Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Conversión de vídeo 360 a proyección plana mediante videotextura

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

Autor: Ferreres Garcia, Francisco
Tutor: Vivó Hernando, Roberto Agustín

Curso 2016/2017

Resumen

El presente proyecto se enmarca en el desarrollo de aplicaciones web para vigilancia de entornos públicos y comerciales mediante el uso de videocámaras 360. Este tipo de dispositivos permiten obtener una fuente de vídeo con un ángulo sólido total que puede ser usada como entrada (streaming) a una aplicación HTML5 mediante la aplicación de nuevas tecnologías WebGL sobre videotextura. La fuente sufre de distorsión que debe ser corregida sobre la superficie de visualización elegida en tiempo real, adaptándose el espacio de la textura al de proyección para minimizar este defecto.

En este documento se describe el problema en cuestión y sus antecedentes, para luego analizar cada fase del problema y las posibles soluciones. A continuación se detallan la arquitectura de la aplicación, los componentes que la conforman y su interacción, explicando en cada caso las tecnologías utilizadas y algunos conceptos específicos. Una vez mostrado todo el proceso de diseño y desarrollo de la aplicación se procede a ejecutar las pruebas especificadas durante el diseño de ésta y sus resultados, y una guía para el usuario que desee probar la aplicación.

Palabras clave: three.js, javascript, vídeo 360, mapeo UV.

Abstract

This project is part of the development of web applications for surveillance of public and commercial environments through the use of 360 video cameras. This type of devices allow to obtain a video source with a total solid angle that can be used as an input (streaming) to an HTML5 application through the application of new WebGL technologies on video textures. The source suffers from distortion that must be corrected on the display surface chosen in real time, adapting the space of the texture to the projection space to minimize this defect.

This document describes the problem and its background, and then analyzes each phase of the problem and its possible solutions. Then, the architecture of the application, its components and its interaction are detailed, explaining in each case the technologies used and some specific concepts. Once the whole process of design and development of the application is shown, the tests specified during the design of the application are executed, and finally there's a guide for the user who wishes to test the application.

Keywords : three.js, javascript, 360 video, UV mapping.

Tabla de contenidos

1. INTRODUCCIÓN

1.1. Motivación

La principal motivación a la hora de realizar este proyecto ha sido la de aprender, aprender cosas nuevas y aprender sobre nuevas tecnologías, y como utilizarlas y explotarlas. Se deben utilizar diversas tecnologías, algunas muy actuales (como las cámaras 360), cuya explotación puede traer muchos beneficios y todavía está en desarrollo. Además, se han utilizado otras tecnologías bastante novedosas como WebGL, jQueryUI, Three.js o Bootstrap, las cuales están relacionadas con HTML5, y pueden ser el futuro de las aplicaciones multiplataforma.

También ha sido una gran motivación el utilizar lo ya aprendido durante el máster en un proyecto real y poder ver hasta dónde era capaz de llegar. El explotar los conocimientos obtenidos y crear algo con ellos, es algo que trae una gran satisfacción personal. Por último, sirve para reforzar los conocimientos adquiridos y profundizar en ellos.

1.2. Objetivos

A la hora de realizar este proyecto, se deben establecer unos objetivos generales que indiquen el objetivo principal de éste, y unos objetivos específicos que indiquen con mayor exactitud cada uno de los distintos detalles del proyecto.

1.2.1. Objetivos generales

El objetivo principal que se pretende cumplir con la realización de este proyecto, es disponer de una aplicación web capaz de visualizar correctamente y de distintos modos la salida de una videocámara de 360 grados.

La aplicación desarrollada, personalizable y de fácil configuración, puede ser utilizada y embebida en cualquier página web, pública o privada, por lo que los posibles usos son ilimitados. Una de las principales aplicaciones sería en la seguridad, ya que mediante su uso se puede crear una consola web de vigilancia. En dicha consola se pueden visualizar diferentes ángulos diferentes ángulos con una sola cámara 360 como si se tratase de diferentes cámaras, con el consiguiente ahorro en gastos e infraestructuras.

1.2.2. Objetivos específicos

- ✓ Obtener correctamente la salida de la cámara 360 en el navegador web.
  - A través de la MediaStream API se debe obtener la salida de la cámara y hacerla disponible al navegador.
- ✓ Proyectar la salida de la cámara en una malla mediante una videotextura.
  - Una vez disponible la salida de la cámara, convertirla en una videotextura y proyectarla en una malla.
- ✓ Conseguir una visualización 360 interactiva de la salida equirectangular de la cámara.
  - Proyectar la salida cuadrangular de la cámara en una esfera y mostrarla desde el interior para obtener la ilusión de estar rodeado por toda la salida de la cámara. No es necesario corregir la distorsión.
- ✓ Obtener una visualización 360 interactiva de la salida cruda de la cámara.
  - Proyectar la salida cruda de la cámara (ojo de pez) en una esfera y mostrarla desde el interior para obtener la ilusión de estar rodeado por toda la salida de la cámara.
  - Se debe corregir la distorsión, ya que en este caso la salida consiste en 2 imágenes de ojo de pez con una distorsión bastante alta.
- ✓ Posibilitar seleccionar el modo de salida de la cámara.
  - Mediante algún tipo de control, posibilitar la selección del modo de salida de la cámara, ya sea equirrectangular o la salida cruda (ojo de pez).
- ✓ Proporcionar la posibilidad de selección del modo de mapeo de cada pantalla de la consola.
  - Posibilitar la selección del modo de mapeo de la salida en cada una de las pantallas.
- ✓ Permitir mostrar varias pantallas independientes en la aplicación web.
  - La aplicación debe posibilitar mostrar varias pantallas independientes, en las que se pueda visualizar la salida de distintos modos y en distintos ángulos.
- ✓ Posibilitar la personalización de la aplicación, creando varias pantallas pre configuradas, por ejemplo para crear una consola de seguridad.

- Se deberían permitir estas personalizaciones y mostrar una aplicación de ejemplo que saque partido de estas características, creando una posible consola de seguridad para un negocio.

1.3. Estructura de la memoria

Esta memoria está dividida en 7 capítulos principales, incluyendo este primer capítulo de introducción. A continuación se resumirá brevemente cada uno de los capítulos:

1. **Introducción**: Este es el capítulo actual, se expone la motivación a la hora de trabajar en este proyecto, descripción de los objetivos preliminares y la sección actual en que se describe cada uno de los capítulos de la memoria.
2. **Antecedentes**: En este apartado, se describe el problema en sí y la situación actual del contexto en general, incluyendo soluciones actuales al problema y las posibles mejoras que se podrían hacer al respecto.
3. **Diseño de la solución**: En este capítulo, se analizan independientemente los problemas que plantea el proyecto y se establecen unos requisitos a cumplir por la aplicación, para luego detallar la solución a estos problemas y el diseño general de la aplicación.
4. **Implementación**: En esta sección se muestran detalles sobre la implementación: comenzando con la descripción de algunas tecnologías y conceptos utilizados para que el lector pueda comprender mejor el resto del capítulo; a continuación se habla de los distintos componentes de la aplicación y de su interacción; y por ultimo se tratan los detalles sobre la interfaz de usuario y su implementación.
5. **Pruebas y resultados**: En el siguiente capítulo, se muestran las diversas pruebas realizadas para asegurar el correcto funcionamiento de la aplicación en diferentes sistemas y los resultados de dichas pruebas.
6. **Conclusiones**: En este apartado, se recuperan los objetivos preliminares para ahora asumir cuales de estos objetivos se han cumplido total o parcialmente y cuales han quedado por resolver. También se habla de las posibles líneas abiertas en el proyecto, ya sean nuevas mejoras u objetivos o nuevas vías de desarrollo respecto al problema aquí tratado.
7. **Bibliografía**: Esta última sección, se trata de las referencias a la bibliografía utilizada en esta memoria.

Finalmente se incluyen dos apéndices: una guía de usuario, la cual se compone del manual de instalación de la cámara, y de un breve ejemplo de uso de la aplicación.

Ya que la memoria es bastante técnica, a un usuario estándar se le recomienda leer los primeros dos capítulos para comprender mejor el porqué de este proyecto y las conclusiones, ya que el resto de capítulos pueden ser demasiado específicos y contener detalles de poco interés para el usuario final de la aplicación.

2. ANTECEDENTES

2.1. Descripción del problema

Antes de hablar del problema se debe explicar la tecnología, ya que a raíz de esta se origina el problema: los videos inmersivos en 360 grados. Estos videos capturan en todo momento lo que sucede alrededor de la cámara, 360x180 grados a su alrededor. El espectador puede tener una experiencia inmersiva y visualizar el momento grabado desde la perspectiva de la persona u objeto que lleva la cámara dirigiendo su visión hacia cualquier ángulo.

Este tipo de videos se pueden obtener con distintos tipos de cámaras específicas que se verán más adelante. En este proyecto se va a trabajar con la cámara Ricoh Theta, que captura su entorno gracias a sus dos lentes de ojo de pez de 180 grados cada una [1].

La forma más común de visualizar este tipo de videos es mostrando un ángulo determinado de la escena, en el que el espectador puede dirigir la visión hacia cualquier punto y creando así la experiencia inmersiva.

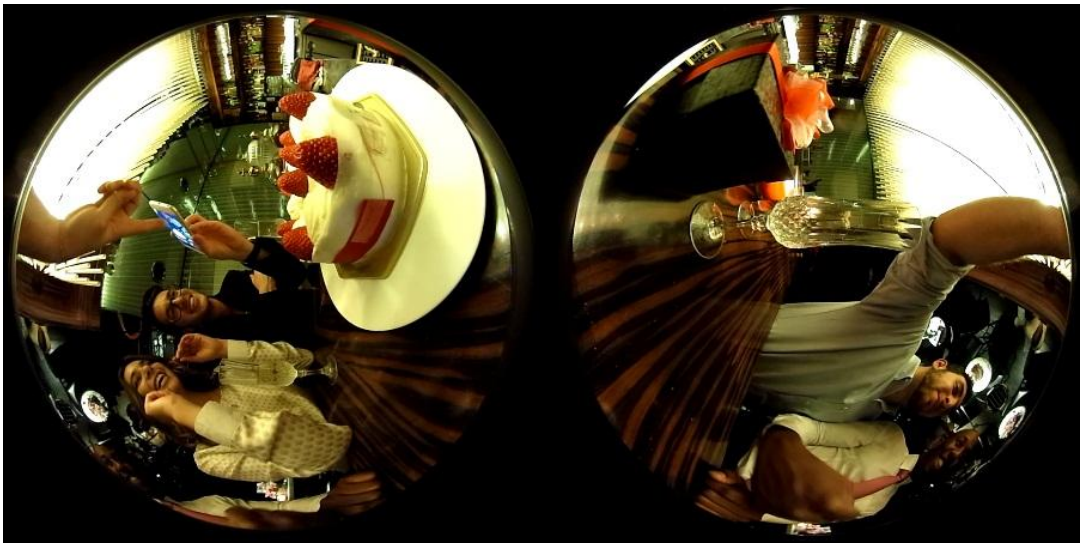

Figura 1: Imagen de 360 grados capturada mediante dos imágenes de ojo de pez de 180 grados.

Dependiendo del dispositivo desde el cual se visualiza el video inmersivo, la forma de dirigir la visión hacia el ángulo deseado puede ser distinta, por ejemplo en un PC podemos manejar el punto de vista con el ratón o con algún sistema de realidad virtual como Oculus Rift, mientras que con un Smartphone normalmente se puede utilizar el giroscopio del dispositivo.

