Implementación de realidad mixta para sistema de búsqueda de rutas en dispositivos móviles

Mixed reality implementation for navigation systems in mobile devices

Por:

Sergio Gavilán Fernández,
Javier Cordero Calvo

UNIVERSIDAD COMPLUTENSE
MADRID

Grado en Desarrollo de Videojuegos
Facultad de Informática

Director:
Pedro Pablo Gómez Martín

Madrid, 2019–2020

Índice general

Índice de figuras

Índice de cuadros

Resumen

La realidad mixta (MR por sus siglas en inglés) ha crecido en los últimos años y ha aumentando su alcance gracias a la accesibilidad de la población a teléfonos móviles. Esta tecnología se basa en la interacción del mundo digital con el mundo físico.

Este trabajo de fin de grado plantea el desarrollo de una aplicación de navegación con rutas a través de un entorno de realidad mixta. Durante el proyecto se estudian las distintas tecnologías de realidad mixta, así como la realidad aumentada y virtual debido a su conexión con esta. Del mismo modo, se ha investigado sobre los distintos servicios de búsqueda de rutas por coordenadas. Estos en su mayoría ofrecen funcionalidades como búsqueda de caminos entre varios puntos, la posición del dispositivo en coordenadas GPS o las distintas rutas que puede tomar una persona para llegar a un destino. Estas últimas dependiendo de tráfico u otras variables. Gracias a esta documentación se han desarrollado dos aplicaciones principales. Por una parte una aplicación que hace uso de la librería ARCore para generar un efecto de oclusión. Con esta aplicación se buscaba generar un entorno de realidad mixta. Por otra parte, una aplicación de navegación utilizando Mapbox. En este caso se crea una ruta desde la ubicación del dispositivo a un destino elegido por el usuario. Esta ruta incluye elementos virtuales para guiar al usuario.

Palabras clave

Realidad Mixta, oclusión, dispositivos móviles, Unity, ARCore, Mapbox, sistemas de navegación, nube de puntos, *Shader*.

Abstract

Mixed reality technology (MR) has developed in the very last years and has increasded its popularity thanks to population's accesibility to mobile devices. This technology is based on the interaction between the digital and the physical world.

This final project presents the development of a navigation application, with routes via a mixed reality environment. Among this project, different mixed reality technologies are studied, as well as both the technologies of augmented and virtual reality, because of the link between those three. In the same way, different route searching services by geographic coordinates have been also studied. Most of them offer different functionalities like path searching between several points, device location in GPS coordinates or the different routes for a person to reach a destination, these last ones depending on traffic or other variables. Thanks to this information, two main apps have been developed. On one hand, it was created an app that uses the ARcore library to create an effect of occlusion. The aim of this app is to generate a mixed reality environment. On the other hand, it was created a navigation app making use of Mapbox. In this case, a route is created from the device location to a destination chosen by the app user. This last route includes virtual elements to guide the user.

Keywords

Mixed Reality, Occlusion, mobile devices, Unity, ARCore, Mapbox, navigation systems, point clouds, *Shader*.

Capítulo 1

Introducción

1.1 Motivación

El acceso generalizado de la población a dispositivos móviles de cada vez mejores características ha ocasionado en la última década el auge de las llamadas tecnologías inmersivas, como la realidad mixta.

En las aplicaciones de realidad mixta, el usuario ve a través de su teléfono móvil una imagen del mundo real tal y como es capturado por la cámara del dispositivo. A la vez, se han incorporado objetos sintéticos que se adaptan e interactúan en la pantalla con los objetos físicos.

Los posibles usos de este tipo de tecnología son ilimitados: desde formación hasta el sector del entretenimiento, pasando por la ayuda en procesos de producción. Este trabajo se centra en las posibilidades de la realidad mixta para la búsqueda de rutas, estudiando las limitaciones y potencialidades de esta fusión para crear una experiencia de navegación inmersiva.

1.2 Objetivos

El objeto principal es estudiar la inclusión de la tecnología de realidad mixta dentro de un sistema de búsqueda de rutas. Para conseguir este objetivo se pueden distinguir dos subobjetivos: Generar un entorno de realidad mixta y colocar elementos digitales en la navegación.

Los pasos para conseguir dicho objetivo principal son los siguientes:

1) Desarrollar un efecto de oclusión para dispositivos móviles en tiempo real.

2) Utilizar el efecto de oclusión para crear un entorno de realidad mixta.

3) Gestionar un sistema de rutas desde un punto A a un punto B.

4) Iniciar la navegación de la ruta a través de elementos de realidad mixta.

1.3 Herramientas usadas durante el proyecto

Existen diferentes librerías que permiten el desarrollo de aplicaciones de realidad aumentada (sección 2.5), en este caso se ha elegido la librería ARCore (sección 2.5.1).

Para el desarrollo de las aplicaciones se ha utilizado uno de los motores de videojuegos más punteros en la actualidad, **Unity**. Este motor ha sido escogido gracias a su compatibilidad con ARCore. La versión que se ha utilizado es Unity 2019.3.1f1.

Para gestionar el código desarrollado se ha elegido Git. Todas las aplicaciones y proyectos se han subido a un repositorio de la plataforma **Github** . Con el objetivo de separar los dos subobjetivos de este Trabajo de Fin de Grado, la realidad mixta y la navegación, se creará una rama "MAPS" donde se trabajará con los proyectos relacionados con los distintos proveedores de búsquedas de rutas. Por otro lado, se utilizará la rama "master" como rama principal para la parte de realidad mixta.

Se puede acceder al repositorio desde el siguiente enlace: https://github.com/JavierCordero/TFG

Este código ha sido implementado utilizando el IDE (del inglés *Integrated Development Environment*) **Visual Studio Community 2020**. Este es el IDE elegido debido a la experiencia que hemos obtenido con su uso durante los distintos cursos del grado y la facilidad que aporta para trabajar con Unity.

Por otra parte, para la navegación se utilizará el sistema de búsqueda de rutas **Mapbox**, ya que al igual que ARCore permite una integración sencilla con Unity gracias los llamados *packages*. A la hora de utilizar ARCore con Unity, si en el proyecto se va a incluir Mapbox, se utilizará la versión de ARCore que viene incluida con Mapbox. En el caso de utilizar ARCore sin incluir Mapbox, se utilizará la versión **ARCore 3.1.3** ya que en esta versión se incluye la actualización donde se permite utilizar la información de la profundidad.

Por último, respecto al desarrollo de la memoria se ha utilizado **Overleaf** debido a su funcionalidad de trabajo colaborativo (al ser únicamente dos personas se puede utilizar sin comprar ningún plan) y la posibilidad de guardar los archivos en la nube así como de descargarlos. Dentro de esta, utilizamos la plantilla *Teflon* para facilitar la estructura del documento.

1.4 Plan de trabajo

Se puede dividir la planificación en dos grandes bloques: efecto de oclusión en dispositivos móviles y búsqueda de rutas.

En primer lugar, en esta primera parte, se deberá investigar sobre el funcionamiento de ARCore a la hora de obtener información sobre la profundidad de los objetos del mundo físico en tiempo real utilizando una única cámara del dispositivo.

Después de conocer dicha información, se realizará una investigación mediante la lectura de libros y artículos sobre realidad mixta para conocer más información sobre la oclusión. Es importante que estos conocimientos estén enfocados a teléfonos móviles y no utilicen cámaras distintas a una única cámara monocular, ya que es la tecnología que se utilizará.

Para poner a prueba los conocimientos obtenidos anteriormente, en esta parte se implementarán 3 aplicaciones:

- **Aplicación del *shader* de profundidad:** Se desarrollará una aplicación para crear un mapa de profundidad (sección 2.2.2).
- **Aplicación del *shader* de oclusión:** Aplicación para probar la efectividad y precisión del *shader* de oclusión (sección 3.2.2).
- **Aplicación de la nube de puntos:** Esta aplicación se utilizará para comprobar la efectividad del *shader* de oclusión a través de una nube de puntos (sección 3.3).

Con estas tres aplicaciones se comprobará la precisión a la hora de generar un efecto de oclusión de distintas formas para establecer cuál es la más precisa y más eficiente.

En la segunda parte, se indagará sobre qué funciones aporta el sistema de búsqueda Mapbox con Unity. Una vez obtenida dicha información, se tomará como objetivo desarrollar una aplicación para probar esas funcionalidades.

Esta aplicación se implementará para obtener unas coordenadas GPS y transformar esa latitud y longitud en coordenadas de Unity. Una vez hecho esto, se transformará la posición obtenida en Unity a una posición en el entorno de ARCore. Todo esto se hará con el objetivo final de poder colocar un objeto virtual dadas unas coordenadas de latitud y longitud con exactitud dentro del entorno virtual.

Chapter 1

Introduction

1.1 Motivation

The widespread access of the general population to mobile devices of better and better characteristics has caused in the last decade thhe rising of the so called ”immersive technologies”, as it is mixed reality.

On mixed reality applications, the user can see, through their mobile device, an image of the real world just asd it is captured by the device's camera. At the same time, sintetic items have been included to interact and get adapted in the screen with the physical objects.

The possible uses of this kind of technology are unlimited: from education to enternainment, going through helping in production processes. This project is centered in the mixed reality possibilities for route searching, studying limitations and potentialities of this fusion to create an immersive navigation experience.

1.2 Goals

The main aim is to study the inclusion of this mixed reality technology within a route searching system. To reach this aim, two subgoals: to generate a mixed reality environment and to include digital elements in the navigation.

The steps to follow to reach this main goal are:

1) To develop an effect of occlusion for mobile devices in real time.

2) To use this effect of occlusion to create a mixed reality environment.

3) To handle a route system from a location ”A” to a location ”B”.

4) Launch the route navigation though mixed reality elements.

1.3 Project Managment

There are different code libraries that allow the development of augmented reality apps (section 2.5), in this specific case the choosen one has been the ARcore library (section 2.5.1).

For the development of these apps it has been used one of the most recognised engines of nowadays: **Unity**. This engine has been chosen thanks to its compatibility with ARcore. The version chosen for this project is *Unity 2019.3.1f1*.

To handle the developed code the software chosen has been Git. Every app and project has been uploaded to **Github's** repository. To sepparate both subgoals of this final project, the mixed reality and navigation, a root named "MAPS" will be created to work with projects related with the different route searching providers. Furthermore, a root named "master" will be used as a main root for the mixed reality part.

The repository can be found here: https://github.com/JavierCordero/TFG

This code has been implemented using IDE *(i.e. Integrated Development Environment)* **Visual studio Community 2020**. This has been chosen because of the usage experience that we have obtained among the different courses of the degree, and its easiness to work with Unity.

Furthermore, the chosen route searching system for the nsvigation will be **Mapbox**, because, as well as ARCore does, it allows a simple integration with Unity thanks to the so called *"packages"*. While using ARCore with Unity, if Mapbook is going to be included in the project, the used version of ARCore will be the one included with Mapbox. If using ARCore without including Mapbox, the chosen version will be **ARCore 3.1.3**, because the upgrade that allows to use the depth information is included.

Finally, about the memory's development, it has been used the online LaTex editor **Overleaf**, because of its collaborative working functionality (the fact of being only two people makes it used without acquiring any payed plan) and because of the allowance of saving any file in the cloud, as well as downloading them. Within this, we use the jig *Teflon* to ease the structure of the document.

1.4 Work plan

The planning can be divided into two big blocks: effect of occlusion in mobile devices and route searching.

On the first place, in this first part, it will be necessary to research about the functioning of ARCore when obtaining information about the depth of the objects of the physical world in real time, using a single camera of the device.

After meeting this information, a research will be done by book and article reading about mixed reality in order to acknowledge more information about the oclusion. It is impor-

tant that this knowledge is focused on mobile phones that do not use different cameras than a single monocular camera, as it is the technology to be used.

To test the previously met knowledge, in this part three apps will be implemented:

- **Depth *shader* application:** An app to create a depth map will be developed (section 2.2.2).
- **Occlusion *Shader* application:** An app to try the effectiveness and precision of the occlusion *shader* (section 3.2.2).
- **Point cloud application:** This app will be used to try the effectiveness of the occlusion *shader* through a point cloud (section 3.3).

These three applications will try the precision while generating an effect of occlusion in different ways, in order to stablish which one is the most precise and efficient.

On the second part, it will be done a research about which functions does this searching system Mapbox provide with Unity. Once this information is obtained, developing an app to try this functionalities will be the aim.

This application will be implemented to obtain some GPS coordinates and transforming this latitude and longitude in Unity coordinates. Then, the obtained position in Unity will be transformed into a position in an ARCore environment. All of this will be done with the final goal of being able to put a virtual object after an exact latitude and longitude coordinates withing the virtual environment.

Capítulo 2

Realidad aumentada, realidad mixta y realidad virtual

En este capítulo se describen las tecnologías de realidad aumentada, realidad mixta y realidad virtual. Además, se habla sobre su uso y los dispositivos necesarios para utilizarlas.

2.1 Definición

Para poder comprender la tecnología de realidad mixta es necesario conocer los conceptos de realidad aumentada y realidad virtual. La realidad aumentada [6] es una tecnología mediante la cual el usuario visualiza el mundo real a través de algún tipo de dispositivo que "sobreimpresiona" objetos sintéticos en tiempo real, es decir que superpone elementos en la imagen del mundo físico (figura 2.1).

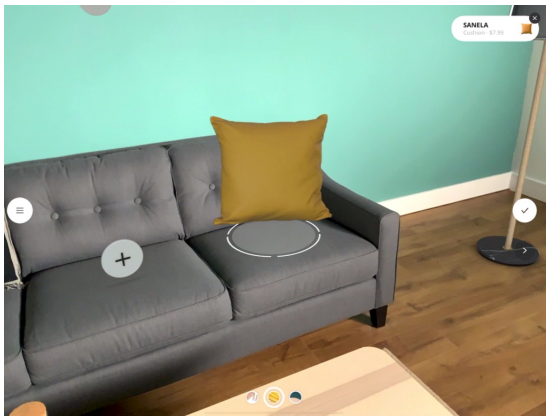


Figura 2.1: Aplicación de Apple IKEA Place[1].

En la realidad aumentada o AR (del inglés *Augmented Reality*) los elementos creados por ordenador que se colocan en el mundo real no tienen conocimiento de los elementos que existen en el entorno donde se están posicionando, de modo que no hay interacción entre ellos.

[1]Fuente: https://www.apple.com/augmented-reality/

Existen dos tipos de realidad aumentada [7], realidad aumentada con marcadores y realidad aumentada sin marcadores. Los marcadores son imágenes en 2D con símbolos o características especiales que son fáciles de captar (figura 2.2). Estos marcadores son usados en la realidad aumentada para colocar los objetos virtuales. En cambio, la realidad aumentada sin marcadores utiliza distintos sensores para obtener la información que aporta un marcador.

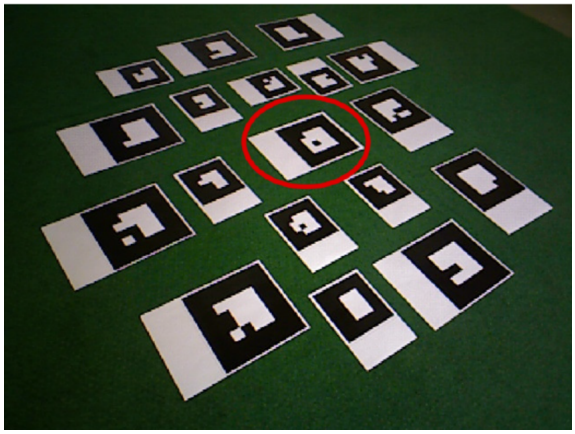


Figura 2.2: Ejemplo de marcadores de AR [1].

Por otro lado, la realidad virtual [8] o VR (del inglés *Virtual Reality*) es una tecnología basada en la simulación de un entorno generado por ordenador que destaca por ser una experiencia inmersiva completa. Por ejemplo, la figura 2.3 muestra un ejemplo donde el usuario maneja un arma con la que dispara a los enemigos de color rojo.

Dicha inmersión se consigue gracias a la sensación de presencia en el entorno virtual que se genera en el usuario principalmente a través de los sentidos de la vista y del oído. En la realidad virtual el usuario se coloca un dispositivo con el que no puede ver nada del mundo físico.

Figura 2.3: Captura del juego de realidad virtual SUPERHOT VR[2].

[2]Fuente: https://www.oculus.com/experiences/quest/1921533091289407/

A diferencia de la realidad aumentada, en la realidad virtual todos los elementos son conscientes de la existencia de los otros objetos, ya que todo el entorno pertenece a la misma simulación y el mundo real no participa en ella.

Por otra parte, se conoce como realidad mixta o MR (del inglés *Mixed Reality*) a la combinación de realidad virtual y realidad aumentada, es decir, a la fusión de la interacción entre elementos físicos y virtuales. En esta tecnología el usuario está presente en un mundo que es tanto virtual como físico, destacando principalmente la posibilidad de interacción de los objetos virtuales con el entorno físico. Por ejemplo, en la figura 2.4 se puede ver en la tableta como se están mostrando elementos virtuales que no existen en el mundo físico mientras la otra persona interactúa con otros elementos de la pared.

Figura 2.4: Experiencia colaborativa de MR con dos tabletas [2].

La definición anterior de realidad mixta es la más aceptada actualmente, aunque una de las primeras apariciones del término le daba un significado ligeramente distinto. En particular, en 1994 Paul Milgram y Fumio Kishino definieron el concepto del continuo de la virtualización [3].

