiovisuales en Internet, en las cuales ya nos vamos centrando en diferentes lenguajes de programación y en programas y conceptos más complejos, con los que iremos ampliando nuestros conocimientos y mejorando nuestras habilidades como programadores. En estas asignaturas hemos aprendido a usar Python, JavaScript y HTML, estas dos últimas son en las que se basan nuestro simulador.

Por otro lado hemos visto también en asignaturas como Protocolos para la Transmisión de Audio y Video en Internet, Arquitectura de Internet y en Sistemas Telemáticos para Medios Audiovisuales, todo lo relacionado con las redes de comunicaciones, fundamentos de estas comunicaciones, protocolos de red, infraestructuras de red, ejemplos y utilización de estas redes... y de este conjunto de asignaturas hemos podido obtener información sobre todo de los protocolos y del comportamiento de los paquetes y las redes, para poderlo aplicar de manera correcta en nuestro simulador.

5.3. Lecciones aprendidas

A lo largo del Trabajo Fin de Grado he podido ampliar mis conocimientos en los siguientes campos:

1. Aprendizaje profundo del framework de A-Frame. Gracias a este framework he podido aprender a crear escenas en 3D, colocar elementos en la escena con diferentes comportamientos y formas, darles animación a elementos de la escena, manejo de luces y cámaras. También he aprendido conceptos de ThreeJS (una de las librerías en la que se basa A-Frame).

2. Crear y manejar componentes enteros dentro de mi programa y poder utilizarlos dentro de otros componentes.

3. Importación de modelos gltf en 3D para incorporarlos en la escena y componerlo con otros componentes de A-Frame para hacer que sean un elemento más en ella.

4. Programacion orientada al manejo de eventos y modificación de elementos a tiempo real mediante el uso de JavaScript.

5. He aprendido a manipular el DOM haciendo uso de las herramientas del DOM mediante JavaScript.

6. He tenido la oportunidad de realizar pruebas con las gafas de realidad virtual, en las que he podido aprender como utilizarlas y que se puede llegar a hacer y ver en ellas.

7. Uso de LaTeX para la edición de textos científicos y técnicos.

5.4. Esfuerzo realizado

El proyecto lo comencé a mediados de Octubre de 2020, por lo que la duración total del proyecto ha sido alrededor de 8 meses. Al comienzo del proyecto y hasta finales de año, tuve menos tiempo para dedicarle al tiempo, ya que tenía que compaginarlo con el horario laboral, y solo podía dedicarle tiempo los fines de semana y entre semana un par de horas por la tarde después de salir de trabajar.

Después en los meses de Enero, Febrero y Marzo, le dediqué prácticamente todos los días unas 4 horas por la mañana y 2 o 3 horas más por la tarde, ya que durante estos meses no tenía trabajo y pude dedicarle mucho más esfuerzo y tiempo al desarrollo del proyecto.

En Abril y Mayo ya volví a trabajar y al tener ya el proyecto más encaminado, me fue más fácil poder dedicarle tiempo y avanzar en él y le dedicaba algunas tardes entre semana y los fines de semana para progresar en el proyecto.

El tiempo aproximado dedicado para cada una de las etapas es el siguiente:

- Etapa 0: Comienzo a mediados de Octubre y finaliza al comienzo de Noviembre.
- Etapa 1: Comienzo a principios de Noviembre hasta mediados de Diciembre.
- Etapa 2: Desde mediados de Diciembre hasta finales de Enero.
- Etapa 3 y 4: Desde Febrero hasta Abril.
- Etapa 5 y 6: A lo largo del mes de Abril y comienzos de Mayo.
- Etapa 7: Durante Mayo.

5.5. Trabajos futuros

En este apartado podemos ver reflejadas las posibles mejoras o implementaciones que se podrían llegar a introducir en un futuro en este proyecto:

- Primeramente se podría mejorar el aspecto en general de la escena, tanto iluminación como el fondo, los paquetes o elementos que se ven en la escena.
- Se podrían incluir más protocolos de los introducidos por el momento en el proyecto, y con ello conseguir que se pueda obtener otro tipo de información en los paquetes.
- Podría crearse una barra de tiempo(estilo YouTube) para ver en que momento de la animación te encuentras, saber cuanto te queda para finalizar la animación, o poder deslizarte sobre ella y avanzar o retroceder la animación.
- Se podría crear un archivo JSON de una red por ejemplo local, con los elementos que la componen y sus direcciones y asignarle un modelo 3D de un elemento diferente(switch, ordenador, móvil...), y con esto podríamos obtener una escena más exacta con los elementos que la componen. Es decir, si el programa detecta una dirección que corresponde a un switch, te dibuje un switch como nodo en vez de un ordenador, o cualquier elemento que queramos incorporarle.
- También es posible coger cualquier funcionalidad de un analizador de paquetes, en este caso de Wireshark, e incorporarlas al simulador para que sea capaz de interpretarlas.

Bibliografía

[1] R. Dubois. A-Frame Gui, May 2021.
https://github.com/rdub80/aframe-gui.

[2] J. Gauchat. *El gran libro de HTML5, CSS3 y JavaScript*. Digitalia Hispánica. Marcombo, S.A., 2012.

[3] J. M. Gonzalez-Barahona. A-Frame-Playground, May 2021.
https://jgbarah.github.io/aframe-playground/interaction-01/.

[4] J. Josa. *Diseño de Juegos 3D para Web - Libro 01: THREE. JS - HTML5 y WEBGL (Versión en Blanco y Negro)*. Independently Published, 2017.

[5] K. Kishan. Writing a simple timer component for A-Frame. May 2021.
https://medium.com/@kewalkishan/writing-a-simple-timer-component-for-a-frame-81889c863589.

[6] Mozilla. A-Frame Documentation, May 2021.
https://aframe.io/.

[7] Mozilla. Timeouts and Intervals, May 2021.
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Timeouts_and_intervals.

[8] K. Ngo. Event Set Componente, May 2021.
https://www.npmjs.com/package/aframe-event-set-component.

[9] C. Yee. A-Frame Widgets, May 2021.
https://github.com/caseyyee/aframe-ui-widgets.