Cabe señalar que estos videos también se pueden visualizar en formato plano, lo que se conoce como una proyección cilíndrica equidistante o equirrectangular. Este formato trata simplemente de mostrar la imagen inmersiva en un plano, se debe corregir su distorsión en gran medida ya que no se puede mostrar su totalidad sin ningún tipo de distorsión.

Figura 2: Proyección equirectangular de una imagen de 360 grados.

El problema que se debe de resolver en este proyecto es relativamente nuevo, la idea de capturar imágenes panorámicas se remonta a los propios inicios de la captura de imágenes, pero hasta 2015 la única forma de capturar videos inmersivos en 360 grados era mediante costosos y complejos sistemas formados por varias cámaras que permitían capturar el entorno [2].

Figura 3: Sistema Freedom 360.

En 2016 surgió el boom de los videos inmersivos en 360, con la aparición de varias cámaras de distintos fabricantes y con precios muy variados (desde 200 dólares hasta 40.000). Estas cámaras permiten capturar este tipo de videos de una forma mucho más cómoda y sin necesidad de procesar imágenes obtenidas por distintas cámaras individualmente, el cual es un proceso muy complejo y laborioso [3].

Figura 4: Cámaras Ricoh Theta y Samsung Gear 360.

Como se puede observar en la Figura 1, no es práctico ni elegante y mucho menos inmersivo visualizar este tipo de videos en su salida estándar, ya que tiene un grado de distorsión muy alto, por lo que se debe presentar de un modo práctico para según qué objetivo. Por ejemplo, es posible que una empresa quiera ofrecer en su intranet la asistencia virtual a ciertas reuniones, en la que dispongan de una cámara 360 en el centro de la mesa o tal vez, el departamento de seguridad quiere disponer de una consola de vigilancia que permita visualizar diferentes ángulos de una habitación a través de una única cámara de este tipo, ahorrando el coste de instalar varias cámaras tradicionales.

Resolver este problema no es algo trivial, ya que se debe trabajar con imágenes distorsionadas y representarlas de varios modos según los intereses y necesidades del usuario. Se debe encontrar la forma de proyectar un área de estas imágenes en la pantalla y ofrecer interactividad respecto al área visualizada.

Pero antes de poder trabajar en esto, se debe obtener la salida de la cámara en el navegador, ya que el proyecto consiste en el desarrollo de una aplicación utilizando HTML5. Se debe poder acceder a las cámaras del sistema y seleccionar el tipo de salida, lo cual es un problema tecnológico, ya que HTML5 permite el acceso a dispositivos multimedia desde no hace mucho y por supuesto también es un problema de seguridad y permisos.

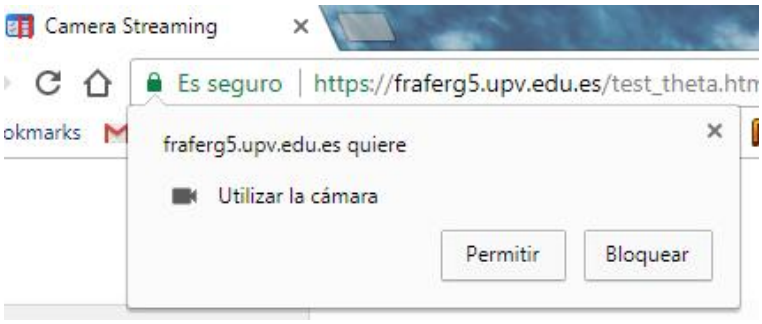

Figura 5: Navegador solicitando permisos para acceder a la cámara.

Para el desarrollo del proyecto se va a trabajar con la cámara Ricoh Theta 360, esta cámara es la que se dispone, y el uso de ésta conlleva trabajar con sus drivers propietarios, pero la salida de imagen se obtiene del mismo modo en cualquier cámara mediante el navegador. El formato de salida de la imagen y su resolución y medidas son otros aspectos a tener en cuenta a la hora de trabajar con una u otra cámara.

2.2. Estado del arte

Hoy en día existen varios modos de visualizar este tipo de videos, ya que el interés en este tipo de contenido está creciendo rápidamente. Dos de los principales distribuidores de contenido del mundo ofrecen esta posibilidad desde hace algún tiempo, Facebook y YouTube. Facebook por ejemplo, permite subir vídeos 360 en formato cilíndrico equidistante, para luego poder visualizarlos inmersivamente desde el ordenador o desde el dispositivo móvil, ofreciendo este último la posibilidad de utilizar el giroscopio del dispositivo como mecanismo de selección del área observada [4].

También existen reproductores de escritorio para este tipo de contenidos, pudiendo destacar el famoso reproductor VLC Player, que introdujo esta característica en noviembre de 2016, o JW Player, que permite visualizar este tipo de videos en páginas web, embebiendo su reproductor del mismo modo que hace YouTube.

Finalmente, algunos de los fabricantes de estas cámaras han desarrollado sus soluciones al problema de procesar la salida cruda de la cámara, y en el caso de Ricoh, añadieron la posibilidad de obtener la salida equirectangular de la cámara, para no tener que trabajar directamente con la salida de ojo de pez, a través de su 'UVC Blender'. Este software se encarga de transmitir la salida de la cámara en vivo al ordenador, y ofrece la posibilidad de obtener esta salida en formato cilíndrico equidistante, el cual es el estándar para su visualización en otras plataformas, como Facebook o YouTube.

Figura 6: Visualización de video 360 en Facebook.

2.3. Conclusiones

Las soluciones existentes a este problema son bastante propietarias, aunque algunas permiten embeber su reproductor, siempre es bajo su marca, y ninguna de las soluciones encontradas es de código abierto. Esto también conlleva que ninguna de las soluciones se puede personalizar, por lo que no se puede crear un producto personalizado para un determinado cliente o usuario, por ejemplo, una consola de seguridad con distintas pantallas.

Teniendo en cuenta estos hechos, se pretende realizar una aplicación que ofrezca una solución a este problema, de código libre y personalizable para todo tipo de aplicaciones, desde entretenimiento, pasando por seguridad y eficiencia de recursos.

Como se puede observar en la figura 6, se ha realizado un Mockup de la posible aplicación, en la que se pueden mostrar tantas pantallas como se desee y personalizar cada una de ellas, con el modo de mapeo deseado y su correspondiente área de visualización, pudiendo elegir también la salida de la cámara deseada (Salida estándar o cilíndrica equidistante).

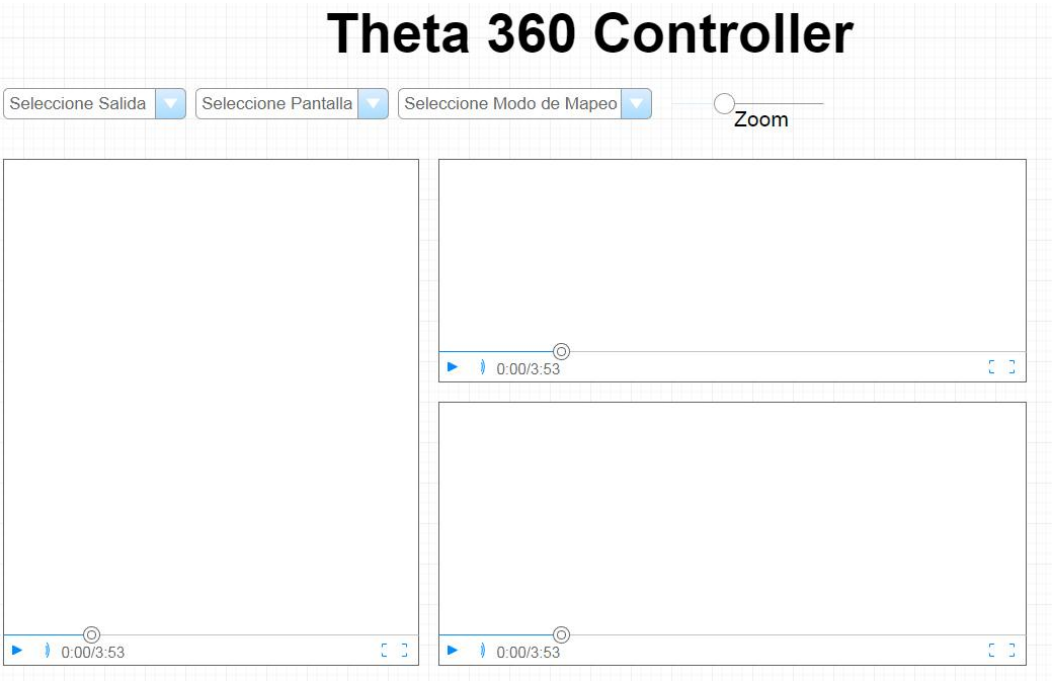

Figura 7: Mockup de la posible aplicación.

3. DISEÑO DE LA SOLUCIÓN

3.1. Análisis

Una vez establecidos los objetivos y exploradas las soluciones actuales, se procede a analizar el problema. En el desarrollo de este proyecto existen varios problemas, que se deben estudiar y resolver individualmente antes de poder llegar a una solución completa.

El primer problema trata de obtener los dispositivos multimedia del equipo utilizado (entre los que se encontraran los dos modos de visualización de la videocámara 360) , desde el navegador. Para lo cual se debe buscar información sobre algún tipo de API o librería que permita el reconocimiento y acceso sencillo a estos dispositivos, para así poder obtener su salida. Es necesario obtener un listado con los dispositivos que ofrecen salida de imagen del sistema, para así poder seleccionar la videocámara 360, o el UVC Blender, el cual muestra la salida equirectangular de la cámara, y se instala en un driver aparte, por lo que el sistema lo reconoce como una cámara distinta.

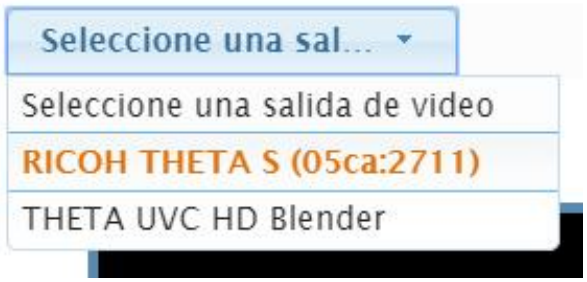

Figura 8: Modos de visualización de la videocámara, obtenidos desde el navegador.

Una vez obtenida la salida de la videocámara, aparece el problema de mostrar esa información al usuario, para lo cual se debe mapear la salida en un objeto tridimensional, a través de una videotextura. Se quieren ofrecer 2 modos de mapeo para cada salida de la cámara, un mapeo en un plano, que muestre la totalidad de la salida, y un mapeo en una esfera. Mapeando correctamente la salida de la cámara en una esfera, se puede ofrecer la sensación de inmersión en el contenido de la imagen, visualizando la escena desde dentro de la esfera. El problema de mapear la salida equirectangular de la cámara en la esfera, no debería ser muy complicado, ya que por ejemplo, es bien conocido el clásico mapamundi, el cual normalmente es una proyección equirectangular

de la tierra, que es una especie de esfera, por lo que mapeando esta imagen equirectangular en una esfera, la distorsión debería desaparecer.