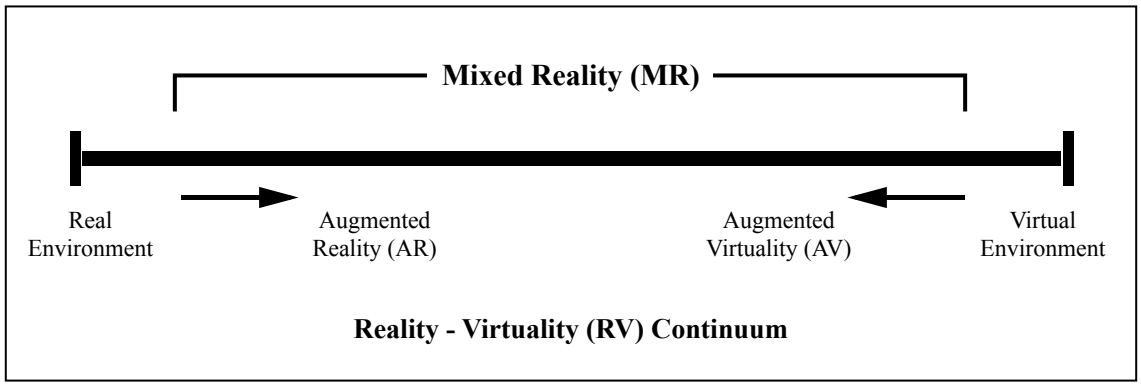


Figura 2.5: Reality-Virtualiy Continuum [3].

Como se puede ver en la figura figura 2.5, en el extremo izquierdo colocaron el mundo físico: objetos, personas, plantas... Elementos que se pueden tocar y sentir. En el extremo derecho colocaron el mundo virtual, es decir, un mundo donde los estímulos son generados

de forma digital. Estos autores definen la realidad mixta como cualquier punto comprendido entre los dos extremos (donde la realidad y la virtualización se mezclan).

Como se ha visto anteriormente, la realidad aumentada consiste en añadir elementos digitales al mundo físico, lo cual según afirman dichos autores estaría en un punto intermedio entre ambos extremos. En cambio, actualmente se considera que la realidad mixta estaría posicionada a la derecha de la realidad aumentada en la figura.

Por último, se conoce con el término de realidad extendida [9] o XR (del inglés *eXtended Reality*) a la tecnología que abarca el conjunto de las tres realidades mencionadas anteriormente, realidad aumentada, realidad mixta y realidad virtual.

2.2 Necesidades tecnológicas

Las tres realidades dentro de las XR que se han tratado utilizan distintos tipos de tecnologías para poder crear las experiencias respectivas a cada realidad.

2.2.1 Tecnologías en realidad aumentada

En primer lugar, un sistema básico de realidad aumentada está formado al menos por tres tipos de elementos: sensores, procesadores y *displays* [10]. Los sensores son los encargados de obtener información del entorno y transmitírsela a la aplicación, los procesadores gestionan dicha información y los *displays*, son los dispositivos en los que se muestra la información de la aplicación.

Actualmente la realidad aumentada se asocia a su uso a través de dispositivos móviles (sección 2.3.1), en los que el sensor es principalmente la cámara, el procesador es el propio móvil (su CPU) y el *display* es la pantalla del teléfono. Aun así, el concepto es más general y se puede ampliar a una gran variedad de sistemas diferentes. En particular estos tres elementos encajan también con el que es considerado el primer visor de realidad aumentada de la historia, la espada de Damocles o *Sword of Damocles* (1968) [4].

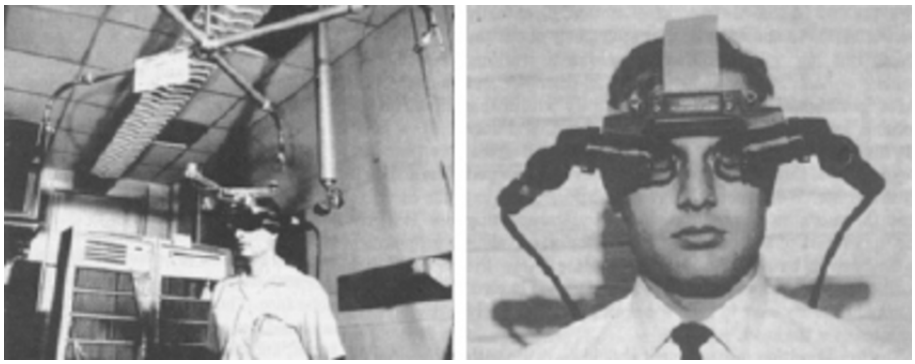


Figura 2.6: Primer visor de realidad aumentada de la historia [4].

Este dispositivo es considerado el primer visor de realidad aumentada de la historia. Dicho aparato utilizaba tubos de rayos catódicos conectados a un computador para generar

imágenes sobreimpresionadas a través de espejos. Como componentes para conocer la posición y orientación del usuario utilizaban un brazo mecánico suspendido del techo que estaba anclado al dispositivo (figura 2.6). En este caso el sensor principal era el brazo mecánico, como procesador se usaba el computador del que dependía y como *display* se utilizaba un estereoscopio[3].

El punto crítico para conseguir una experiencia satisfactoria en AR es el seguimiento de la posición del usuario, de modo que sea posible integrar los objetos virtuales en las posiciones correctas de la imagen que el usuario ve, incluso aunque éste se desplace por el entorno. A este seguimiento se le conoce como *tracking* y se consigue a través de los ya mencionados sensores.

Se pueden distinguir distintos tipos de *tracking* en función de los componentes que se utilicen para generar la información referente al usuario. Esta búsqueda de la pose del individuo se puede dividir en las siguientes técnicas [11]:

- **Técnicas basadas en sensores:** Se utilizan sensores magnéticos, acústicos, inerciales o mecánicos entre otros. Por ejemplo, el *tracking* con sensores mecánicos se realiza atando enlaces al objeto al que se quiere realizar el seguimiento. Los enlaces tienen sensores que aportan información sobre el ángulo que se forma entre la unión de varios enlaces. Es el caso del sistema de la espada de Damocles descrito, donde el seguimiento se realizaba sobre la posición del usuario.

  Normalmente se utiliza un potenciómetro (figura 2.7) en las uniones para obtener el voltaje, de este modo, cuando el ángulo entre los enlaces cambia, varía la resistencia del potenciómetro haciendo que cambie el valor del voltaje. Gracias a este componente se puede conocer la posición y orientación del objeto.

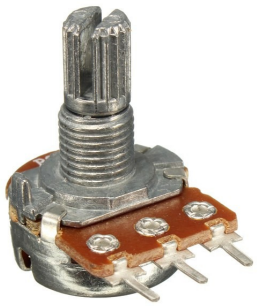


Figura 2.7: Potenciómetro[4].

  Por otra parte, el seguimiento mediante sensores magnéticos [12] es otra de las opciones del *tracking* con sensores. El *tracking* magnético se basa en la generación de un campo magnético de un transmisor para posteriormente, medir dicho campo en un receptor colocado en el objeto al cual se quiere hacer el seguimiento [13].

[3]Aparato en el que, mirando con ambos ojos, se ven dos imágenes de un objeto, que, al fundirse en una, producen una sensación de relieve por estar tomadas con un ángulo diferente para cada ojo. (Fuente: https://dle.rae.es/estereoscopio)

[4]Fuente: https://www.iberobotics.com/producto/potenciometro-lineal-b10k-10k/

Los campos magnéticos generados por el emisor pueden ser distorsionados por cualquier elemento metálico cercano, es por ello que este proceso no es una forma de seguimiento fiable por si solo para calcular la pose del objeto de forma exacta, ya que genera error en su estimación.

Aunque su uso no es tan habitual en las aplicaciones de realidad aumentada para hacer el seguimiento, otro sensor habitual es el GPS (del inglés *Global Positioning System*). El GPS es un servicio que proporciona a los usuarios información sobre posicionamiento, navegación y cronometría [14]. Se puede obtener la ubicación del usuario gracias a una red de satélites que orbita sobre la Tierra.

Para finalizar, se pueden distinguir 3 sensores comúnmente utilizados para el seguimiento del usuario: acelerómetro, giroscopio y magnetómetro (campo magnético).

El acelerómetro es el sensor encargado de medir la aceleración de un dispositivo. La fuerza generada por el movimiento hace que se generen unas cargas eléctricas con las que se puede obtener la aceleración. Por otro lado, el giroscopio es un sensor mecánico que se utiliza para obtener la rotación del dispositivo.

Por último, el magnetómetro sirve para detectar los campos magnéticos. El uso más habitual es para detectar el campo magnético de la tierra, por lo que sirve de brújula y permite medir la orientación del dispositivo en relación a él. Aun así, se puede utilizar para otros usos como el visto en las *Google Cardboard*[5] donde se puede accionar el móvil sin tocarlo gracias a las variaciones del campo magnético creadas al accionar un botón.

Estos tres sensores se utilizan en conjunto para formar la unidad de medición inercial o IMU (del inglés *Inertial Measurement Unit*). La IMU es un dispositivo electrónico que mide la velocidad, aceleración y fuerzas gravitacionales de un dispositivo.

- **Técnicas basadas en la visión:** Estas son las técnicas más comunes. En ellas, se utiliza la visión por computador [15] para obtener información sobre la posición (o el movimiento) del usuario. Dicha técnica de visión por computador se basa en encontrar una correspondencia entre puntos característicos de la imagen 2D y el entorno en 3D.

  De esta forma, la pose del elemento del cual se quiere hacer el seguimiento, se puede obtener proyectando las coordenadas 3D de los puntos característicos sobre las coordenadas de la imagen 2D y minimizando la distancia hacia estos (lo cual es un proceso lento).

- **Técnicas híbridas:** Las técnicas híbridas surgen de la necesidad de una precisión mayor en algunas aplicaciones debido a la robustez de las mediciones que se obtienen al hacer *tracking* mediante sensores o por visión. Se pueden combinar las técnicas por sensores con las técnicas que utilizan visión por computador, de esta manera,

[5] *Google Cardboard*: https://arvr.google.com/cardboard/

se saca partido a la velocidad de los sensores frente a la visión por computador y se complementa la robustez de los sensores con la precisión de las técnicas por visión.

Estas combinaciones permiten afinar el *tracking*, por ejemplo, en una aplicación en el exterior en entornos como ciudades [16] donde el nivel de distorsión es elevado y los sensores utilizados tradicionalmente (GPS y brújula magnética) son imprecisos.

Para finalizar, un ejemplo de modelo híbrido es el modelo que combina información obtenida por el giroscopio y visión por computador basada en líneas [17]. En este modelo se hace uso del giroscopio para predecir la orientación y la posición de las líneas de las imágenes que luego es corregida por la visión por computador.

Otra técnica comúnmente utilizada en realidad aumentada es la tecnología SLAM [18] (del inglés *Simultaneous Localization and Mapping*) la cual consiste en una serie de algoritmos utilizados para calcular la posición del objeto del cual se hace el seguimiento, además, realiza una reconstrucción 3D del entorno. En el caso de estar utilizando un dispositivo móvil a la hora de aplicar SLAM, se utiliza la información de la cámara, acelerómetro y giroscopio, también, puede complementarse esa información con la obtenida por el GPS, sensores de luz o incluso cámaras de profundidad.

El SLAM se utiliza en AR. En cambio, si estamos hablando de realidad virtual se pueden diferenciar las técnicas de *tracking* conocidas como *inside-out* y *outside-in*.

La diferencia entre ellas es la localización de las cámaras que hacen el seguimiento del usuario. La técnica *inside-out* utiliza las cámaras que posee el dispositivo de realidad virtual que se esté utilizando (colocado en la cabeza), en cambio, *outside-in* utiliza cámaras colocadas en distintos puntos del entorno (figura 2.8).

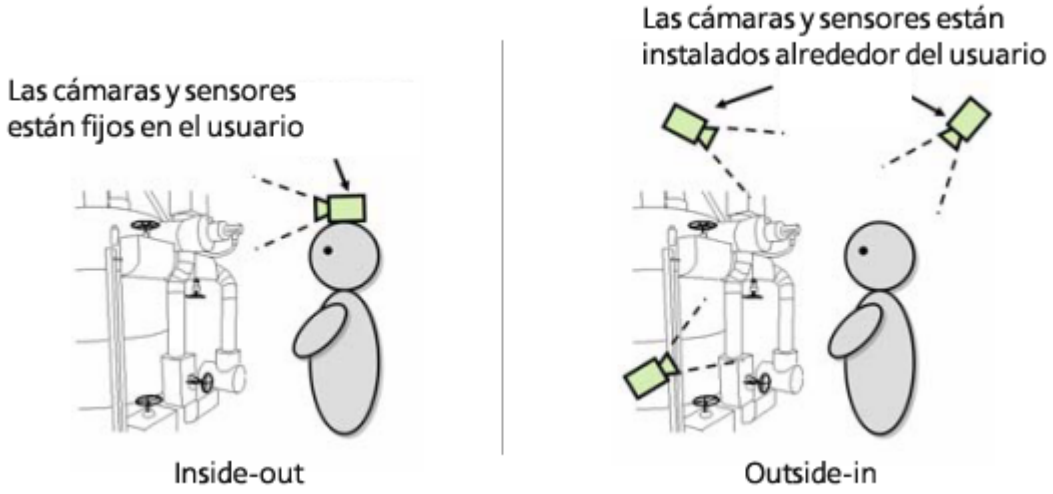


Figura 2.8: Distinción entre *tracking* del tipo *inside-out* frente al *outside-in*[6].

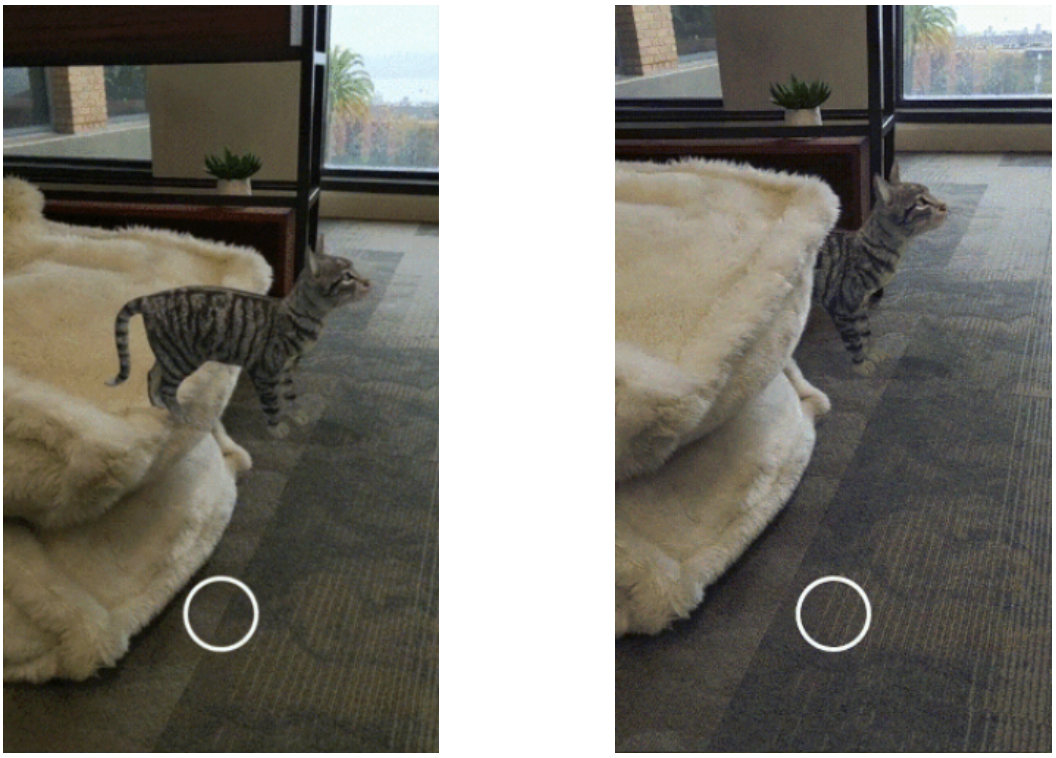
2.2.2 Tecnologías en realidad mixta

En la realidad mixta se hacen uso de las mismas tecnologías que las vistas en el apartado 2.2.1 de realidad aumentada. Aun así, para una correcta experiencia de realidad mixta

es necesario también un buen efecto de oclusión.

La oclusión [19] ocurre cuando los objetos del mundo real están por delante de los objetos virtuales en la escena. Si no hay oclusión, el objeto virtual se superpondrá al real creando la sensación de que el objeto real está detrás cuando realmente no es así (figura 2.9).

Ya que en la realidad aumentada los objetos virtuales no tienen constancia de la existencia de los objetos del mundo físico, puede ocurrir que no haya una correcta oclusión entre elementos.



(a) Realidad aumentada sin oclusión. (b) Realidad aumentada con oclusión.

Figura 2.9: Ejemplo del efecto de oclusión[7].