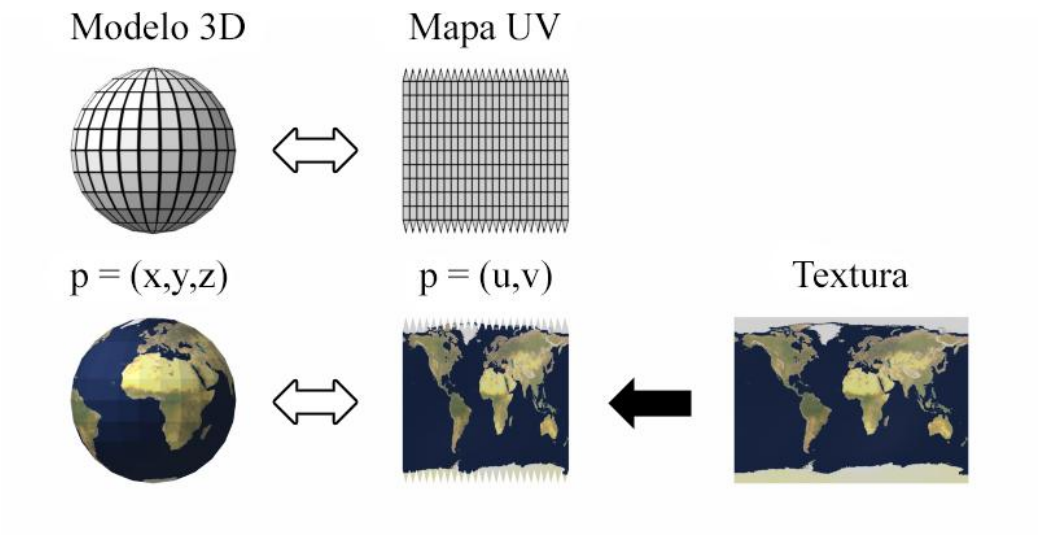

Figura 9: Ejemplo de mapeo UV equirectangular, con la imagen de la tierra.

El problema de mapear en la esfera la salida cruda de la cámara es mucho más complicado, ya que se compone de una textura con mucho espacio inservible en negro y la propia captura de la cámara, la cual se compone de dos imágenes en ojo de pez con un grado de distorsión muy alto, como se puede apreciar en la figura 9.

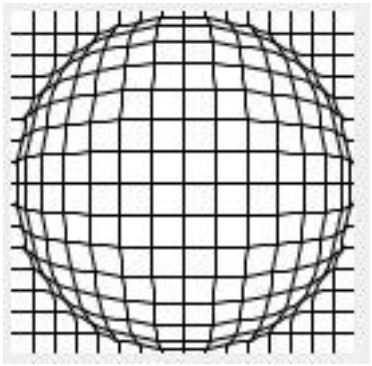

Figura 10: Representación de la distorsión que produce una lente de ojo de pez.

Para representar correctamente esta salida en la esfera, se debe realizar un mapeo UV, seleccionando correctamente las coordenadas de los dos círculos en los que se muestras las imágenes en ojo de pez, y aplicarle un algoritmo de corrección adecuado.

Otro problema principal a solucionar es el control de los atributos de la aplicación. El usuario debería poder personalizar los atributos de manera sencilla y amigable, a través de una interfaz estética, uniforme y consistente. El código debe ser limpio, modular y robusto.

Con estos problemas resueltos, se podrá comenzar a desarrollar la aplicación de ejemplo, que se tratará de una consola de seguridad para un posible negocio que quiera mejorar la eficiencia de su sistema de seguridad.

Una vez analizados los principales problemas, se pueden establecer los requisitos a cumplir por la aplicación:

- Debe funcionar correctamente en los principales navegadores: Mozilla Firefox, Google Chrome y Opera.
- Debe permitir seleccionar la salida cruda de la cámara y la salida equirectangular proporcionada por el driver UVC Blender.
- Debe permitir mostrar cada salida por completo en un plano, o en una proyección envolvente, reduciendo al máximo la distorsión, y ofreciendo la sensación de estar rodeado por el entorno de la cámara.
- El código debe ser modular y limpio, y debe permitir la fácil personalización del número de pantallas y su configuración.
- Debe disponer de unos controles que permitan la configuración de cada una de las pantallas.
- Debe permitir modificar el ángulo hacia el cual se dirige cada una de las pantallas, así como ofrecer la posibilidad de tener cada pantalla un ángulo de visualización predefinido.

3.2. Diseño

En este apartado se describe el diseño de las soluciones a los problemas independientes previamente identificados. También se describen los principales elementos de la aplicación, con sus componentes y funcionalidades, y alguno de sus algoritmos. Finalmente se describen brevemente las pruebas a realizar una vez terminado el desarrollo de la aplicación.

3.2.1. Obtención de dispositivos multimedia

Para obtener los dispositivos multimedia del sistema, de los cuales se quieren obtener las webcams, se ha recurrido a las APIs de WebRTC, que nos ofrecen funciones como *getUserMedia* y *enumerateDevices*. Al iniciar la aplicación se debe llamar a *enumerateDevices* para obtener las cámaras y alimentar el ComboBox de selección de salida, y al seleccionar una de las salidas se debe recurrir a *getUserMedia* para obtener un objeto de tipo MediaStream, que contiene la salida de

la videocámara. Para acceder a la cámara se necesita permiso del usuario, por lo que la función no puede devolver inmediatamente el MediaStream, así que lo que se obtiene es una promesa, que contendrá la salida de la cámara si el usuario otorga los permisos [5].

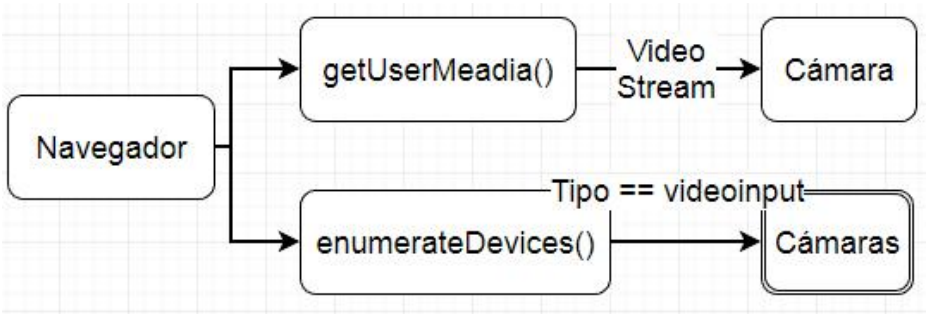

Figura 11: Esquema de las funciones de WebRTC utilizadas.

.

3.2.2. Mapeado de la videotextura

Una vez obtenido el Stream de la cámara, se obtiene una URL con la cual poder acceder a este a través del código JavaScript, lo cual se consigue con la función *createObjectURL()*. Esta URL se envía a los controladores de cada una de las pantallas para cambiar la fuente de vídeo que alimenta la videotextura. La videotextura es simplemente un objeto interno a la aplicación que contiene la imagen a superponer en el objeto deseado, este proceso se denomina 'mapeado'. Para mapear un objeto es necesario disponer de una textura, que es el 'dibujo' a superponer, y un material, el cual contiene la textura y las propiedades visuales que tendrá el objeto mapeado, por ejemplo reluctancia.

Esta videotextura se debe mapear en los correspondientes objetos que harán el papel de pantalla de proyección, los cuales serán un plano y dos esferas (una para cada tipo de mapeado). Antes de mapear la textura se crea un material, el cual contendrá la videotextura y es el que realmente se va a aplicar a los objetos. Para mapear este material en un objeto, solo se le debe indicar al constructor del objeto, por lo que es un proceso relativamente sencillo en el caso del mapeado en un plano, el cual será observado de frente sin necesidad de mover la cámara de la escena.

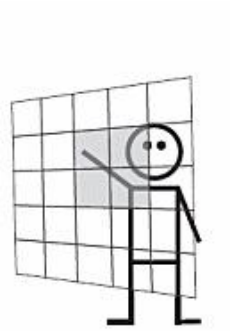

Figura 12: Representación del mapeado en un plano y el punto de vista del usuario.

En el caso de la proyección envolvente de la salida equirectangular se debe mapear del mismo modo el material en una esfera, y se debe colocar la cámara de la escena en el centro de la esfera, e invertir la escala de la textura, ya que se observará desde el interior.

Figura 13: Representación del mapeado en una esfera y el punto de vista del usuario.

Para mapear correctamente la salida cruda de la cámara (ojo de pez) en la esfera no basta simplemente con esto, y se debe trabajar con el mapeado UV, el cual asigna coordenadas de la textura 2D (UV) a coordenadas del objeto 3D (XYZ). A través de esta asignación se pueden seleccionar las coordenadas de la textura (ya que la imagen de ojo de pez contiene mucho espacio inservible en negro) y aplicarlas a los puntos correspondientes en el modelo 3D para de este modo corregir la distorsión.

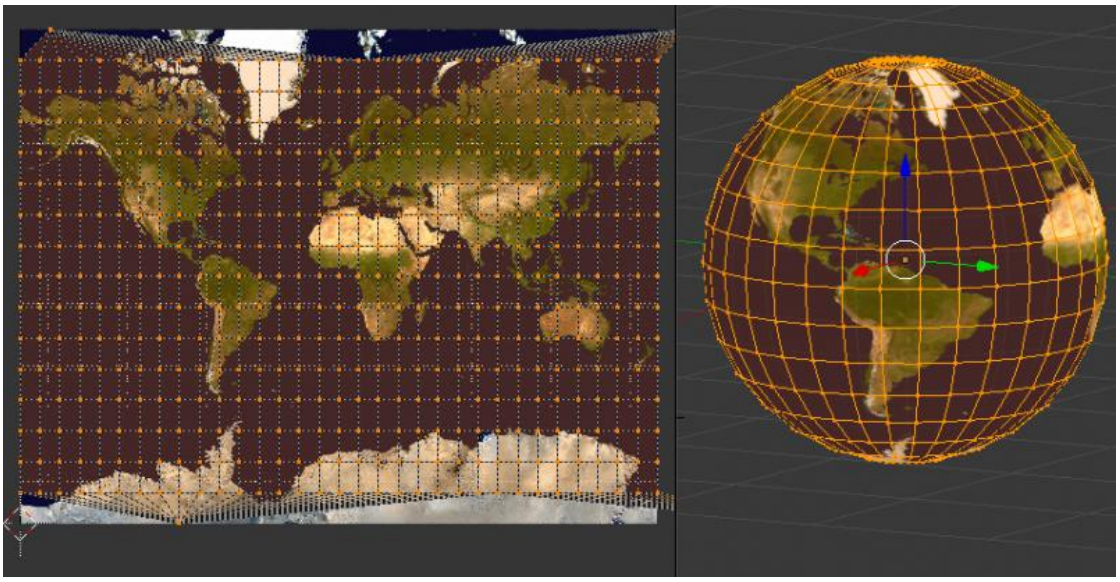

Figura 14: Ejemplo de mapeado UV de una textura equirectangular.

Para la proyección equirectangular no es necesario hacer ningún ajuste a este mapeado, ya que representa el propio despliegue de la textura de una esfera, pero en el caso de la proyección de ojo de pez es necesario indicar las coordenadas del circulo que contiene la textura y su correcto mapeo en cada una de las semiesferas.