Para generar una buena oclusión en realidad aumentada, es necesario poder reconstruir el mundo real en 3D. Este es un proceso complicado ya que los dispositivos usados actualmente para AR no poseen los componentes necesarios para hacer esta reconstrucción de forma precisa y rápida. Aun así, a día de hoy existen distintos métodos para generar oclusión [20]:

a). **Mapa de profundidad**: En este caso se utiliza un sensor de profundidad. Es común que el sensor de profundidad que se utilice sea un sensor *ToF* (del inglés *Time of Flight*). Un sensor *ToF* mide la profundidad en función a la distancia que tarda la emisión y recepción de un haz de luz infrarrojo. Con esa información se crea un mapa de profundidad en 2D (figura 2.10). Este mapa no es perfecto ya que hay huecos en él. Además, la resolución del mapa suele ser menor que la resolución de la cámara por lo que hay que escalar el mapa generando imperfecciones.

[7]Fuente: https://developers.googleblog.com/2019/12/blending-realities-with-arcore-depth-api.html

b). **Mallas de oclusión**: En este método se crea una nube de puntos (apartado 3.1) basados en la profundidad para posteriormente unir dichos puntos. De este modo, se crean mallas que reconstruyen el entorno en 3D y sirven como elemento para generar la oclusión.

Ya que los dispositivos que se utilizan para AR (generalmente teléfonos móviles, de los que se habla en la sección 2.3.1) generan una oclusión muy imprecisa en tiempo real con estos métodos, existen otros métodos para generar oclusión, pero en este caso, no se realiza la reconstrucción en tiempo real.

Por ejemplo, el estudio [5] donde hacen una grabación del elemento sobre el que quieren generar la oclusión y aplican un algoritmo de visión por computador. Este mapa generado (figura 2.10) es muy preciso y de este modo se puede crear un efecto de oclusión sin grandes desajustes, todo esto utilizando únicamente una cámara monocular.

Por otro lado, si se conoce el entorno en el que se va a realizar la experiencia de realidad aumentada, se puede, previamente, recorrer el entorno repetidas veces generando una reconstrucción 3D del entorno más precisa que en tiempo real (generando una malla de oclusión). Posteriormente, esta reconstrucción se utiliza como modelo 3D para generar la oclusión.

Figura 2.10: Ejemplo de mapa de profundidad obtenido en el estudio [5].

2.2.3 Tecnologías en realidad virtual

En el caso de la realidad virtual es necesario tener en cuenta otros factores para una buena experiencia aparte de la posición y orientación del usuario, como por ejemplo, la

posición y gestos de las manos o la distancia interpupilar.

Primero de todo, hay que tener en cuenta el concepto de DoF (del inglés *Degrees of Freedom*). Los DoF hacen referencia a los distintos movimientos o rotaciones que puede tener en cuenta o realizar un elemento. En los dispositivos de realidad virtual se distingue entre 3DoF y 6DoF (figura 2.11). Por ejemplo, en un dispositivo con 3DoF solo se tiene en cuenta la rotación mientras que si posee 6DoF se tiene en cuenta estas 3 rotaciones y el movimiento del dispositivo en 3 ejes.

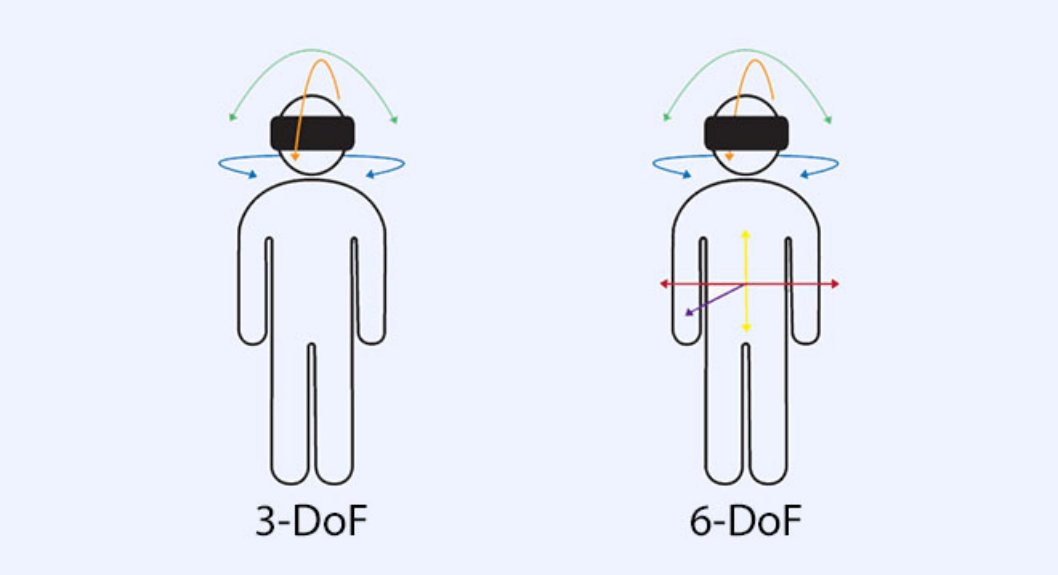


Figura 2.11: Diferenciación entre 3DoF y 6DoF[8].

Por otra parte, el *hand tracking* o seguimiento de manos es un proceso mediante el cual se obtiene la posición de las manos y así como los gestos que está realizando, como cerrar el puño o agarrar algún objeto entre otros. Un modelo que se presenta para hacer este seguimiento [21] habla de la dificultad del proceso debido a la gran cantidad y variación de DoF que pueden tomar las manos. Este modelo se basa en una única cámara de profundidad mediante la cual se obtienen imágenes que luego procesan utilizando aprendizaje automático, para finalmente poder obtener la posición de las manos y su rotación en tiempo real.

Otra tecnología aplicada a la realidad virtual es el *eye tracking*. El *eyetracking* es una solución tecnológica que pretende extraer información del usuario analizando sus movimientos oculares. Esto se consigue lanzando rayos infrarrojos a los ojos del usuario[9].

Aunque no es obligatorio para un correcto funcionamiento de la realidad virtual que los dispositivos utilicen *hand tracking* (ya que se puede utilizar un par de mandos en su lugar) es un factor que incrementa la inmersión del usuario notablemente. Del mismo modo, no es obligatorio el uso de *eye tracking* o seguimiento de ojos pero puede ser beneficioso para la experiencia. Gracias al *eye tracking* se puede obtener información útil [22] como

[8]Fuente: https://virtualspeech.com/blog/degrees-of-freedom-vr
[9]Fuente: https://www.solucionesc2.com/que-es-el-eye-tracking-y-para-que-nos-sirve/

las regiones de interés del espacio 3D y saber en qué momento se ha mirado hacia estas regiones. También se puede cambiar la información en pantalla basándose en dónde está mirando el usuario o incluso controlar las interfaces virtuales con la mirada.

Cuando se hace uso de un visor de realidad virtual, cada ojo obtiene una imagen ligeramente desplazada en el espacio[10]. El cerebro crea una imagen única a partir de esas dos imágenes desplazadas. La distancia interpupilar o IPD (del inglés *Interpupillary Distance*) es otro factor a tener en cuenta a la hora de una experiencia virtual satisfactoria debido al proceso del cerebro de crear dicha imagen única. La IPD es la distancia (normalmente medida en milímetros) entre los centros de los dos ojos (figura 2.12).

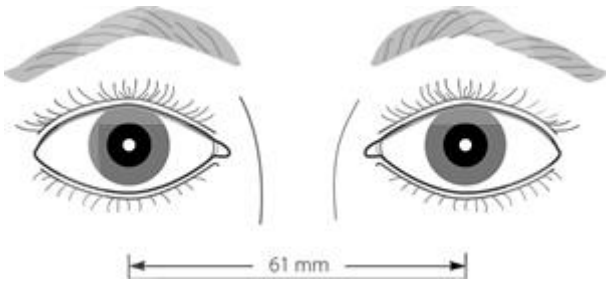


Figura 2.12: Medición de la distancia interpupilar[11].

Según un estudio realizado con distintas personas haciendo variaciones de la IPD [23] en un HMD (del inglés *Head Mounted Display*) o casco de realidad virtual, se llegó a la conclusión de que no había alteraciones a la hora de distinguir los tamaños de los objetos virtuales o afectaciones en la claridad de la visión. En cambio, los usuarios notaron una mayor fatiga con una configuración en el HMD de 50mm y 74mm en la distancia interpupilar en comparación a un valor de la IPD adecuado a su distancia interpupilar anatómica.

También está relacionado con la visión el concepto de FOV, del inglés *Field Of View*. El FOV hace referencia a la amplitud del campo de visión que ofrece el dispositivo. Los seres humanos pueden llegar hasta 180° gracias a los dos ojos.

Para finalizar, respecto a la autonomía de los dispositivos, estos pueden ser inalámbricos o depender de algún cable como fuente de energía. Los dispositivos inalámbricos suelen utilizar pilas o baterías recargables con distintos tiempos de autonomía y en caso de necesitar conectarse a un dispositivo utilizan *bluetooth*. En el caso de no ser inalámbricos los dispositivos suelen conectarse a un ordenador a través de cables HDMI o DisplayPort y USB.

2.3 Dispositivos

Dado que la realidad aumentada y la realidad virtual tienen distintas necesidades tecnológicas como se ha tratado en la sección 2.2, sus dispositivos correspondientes también tienen componentes hardware distintos para poder satisfacerlas.

[10]Fuente: https://www.vistaoftalmologos.es/realidad-virtual-atencion-los-ojos/
[11]Fuente: https://www.aao.org/image/interpupillary-distance-2

En el sector de la realidad aumentada destaca principalmente el uso de teléfonos móviles, debido a su fácil disponibilidad y a la facilidad añadida de crear aplicaciones para estos mediante la aparición de las tecnologías ARCore y ARKit (sección 2.5.1). Se puede disfrutar de esta experiencia usando gafas de realidad aumentada y HMDs.

Por otra parte, en el campo de la realidad virtual destacan los HMD. Estos dispositivos son variados entre ellos en cuanto a los sensores, cámaras o conectividad, entre otras características.

Al ser la realidad mixta una combinación entre las dos realidades mencionadas anteriormente, la MR puede ser utilizada a través de cascos enfocados a experiencias de realidad virtual (aunque estos cascos no se van a tratar en esta sección), dispositivos enfocados para su uso en realidad aumentada o incluso desde teléfonos móviles.

Del mismo modo, se pueden distinguir dos tipos de dispositivos específicos de realidad mixta en el campo de los HMD. Por un lado los cascos en los que se ve directamente el mundo físico y por otro aquellos HMD en los que la visión del usuario está completamente tapada por el casco y el mundo real se ve a través de las cámaras del dispositivo.

También, cabe destacar que un gran número de dispositivos de realidad mixta se utilizan a través de la plataforma WMR[12] (*Windows Mixed Reality*). Esta plataforma ofrece acceso a numerosas aplicaciones de realidad mixta y realidad virtual. Para poder hacer uso de WMR es necesario un ordenador con Windows 10 y una tarjeta gráfica y procesador potentes.

Por último, los teléfonos móviles están empezando a ganar importancia en el sector de la realidad mixta. Según un estudio realizado por Sam Barker [24], los avances en el *Edge Computing* y la aparición de redes 5G acelerarán el desarrollo de la tecnología de realidad mixta en móviles.

Por una parte, el *Edge Computing* [25] es un paradigma que permite que los servicios de computación en la nube[13] sean más cercanos al usuario final.

Por otra parte, el 5G es la quinta generación de tecnología inalámbrica. Las redes de 5G se caracterizan por un aumento de la velocidad y menor latencia frente al 4G.

Dicho autor afirma que las conexiones inalámbricas que se utilizan hoy día para procesar la información de los dispositivos, deberán alcanzar velocidades mayores (usando el 5G y el *Edge Computing*) para una experiencia inmersiva total. En este estudio se calcula que gracias a estos avances se alcanzará un mercado de 43 billones de dólares en el año 2024, a diferencia de los 8 billones alcanzados en 2019.

2.3.1 Teléfonos móviles

Los teléfonos móviles son un gran dispositivo ya que combinan GPS, cámara, brújula y

[12]https://www.microsoft.com/en-us/mixed-reality/windows-mixed-reality

[13]El *cloud computing* o computación en la nube [26] es una tecnología de computación que provee unidades computacionales de bajo coste.

un acelerómetro, cubriendo así las necesidades de una aplicación de realidad aumentada [27]. El teléfono móvil es un dispositivo muy común entre la población lo cual facilita el desarrollo y expansión de esta realidad. Por otro lado, dichos dispositivos se pueden utilizar también para realidad mixta. Además, existen cascos en los que se introduce el teléfono móvil para generar experiencias de MR.

Existen distintos HMD que se utilizan para estas experiencias de realidad mixta acoplando los móviles a alguna parte de las gafas, como por ejemplo, las *Holoboard Enterprise Edition* de la mano de la empresa *TESSERACT* (figura 2.13). Este dispositivo es compatible a partir de Android 6.0 y como especificaciones técnicas se pueden destacar sus 82° de FOV, utiliza la tecnología SLAM para el *tracking*, posee un controlador 6DoF y utiliza el *tracking* por sensores IMU, además, permite experiencias colaborativas en la nube.

Figura 2.13: *Holoboard Enterprise Edition*[14].

En cambio, si estamos hablando de un dispositivo para utilizar en móviles con iOS podemos hablar del casco *Bridge* (figura 2.14) de la empresa *Occipital*. Este HMD requiere de un componente extra llamado *Occipital Structure Sensor*, el cual, se utiliza para el escaneo 3D del entorno.

Figura 2.14: *Occipital Bridge*[15].

Las gafas *Bridge* tienen un FOV de 120°, un controlador de 6DoF y destacan por que gracias a estas se puede utilizar aplicaciones de realidad aumentada, realidad mixta y realidad virtual, también, utiliza la técnica de *tracking* conocida como *inside-out*.

Pese a que la principal diferencia entre las dos gafas es el sistema operativo para el que están destinadas, se puede observar que los precios son similares y que las gafas destinadas para iOS poseen un FOV notablemente superior.

[14]Fuente: https://www.aniwaa.com/product/vr-ar/tesseract-holoboard-enterprise-edition
[15]Fuente: https://www.aniwaa.com/product/vr-headsets/occipital-bridge/

Dispositivo	Precio	DoF	FOV	*SO*	*Tracking*
Holoboard Enterprise Edition	$399	6DoF	82°	Android	SLAM
Occipital Bridge	$349	6DoF	120°	iOS	*Inside-out*

Cuadro 2.1: Comparación entre ambas gafas de MR para móviles.

2.3.2 Hololens 2

Las *Hololens 2* (figura 2.15) son un casco desarrollado por Microsoft que salió al mercado en 2019. Es un dispositivo dotado de tecnología de *eye tracking* y *hand tracking* capaz de detectar las dos manos, esto añade un extra de inmersión a la experiencia y de comodidad ya que no se necesitan mandos para manejarlo y se pueden tener las manos libres. Respecto a los sensores, goza de acelerómetro, giroscopio y magnetómetro para medir la velocidad, orientación y fuerzas gravitacionales del dispositivo.

(a) Vista lateral. (b) Vista superior.

Figura 2.15: Vistas de las Microsoft HoloLens[16].

Las *Hololens 2* funcionan de forma independiente ya que tienen integrado su propio procesador y utilizan una batería recargable con una autonomía de entre 2 y 3 horas. Además, permiten capturar el movimiento y los giros del usuario gracias a su control de 6DoF. Por otra parte, respecto a la conectividad, este HMD tiene WiFi, *bluetooth* y permite conexiones mediante un USB tipo C. Por ejemplo, se puede utilizar la conexión WiFi para navegar por internet (a través del navegador *Microsoft Edge* incorporado en las gafas) o para conectarse a reuniones en línea con otros usuarios de las mismas gafas.

Del mismo modo, este casco está diseñado con un sistema ergonómico de tal forma que no hay necesidad de quitarse las gafas, ya que se puede colocar directamente sobre ellas (figura 2.16).

Figura 2.16: Diseño ergonómico del dispositivo[16].

[16]Fuente: https://www.microsoft.com/es-es/hololens/hardware

Para finalizar, este HMD tiene un FOV de 43°, una resolución de 2.048×1.080 píxeles por ojo (4.096x1.080 en combinación) y una IPD que se ajusta automáticamente según la posición de los ojos que es detectada por un sensor. Además, tiene un sensor de 1 megapíxel de profundidad *ToF*. Este sensor calcula distancias entre objetos en función a la distancia que tarda la emisión y recepción de un haz de luz infrarrojo. El peso del dispositivo es de 566 gramos.

2.3.3 HMD Odyssey+

El casco *HMD Odyssey+* (figura 2.17) es un dispositivo desarrollado por la empresa Samsung que salió al mercado en 2018. No es un casco independiente ya que requiere estar conectado a un ordenador compatible con la plataforma *Windows Mixed Reality* para funcionar, se conecta al ordenador mediante un cable HDMI 2.0 y un USB 3.0.

Figura 2.17: *HMD Odyssey+ dispositivo al completo*[17].

En cuanto a la parte de monitorización de la imagen, este casco tiene una resolución de 1.440 x 1.600 píxeles por ojo, un total de 2.880 x 1.600 píxeles combinando los dos ojos, además, el *Field of View* abarca un total de 110°. Este dispositivo no posee tecnología de *hand tracking* (por lo que controla el movimiento de manos con dos mandos que funcionan con pilas) y tampoco tiene *eye tracking*.

Por otra parte, está dotado de sensores como acelerómetro, giroscopio, brújula y sensor de proximidad (este último se utiliza para saber en qué momento se lleva puesto el casco). Del mismo modo, posee una diadema que se puede modificar para ajustar el casco a la cabeza de cada individuo. Sin embargo, este dispositivo no está diseñado para ser utilizado con gafas.