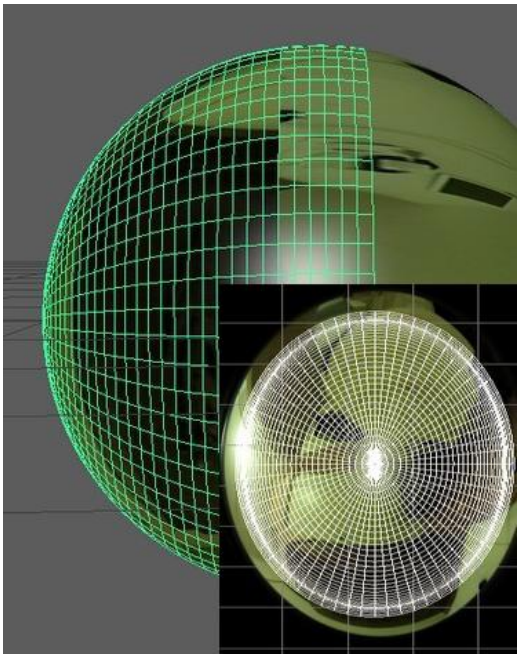

Figura 15: Ejemplo de mapeado UV en una esfera a partir de una textura en ojo de pez.

3.2.3. Corrección de la distorsión

Para corregir la distorsión de esta textura circular se debe usar una función trigonométrica que calcula las coordenadas correspondientes de la textura para cada vértice de la esfera. Para realizar los cálculos se deben obtener ciertas medidas de la imagen cruda de la cámara [6]:

- Coordenadas del centro de cada circulo.
- Altura y anchura de la textura completa.
- Radio del circulo que contiene la imagen ojo de pez.
- Distancia desde el centro de los círculos hasta el punto más bajo de la textura.

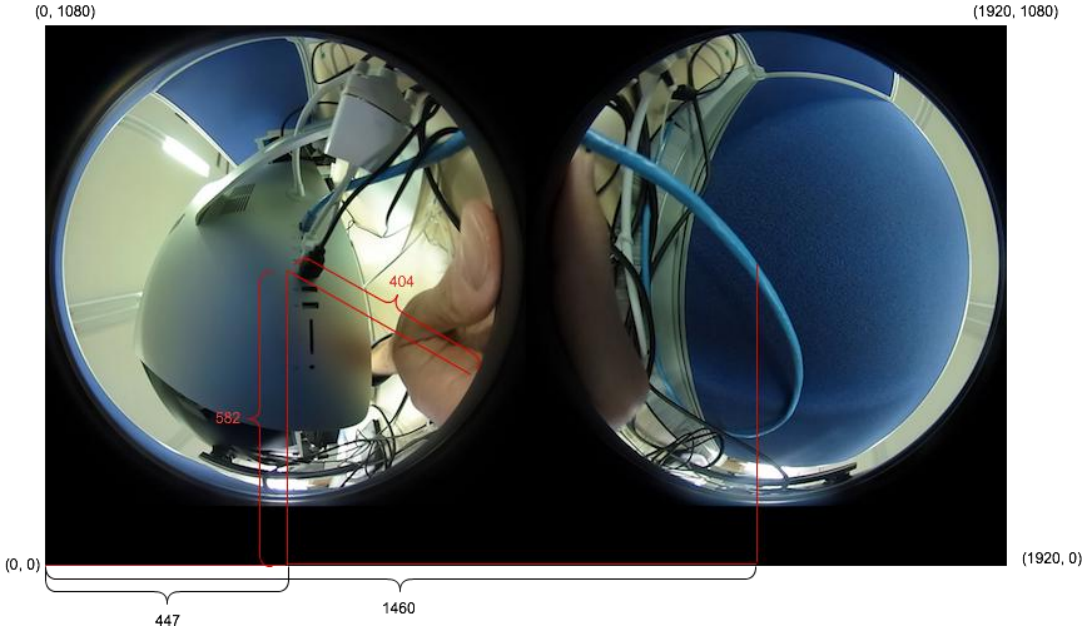

Figura 16: Ejemplo de medidas necesarias para la corrección de la distorsión en una textura de ojo de pez.

Las medidas deben ser tomadas sobre la imagen cruda obtenida de la cámara, y ser realizadas con la máxima precisión posible, ya que de ello dependerá la efectividad de la formula de corrección de la distorsión, y una vez obtenidas éstas, se puede aplicar la formula a la geometría de la esfera. En Threee.js se deben obtener los mapas UV de las caras de la esfera (triángulos de los que se compone su geometría) y aplicar la formula correspondiente a cada uno de sus tres vértices.

A continuación se muestra el pseudocódigo que representa la porción de código encargada de eliminar la distorsión.

```
para(cada cara de la esfera){
        var UVs = (coger UVs de los 3 vértices);
        var cara = (guardar aquí la cara(triangulo de la geometría de la esfera));
                para(cada uno de los 3 vértices de la cara){
                        aplicar_formula_correccion(UVs,cara);
                }
}
```

Para aplicar la formula de corrección se cogen las coordenadas (x,y,z) del vector normal de la cara y se aplica la siguiente fórmula:
$(\text{acos}(y) / \sqrt{x^2 + z^2}) * (2/\pi)$ [6].

```
var faceVertexUvs = sphereGeom1.faceVertexUvs[ 0 ];// cogemos los UVs de cada cara de la esfera
for ( i = 0; i < faceVertexUvs.length; i++ ) {// para cada cara de la esfera(que contiene 3 vertices):
        var uvs = faceVertexUvs[ i ]; //cogemos los UVs de sus tres vertices(cada UV es un vector con 2 elementos[X e Y] con las coordena
        var face = sphereGeom1.faces[ i ]; //cogemos la cara en si (clase Face3 en Three.js) -> https://threejs.org/docs/#api/core/Face3
        for ( var j = 0; j < 3; j ++ ) {//para cada vertice de la cara:
                var x = face.vertexNormals[ j ].x;
                var y = face.vertexNormals[ j ].y;//cogemos las coordenadas de su normal(que indica hacia donde mira la cara)
                var z = face.vertexNormals[ j ].z;
                //aplicamos la formula de correccion
                if (i < faceVertexUvs.length / 2) {//---Si pertenece a la primera semiesfera
                        var correction = (x == 0 && z == 0) ? 1 : (Math.acos(y) / Math.sqrt(x * x + z * z)) * (2 / Math.PI);
                        uvs[ j ].x = x * (radioCorreccion / anchuraRaw) * correction + (centroEsfera1 / anchuraRaw);
                        uvs[ j ].y = z * (radioCorreccion / alturaRaw) * correction + (centroAlturaEsferas / alturaRaw);
                } else {                        //---Si pertenece a la segunda semiesfera
                        var correction = ( x == 0 && z == 0) ? 1 : (Math.acos(-y) / Math.sqrt(x * x + z * z)) * (2 / Math.PI);
                        uvs[ j ].x = -1 * x * (radioCorreccion / anchuraRaw) * correction + (centroEsfera2 / anchuraRaw);
                        uvs[ j ].y = z * (radioCorreccion / alturaRaw) * correction + (centroAlturaEsferas / alturaRaw);
                }
        }
```

Figura 17: Fragmento del código que corrige la distorsión de la textura.

3.2.4. Diseño de la interfaz grafica

La interfaz gráfica debe ofrecer las funcionalidades especificadas en los requerimientos, y ser lo más atractiva y sencilla posible, por lo que se recurrirá a las conocidas librerías de jQuery y jQueryUI. Estas librerías, además de ofrecer un aspecto visual mucho más atractivo que los clásicos controles del navegador, hace mucho más sencillo el trabajo con estos controles y elementos de la página web, y con sus funciones de callback.

En definitiva, la interfaz grafica debe permitir:

- Seleccionar la salida de video deseada
- Seleccionar el modo de mapeo deseado en la pantalla
- Modificar el zoom de la pantalla seleccionada
- Y obviamente poder cambiar la selección de la pantalla a configurar

El modo más sencillo de realizar esto es mediante controles ComboBox, y un Slider para modificar el nivel de zoom sobre un rango predefinido.

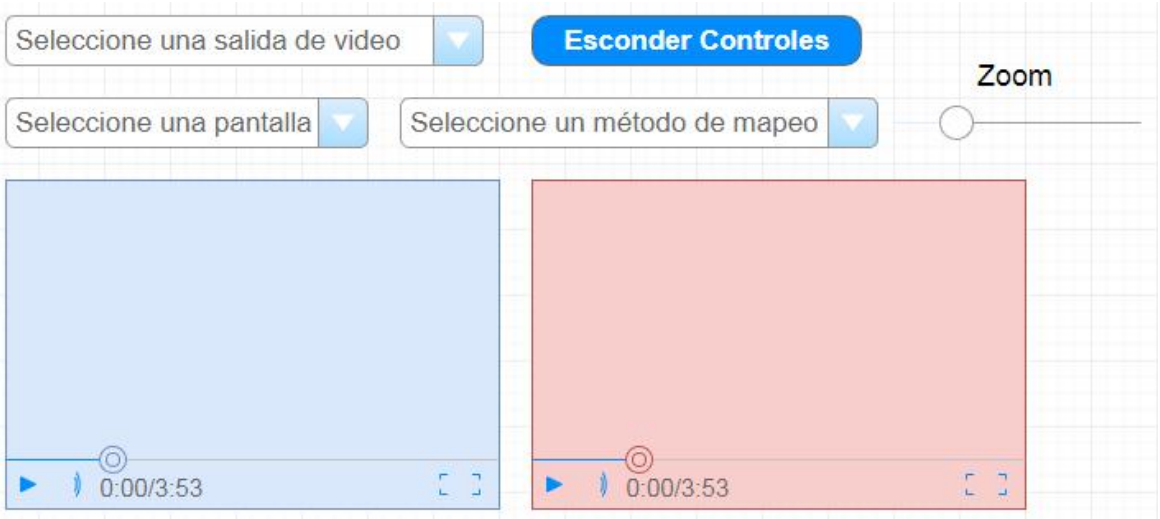

Figura 18: Mockup final de los controles de la aplicación.

Mediante jQueryUI la implementación de estos controles es algo relativamente sencillo, y gracias a los selectores de jQuery el código es mucho más sencillo de mantener, además de disponer de mucha documentación [7].

3.2.5. Diseño de pruebas

La aplicación a desarrollar no tiene complejos algoritmos los cuales requieran un análisis de coste temporal, pero al ser una aplicación multiplataforma debe ser probada en los principales navegadores y analizar su correspondiente rendimiento, tanto las sensaciones que ofrece, dependientes de la tasa de refresco del canvas WebGL, como su consumo de recursos en el sistema (memoria). Para lo cual se utilizarán las herramientas provistas por cada navegador para analizar el uso de memoria de la aplicación, siempre que ésta funcione correctamente.

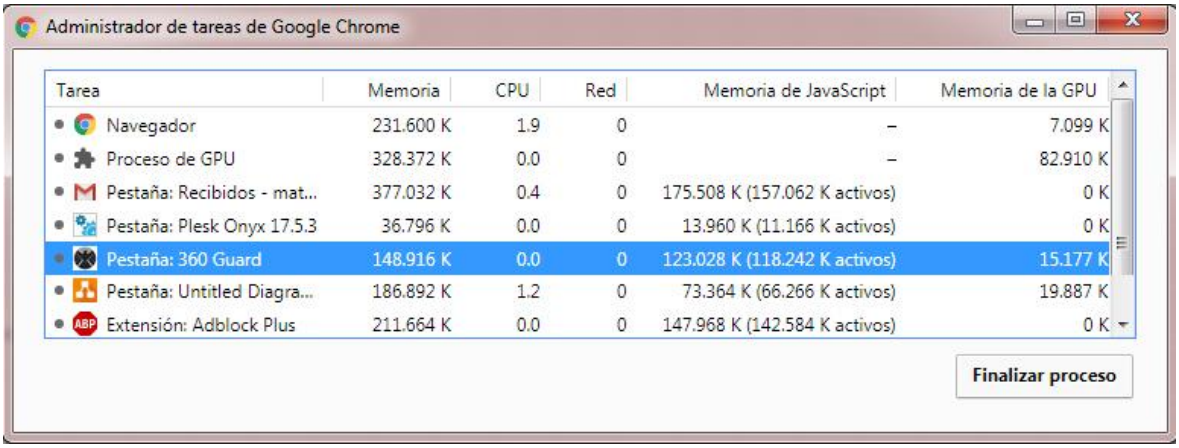

Figura 19: Administrador de tareas de Chrome, permite analizar el rendimiento de las pestañas en ejecución.

3.3. Conclusión

El desarrollo de este proyecto conlleva la resolución de varios problemas, y para ello se ha diseñado una solución para cada uno de ellos, en resumen, se deben cumplir los requisitos descritos brevemente a continuación:

- Permitir seleccionar los dos diferentes modos de visualización de la videocámara.
    - Este problema se ha resuelto gracias al uso de la API WebRTC implementada en la mayoría de navegadores modernos, y en especial gracias a las funciones *getUserMedia* y *enumerateDevices.*
- Mostrar la salida de la cámara en un plano o en un modo de visualización envolvente en el que se pueda seleccionar fácilmente la dirección de la cámara.

    - La solución a este problema se ha conseguido gracias al uso de la librería Three.js, mapeando la videotextura conteniendo la salida de la cámara en un plano o en una esfera según el mapeado deseado.
- Eliminar la distorsión de la salida cruda de la cámara (en ojo de pez).
    - Este problema se ha resuelto mediante un algoritmo de mapeado UV, el cual ajusta las coordenadas de la videotextura a los vértices de cada cara de la esfera (las caras de la esfera son los pequeños triángulos de los que se compone su geometría).
- Ofrecer una interfaz de usuario sencilla y atractiva.
    - Gracias al uso de jQuery y jQueryUI se ha diseñado una interfaz grafica agradable y sencilla de utilizar, aprovechando toda la potencia de JavaScript.

4. IMPLEMENTACIÓN

4.1. Tecnologías y conceptos

En esta sección se explicarán brevemente las principales tecnologías utilizadas en el proyecto, y algunos conceptos que harán más sencilla la comprensión de éste. Antes de conocer los conceptos más específicos que se tratarán durante la explicación del proyecto, se detallarás las principales tecnologías:

- **Javascript**: Javascript es un lenguaje de programación interpretado, que se ejecuta normalmente en el cliente del navegador web, y permite hacer las páginas web mucho más dinámicas. Este es el lenguaje en el que está escrita la librería Three.js, la cual se va a utilizar para el mapeado de las videotexturas creadas a partir de la salida de la videocámara.
- **HTML5**: Esta es la quinta versión del lenguaje de programación web por excelencia, en la que se añaden novedosas mejoras, y para el desarrollo de este proyecto es algo crucial la inclusión del Canvas 3D y las mejoras a los accesos a dispositivos multimedia a través del navegador [8].
- **WebRTC**: Las siglas de WebRTC hacen referencia a Web Real-Time Communication, o lo que es lo mismo, comunicación web en tiempo real. Esta tecnología permite el acceso sencillo del navegador a los dispositivos multimedia en tiempo real, entre otras cosas, utilizando una sencilla API en Javascript [9]. En este proyecto se va a sacar partido del componente que permite acceder a las cámaras y otros dispositivos multimedia: 'getUserMedia'.
- **JQueryUI**: No es indispensable, pero es una colección de objetos, animaciones y temas, que son capaces de mejorar visualmente una página web en gran medida, de una forma sencilla y a través de la librería jQuery.
- **WebGL**: Es una API de Javascript que permite renderizar gráficos 3D en el navegador sin necesidad de usar ningún tipo de plugin. Es el componente que permitirá renderizar el objeto en el que se proyectará la videotextura de la captura de la videocámara.
- **Three.js**: Es una librería escrita en Javascript que facilita el trabajo con varias APIs de renderizado de gráficos, como HTML5 Canvas, SVG o WebGL. En el caso de

este proyecto se va a utilizar para trabajar cómodamente con WebGL, el cual es uno de sus usos más populares.

Teniendo claro esto, es mucho más sencillo conocer la naturaleza del proyecto, pero es posible que más adelante surjan dudas sobre algún concepto, por lo que aquí se explicaran brevemente algunos de estos.

- **Contenedor**: Un contenedor es un elemento de la pagina web, en el que se va a mostrar la escena de Three.js de su correspondiente controlador. Se trata simplemente de un elemento 'div' en el cual se va a crear un canvas WebGL, para mostrar la escena contenedora del objeto tridimensional en el cual se va a mapear la textura con la salida de la videocámara.
- **Controlador**: Un controlador es una instancia de la clase *camera_controller.js*, la cual se encarga de crear y configurar el espacio de Three.js, cámara, escena, pantallas, etc.
- **Escena**: Una escena es un componente de Three.js en el cual se colocan todos los objetos, luces y cámaras, es lo que se va a renderizar a través del motor, en este caso WebGL.
- **Cámara de la escena**: En una escena puede haber varias cámaras, y representan al usuario dentro de la escena, por lo que el renderizador mostrará por pantalla aquello que la cámara de la escena tenga delante.
- **Mapeo UV**: Es una parte del proceso de modelado 3D, en el que se proyecta una imagen o textura 2D en una figura 3D. UV denotan las 2 coordenadas X e Y de la textura, y se usan estas ya que X, Y, y Z ya están utilizadas para representar las coordenadas del objeto 3D [10].
- **Pantalla de la escena**: En cada escena de Three.js, existe una 'pantalla' en la que se mapea la salida de la cámara. No es más que un objeto tridimensional, como un plano o una esfera, en la que se mapea la videotextura que contiene la salida de la videocámara, por lo que dentro de la escena hace el papel de pantalla.
- **GetUserMedia**: Este es el método de la API del navegador, relacionado con WebRTC, que permite acceder a los dispositivos multimedia, en este caso, la videocámara 360.
- **Stream**: Este objeto es el que contiene el flujo de datos de la cámara, y el que se debe enviar a Three.js para su manipulación y conversión en videotextura mapeable.

4.2. Estructura del software

La aplicación consta de 3 ficheros que interaccionan entre sí a lo largo de la ejecución. A continuación se describirán las principales interacciones entre las

diferentes partes de la aplicación, y se mostrará un esquema en el que poder visualizar con mayor facilidad las interacciones previamente descritas.

Antes de describir las interacciones entre las diferentes partes, se procederá a describirlas. La aplicación consta de un fichero *html*, y 2 ficheros *javascript*, además de varias hojas de estilo *css* y otros archivos *javascript* con las librerías utilizadas.

- **aplicación.html**: Este fichero contiene la aplicación en sí, y puede ser personalizado para mostrar el numero deseado de pantallas con su correspondiente disposición sencillamente.
- **theta_controls.js**: Este fichero se encarga de trabajar con la API de WebRTC, para así obtener los dispositivos multimedia del sistema (las cámaras), y gestiona su obtención y procesado. También se encarga de inicializar el código jQuery de los controles de la pagina, y de sus correspondientes funciones, para poder interaccionar con el controlador de la pantalla seleccionada.
- **camera_controller.js**: Este fichero se encarga de todo el procesado del canvas WebGL de cada pantalla de la aplicación, por lo que se debe instanciar uno por cada pantalla. Se encarga de inicializar la librería de Three.js y crear y configurar la escena con sus correspondientes objetos y texturas, también ofrece las funciones para seleccionar la videocámara a visualizar, el modo de mapeado, y el nivel de zoom.

A continuación se describirán brevemente y cronológicamente los pasos por los que puede pasar la aplicación, y sus correspondientes interacciones. Para visualizar mejor estas interacciones, cada paso esta numerado, y estos se pueden observar en la Figura 10, la cual se muestra a continuación de sus descripciones.

El primer paso al abrir la aplicación, es la configuración de los componentes:

1. Se instancia un controlador por cada una de las pantallas a mostrar. Estos se obtienen de los elementos *'div'* de la pagina que comienzan por *'container'*, y se crea un Canvas WebGL en cada uno de ellos.
2. Se obtienen los dispositivos multimedia del sistema para mostrar las videocámaras disponibles en el ComboBox de selección de salida de video.

Una vez inicializados los componentes, se puede comenzar la interacción entre el usuario y la aplicación, que a su vez genera la interacción entre los distintos componentes.

3. Al seleccionar una de las salidas de video en el ComboBox, el fichero *theta_controls.js* recibe el id del dispositivo seleccionado, y a través de *getUserMedia* obtiene la URL del dispositivo y lo envía a los controladores de todas las pantallas, ya que no se pueden visualizar dos cámaras distintas al mismo tiempo.
4. El controlador ofrece la función *setVideoSrc()* que se encarga de obtener la nueva fuente de video y actualizar la fuente de la videotextura a través de la URL suministrada.
5. Al seleccionar un modo de mapeo mediante el ComboBox de la aplicación, theta_controls.js se encarga de gestionar la solicitud de cambio de modo de mapeo a su correspondiente controlador (aquel de la pantalla seleccionada).
6. El controlador ofrece la función *updateVideoScreen()* que se encarga de mostrar el modo de mapeo seleccionado (a través del identificador de mapeo deseado) en su correspondiente pantalla de la aplicación.

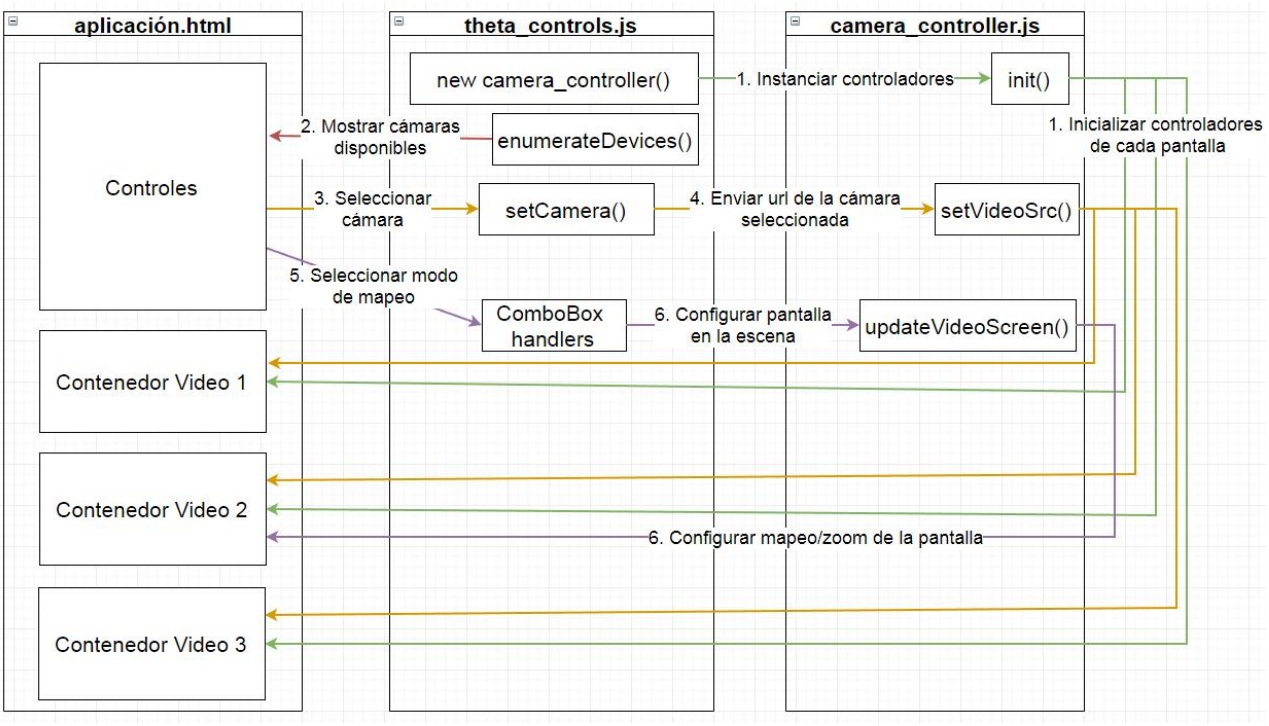

Figura 20: Interacción entre los scripts de la aplicación.