Por último, tiene un sensor que ajusta automáticamente la IPD, conexión *bluetooth* y un peso total de 644g teniendo en cuenta solo el HMD y un añadido de 176g si se cuenta el peso del cable.

2.3.4 Magic Leap One

El dispositivo *Magic Leap One*[18] (figura 2.18) fue lanzado al mercado en el año 2018 de manos de la compañía Magic Leap. Principalmente este casco destaca por su ligero peso de 316g frente a otros dispositivos, por otra parte, son un HMD independiente ya que

[17]Fuente: https://www.samsung.com/hk_en/news/product/reality-headset-hmd-odyssey-plus/

viene con un sistema (procesador y tarjeta gráfica) para no tener que ser usado mediante conexión con un ordenador. Aunque no es necesario conectarlo al ordenador, el dispositivo tiene conexión *bluetooth*, WiFi y USB para poder conectarse al PC y manejar archivos o instalar aplicaciones.

Por otra parte, pese a que dispone de tecnología de *hand tracking*, viene con un mando para simular una de las manos del usuario, asimismo, utiliza la tecnología de *eye tracking* para poder interactuar con el entorno a través de la mirada. Del mismo modo, el dispositivo captura 6DoF, tiene un campo de visión de 50° y una resolución de 1.280x960 píxeles por ojo (un total de 2.560x960 píxeles en combinación con los dos ojos).

(a) Vista del casco completo (b) Vista del mando

Figura 2.18: Dispositivo completo de las *Magic Leap One*[19].

Por último, la batería recargable de litio de las *Magic Leap One* permite un uso continuado de 3 horas. Además, tiene un diseño ergonómico mediante el cual se puede utilizar el casco con gafas, ya que hay un compartimento enfrente de los ojos adaptado para dicho uso.

2.3.5 DAQRI Smart Helmet

El *DAQRI Smart Helmet* (figura 2.19) es un HMD enfocado al uso industrial. Está formado por un dispositivo de realidad mixta sin cables integrado en un casco duro (cascos utilizados por ejemplo en la construcción). Ya que está pensado para un uso industrial está dotado de 4 cámaras para poder capturar todo el entorno, es decir, abarcar 360°.

Figura 2.19: Vista del *DAQRI Smart Helmet*[20].

[18]Especificaciones: https://www.businessinsider.com/magic-leap-one
[19]Fuente: https://www.estiloextra.net/magic-leap-one

Con el objetivo de proteger al usuario en entornos industriales peligrosos, el dispositivo cuenta con una cámara termográfica (para poder controlar lugares potencialmente peligrosos debido a su temperatura) y barómetro para poder medir la presión del entorno.

Para el seguimiento, el *DAQRI Smart Helmet* utiliza la tecnología SLAM y posee 6DoF para el movimiento y los giros del usuario. Del mismo modo, ya que el objetivo final es proteger al usuario, el dispositivo cuenta con reconocimiento de órdenes por voz y de control de movimientos de la cabeza para manejar el HMD, de esta forma se pueden tener las manos libres.

Finalmente, cabe recalcar que el casco goza de procesadores y un sistema operativo Android. Para ser totalmente independiente también utiliza unas baterías intercambiables de 5.700 miliamperios por hora y dispone de conectividad WiFi para poder comunicarse por vídeo en tiempo real remotamente con otros usuarios. El casco tiene un peso total de 1 kilo y 500 gramos y su precio actual es de 15.000 dólares.

2.3.6 HP VR1000-127il

Este HMD de la empresa HP salió al mercado en el año 2018. Es un dispositivo dependiente de un ordenador compatible con la plataforma *Windows Mixed Reality*. Este casco se conecta a dicho ordenador a través de una conexión 2 en 1 que combina un HMDI 2.0 y USB 3.0.

Las gafas *HP VR1000-123il* (figura 2.20) utilizan dos controladores inalámbricos (que se conectan mediante *bluetooth*) para controlar el movimiento de las manos. Estos controladores utilizan 2 pilas AA cada uno. El dispositivo tiene un campo de visión de 90° y una resolución de 1.440×1.440 píxeles por ojo (un total de 2.880x1.440 píxeles con ambos ojos), por otro lado, el casco cuenta con 6DoF y la IPD no se puede regular.

Figura 2.20: *HP VR1000-123il* y sus controladores[21].

[20]Fuente: https://www.stereoscape.com/blog/2017/04/25/daqri-smart-helmet
[21]Fuente: https://store.hp.com/HPVR1000-123il

Finalmente, este HMD no tiene un diseño centrado en su uso con gafas y tiene un peso de 834 gramos.

2.3.7 Asus HC102

Este dispositivo (figura 2.21) apareció en el año 2018 de manos de la empresa Asus, al igual que las *HMD Odyssey+* (punto 2.3.3). Es un dispositivo que requiere ser utilizado junto a un ordenador compatible con *Windows Mixed Reality* y está dotado de sensores para calcular la orientación del usuario como giroscopio, acelerómetro, magnetómetro y sensor de proximidad.

Figura 2.21: *Asus HC102* al completo[22].

Este casco goza de un FOV de 105°, una resolución de 1.440x1.440 por ojo (2.880x1440 en total) y cuenta con dos mandos (cada uno de ellos con un sensor 6DoF) que funcionan con 2 pilas del tipo AA, es decir, 4 pilas AA en total.

Finalmente, este HMD permite regular la IPD entre 55mm y 71mm, tiene dos cámaras para hacer un *tracking* de tipo *inside-out* y un peso de 399 gramos.

2.3.8 Acer AH101-D8EY

Este dispositivo de la marca Acer (figura 2.22) fue lanzado al mercado en 2017. Para utilizarlo es necesario un ordenador compatible con *Windows Mixed Reality*. A diferencia de los otros HMD que se utilizan con la plataforma WMR, este dispositivo se conecta al ordenador a través de conexión *bluetooth*.

(a) Vista del casco completo (b) Vista de los controladores

Figura 2.22: Dispositivo *AcerAH101-D8EY*[23].

[22]Fuente: https://www.asus.com/Headset/ASUSHC102/specifications/

El campo de visión del casco es de 100° y respecto a la resolución, es capaz de mostrar 2.880x1.440 píxeles combinando los dos ojos (1.440x1.440 píxeles en cada ojo). La distancia interpupilar no es ajustable y viene configurada por defecto a 62mm.

Por otro lado, ya que se conecta mediante *bluetooth* necesita una fuente de energía de 2 pilas tipo AA. Del mismo modo, el dispositivo viene con dos controladores que igualmente funcionan con 2 pilas del tipo AA cada uno y tienen 6 grados de libertad.

Los sensores que tiene este casco son: acelerómetro, giroscopio, magnetómetro y sensor de proximidad, además, el peso del dispositivo es de 848g.

2.3.9 Lenovo Explorer

La marca Lenovo puso a la venta el dispositivo *Lenovo Explorer* (figura 2.23) en el año 2017. Es un casco de realidad mixta que se utiliza a través de *Windows Mixed Reality*, es decir, depende de un ordenador para ser utilizado. Este HMD se conecta a través de un cable en Y con conexión HDMI y USB 3.0 y goza de conexión *bluetooth*.

Este dispositivo está dotado de sensor de proximidad, giroscopio, acelerómetro y magnetómetro, además, posee dos cámaras para realizar el *tracking inside-out*.

Por otra parte, tiene un FOV de 110° y una resolución con ambos ojos de 2.880x1.440 píxeles (1.440x1.440 con cada ojo). Tiene control de 6 grados de libertad y su IPD no se puede ajustar, viniendo preestablecido con un valor de 62mm. Este dispositivo no tiene tecnología *hand tracking* ni *eye tracking*. Para sustituir el seguimiento de manos se utilizan dos controladores que funcionan con pilas del tipo AA. Su peso es de 380 gramos.

Figura 2.23: *Lenovo Explorer*[24].

2.3.10 Comparación de dispositivos

Las características de los HMD son variadas, desde diferentes tipos de *tracking* para crear un mapa del entorno y conocer la posición y orientación del usuario hasta utilizar controladores o *hand tracking* (incluso ambos a la vez) para controlar las manos del usuario.

En estas comparaciones se excluyen los teléfonos móviles (ya que son muy distintos a todos los demás respecto a características hardware) y el casco *DAQRI Smart Helmet*

[23]Fuente: https://www.acer.com/model/VD.R05AP.002
[24]Fuente: https://www.lenovo.com/Lenovo-Explorer/p/G10NREAG0A2

(punto 2.3.5) ya que este está enfocado a usos industriales y por ello sus componentes son notablemente diferentes a los demás (termómetro, barómetro, control por voz...).

Dispositivo	FOV	Resolución	IPD Ajustable	Independiente
HMD Odyssey+	110°	2.880x1.600	✓	×
HP VR1000-127il	90°	2.880×1.440	×	×
Magic Leap One	50°	1.280 x 960	-	✓
ASUS HC102	105°	2.880×1.440	✓	×
Acer AH101-D8EY	100°	2.880×1.440	×	×
Lenovo Explorer	110°	2.880×1.440	×	×
Hololens 2	43°	4.096×1080	✓	✓

Cuadro 2.2: Comparación del apartado visual de los *HMD*.

Como se puede observar en la tabla 2.2, los dispositivos independientes, es decir, que poseen sus propios procesadores y tarjetas gráficas (no como los otros que utilizan los componentes de un ordenador para funcionar) tienen un FOV o campo de visión reducido respecto a los dispositivos dependientes. Por ejemplo, resaltan los 50° de las *Magic Leap One* frente a los 100° de las gafas Acer, en cambio, en el ámbito de los píxeles en pantalla, sobresalen notablemente las *Hololens 2* con una resolución horizontal muy superior a las demás pese a ser un dispositivo independiente.

Por otro lado, son cada vez más los dispositivos que permiten ajustar la distancia interpupilar para una experiencia del usuario satisfactoria sin fatiga ocular. Aun así, pese a su importancia, dispositivos como las *Lenovo Explorer* no permiten al usuario ajustar este valor. Por el contrario, destaca el sistema de regulación automático de las *Hololens 2*.

Dispositivo	Alimentación del HMD	Alimentación de los controladores
HMD Odyssey+	Cable enchufado al PC	4 Pilas tipo AA
HP VR1000-127il	Cable enchufado al PC	4 Pilas tipo AA
Magic Leap One	Batería recargable (3h de uso)	Batería recargable (6h de uso)
ASUS HC102	Cable enchufado al PC	4 Pilas tipo AA
Acer AH101-D8EY	2 Pilas tipo AA	4 Pilas tipo AA
Lenovo Explorer	Cable enchufado al PC	4 Pilas tipo AA
Hololens 2	Batería recargable (2h - 3h de uso)	No tiene controladores

Cuadro 2.3: Comparación de baterías del casco y los controladores.

Si comparamos la autonomía energética de los dispositivos (tabla 2.3), sobresale que por la parte de los controladores la mayoría utiliza 2 pilas de tipo AA en cada mando, sin embargo, las *Hololens 2* se basan únicamente en su tecnología de *hand tracking* para el control de las manos y las *Magic Leap One* utilizan un único mando recargable para complementar su uso del *hand tracking*.

Respecto a la autonomía del casco, los dispositivos que utilizan la plataforma *Windows Mixed Reality* utilizan la energía del ordenador para funcionar, al contrario lo hacen los

dispositivos independientes que utilizan baterías recargables que permiten de 2 a 3 horas de uso o el dispositivo que utiliza 2 pilas de tipo AA ya que se conecta al ordenador mediante *bluetooth*.

Dispositivo	Peso	Precio ($)	Independiente
HMD Odyssey+	644g	499	×
HP VR1000-127il	834g	400	×
Magic Leap One	316g	2.295	✓
ASUS HC102	399g	492	×
Acer AH101-D8EY	848g	509	×
Lenovo Explorer	380g	200	×
Hololens 2	566g	3.500	✓

Cuadro 2.4: Pesos y precios de los dispositivos.

Por último, en la tabla 2.4 se puede observar la variabilidad entre los pesos de los dispositivos, en ella se incluye de nuevo la información sobre si son independientes o no para comodidad del lector.

Aunque se podría imaginar que los dispositivos independientes deberían ser más pesados frente a los que se utilizan conectados a un ordenador, resalta el peso de las *Magic Leap One* y las *Hololens 2* frente a los demás cascos. También, se aprecia el gran peso en comparación a los otros del dispositivo de HP y de Acer pese a ser un dispositivo que utiliza los recursos de una CPU.

En el ámbito del coste de los HMD, los dos dispositivos que no dependen de un ordenador tienen un precio que sobrepasa los otros dispositivos los cuales tienen un precio cercano entre ellos.

2.4 Usos

Las tecnologías inmersivas están ganando popularidad y haciéndose más accesibles para los consumidores. Esto está provocando que se empiece a usar en sectores como la educación. Además, la realidad mixta está ganando importancia frente a la realidad virtual y la realidad aumentada, ya que utiliza lo bueno de las dos tecnologías, lo cual provoca que se utilice a día de hoy en diversos sectores recogidos en este apartado.

2.4.1 Educación y formación

Las tecnologías inmersivas hacen una gran aportación a la educación en forma de motivación y como forma innovadora de aprender para los estudiantes. Alice Bonasio [28] afirma que la realidad mixta ofrece beneficios en la educación en el proceso cognitivo, sirve como acogida social, mejora el aprendizaje emocional y permite un cambio de comportamiento en los aprendices.

Por un lado, la realidad mixta ofrece un entorno seguro para practicar y perfeccionar habilidades y reduce el cuello de botella debido al exceso de información generando un mayor razonamiento abstracto y pensamiento crítico.

Por otra parte, al aumentar el uso de estas tecnologías en instituciones educativas se está brindando su uso a personas que anteriormente no tenían acceso a ellas. Igualmente, al permitir experiencias colaborativas se mejoran las relaciones sociales entre usuarios.

Las simulaciones permiten a los usuarios practicar situaciones rutinarias o acceder a experiencias que serían inalcanzables en el mundo real. Los estudiantes que aprenden a través de la realidad mixta presentan un mayor nivel de concienciación con el entorno. También, gracias a la realidad mixta se puede realizar una formación de empleados más eficiente.

Muchos tipos de formación pueden ser remplazadas por un entorno en realidad mixta de situaciones reales o instrucciones con el mundo real frente a los empleados. Esto hace que al enfrentarse a estas situaciones reales (no como lo harían sin esta tecnología, solo con casos teóricos) ganen aptitudes de una manera más rápida.

En conclusión, la realidad mixta en el ámbito de la educación reduce la carga cognitiva, aumenta la retención de conocimientos y desencadena la empatía [28].

Desde la perspectiva de las aplicaciones, se pueden encontrar alguna de ellas como por ejemplo, *VR Frog Dissection: Ribbit-ing Discoveries* (figura 2.24) una experiencia de realidad mixta a través de la cual los usuarios pueden aprender cómo se disecciona una rana.

Figura 2.24: *VR Frog Dissection: Ribbit-ing Discoveries*[25].

Otras de las muchas experiencias que nos ofrece la plataforma *Windows Mixed Reality* son *HoloTour* (una combinación de vídeo de 360 grados, sonido espacial y paisajes holográficos) o la aplicación de navegación 3D a través del cuerpo humano llamada *Holo-Human*. Se puede usar esta experiencia de forma colaborativa para descubrir la anatomía de un cuerpo humano (figura 2.25).

[25]VR Frog Dissection: Ribbit-ing Discoveries: https://www.microsoft.com/es-es/p/vr-frog-dissection

Figura 2.25: Uso de la aplicación *Holo-Human*[26].

2.4.2 Industria

Otro ámbito que es potenciado por la realidad mixta es la industria. Esta tecnología beneficia a este sector de las siguientes formas:

- **Proceso de control de calidad más rápido:** Estos largos procesos de control de calidad se ven reducidos notablemente haciendo uso de imágenes, textos o modelos 3D. Esto da lugar a ensamblajes más rápidos y con menos errores.
- **Mantenimientos más rápidos:** Se pueden sustituir manuales de reparación tediosos por simples instrucciones en el entorno de realidad mixta. Otro uso para agilizar el mantenimiento es el que hace, por ejemplo, la empresa de ascensores *ThyssenKrupp*. En ella, sus empleados hacen uso de estos cascos de realidad mixta para realizar llamadas al soporte técnico que les guía con facilidad gracias a poder superponer en tiempo real distintos elementos del ascensor a través de la retransmisión que ofrecen las gafas.
- **Reuniones colaborativas:** Los cascos de realidad mixta aportan utilidad de cara a trabajar de forma colaborativa mediante reuniones a distancia en tiempo real. Se crean hologramas de los participantes en un entorno concreto en el que pueden colaborar en diversos proyectos o participar de forma conjunta en lluvias de ideas.

Todos los puntos anteriores fomentan el crecimiento de la empresa en forma de automatizar procesos, minimizar tiempos y reducir costes, es por esto que el uso de dicha tecnología es necesaria para ser competitivos.

La empresa de diseño, fabricación y venta de aviones civiles Airbus es una de las beneficiadas por la realidad mixta. Esta empresa afirma que el uso de gafas de realidad mixta en su producción aumenta su productividad y su eficacia. Además, utiliza estos HMD para formar a sus empleados (figura 2.26). Del mismo modo, aseguran que la tecnología *hand tracking* de algunas de las gafas aporta seguridad a los empleados gracias a tener las manos libres.

[26] *Holo-Human*: https://www.microsoft.com/es-es/p/holo-human

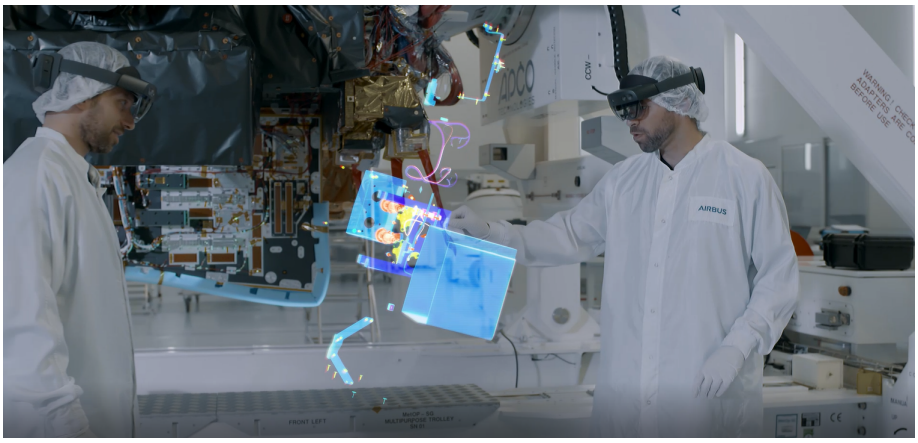


Figura 2.26: Diseñadores de Airbus haciendo uso de la realidad mixta[27].

Por último, otro uso industrial de la realidad mixta es el que se puede aplicar al diseño de apariencia y estética de productos industriales como coches o motocicletas [29]. En este caso, la aplicación *Spacedesign* sustituye las técnicas de prototipado rápido por una experiencia de realidad mixta para realizar esos bocetos en tres dimensiones.

2.4.3 Atención médica

Existen diversos usos de la realidad mixta en la atención médica, desde educación médica hasta propias aplicaciones de atención médica. En primer lugar, la empresa *CAE Healthcare* es una de las líderes en simulaciones tecnológicas y en recursos para mejorar el desempeño en las clínicas. *CAE VimedixAR* es un simulador de ultrasonido diseñado por dicha empresa.

Esta aplicación superpone hologramas de lo que se vería al realizar un ultrasonido. Estos hologramas se superponen sobre un maniquí del mundo físico. De esta forma, los profesionales pueden manipular partes anatómicas (rotándolas y escalándolas).

Figura 2.27: Uso de la aplicación *CAE Lucina*[28].

[27]Fuente: https://news.microsoft.com/es-es/2019/06/18/

Otra aplicación diseñada por esta empresa es *CAE Lucina* (figura 2.27). Se trata de un simulador de partos donde los profesionales pueden realizar la extracción completa del feto en cualquier situación excepcional (estando así preparados para cualquier imprevisto).

Por otro lado, la empresa Pearson ha desarrollado una aplicación llamada *HoloPatient* la cual sirve para formar estudiantes de enfermería. El programa se basa en una experiencia que permite aprender sin necesidad de actores o pacientes que se necesitarían sin usar esta tecnología. *HoloPatient* salió al mercado en abril del año 2018.

Existen otros proyectos como el que viene de la mano de SphereGen en colaboración con *St. George's University*, bautizado con el nombre de *Learning Heart* (figura 2.28). Esta aplicación permite a sus usuarios aprender sobre el corazón (de manera individual o colaborativa) viéndolo desde cualquier ángulo y posición.

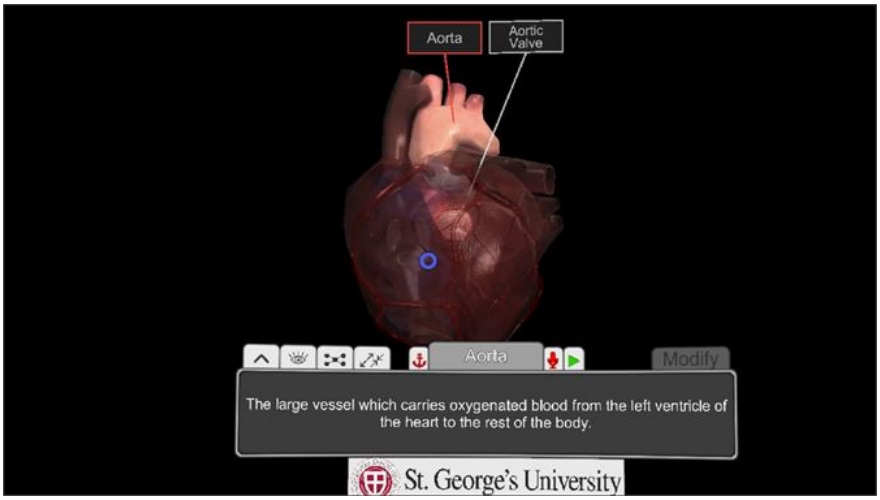


Figura 2.28: *Learning Heart* en ejecución[29].

En cuanto a la radiología, *DICOM Director* aporta una gran ayuda a estos expertos permitiéndoles ver y analizar todo tipo de escaneos en 3D. Los radiólogos y doctores pueden ver estas radiografías haciendo uso de sus cascos de realidad mixta de Windows. Igualmente, la aplicación genera un modelo 3D de esas imágenes (figura 2.29) los cuales se pueden rotar y analizar. De esta manera, *DICOM Director* se utiliza también como herramienta de comunicación entre doctores para compartir estos modelos.

[28]Fuente: https://blogs.windows.com/windowsexperience/2018/03/08/
[29]Fuente: https://www.microsoft.com/en-us/p/learning-heart

Figura 2.29: Usuario rotando el modelo 3D de una radiografía en*DICOM Director*[28].

Para finalizar, la empresa *Visual 3D Medical Science and Technology Development CO. LLC* ha desarrollado un producto que ha sido usado en China para realizar 200 operaciones (de rodilla, cadera y columna vertebral) gracias a la realidad mixta. Incluso se está utilizando para preparar a los cirujanos antes de entrar a la sala de operaciones, reproduciendo la operación a través de las gafas.

2.4.4 Videojuegos

El impacto de la realidad mixta tampoco podía quedar de lado dentro del mundo de los videojuegos. Pese al gran potencial que esta tecnología presenta para la industria del videojuego, es un terreno todavía muy poco explorado en comparación a los videojuegos en Realidad Aumentada. En AR existen videojuegos de éxito como *Pokémon GO* o el juego basado en AR con marcadores *Invizimals*.

Los videojuegos que a día de hoy existen de realidad mixta están únicamente pensados para dispositivos móviles (Android e iOS) y basados en ubicación por GPS. No suelen ser videojuegos cerrados sino más bien prototipos o pruebas de concepto con escasa funcionalidad.

Se pueden destacar los siguientes títulos dentro de este ámbito:

- **SpecTrek:** El objetivo principal de este videojuego (figura 2.30) es que el usuario realice actividad física. Para ello, plantea una serie de lugares que visitar en los cuales existen varios de enemigos que habrá que derrotar. Cuenta con tres modos de juego distintos, según el tiempo que el usuario desee estar realizando actividad física. El videojuego utiliza la ubicación GPS del jugador a medida que este se mueve para medir la distancia hacia estos enemigos, y una vez se encuentre delante de uno de ellos, tendrá que encontrarlo con la cámara del dispositivo móvil para eliminarlo.

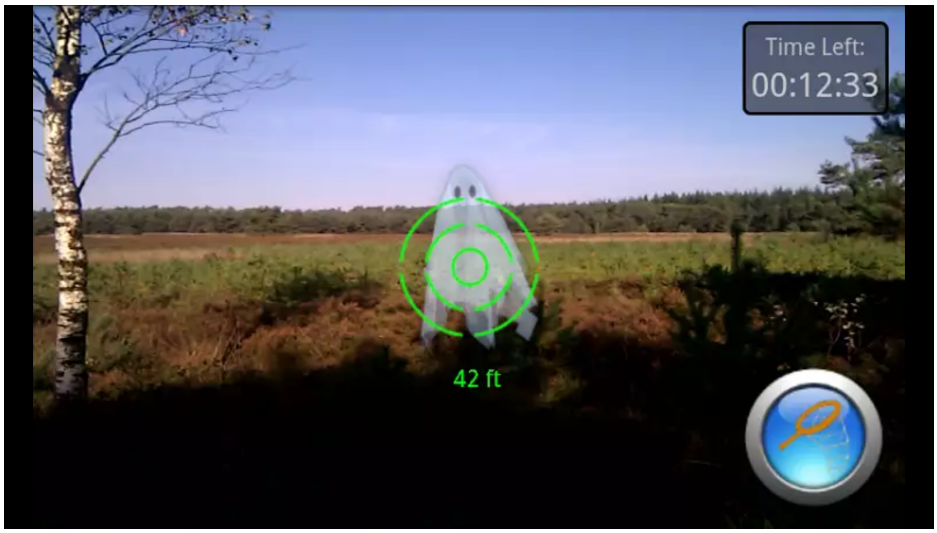


Figura 2.30: Captura de pantalla de *SpecTrek*[30].

- **Ingress:** Dentro de este videojuego, el objetivo será moverse a través del mundo real para conocer distintas obras de arte urbanas así como monumentos o lugares de interés dentro de una ciudad. Para ello, el videojuego sitúa dentro del mundo una serie de portales virtuales que se podrán visitar. Es considerado videojuego ya que mientras más lugares de interés se visiten, más aumentará el nivel del jugador. También dispone de una serie de "minijuegos" a realizar por el jugador y facilitar que este pueda subir de nivel.
- **Gbanga Zooh:** Gbanga está definido como una experiencia de "Gaming social". En este, las tres actividades a realizar son explorar el entorno, recolectar objetos y recursos del mismo y finalmente intercambiarlos con otros jugadores que hayan seguido el mismo proceso.
- **Can You See Me Now?:** Este videojuego es una imitación del tradicional juego de persecución en el que un jugador tiene que alcanzar a otro y este último huir de él. En este caso, ambos jugadores se encuentran en una ciudad y disponen de un equipo GPS así como un Walkie-Talkie. El equipo GPS transmite constantemente su posición a un ordenador y el Walkie-Talkie sirve para comunicarse con la persona que se encuentra en este ordenador. Esta última persona, que no se encuentra en la ciudad (una por jugador o una por equipo), tiene que guiar a los jugadores para que cumplan su objetivo simplemente viendo sus posiciones en el ordenador y comunicándose mediante voz.

2.5 APIs

A la hora de generar experiencias de realidad mixta existen diversas APIs (del inglés *Application Programming Interface*) que pese a no estar enfocadas a crear entornos de realidad mixta, pueden ser utilizadas para ello.

[30]Fuente: https://play.google.com/store/apps/details?id=com.spectrekking

En esta sección se tratan dos APIs destacables a la hora de generar dichos entornos: ARcore y Vuforia.

2.5.1 ARCore

ARCore es un kit de desarrollo de Google [30] enfocado a construir experiencias de realidad aumentada en dispositivos Android. Tiene distintas APIs que permiten utilizar el teléfono para entender el mundo real e interactuar con él. ARCore utiliza tres pilares fundamentales para integrar el contenido virtual en el mundo físico:

1) ***Motion tracking*:** ARCore utiliza unos **puntos característicos** *(features en inglés* como elementos de referencia para detectar si se produce un cambio en la posición. Esta información se combina con los datos de la velocidad, orientación y fuerzas gravitacionales del dispositivo para estimar la posición y orientación de la cámara.

2) ***Environmental understanding*:** Esta tecnología utiliza concentraciones de estos puntos característicos mencionados anteriormente uniéndolos de manera horizontal o vertical para formar planos. Esta información se puede utilizar para colocar elementos en superficies lisas.

3) ***Light estimation*:** Para poder dar una iluminación a nuestros objetos virtuales acorde al mundo real, la plataforma detecta la información de la luz del entorno y realiza una estimación de la intensidad y el color.

A la hora de colocar un elemento virtual en el mundo físico, cuando el usuario interactúa con la pantalla en un punto concreto, ARCore comprueba si existe algún plano o punto característico en ese punto y coloca el objeto ahí utilizando un **ancla** (posiciones en el mundo físico para mantener el control de los objetos en el tiempo).

Por ejemplo, si colocamos un cubo en una puerta y nos desplazamos, al volver a apuntar a la puerta podremos ver que el cubo sigue ahí (gracias a las anclas). No solo se pueden utilizar estas anclas en un mismo dispositivo, sino que ARCore ofrece la *ARCore Cloud Anchor API* mediante la cual se guarda la posición de las anclas en la nube, de tal forma que varios móviles pueden observar el mismo objeto virtual colocado por una persona. Esto permite compartir los entornos con varios usuarios simultáneamente.

Recientemente ARCore ha introducido la *ARCore Depth API* lo cual supone un gran avance a la hora de calcular profundidades en el mundo virtual, ya que funciona con una única cámara (antes se requerían dispositivos con grandes funcionalidades hardware). La profundidad es calculada mediante un algoritmo que toma múltiples fotos de distintos ángulos y estima la distancia a cada píxel. Esto permite a los desarrolladores crear mapas de profundidad para generar un efecto de oclusión (sección 2.2.2).

Aunque esta API no es dependiente de cámaras o sensores específicos, la experiencia mejorará notablemente al utilizar dispositivos con un mejor hardware como sensores ToF, dado que permitirá activar la funcionalidad de la oclusión dinámica, es decir, poder hacer el efecto de oclusión con objetos en movimiento.

Actualmente ARCore se puede utilizar en la plataforma de desarrollo Unity, en Unreal Engine o a través del entorno de desarrollo integrado Android Studio. Aunque ARCore es gratuito, no está soportado por todos los dispositivos móviles o tabletas, existe una lista de dispositivos soportados[31].

Si se quiere desarrollar algo para iOS, la alternativa a ARCore se llama ARKit [31]. Este kit de desarrollo de Apple ofrece opciones muy similares a ARCore como detección del entorno, cálculo de la profundidad o sesiones colaborativas, todo para dispositivos con iOS.

2.5.2 Vuforia

Vuforia Engine [32] es una plataforma enfocada al desarrollo de aplicaciones de realidad aumentada. Se puede utilizar en todos los dispositivos móviles y tablets. Además, permite a los desarrolladores añadir funcionalidades de visión por computador. Con Vuforia se pueden crear aplicaciones para dispositivos Android, iOS y para dispositivos Windows.

Esta librería ofrece las siguientes herramientas:

- ***Area Target Generator***: Aplicación de escritorio que toma como entrada el escáner de un modelo 3D del entorno y genera un *area target* (en el ámbito de Vuforia se refiere a un entorno donde se controla el *tracking* de objetos y se permite colocar elementos).
- ***Area Targets Test App***: Se trata de una aplicación para poder comprobar la calidad del *area target* generado al hacer el escáner de forma rápida, es decir, un área de pruebas de entornos 3D generados.
- ***Model Target Generator***: El *Model Target Generator* es otra aplicación de escritorio mediante la cual se puede generar un *area target* tomando como entrada un modelo 3D ya existente (en lugar de realizar un escáner). Esta aplicación soporta extensiones de archivo como *.obj*, *.fbx* o *.stl* entre otros.
- ***Model Target Test Application***: En este caso se trata de una aplicación para móviles Android que se utiliza para evaluar *model targets* (modelos que se utilizan para reconocer y hacer el *tracking* de determinados elementos del mundo real gracias a su forma).
- ***Vuforia Object Scanner***: Aplicación de Android que se utiliza para escanear elementos del mundo real y generar archivos compatibles con Vuforia para definir un *model target*.
- ***Target Manager***: Herramienta web que permite al desarrollador crear y manipular sus *targets*.

Respecto al *tracking*, Vuforia calcula la posición del usuario en función de detalles en el entorno tomados por la cámara (como los puntos característicos de ARCore). Asimismo, utiliza la Unidad de Medición Inercial para controlar los 6DoF del dispositivo.

[31]Dispositivos que soportan ARCore: https://developers.google.com/ar/discover/supported-devices

A diferencia de ARCore, para utilizar este kit de desarrollo es necesario pagar alguno de sus planes.

Resumen

En este capítulo se han tratado los conceptos de las tres realidades que abarcan las XR. Dichos conceptos han servido para poder distinguir las distintas realidades entre ellas. Además se han contado las tecnologías que hacen falta para sus usos. Dentro de estas tecnologías cabe destacar la importancia del *tracking* a través de distintas técnicas.

Una vez conocidos lo componentes necesarios para su funcionamiento se puede hablar de los distintos dispositivos que permiten su uso, desde teléfonos móviles hasta HMDs donde se ha podido ver que estos tienen distintos tipos de conectividad, pueden utilizar mandos o no e incluso estar dotados de tecnología *eye tracking*.

Por último, la realidad mixta potencia distintos campos como la educación, la atención médica, la industria o los videojuegos. Para poder desarrollar aplicaciones de realidad aumentada y realidad mixta destacan distintas APIs como ARCore, ARKit o Vuforia.

En el siguiente capítulo se podrán ver como se han desarrollado las distintas aplicaciones haciendo uso de la API ARcore para alcanzar el primero de los subobjetivos.

Capítulo 3

Oclusión en tiempo real con ARCore y Unity

La oclusión en realidad mixta se encarga de que los objetos del mundo real sean capaces de superponerse y ocluir los objetos del entorno virtual, según el punto en el que ambos se encuentren.