4.3. Interfaz de usuario

La interfaz de usuario de la aplicación es muy sencilla e intuitiva, ya que consta de 3 selectores 'ComboBox', un botón y un 'Slider' para el nivel de zoom.

Al iniciar la aplicación, solo se muestra el selector de salida de video, ya que el resto de controles no tienen utilidad hasta que se seleccione una videocámara. Una vez seleccionada la videocámara y mostrados los controles, se dispone de un botón que permite esconder y mostrar los controles, para simplificar la interfaz al usuario cuando no es necesario modificar más parámetros.

En cada pantalla se pueden modificar 2 parámetros, los cuales son el modo de mapeo de la salida de video y el nivel de zoom de la escena. Para seleccionar la pantalla a modificar se puede usar su correspondiente *ComboBox*, o por conveniencia y facilidad de uso también se ha implementado la posibilidad de hacer clic directamente en una pantalla para seleccionarla. La pantalla seleccionada mostrará un marco rojo, diferente al marco azul del resto de pantallas, para mejorar su facilidad de uso y eficiencia.

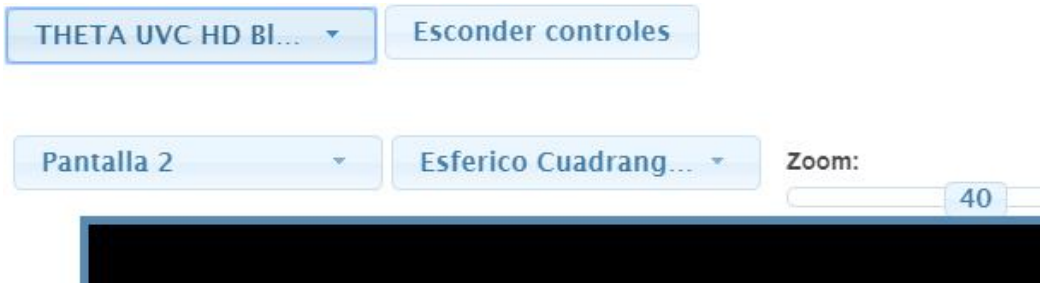

Figura 21: Controles de la aplicación.

La implementación de los controles se ha realizado a través de jQueryUI, para mostrar unos controles visualmente mucho más atractivos y más funcionales. Cada elemento dispone de una función de callback(en el fichero theta_controls.js) encargada de llamar a las funciones correspondientes del controlador en base a los parámetros seleccionados. En la siguiente figura se puede observar la función de callback del selector de mapeo, que una vez modificado el valor del *ComboBox*, llama a la función *updateVideoScreen()* del controlador de la pantalla seleccionada, enviando como parámetro el índice del elemento seleccionado.

```
$( "#selectMapping" ).on( "selectmenuchange", function( event, ui ) {
        if(ui.item.index>0){
                $("#firstSM3").attr("disabled", true);// Para que no se pueda sele
                $("#selectMapping").selectmenu("refresh");

                controllers[controllerSelected].updateVideoScreen(ui.item.index);
                controllerMaps[controllerSelected]=ui.item.index;
                if(ui.item.index==3){//Modificar el slider del zoom a su valor po
                        $("#slider").slider('value',3);
                        handle.text( $("#slider").slider( "value" ) );
                        $("#slider").slider('disable');
                }else{
                        $("#slider").slider('enable');
                        $("#slider").slider('value',40);
                        handle.text( $("#slider").slider( "value" ) );
                }
        }
} );
```

Figura 22: Función de callback del selector de mapeo.

4.4. Conclusión

En este capítulo se ha hablado de la estructura de la aplicación y la interacción de sus componentes, así como de ciertos detalles sobre la interfaz grafica. En resumen, una aplicación se compone de su correspondiente fichero HTML, el cual se puede personalizar con el numero de pantallas deseado, cada una con su correspondiente configuración. Los ficheros que toman el control de la aplicación propiamente son dos ficheros javascript: *theta_controls.js* y *camera_controller.js*, el primero se encarga de gestionar la interfaz grafica y sus correspondientes callbacks, así como de obtener la salida de video deseada. El segundo fichero, *camera_controller.js*, se encarga de crear la escena Three.js y todos los objetos necesarios, así como de implementar las funciones a las que llaman los callbacks de la interfaz grafica.

5. PRUEBAS Y RESULTADOS

La aplicación desarrollada en este proyecto no posee costosos algoritmos los cuales requieran de un gran tiempo de ejecución, ya que se ejecuta en tiempo real, por lo que no es necesario realizar un análisis de tiempo de ejecución. Se han realizado pruebas y análisis del consumo de memoria en los principales navegadores, ya que no todos tratan del mismo modo el canvas WebGL ni la ejecución de código JavaScript.

En Google Chrome la aplicación funciona perfectamente y con un consumo aproximado de memoria de 258MB (Pestaña de la aplicación + Proceso de GPU). El canvas WebGL funciona con gran fluidez y todas las librerías funcionan correctamente.

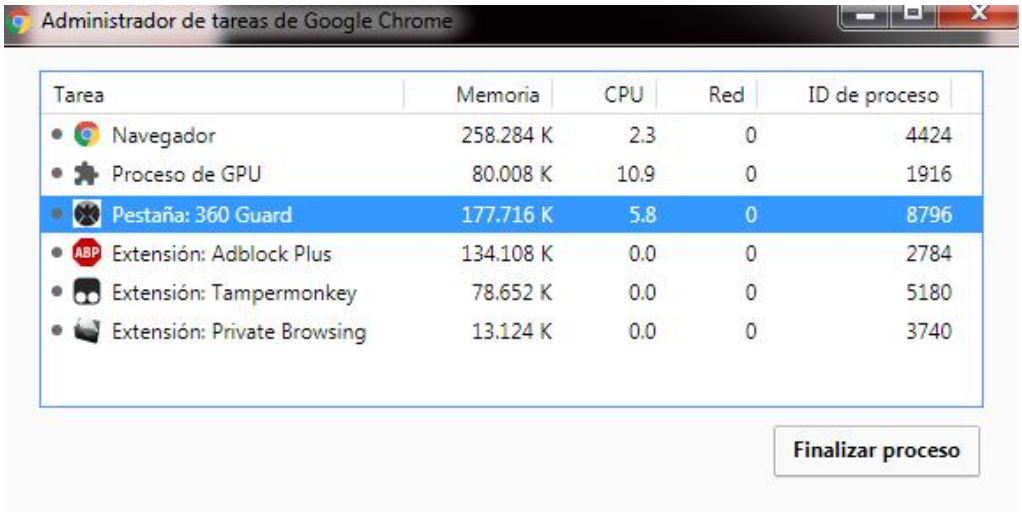

Figura 23: Detalle del consumo de memoria de la aplicación en Chrome.

En Mozilla Firefox la aplicación funciona correctamente, pero el rendimiento del canvas WebGL es menor, haciendo que la aplicación funcione un poco más lenta, pero sigue siendo totalmente aceptable. Este problema parece ser común en los usuarios de Firefox, ya que está reportado por ser considerablemente más lento que Chrome en cuanto al rendimiento de la ejecución de aplicaciones WebGL [11]. El consumo de memoria también es mayor en Firefox, con un consumo aproximado de 333MB.

Figura 24: Detalle del consumo de memoria de la aplicación en Firefox.

En Opera la aplicación también funciona correctamente, si bien se ejecuta algo más lenta que en Chrome, sigue siendo mas rápida que en Firefox. Al parecer no es tan eficiente como el navegador de Google ya que su consumo de memoria también es algo mayor, alcanzando casi los 300MB.

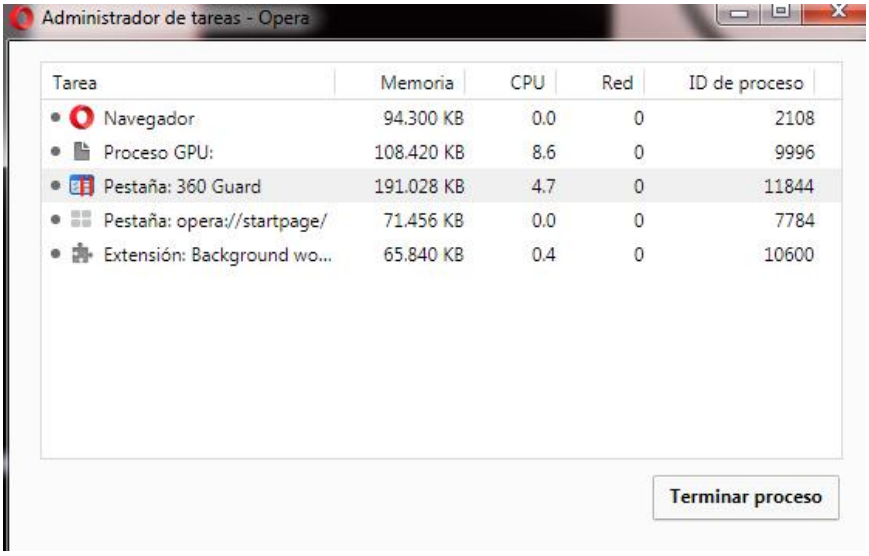

Figura 25: Detalle del consumo de memoria de la aplicación en Opera.

En Internet Explorer 11 la aplicación no funciona correctamente, ya que ésta hace uso de promesas para trabajar con la API de WebRTC a la hora de obtener los dispositivos multimedia del sistema (los cuales no están disponibles inmediatamente). Este tipo de funciones están implementadas desde Junio de 2015 en el nuevo ECMAScript v6, el cual IE11 no soporta [12] [13]. Para solucionar este problema se debería hacer uso de una librería externa que otorgue funcionalidad de promesas a IE, por ejemplo Bluebird, pero ya que Internet Explorer es un navegador poco usado, y ya

en desuso gracias al lanzamiento de Microsoft Edge, este es un trabajo demasiado costoso para las ventajas que podría obtener.