Dada que la idea del trabajo es la generación de un sistema de rutas en realidad mixta, una de las partes más importantes dentro de esta es la oclusión de objetos. Por ello, este capítulo se centra en la forma de generar oclusión en tiempo real en un dispositivo móvil. Las tecnologías seleccionadas para dicho fin son Unity como plataforma de desarrollo y ARCore con su integración para dicho editor.

Es conveniente también diferenciar entre la generación de dicha oclusión en tiempo real o de un entorno anteriormente escaneado. Por ejemplo, como se puede apreciar en la figura 3.1, se genera la oclusión con la pared lateral derecha, pero es en un lugar escaneado con anterioridad.

Figura 3.1: Augmented Reality Indoor Navigation. [33]

La oclusión se puede conseguir de diversas formas, como se explica en la sección 2.2.2. Si los dispositivos en la actualidad integrasen sensores *ToF*, se facilitaría la generación de la oclusión. Sin embargo, dado que dichos sensores tiene un alto coste y los dispositivos no los tienen incorporados, se hará uso de las llamadas *nubes de puntos* (3.1).

El uso de esta *nube de puntos* viene dado ya que la oclusión en tiempo real se genera con una serie de puntos en lugares clave de la imagen (*features*), los cuales toman la posición de los vértices de los objetos del mundo real.

El objetivo final es la generación de dicha nube de puntos y, por consiguiente, una oclusión del entorno que funcione de manera fluida en un dispositivo móvil.

3.1 Nube de puntos

3.1.1 Definición

La definición de nube de puntos depende del ámbito en la que esta se aplique. Es por ello, que encontramos varias definiciones dentro de incluso un mismo proyecto para esta [34, 35, 36]. Sin embargo, todas coinciden en el mismo aspecto, y es que *una nube de puntos es un producto resultante del escaneo láser o por fotogrametría (en tiempo real o en diferido). Este está expresado por millones de puntos posicionados tridimensionalmente en el espacio (con coordenadas X, Y, Z) formando una entidad física y destinados a representar virtualmente la superficie externa de esta*. Esta nube de puntos, además de contener la información de posiciones de los mismos también puede contener información añadida como el color de cada punto, su reflexión o la iluminación que recibe o el tamaño.

3.1.2 Generación

Pese a todas estas formas de generación de nubes de puntos que se han visto en la introducción (escaneo láser, fotogrametría o sensores de luz infrarroja ToF), en este caso se profundizará en la llamada *Structure for motion* (figura 3.2). Esta entra dentro de la fotogrametría y es la forma más común de uso y la utilizada por ARCore.

En concreto, ARCore y ARKit (la versión de ARCore para iOS) utilizan la técnica llamada *visual-inertial odometry* [37]. Este proceso combina información del dispositivo como puede ser el giróscopo o el acelerómetro con análisis de visión por computador de la escena visible a la cámara [38].

Es usado en una gran variedad de aplicaciones, sobre todo robóticas, como por ejemplo en la exploración de Marte con el *Rovers* [39].

La tecnología de ARCore utiliza la cámara del dispositivo para identificar puntos interesantes, llamados *features*, y rastrea su movimiento a lo largo del tiempo. Esto último lo realiza con una combinación del movimiento de esos puntos y la lectura de los sensores de inercia del dispositivos (tanto la posición como la orientación en el espacio). Cabe destacar que estas *features* no tienen por qué verse finalmente representadas en la nube

de puntos. Por ejemplo, se puede identificar una *feature* pero no ser capaz de continuar su movimiento, por lo que no sería añadida.

Como extras dentro de estas mediciones, ARCore también detecta superficies planas (tanto horizontales como verticales) y puede realizar una estimación de la iluminación del entorno. Estas funcionalidades combinadas son capaces de generar la nube de puntos. [40]

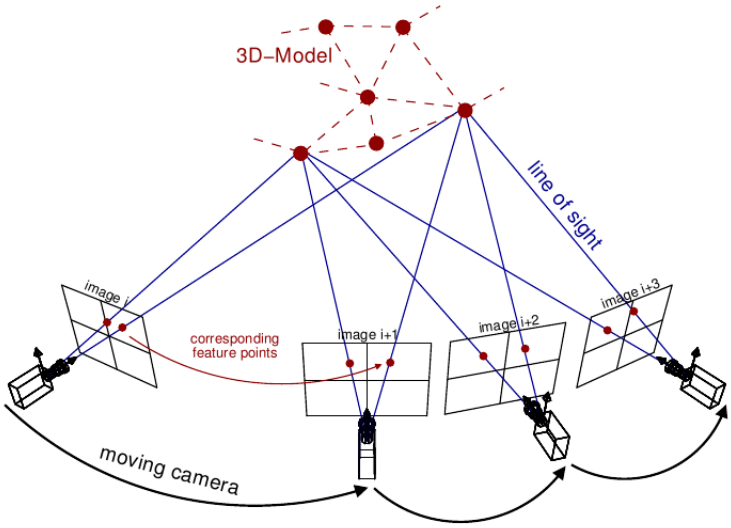


Figura 3.2: Representación de *Structure From Motion (SfM).* [32]

Como muestra la figura 3.3, se seguirán unas reglas básicas trigonométricas para resolver la posición de un único punto en el mundo real.

Supongamos, por ejemplo, que se desea conocer la posición del punto A de la figura 3.3, y nuestro dispositivo se encuentra en la posición B de la misma figura. En este punto, sólo conocemos la rotación de nuestro dispositivo pero no la posición. Esta no es determinante aunque se utilizará como referencia para el cálculo de la siguiente posición (por ejemplo, se supone que el punto B es el centro de coordenadas para calcular los futuros desplazamientos). Ahora realizamos un movimiento desde la posición B a la posición C. En este punto, gracias a las mediciones de los sensores del dispositivo vistos en la sección 2.2.1, conocemos la nueva posición y rotación (C, γ), y por tanto la distancia que une a estas dos posiciones ($\overline{BC}$) que en la figura se representa por a, así como el punto B.
Una vez se tenga tanto la posición B y C, y como sabemos la rotación del dispositivo mediante el giróscopo del mismo, tenemos los ángulos β, γ. Teniendo estos dos ángulos y dado que los ángulos de cualquier triángulo han de formar 180°, también tenemos α:

$$\alpha = 180 - \gamma - \beta$$

[32]Fuente: https://n9.cl/8up8

Finalmente, conocidos todos estos datos y mediante el Teorema del seno, se puede conocer tanto la distancia desde $\overline{AB}$ como $\overline{AC}$, y por consiguiente el punto A que se situará dentro de la aplicación:

$$\frac{Sen(\gamma)}{\overline{AB}} = \frac{Sen(\alpha)}{\overline{CB}} \Rightarrow \overline{AB}$$

$$\frac{Sen(\beta)}{\overline{AC}} = \frac{Sen(\alpha)}{\overline{CB}} \Rightarrow \overline{AC}$$

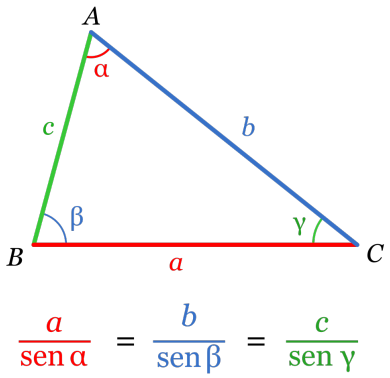


Figura 3.3: Principio trigonométrico para calcular distancias. [33]

Una vez obtengamos el punto A, sólo habrá que repetir el proceso tantas veces como *"features"* haya dentro de la imagen y en consecuencia construir la nube de puntos.

Cuando toda la información de estos puntos esté disponible, existen varias maneras de tratarlos. Para nuestro estudio realizamos distintas pruebas aunque existen otras muchas formas de dar uso a esta nube de puntos. Por ejemplo, la creación de una malla tridimensional para generar oclusión o simplemente obtener la información de dichos puntos para construir un objeto en tres dimensiones dentro de la aplicación. Esta nube de puntos la facilita automáticamente ARCore.

3.2 Renderizado y shaders

Una vez disponemos de la nube de puntos, para generar el efecto de oclusión se hará uso del renderizado digital proporcionado por Unity. Para ello se han utilizado los llamados *shaders*.

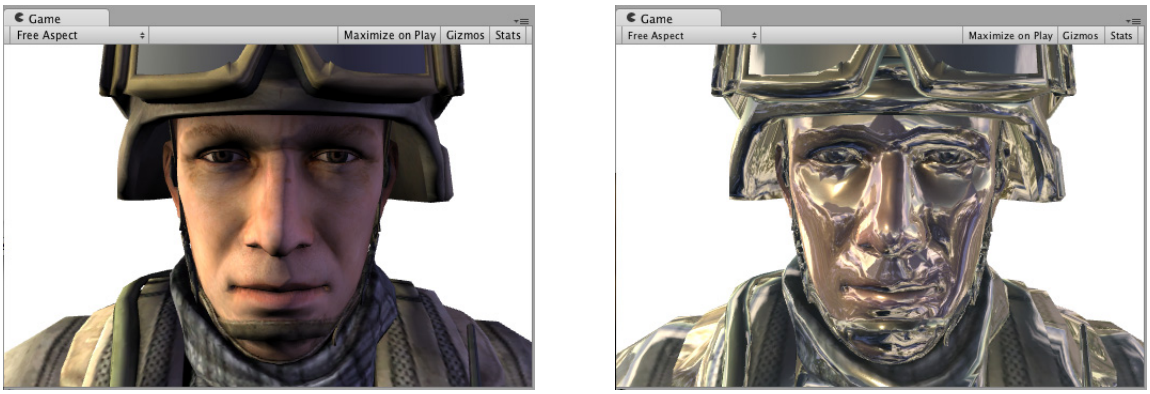
Los *shaders* son pequeños *scripts* diseñados para ser ejecutados en algún estado del procesador gráfico que contienen cálculos matemáticos y algoritmos para calcular el color de cada píxel que se renderiza. Estos cálculos están basados en la entrada de luz y la configuración del material. Normalmente se utilizan para controlar el sombreado y la iluminación. Además, estos programas pueden utilizarse también (aunque no es lo más común) para computación en la GPU [41, 42].

[33] Fuente: https://es.wikipedia.org/wiki/Teorema_de_los_senos

Por ejemplo, en la figura 3.4 se puede ver el cambio en la imagen del soldado al aplicar un *shader* llamado *cubemap reflection*. Este *shader* aplica una reflexión en el color lo cual hace que se vea un solado con un aspecto metálico.

Existen tres maneras distintas de desarrollar *shaders* en Unity en función de lo que se quiera conseguir[34]. Si el objetivo es trabajar con las luces y las sombras los ***surface shaders*** son la mejor opción. Este tipo concreto permite crear *shaders* complejos de una manera compacta (en pocas líneas de código). Además, son una capa de abstracción de alto nivel para trabajar con la tubería de iluminación de Unity.

Para escribirlos se utilizan los lenguajes de *shading* Cg o HLSL, envueltos en una capa de **ShaderLab**.



(a) Imagen base. (b) Imagen con el *shader* aplicado.

Figura 3.4: Ejemplo de *shader* cubemap reflection[35].

Por otro lado, si lo que se busca es un *shader* que no interactúe con la iluminación, se utilizan los ***vertex and fragment shaders***. También se utilizan si se busca crear efectos que no se pueden alcanzar con los *surface shaders*. En este caso son *scripts* de mayor tamaño, este es el precio a pagar para gozar de mayor flexibilidad a la hora de crear distintos efectos. Al igual que los *surface shaders* estos se escriben en los lenguajes de *shading* Cg o HLSL, de nuevo envueltos en una capa de abstracción proporcionada por **ShaderLab**.

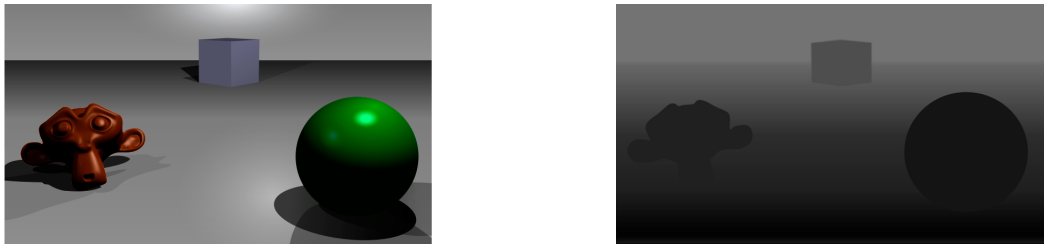
Por último, los ***fixed function shaders*** se utilizan para crear efectos muy simples. Estos son *shaders* de legado, es decir, es una opción obsoleta. En este caso se escriben completamente utilizando el lenguaje de *shading* de Unity llamado ***ShaderLab***.

Es importante conocer este lenguaje ya que aunque se utilice cualquiera de las dos opciones anteriores (*surface shaders* y *vertex and fragment shaders*) ambas están envueltas en una capa escrita en *ShaderLab*.

Por otra parte, al estar trabajando en este caso con la información de profundidad de los objetos es necesario conocer el concepto de *depth buffer* o *Z-Buffer*.

[34] Unity *shaders*: https://docs.unity3d.com/Manual/ShadersOverview.html
[35] *Cubemap reflection*: https://docs.unity3d.com/Manual/SL-SurfaceShaderExamples.html

El *Z-Buffer* consiste en un almacenamiento de memoria que guarda la coordenada *Z* de cada píxel de una imagen. Este valor es la profundidad de cada píxel que se obtiene desde el plano de proyección[36]. En concreto, en la figura se puede observar un ejemplo de como funcionaría el *Z-buffer*.



(a) Escena simple en tres dimensiones. (b) Representación del *Z-Buffer*.

Figura 3.5: Ejemplo de resultado del *Z-Buffer*[37]

A la hora de representar el *Z-Buffer* el color concreto de los objetos no es importante, ya que en este caso los píxeles no codifican color, sino información de profundidad. Se puede observar que los objetos cercanos aparecen de un color más oscuro que los lejanos.

3.2.1 Shader de profundidad

Una vez obtenida la nube de puntos de ARCore en Unity y antes de crear el efecto de oclusión se pasó a comprobar su funcionamiento a la hora de ser renderizada. Se quería comprobar si ARCore colocaba los puntos respetando la información de profundidad obtenida o, en cambio, variaba los tamaños de los puntos para dar la sensación de que estaban más alejados. Para comprobar esto se creó un *shader* de profundidad.

Este *shader* (del tipo *surface shader*) se centra en utilizar la textura de profundidad que genera la cámara de Unity[38]. Esta textura es creada a partir de la información obtenida del *Z-buffer*. Con esa información, se le aplica a los objetos un material con una variable "*DepthLevel*" la cual contiene el valor de profundidad obtenido en la textura generada por la cámara.

Al tener la información de profundidad se escribe el *shader* de profundidad. En él, se modifica el valor de transparencia de los objetos en función del valor de *"DepthLevel"*.

En primer lugar se hizo una prueba del *shader* con un cubo que podía acercarse y alejarse con unos botones, como se puede observar en la figura 3.6. El valor de transparencia del cubo se ve afectado en función de su profundidad respecto al dispositivo.

[36] *Depth buffer:* https://docs.unity3d.com/Manual/Glossary.html#depthbuffer
[37] *Ejemplo de Z-Buffer:* https://en.wikipedia.org/wiki/Z-buffering
[38] Textura de profundidad de Unity: https://docs.unity3d.com/Manual/SL-CameraDepthTexture.html

Figura 3.6: *Shader* de profundidad aplicado a un cubo con profundidad variable.

Dicho cubo era un elemento fijado a la cámara, es decir, seguía su movimiento. Además, el *shader* se le aplicaba a todos los elementos de la escena que poseían el material que tenía la variable de profundidad mencionada anteriormente. Aunque lo más común es aplicar el *shader* a un material concreto, como en este caso se utilizaba la textura de profundidad de la cámara, estaba aplicado a ella.

En resumen, la cámara generaba una textura de profundidad en cada *frame* donde se modificaba el valor de transparencia de los objetos que estuviesen dotados de un material en concreto, generando mayor transparencia a mayor profundidad.

Como esta prueba había sido exitosa y el objetivo final del *shader* era comprobar el funcionamiento del renderizado de la nube de puntos, se intentó aplicar este efecto de post-procesado a dichos puntos (sección 3.1) generados por ARCore.

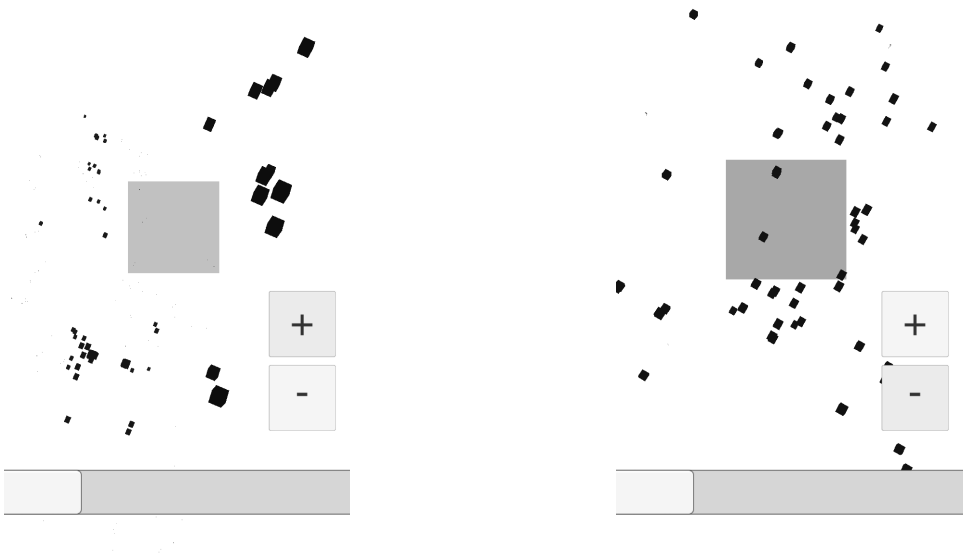


Figura 3.7: *Shader* de profundidad modificando el cubo pero no la nube de puntos.