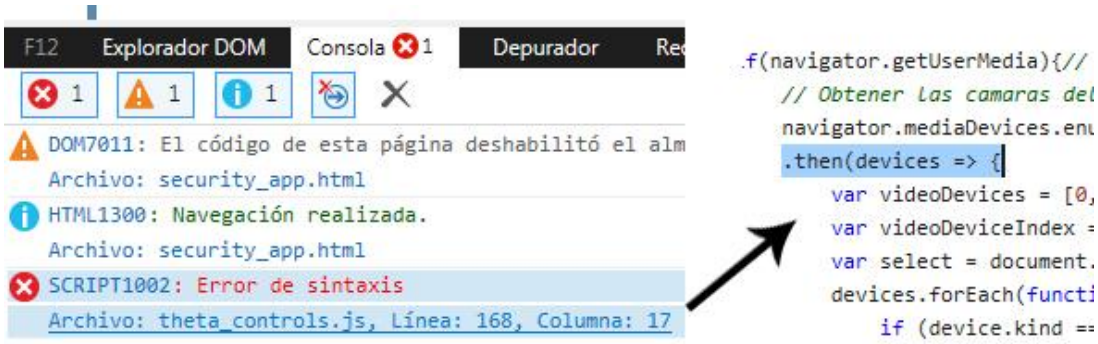

Figura 26: Error de promesas no soportadas en IE11.

Como conclusión de las pruebas se puede afirmar que el mejor navegador para ejecutar la aplicación es Google Chrome, al ser más eficiente en cuanto a recursos consumidos y ejecución del canvas WebGL.

A continuación se muestran unas capturas de la aplicación en funcionamiento, mostrando los distintos modos de mapeo.

Figura 27: Captura de la pantalla inicial tras pulsar en el ComboBox de selección de video.

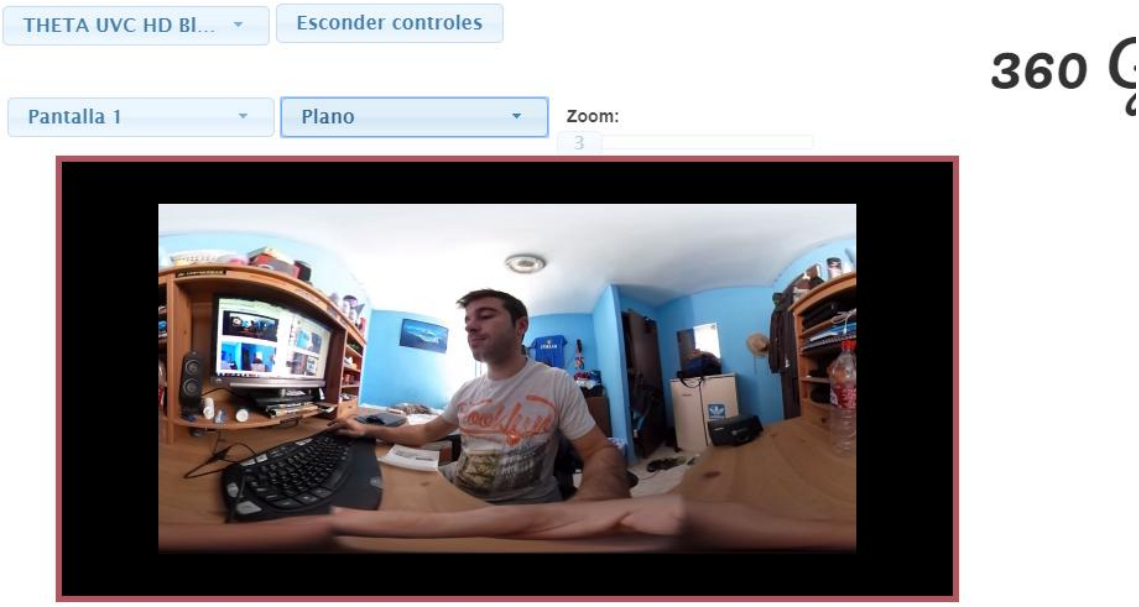

Figura 28: Captura del modo de mapeo en plano de la proyección equirrectangular.

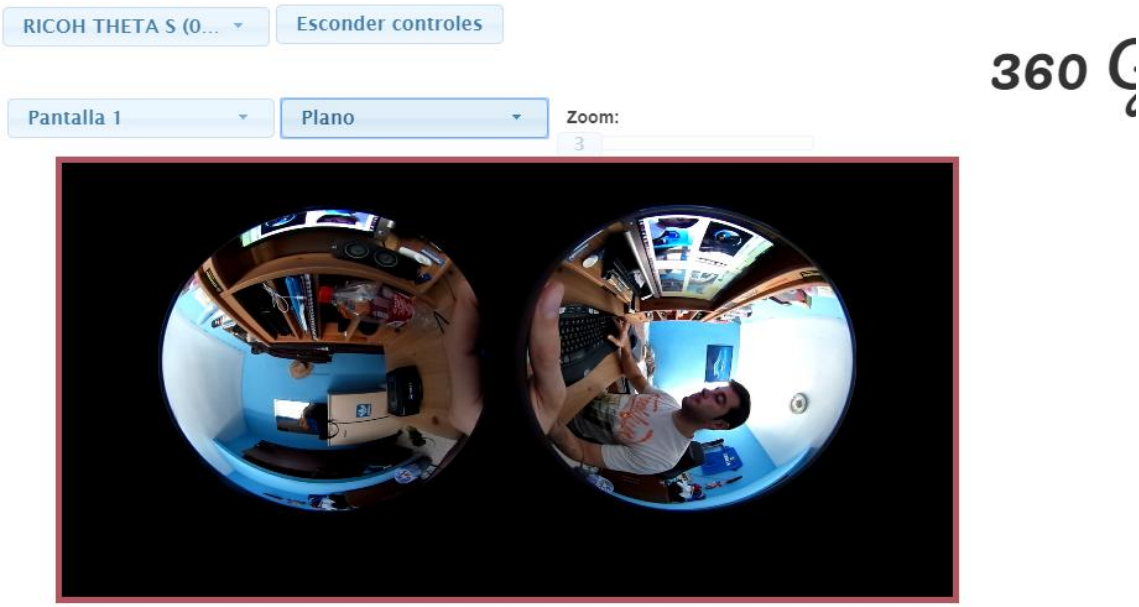

Figura 29: Captura del modo de mapeo en plano de la proyección cruda en ojo de pez.

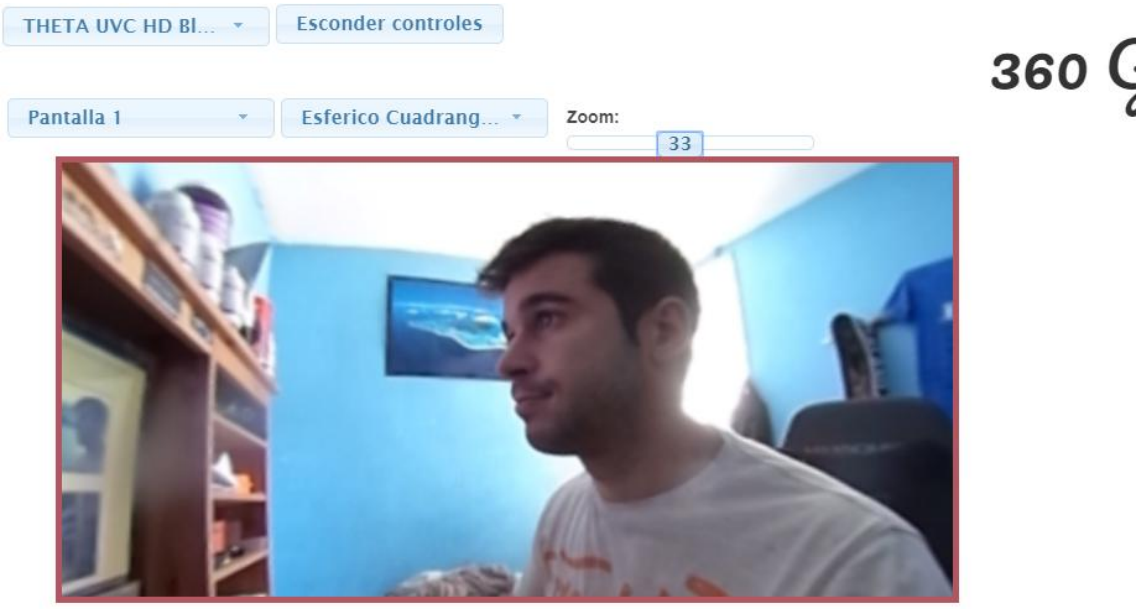

Figura 30: Captura del modo de mapeo en esfera envolvente de la proyección equirrectangular.

Figura 31: Captura del modo de mapeo en esfera envolvente de la proyección cruda en ojo de pez.

Como se observa en las dos últimas capturas (en las cuales se muestra el modo de mapeo en esfera envolvente de las dos proyecciones), la distorsión está reducida al máximo y de un modo similar en ambas imágenes. Tras las pruebas realizadas el resultado es muy satisfactorio, especialmente en la reducción de la distorsión de la proyección cruda la cual era muy alta.

La aplicación y su código se encuentran en la siguiente web, que requiere disponer de una cámara Ricoh 360:

https://fraferg5.upv.edu.es/security_app.html

6. CONCLUSIONES

6.1. Objetivos alcanzados

- ✓ Obtener correctamente la salida de la cámara 360 en el navegador web.
    - Este objetivo se ha conseguido gracias a la API WebRTC, la cual está implementada en la mayoría de navegadores modernos, y ofrece varias funciones de gran utilidad a la hora de trabajar con dispositivos multimedia, entre otras cosas.
    - Se ha usado la función *navigator.getUserMedia()* a la cual se le puede enviar como parámetro un conjunto de restricciones, por lo que se le envía el identificador de la cámara seleccionada en el ComboBox de selección de salida de video para que así devuelva el Stream de dicha videocámara.
- ✓ Proyectar la salida de la cámara en una malla, mediante una videotextura.
    - Este objetivo se ha cumplido gracias a la librería Three.js, con la cual se ha creado una escena 3D en un canvas WebGL. En dicha escena se ha creado una textura base a la que se le ha proyectado la salida de la cámara, la cual se obtiene gracias a su URL, obtenida gracias a la función anteriormente mencionada, *navigator.getUserMedia()*. Dicha función devuelve el Stream de la salida de la videocámara, del cual se puede obtener su URL.
- ✓ Obtener una visualización 360 interactiva de la salida equirectangular de la cámara.
    - Esto se ha cumplido fácilmente gracias al uso de la librería Three.js, creando una esfera, en la cual se mapea directamente la salida equirectangular de la videocámara (contenida en una textura como se explica en el objetivo anterior). La esfera se coloca en el centro de la escena, y justo en el centro de dicha esfera se coloca la cámara de la escena, por lo que el usuario está totalmente rodeado por la salida de la cámara. De este modo apenas existe distorsión, ya que la propia salida equirectangular tiene la distorsión de haber sido obtenida de una esfera, por lo que al mapearla de nuevo en una esfera recupera su apariencia

original (siempre que el nivel de zoom, y por lo tanto el campo de visión no sea demasiado alto).