Se llegó a la conclusión de que este *shader*, al ser un efecto que modificaba la textura de la cámara, no se aplicaba a la nube de puntos obteniendo los resultados de la figura 3.7 donde se aprecia que se está modificando la transparencia del cubo pero no de los puntos generados por ARCore.

3.2.2 Shader de oclusión

El *shader* de oclusión es el efecto necesario para obtener la experiencia de realidad mixta en este trabajo. Con este efecto se busca evitar el pintado de ciertos objetos del entorno aunque su representación física continuase en la escena.

Para ello, se utiliza la nube de puntos para generar una malla de oclusión (explicada en la sección 2.2.2) y se le aplica dicho *shader* a esta malla para que actúe como plano que ocluya los objetos virtuales.

Para conseguir este efecto se han utilizado conceptos relacionados con el *culling*[39] que ofrece *ShaderLab*. El *culling* es una optimización utilizada para no renderizar polígonos lejanos al observador. También se ha hecho uso de conceptos relacionados con el *depth testing*. Esta característica de renderizado se asegura de que sean únicamente los elementos cercanos a la cámara los que se pinten en la escena.

Es importante tener en cuenta que este *shader* (del tipo *surface shader*) va a ser aplicado sobre los materiales de la malla generada por la nube de puntos.

Para escribir este *shader* es necesario crear un *shader* estándar de Unity. A continuación se van a tratar distintas características que hay que escribir en el *shader*. Estas características relacionadas con el *culling* y el *depth testing* para que el *shader* de oclusión funcione son:

- ***ZWrite***: Se debe crear un campo *ZWrite* en el *shader*. Este valor tiene que estar activo para que los píxeles del objeto se escriban en el *Z-buffer* (sección 3.2). Esto es necesario ya que se va a trabajar con el renderizado en función de la profundidad de los elementos.
- ***ZTest***: Esta variable puede tomar distintos valores. Debe tener el valor *LEqual* para que los elementos se pinten unos encima de otros en función de la distancia entre ellos (oculta los objetos que están por detrás del que porta el *shader*).
- ***ColorMask***: Ya que lo que se busca es que no se vean los planos que hagan el efecto de oclusión (se busca que se pinten como transparentes) es necesario su valor a 0. De este modo el color del objeto no se pinta en pantalla.

Para la prueba de este *shader* en primer lugar se escanea el área para que ARCore genere la nube de puntos y una dichos puntos entre sí para generar planos con distintos ángulos en los que detecta objetos. Para visualizar mejor estas mallas de oclusión se les ha aplicado un contorno negro (figura 3.8).

[39] *Unity ShaderLab culling*https://docs.unity3d.com/Manual/SL-CullAndDepth.html

Figura 3.8: Generación de malla de ARCore con contorno.

Es necesario escanear el área varias veces para que ARCore genere todos los planos del área en el que se quiere crear el efecto de oclusión. Una vez hecho esto, se coloca un objeto (en este caso un cubo de color rojo) en uno de los planos.

Figura 3.9: Efecto de oclusión generado con el *shader* de oclusión.

En la figura 3.9 se puede apreciar como el proceso de aplicar el *shader* de oclusión a los planos genera un buen efecto de oclusión. Esto ocurre ya que se coloca el cubo en un

plano a la altura del suelo y, por encima de este, existe el plano generado debido a la mesa que hace de malla de oclusión para el elemento virtual.

Después de obtener estos resultados en elementos pequeños y cercanos al dispositivo, se pasó a comprobar la eficacia del *shader* en elementos más grandes y más distantes del dispositivo.

Como se puede ver en las pruebas que se realizaron en una esquina y en un objeto grande y detallado como una silla (figura 3.10), la generación de mallas de ARCore al menos en dispositivos con cámara monocular se ajusta poco a la realidad. Es decir, los planos generados no son precisos. Esto provoca que el efecto de oclusión no sea preciso en estos casos en los que los elementos son grandes y distantes al dispositivo o grandes y con geometrías detalladas.

(a) Esquina de una pared. (b) Contorno de una silla.

Figura 3.10: Pruebas del *shader* de oclusión en bordes y objetos con geometrías detalladas.

Dado que el problema en este caso era que los planos generados por ARCore uniendo la nube de puntos eran imprecisos y el *shader* de oclusión funcionaba correctamente, se pasó a dejar de unir los puntos de la nube para generar planos y hacer una malla de oclusión, a simplemente generar la nube y colocar primitivas (las cuales tenían el *shader* de oclusión aplicado) en dichos puntos para generar el efecto de oclusión.

3.3 Pruebas con nubes de puntos

3.3.1 Introducción

Teóricamente las nubes de puntos deberían funcionar sin demasiada complicación debido a la sencillez de su cálculo, pero en la práctica encontramos varios fallos que impiden su correcto funcionamiento.

Existen varias maneras de utilizar la nube de puntos para distintos fines. Para el propósito de este estudio, se hace uso de esta para generar la oclusión de los objetos virtuales con los que se encuentran en el mundo real utilizando el *shader de oclusión*.

En este caso, se dará uso de las posiciones generadas por ARCore para crear objetos propios para la oclusión. Este será el resultado de analizar tanto el rendimiento de la nube de puntos (tanto en tiempo como en consumo de recursos) como el porcentaje de acierto de la oclusión resultante.

3.3.2 Aplicaciones de pruebas

Debido al extenso número de factores a considerar a la hora de realizar las pruebas, han sido implementadas tres aplicaciones para automatizarlas y extraer los datos de ellas.

El resultado obtenido no se limita a una única ejecución de la prueba para evitar producir sesgo en los datos. Cada prueba es realizada desde distintas perspectivas (mínimo 2) y así poder considerar varios factores. Estos pueden ser, por ejemplo, la luz natural del entorno o los objetos que en este se encuentren.

- **Aplicación ARCore**: Esta aplicación realiza todas las comprobaciones necesarias para valorar el estado de la oclusión de un conjunto de elementos determinados así como del rendimiento de los mismos.

  El uso ideal se conseguiría mediante un archivo de vídeo sobre el que se iterarían todas las pruebas. Sin embargo ARCore no utiliza únicamente los datos extraídos de la entrada de vídeo para calcular la profundidad del entorno. A estos datos les añade los obtenidos de los distintos sensores como pueden ser el giróscopo o el acelerómetros, para medir rotaciones y posiciones respectivamente como se pudo ver en la sección 2.2.

  La aplicación cuenta con un *Asset* [40]para facilitar la medición de los datos llamado *Graphy* [43]. Este proporciona la medida de los *Frames por segundo (FPS)* de la aplicación, el uso de la memoria y el consumo de la CPU, así como todas las características internas del dispositivo.

En la interfaz de la figura 3.11 se pueden encontrar todas las opciones necesarias para generar las pruebas:

- **1. Start Point Cloud**: Sirve para generar la nube de puntos de manera que esta se pueda ver desde la aplicación.
- **2. Test Visualization**: Siempre y cuando la nube de puntos esté activada, esta opción ejecuta la batería de pruebas para comprobar la oclusión.
- **3. Start Time Test**: Activa y desactiva la prueba para comprobar el tiempo de generación de la nube de puntos.
- **4. Chg. Max. Ptos**: Cambia el número de puntos máximos que puede tener la nube de puntos.
- **5. Chg. Ptos. *Frame***: Modifica el número de features en la nube máximo que se pueden considerar por *frame*.
- **6. Chg. Forma**: Intercambia las distintas formas para probar las variantes de la oclusión.
- **7. Capturar pantalla**: Siempre que la nube de puntos esté activada, genera una captura del estado de la pantalla.
- **8. Medidor de pruebas**: Utilizado para visualizar todos los estados de las interacciones anteriores.
- **9. Datos Graphy**: Datos de *Graphy* como el rendimiento en *FPS* o la memoria utilizada.

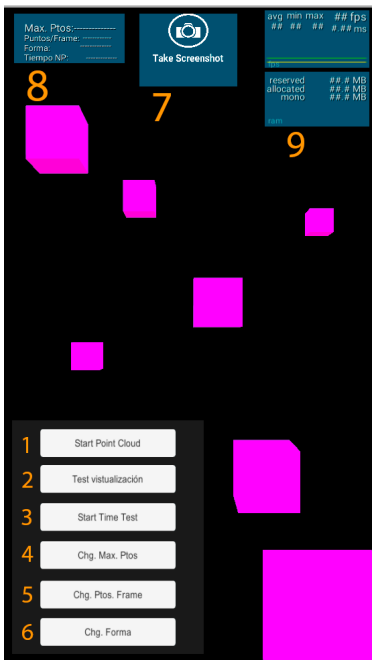


Figura 3.11: Vista principal de la aplicación para la generación de pruebas.

[40]Definición de Asset: https://docs.unity3d.com/Manual/AssetWorkflow.html

- **Test Visualization**: Debido a que esta prueba engloba todos los apartados dentro de la aplicación anterior, se detallará en más profundidad su uso.

  Para el uso de este test se ha de activar la nube de puntos, por lo que se pulsará el botón *1. Start Point Cloud*.

  Una vez esté completa la nube de puntos deseada para generar la oclusión, habrá que pulsar el botón *2. Test Visualization* para comenzar la prueba. Cabe destacar en este punto que para el correcto funcionamiento de la aplicación, el dispositivo ha de encontrarse totalmente estático. Este proceso se explicará más en detalle en el apartado *Errores de las pruebas* 3.3.4.

  Pulsado este botón, la aplicación ejecutará el script C# que controla toda la prueba. Este realizará la siguiente serie de acciones:

  - **Obtención de la nube de puntos**: Gracias a la creación del script *PointCloudEditor*, es posible acceder a la nube de puntos en cualquier momento o estado. Esta se guardará en una lista y desactivará la generación para que no interfiera más en las pruebas.
  - **Ocultación del HUD**: Para que las pruebas se realicen con la máxima fiabilidad posible, se procede a ocultar todo el HUD de la aplicación.

    Todo el proceso descrito a continuación se engloba dentro de una *corrutina*. Estas impiden que una acción tenga que realizarse forzosamente dentro de un único frame [41].
  - **Creación de los puntos**: En cada una de las posiciones de la nube de puntos se generará un objeto mediante la función *Instantiate* de Unity, que clona un objeto y lo crea en la escena [42]. Este se almacenará dentro de una lista diferente a la de la nube de puntos para poder utilizarlos en el futuro.
  - **Bucle principal**: Dentro de este apartado, se realizará la comprobación por cada forma y por cada tamaño. Por defecto, la aplicación cuenta con cinco formas distintas (Esfera, cubo, quad, cilindro y cápsula) y con cinco tamaños distintos (desde 0,01 hasta 0,05). Existen varios bucles anidados que recorren todas estas posibilidades. En primer lugar, un bucle recorre todas las formas y a cada uno de los puntos anteriormente creados, les aplica la forma que le corresponda. En segundo lugar, dentro de ese mismo bucle, existe otro que recorre todos los tamaños posibles y aplica a cada objeto el tamaño que en este caso tenga que comprobar.
  - **Captura**: Una vez se tenga un tamaño concreto para todas las posiciones de la nube de puntos, se esperará a que el *frame* termine para poder realizar la captura de pantalla. Esto se debe a que al estar dentro de la corrutina, separada del bucle principal de Unity, la captura de pantalla se puede realizar sin que se haya pintado el *frame* anterior. Así se asegura que el paso de *renderizado* de Unity ya se haya realizado. Acto seguido, realizamos la captura y

[41] Unity Coroutines: https://docs.unity3d.com/Manual/Coroutines.html
[42] Unity Instantiate: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html

volvemos a realizar el mismo paso de esperar al final del *frame* para guardarla correctamente.

En la figura 3.12 se puede apreciar que, si no se esperase al final del *frame*, la aplicación podría realizar una captura antes del renderizado de la escena, justo después del update principal de Unity.

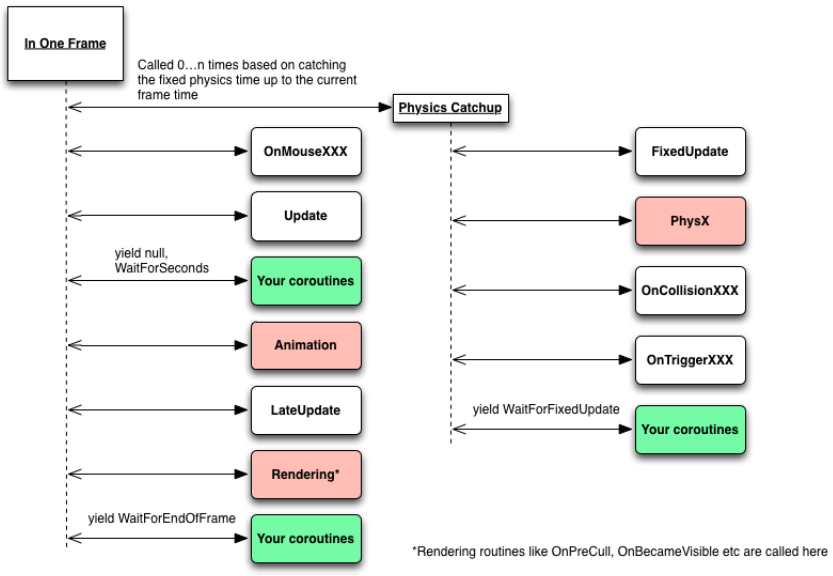


Figura 3.12: Corrutinas C++ en el bucle principal de Unity. [43]

  - **Imágenes vacías**: Finalmente, se crearán dos imágenes particulares. Una de ellas, es una imagen sin ningún tipo de elemento virtual, únicamente con la imagen de la cámara. La otra consiste en un fondo negro con los elementos virtuales. Estas dos imágenes en el futuro servirán para crear la que llamaremos imagen "Desired", que representará el estado óptimo esperado por la oclusión.

- **Aplicación de comparación total de imágenes**: En este caso la aplicación realiza toda la comprobación de imágenes necesaria para la prueba *Test Visualization* de la aplicación anterior. Consiste en un script escrito en Python.

  Esta busca dentro de un directorio dado todas las carpetas que este contenga. En nuestro ejemplo, ya que se realizaron 5 pruebas, se generaron 5 carpetas llamadas *VistaN*, donde N es el numero de la prueba. Acto seguido, buscará una subcarpeta llamada *Images*, donde se encuentran todas las imágenes generadas por la aplicación de Unity.

  En el mismo directorio *VistaN*, también encontramos la imagen *Desired* antes mencionada así como las dos originales que dan forma a esta. Una vez encontradas todas las imágenes, se comparan una a una con *Desired*, generando en una nueva carpeta llamada *Structural Similarity* una imagen por cada imagen tomada, del resultado

[43] Fuente: https://unitygem.wordpress.com/coroutines/

de comparación entre *Desired* y las originales. Esto sólo nos servirá como referencia para validar que dicha comparación es correcta.

A su vez, en el directorio raíz donde se ejecuta la aplicación también se generan una serie de archivos CSV, uno por cada carpeta *VistaN*, llamado *ResultsVistaN*. En ellos encontramos todos los datos de cada una de las pruebas. La forma del objeto, su tamaño y el porcentaje de acierto que ha obtenido. Para finalizar, se genera un último archivo CSV, llamado *Final_Results*. En este encontramos la media de todas y cada una de las pruebas unidas. Es decir, la media del resultado de cada una de las formas y tamaños para todas las pruebas *VistaN*.

Figura 3.13: Estructura de las carpetas para comparar imágenes.

- **Aplicación de comparación parcial de imágenes**: Esta aplicación es una variable de la anterior sobre comparativa de imágenes en Python. En este caso, la aplicación no compara las imágenes por completo, sino que únicamente compara las partes de esta donde se encuentra un elemento virtual. Para ello, la aplicación utiliza como referencia la imagen con el fondo negro que contiene únicamente los elementos virtuales. En ella comprueba en qué píxeles concretos se encuentra un elemento virtual (ya que el fondo es negro, los píxeles elegidos serán los que tengan un color distinto a negro). Una vez escogido uno de los píxeles, comprueba en la imagen *Desired* si el color del píxel en la posición de la foto de referencia coincide con el color del píxel de la imagen generada por la aplicación de pruebas. Si estos colores coinciden, se considerará acierto. Esto se repetirá por todos y cada uno de los píxeles que contenga la imagen.