- ✓ Obtener una visualización 360 interactiva de la salida cruda de la cámara.
  - Este ha sido uno de los objetivos más difíciles de cumplir, ya que la salida de la videocámara en este modo está totalmente distorsionada, y esto se debe corregir con un complejo algoritmo aplicado tras la selección del área de mapeado UV. Una vez corregido el mapeado UV, el proceso es el mismo de el paso anterior, simplemente colocando la cámara de la escena en el centro de la esfera en la cual se aplica el mapeado.
- ✓ Permitir seleccionar el modo de salida de la cámara.
  - Este objetivo también se ha cumplido gracias a la API de WebRTC, al permitir listar todos los dispositivos multimedia del sistema, o solo los que cumplan ciertas restricciones, en este caso, solo los que dispongan de salida de video.
  - Se ha usado la función *navigator.mediaDevices.enumerateDevices()* para obtener los dispositivos multimedia del sistema y así disponer de los identificadores de las videocámaras, entre los que se encuentran los dos modos de salida de la videocámara 360. Dichos identificadores alimentan el ComboBox desde el cual se selecciona la salida de video.
- ✓ Permitir seleccionar el modo de mapeo de cada pantalla de la consola.
  - Este objetivo ha sido relativamente sencillo de cumplir, ya que para ello, se han creado en la escena de Three.js los diferentes objetos que hacen el papel de pantallas, con sus correspondientes mapeos, y se han ocultado todos menos el modo actualmente seleccionado. Al seleccionar un nuevo modo de mapeo en la interfaz grafica, se llama a una función que oculta de nuevo la ‘pantalla’ actual y muestra la que contiene el nuevo modo de mapeo.
- ✓ Permitir mostrar varias pantallas independientes en la aplicación web.
  - Para esto, se ha modularizado el código que controla toda la escena Three.js desde el inicio, conteniéndolo en una clase instanciable y parametrizada. Para crear una nueva pantalla en el código HTML de la pagina, simplemente se debe añadir un nuevo elemento ‘div’ cuyo identificador comience con la palabra ‘container’.

- ✓ Permitir la personalización de la aplicación, creando varias pantallas pre configuradas, por ejemplo para crear una consola de seguridad.
    - Este objetivo se ha conseguido de nuevo gracias a la modularidad de la aplicación desarrollada, y obteniendo el ángulo predefinido de cada pantalla de su identificador. Por lo tanto, el identificador de cada elemento 'div' de las correspondientes pantallas se forma de la siguiente manera: "*container*" + (ángulo deseado). El controlador de cada pantalla se encarga de girar la cámara de la escena para mostrar el área correspondiente.

6.2. Líneas abiertas

La aplicación está orientada a aplicaciones HTML que normalmente se van a usar en un sistema de escritorio o laptop, pero ofrece muchas posibilidades de mejora y ampliación, aprovechando nuevos dispositivos y tecnologías:

- Desarrollo de una página que muestre la salida de la cámara en la pantalla de un dispositivo móvil, el cual haga uso de su giroscopio para dirigir el ángulo de visión.
- Soporte de las pantallas mostradas en la aplicación para mostrarse en un sistema de realidad virtual, como es Oculus Rift.
- Posibilidad de mostrar en la página cualquier cámara 360 remotamente, para lo cual se debería realizar el streaming de la videocámara a través de su correspondiente aplicación, para luego alimentar a la aplicación con la URL de dicho streaming.

7. BIBLIOGRAFÍA

[1] L. Ricoh Company, «Theta Developer documentation - Overview,» [En línea]. Available: https://developers.theta360.com/en/docs/introduction/.

[2] C. Scott, «Journalism.co.uk - How to get involved in the rise of 360-degree video,» [En línea]. Available: https://www.journalism.co.uk/news/how-to-get-involved-in-the-rise-of-360-degree-video-/s2/a611715/.

[3] E. Betters, «Pocket Lint - Best 360 cameras,» [En línea]. Available: http://www.pocket-lint.com/news/137301-best-360-cameras-the-best-vr-and-360-video-cameras-no-matter-your-budget.

[4] FacebookMedia, «Primeros pasos con Facebook 360,» [En línea]. Available: https://www.facebook.com/facebookmedia/get-started/360.

[5] «Mozilla Developer - MediaDevices.getUserMedia(),» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia.

[6] «Qiita.com - THETA Dualfisheye Distorion,» [En línea]. Available: http://qiita.com/mechamogera/items/b6eb59912748bbbd7e5d.

[7] «jQuery documentation,» [En línea]. Available: https://api.jquery.com/.

[8] A. Harris, «Dummies.com - Notable changes in HTML5,» [En línea]. Available: http://www.dummies.com/web-design-development/html/notable-changes-in-html5/.

[9] H. Alvestrand, «Google release of WebRTC source code,» [En línea]. Available: http://lists.w3.org/Archives/Public/public-webrtc/2011May/0022.html.

[10] T. Mullen, Mastering Blender. 1st ed., Indianapolis: Wiley Publishing, Inc., 2009.

[11] «BugZilla - WebGL demos and games are slower on Firefox than on Chrome,» [En línea]. Available: https://bugzilla.mozilla.org/show_bug.cgi?id=856900.

[12] B. Ilegbodu, «Learning ES6: History of ECMAScript,» [En línea]. Available: http://www.benmvp.com/learning-es6-history-of-ecmascript/.

[13] «ECMAScript 6 compatibility,» [En línea]. Available: https://kangax.github.io/compat-table/es6/.

APÉNDICE A. GUÍA DEL USUARIO

I. Manual de instalación de la cámara RICOH THETA 360

La cámara puede hacer streaming en diferentes formatos, dependiendo del tipo y versión de driver utilizado:

Driver	**Formato**	**Resolución y FPS**	**Formato Streaming**
THETA S	dual-fisheye	1280x720 @ 15fps	MotionJPEG
THETA S (firmware 01.82)	dual-fisheye	1920x1080 @ 30fps	H.264
THETA UVC HD Blender	Equirectangular	1280x720 @ 15fps	MotionJPEG
THETA UVC FullHD Blender (firmware 01.82)	Equirectangular	1920x1080 @ 30fps	H.264

I.1. Instalación de los drivers

Para instalar los drivers de la cámara, se debe acceder a su página de soporte: https://theta360.com/en/support/download/

Y descargar la ' Live-streaming app' correspondiente a el sistema operativo utilizado. (Para el correcto funcionamiento de la aplicación se recomienda actualizar Adobe air: https://get.adobe.com/air/).

Se debe instalar la aplicación con la cámara apagada y desconectada, y al finalizar la instalación se pedirá conectar la cámara apagada al puerto USB.

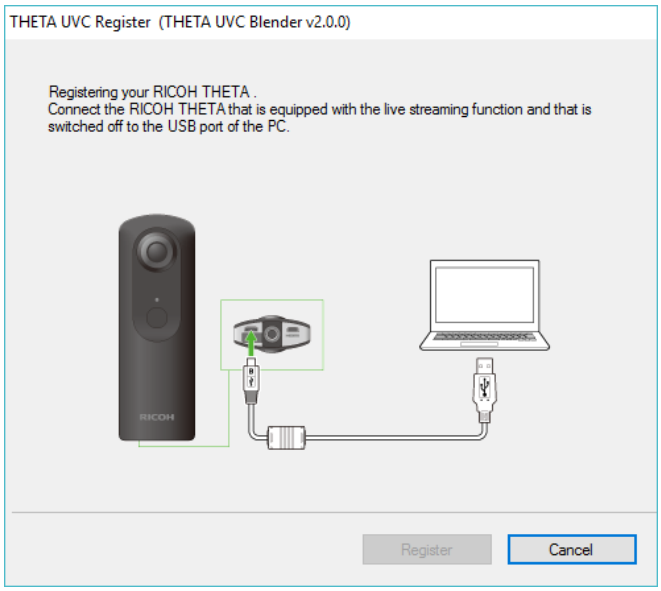

Figura 32: Captura del momento en el que se debe conectar la cámara al sistema.

Una vez conectada y reconocida por el sistema, se puede hacer clic en "*Register*" y la cámara quedará instalada y registrada en el sistema.

I.2.Prueba de la cámara

Una vez instalada la cámara, se puede probar su correcto funcionamiento, para esto se debe apagar y desconectar.

Ahora se debe encender la videocámara en modo Live Streaming, para lo cual se deben pulsar a la vez los botones de encendido y de captura.

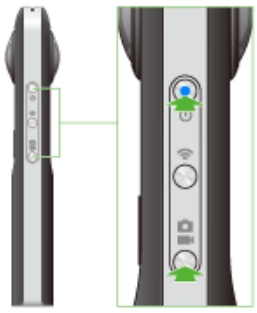

Figura 33: Detalle de los botones que se deben presionar al mismo tiempo.

Tras presionar simultáneamente estos botones, se encenderá una luz azul en el frontal del dispositivo, donde está escrito el texto "Live". Ahora se puede conectar la cámara al sistema, y se debe usar un programa capaz de visualizar el dispositivo en sí, por ejemplo, Skype o Media Player Classic.

Para visualizar la cámara correctamente en Media Player Classic se deben realizar los siguientes pasos:

1. Seleccione [Ver] → [Opciones] en el menú de Media Player Classic.

2. Seleccione [Capturar] para [Reproducir] en el menú de opciones de Media Player Classic, seleccione [THETA UVC FullHD Blender] o [THETA UVC HD Blender] para [Vídeo] y, después, seleccione [OK]
3. Seleccione [Archivo] → [Abrir Dispositivo] en el menú de Media Player Classic.

II. Ejemplo de uso

La aplicación comienza con las pantallas en negro y esperando a que el usuario seleccione una salida de video, en el único selector disponible al inicio, tras lo cual aparecerán el resto de controles.

Figura 34: Pantalla inicial tras pulsar en el ComboBox de selección de video.

Una vez seleccionada una salida de video se mostrará la correspondiente salida en las pantallas, con sus correspondientes configuraciones por defecto, si éstas han sido establecidas.

La configuración de cada pantalla se puede cambiar haciendo clic sobre la pantalla o seleccionándola mediante el selector de pantalla. Una vez seleccionada se puede modificar su modo de mapeo, así como su nivel de zoom. También se puede modificar la dirección de la visión simplemente haciendo clic y arrastrando el puntero hacia la dirección deseada. Cabe señalar que en la proyección sobre plano no se permite cambiar la dirección ni el nivel de zoom, ya que no es algo necesario en estos casos.

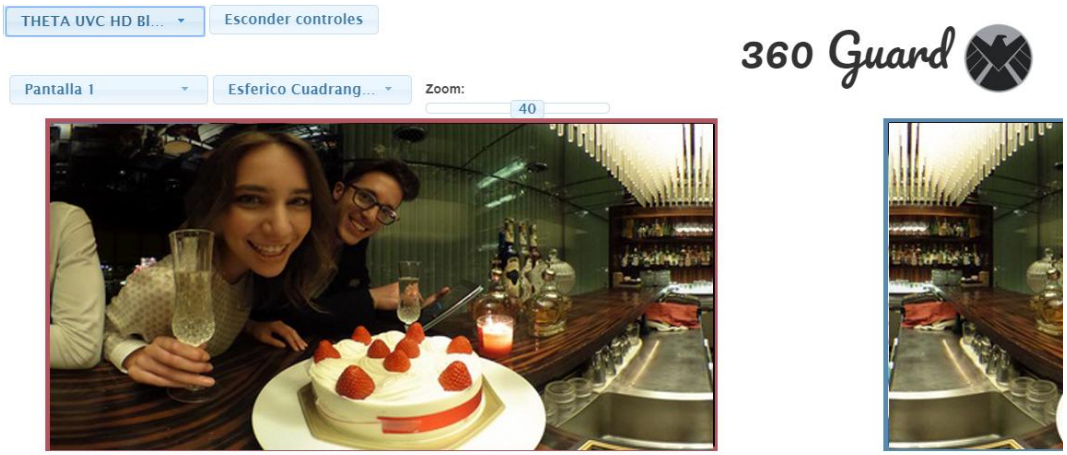

Figura 35: Aplicación con la salida de video seleccionada, que muestra todos los controles.