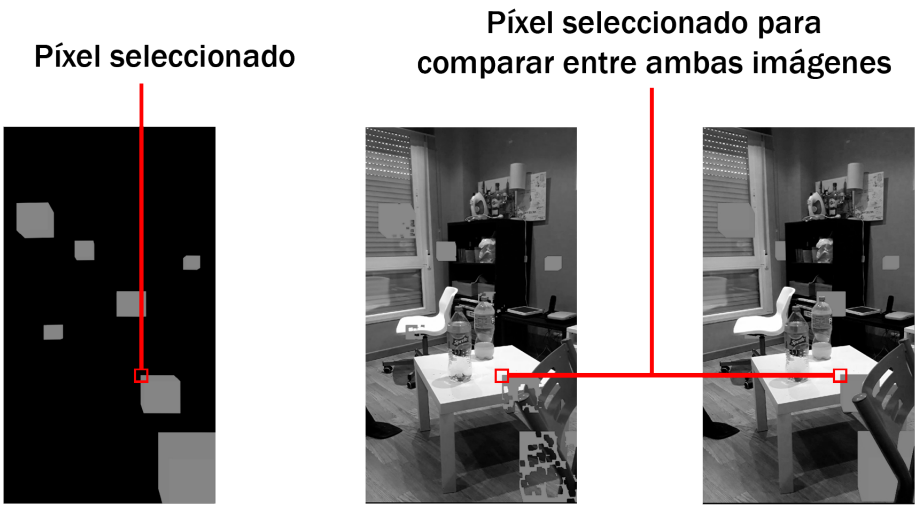


Figura 3.14: Comparación de imágenes por elementos virtuales.

Finalmente, se contabilizarán todos los aciertos y los fallos y se generarán los archivos CSV con los resultados obtenidos.

La diferencia entre estas dos aplicaciones es que mientras esta segunda nos muestra el porcentaje de acierto total relativo a los elementos virtuales, la anterior nos lo muestra en referencia a la imagen completa.

3.3.3 Pruebas

Para poder obtener una serie de conclusiones acertadas, en primer lugar se han definido tres pruebas distintas.

En el primer escenario que se tendrá en cuenta se resumen todas las variables a considerar (Forma de los puntos, máximo numero de puntos y máximo número de *features* a añadir por *frame*). Con ello se obtendrá la media de rendimiento de cada una de estas situaciones. Encontramos 5 tipos distintos de formas a considerar (Esfera, Cubo, Quad, Cilindro y Cápsula). Con cada una de estas formas se generarán pruebas combinando los distintos valores del máximo de puntos posibles que puede contener nuestra escena (1000, 10000 y 20000) y el número máximo de *features* que podemos añadir por cada *frame* (1, 10 o 100). Este último representa el número máximo de *features* que la aplicación puede considerar por cada vuelta en el bucle principal. Estos valores, tanto para el número máximo de puntos como para el máximo de puntos añadidos por *frame*, no son aleatorios.

En el caso del máximo número de puntos posibles, 1000 es un valor que permanece neutro mientras que 20.000 es un valor demasiado alto que en muy pocas circunstancias se llega a alcanzar. En lo referente al máximo número de *features* por *frame*, se ha de tener en cuenta que el número aquí introducido influirá directamente en el tiempo que tarde en realizarse el bucle principal de la aplicación. A partir de 100 la aplicación tiene un rendimiento totalmente nulo y 1 es el valor por defecto y el mínimo que se puede aplicar. Esto

[43]Fuente: https://unitygem.wordpress.com/coroutines/

deja un total de 45 pruebas a realizar por cada una de las perspectivas que se quieran tener en cuenta.

Una vez estén todos estos datos encapsulados, se procede a realizar la media de rendimiento por cada una de las formas que hemos utilizado. Así encontramos una primera aproximación a los datos que queremos analizaremos más adelante.

Forma	Media de rendimiento (FPS)
Esfera	17.88
Cubo	15.72
Quad	15.27
Cilindro	11.33
Cápsula	9.55

Cuadro 3.1: Media de *Frames* Por Segundo (FPS) para cada una de las formas.

Con esta primera aproximación en la que se puede ver lo que ha costado todo el proceso, desde la generación de la nube de puntos hasta el renderizado de la misma, fácilmente se aprecia que la forma con la que se obtiene mejor rendimiento (de media) es la esfera.

Este rendimiento está influido por la forma de la figura ya que en esta varía el número de polígonos a la hora de ser renderizada, por lo que el resultado final sólo es orientativo para proseguir con las pruebas.

Sabiendo que la esfera es de promedio la forma con mejores resultados, se utilizará para realizar la siguiente prueba. Esta consiste en probar únicamente el tiempo en el que una nube de puntos tarda en estabilizarse. Para ello, se tendrá en cuenta el instante en el que comenzamos la prueba y el instante en el que la nube de puntos ha alcanzado su estabilidad.

Esta estabilidad se da por asumida cuando se han generado el máximo número de puntos establecido. Se realizará únicamente con las medidas de máximo de puntos que se pueden añadir a la escena y con el máximo de *features* que se pueden añadir por *frame*. Las mediciones se llevarán a cabo dos veces para obtener la media de ambas pruebas y poder tener en cuenta otros factores como la iluminación del entorno o el número de objetos que en él se encuentren.

Máx. Puntos	*Features* por *frame*	Media de tiempo (s)
1.000	1	15.03
1.000	10	7.75
1.000	100	5.97
10.000	1	18.98
10.000	10	11.63
10.000	100	7.84
20.000	1	22.21
20.000	10	16.19
20.000	100	11.31

Cuadro 3.2: Tiempo empleado por la aplicación para generar una nube de puntos estable.

Realizada esta prueba se puede apreciar que la combinación que toma menos tiempo para realizar la nube de puntos es la que cuenta con un número máximo de 1.000 puntos y un número de 100 *features* por *frame*. Esta medida era lógica y esperada, ya que 1.000 es el número menor de puntos y que generando 100 *features* por *frame* tenemos un crecimiento bastante rápido de la nube. Evidentemente esta última medida afecta notoriamente al rendimiento de la aplicación. Este resultado en el futuro nos ayudará a decidir cual es la mejor manera de generar la oclusión dentro de la aplicación.

La siguiente prueba a realizar consistirá en unir todas las formas mencionadas y distintos tamaños para la oclusión. Para ello utilizaremos la prueba *Test Visualization* de la aplicación explicada en profundidad en el apartado anterior.

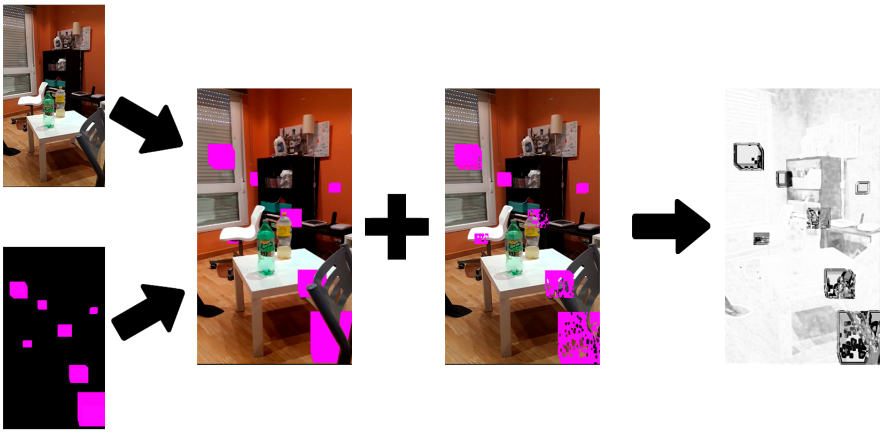


Figura 3.15: *Workflow* de comparación de imágenes.

En la siguiente tabla mostramos cual ha sido el mejor tamaño para cada una de las formas en base a su acierto medio.

Forma	Tamaño	Acierto
Esfera	0.05	85,92 %
Cubo	0.04	85,79 %
Quad	0.04	86,85 %
Cilindro	0.01	86.65 %
Cápsula	0.05	88.22 %

Cuadro 3.3: Distintas formas para general la oclusión y tamaños con sus correspondientes porcentajes de acierto.

Sin embargo esta prueba ha de ser meramente orientativa y exclusiva para este trabajo. Esto se debe a que a la hora de la comparación se tiene en cuenta toda la imagen y no sólo la parte de los elementos virtuales que en ella se encuentran. Por lo tanto, si se añadiese un nuevo elemento virtual a las imágenes anteriores, el porcentaje de acierto bajaría considerablemente. Por ello, se realizó una prueba más exhaustiva para determinar únicamente el acierto en los objetos virtuales. En esta se tienen en cuenta exclusivamente las secciones de la imagen donde existe un objeto virtual (gracias a la imagen de referencia con fondo negro y únicamente con los elementos virtuales en ella). Una vez sabemos dónde están todos ellos situados, se guardan las posiciones de dichos píxeles y se comparan los de la imagen *Desired* con los de imagen obtenida de la oclusión. Si los píxeles de la posición *(x,y)* de la imagen *Desired* coinciden con los *(x, y)* de la imagen de oclusión, se contabilizará como un acierto. Este proceso se realizará por cada uno de los píxeles cuyo color sea distinto a negro de la imagen con objetos únicamente virtuales.

Forma	Acierto medio
Esfera	56,10 %
Cubo	52,28 %
Quad	55,76 %
Cilindro	50,09 %
Cápsula	51,90 %

Cuadro 3.4: Acierto medio por cada forma considerando únicamente los elementos virtuales.

Este resultado indica que la forma *Esfera* es la que mejor porcentaje de acierto tiene de entre todas las pruebas realizadas. Si se observa el cuadro 3.1, también podemos apreciar que es la que mejor rendimiento obtiene.

Si se buscase un termino medio que aunase el resultado de la oclusión, con el número de puntos generados y el tiempo que tardan en generarse estos, así como la media de rendimiento de la aplicación, basándonos en los tres cuadros anteriores podríamos apreciar que:

- La esfera tiene una tasa de acierto media muy baja aunque aceptable en comparación con las otras formas(56,10 %)
- La esfera tiene la mejor media de rendimiento (17.88 FPS)
- Una media razonable de puntos a añadir es de 10.000 puntos con 10 puntos añadidos por *frame*, ya que tiene un valor intermedio en tiempo de creación (11.63s) y no afecta extraordinariamente al rendimiento.

Sin embargo, como se puede apreciar en las imágenes generadas por la oclusión, este porcentaje de acierto es demasiado bajo y no llega a generar una oclusión aceptable.

3.3.4 Errores de las pruebas

Para la generación de estas pruebas en una primera instancia la aplicación se ejecutó en un dispositivo móvil con ARCore, pero presentó varios errores durante el proceso:

- **Tiempo de prueba**: Debido al gran número de pruebas que se debían realizar, la aplicación tomaba demasiado tiempo en llevarlas todas a cabo, pese a toda la optimización que se añadió dentro de la misma.
- **Batería del dispositivo**: La mayoría de dispositivos tiene un tiempo máximo de espera para que la pantalla se apague automáticamente. En los dispositivos probados el tiempo máximo era de 10 minutos, por lo que si la aplicación tardaba más en realizar las pruebas, estas quedaban invalidadas. Por ello hubo que añadir a la aplicación la funcionalidad de que mantuviese siempre encendida la pantalla del dispositivo. Esto, unido a la gran cantidad de recursos que necesitaba la propia prueba, hacía que la batería del teléfono se agotase al poco tiempo.
- **Temperatura del dispositivo**: Junto con lo anterior mencionado, al hacer tanto uso del dispositivo este sufría varias subidas de temperatura. Se redujo considerablemente tanto el número como tamaño de las pruebas.
- **Movimiento del dispositivo**: Para que el comparador de imágenes funcionase correctamente en Python, estas debían de ser lo más parecidas posible. Tener el dispositivo en la mano era completamente inviable para la realización de las pruebas.

Finalmente, la solución fue evitar que el dispositivo móvil realizase la batería de pruebas y delegase en otro equipo más potente. Gracias a la aplicación *ARCore Instant Preview* y *Unity* se logró este objetivo. Con ella, el móvil se conectaba a un ordenador portátil mediante USB. Este, con el proyecto de Unity abierto, hacía todo el trabajo de procesar la información mientras que el móvil lo único que realizaba era la emisión de los datos como posición o rotación del mismo. Con esto las pruebas que tardaban de promedio 1 hora o más en realizarse pasaron a una media de 5 segundos en estar completas, obteniendo el mismo resultado visual que se esperaba con el dispositivo móvil.

3.4 Optimización

El trabajo realizado por la aplicación consiste en generar los puntos de la nube oclusión y renderizarlos, así como medir todos los datos y cambiar entre las distintas pruebas. Esto hace que la aplicación no sea completamente estable o que en general los fotogramas por segundo sean demasiado bajos.

Para optimizar todo este proceso los recursos que ARCore proporcionaba tienen que ser modificados. Ya que los *scripts* que ARCore ofrece no se pueden modificar directamente en el editor de Unity para realizar todas las pruebas, se generó uno nuevo, llamado *Point Cloud Editor*. Este *script* permite controlar de manera pública todos los parámetros necesarios para la generación de la nube de puntos.

Ahora que existe un control sobre esta nube, se puede decidir cómo se generan los objetos dentro de la misma. Se crea una *pool* de puntos para agilizar todo el proceso de la creación de la nube, cosa que ARCore no hace de serie al generar en cada bucle de la aplicación una malla distinta. *Object pool* es el nombre que se le da a un patrón de diseño. Este actúa de contenedor y tiene una lista de objetos los cuales están listos para ser usados. Una vez usados, estos objetos se desactivan aunque se mantienen dentro de la *pool* para poder ser reutilizados.

Esta *pool*, al comenzar la aplicación, genera *Max Point Count* puntos, que es el número máximo de puntos del que va a disponer la nube de puntos y que está de manera pública en el editor. Una vez generados los puntos (todos desactivados), estos se activarán o desactivarán a medida que la nube de puntos vaya creciendo o decreciendo. Si se desean añadir más puntos, la aplicación generará de nuevo los necesarios. Si fuesen menos, borrará los que no se necesitasen para evitar el excesivo consumo de memoria.

Ahora, en lugar de generar los puntos en cada vuelta bucle de aplicación, simplemente será necesario que se activen o desactiven y ponerlos en su posición correcta.

El siguiente *script* que requería una optimización fue el que realiza todas las pruebas de oclusión, llamado *Visualization Test*. Este, al realizar un número bastante elevado de pruebas, debe ser lo más eficiente posible para que el tiempo de realización de estas no fuese excesivo.

Para ello el bucle principal de la aplicación se transformó en una corrutina que controlaba la prueba. En una primera aproximación, contaba con tres bucles anidados. El primero, controlaba el número de formas que había que probar, el intermedio todos los tamaños para dichas formas y finalmente el último recorría todos los puntos de la nube, instanciaba el objeto con la forma dada y el tamaño dado. Una vez estaban todos, realizaba la captura de pantalla pertinente, destruía todos los objetos y cambiaba el tamaño para volver a repetir el proceso con todas las formas y tamaños. Tampoco se dejaba de construir la nube de puntos de ARCore, por lo que en segundo plano esta seguía generando objetos a los que no se daba uso.

Evidentemente este procedimiento era costoso y una primera aproximación para comprobar que, con pocos puntos, era viable realizar la prueba. Una vez crecía el tamaño de la

prueba, la aplicación era incapaz de finalizar su ciclo de vida.

El siguiente paso fue en generar una nueva *pool* de objetos. Esta vez se instancia un objeto nada mas comenzar la corrutina por cada uno de los puntos de la nube, con una forma y un tamaño sin relevancia, y en la posición de cada punto. La nube de puntos que genera ARCore en este punto también se deja de usar y se elimina, ya que ahora la tenemos guardada para la prueba.

En lugar de anidar los anteriores tres bucles, ahora existen cuatro bucles. Lo que en un principio parece contraproducente, resulta ser más eficiente que los tres bucles anteriores. En este nuevo estado, existe un bucle exterior, cuyo cometido es simplemente recorrer todas las formas que podemos intercambiar. Dentro de este, existen otros dos bucles. El primero recorre toda la nube de puntos y cambia el *mesh* de cada uno de los objetos por el de la forma actual que se tiene que probar. El segundo, recorre todos los tamaños que se quieren probar. Este segundo a su vez incorpora otro bucle más, que de nuevo recorre toda la nube de puntos y simplemente cambia el tamaño de cada uno por el tamaño actual para la prueba. Una vez hecho todo esto, se genera una captura de pantalla y se vuelve a repetir todo el proceso para los tamaños y formas restantes.

Se comprueba que esta forma es más eficiente ya que el acto de asignar un nuevo *mesh* a cada forma y a la vez cambiar el tamaño de esta por cada punto, por cada forma y por cada tamaño es mucho más costoso que, en primer lugar, cambiar la forma una única vez y después por cada forma y por cada punto cambiar el tamaño.

3.5 Resumen y problemas de ARCore en Unity

Pese a la publicidad de ARCore y de su tecnología, una vez comprobada esta se puede asegurar que las opciones que ofrecen están todavía muy lejos del efecto deseado y que contienen varios problemas fundamentales.

En varias de las imágenes mostradas por la empresa se puede apreciar como se realiza una perfecta oclusión de los objetos virtuales junto a los objetos del mundo real, incluso generando mapas de calor para conocer con total seguridad la profundidad de la escena que se ve en la figura 3.16.

Sin embargo, tras probar toda la tecnología, sabemos que los *features* mencionados en capítulos anteriores se centran en encontrar los bordes de los objetos reales para generar la nube de puntos. Pese a ello, para crear los mapas de calor o una nube de puntos óptima también sería necesario que dichas *features* se generasen en superficies planas.

Una vez se conocen todos estos detalles, se puede apreciar a simple vista que los vídeos de las campañas de ARCore han sido modificados después de ser grabados para generar estos efectos. Por ejemplo, en la figura 3.16 el dedo pulgar de la persona que tiene el dispositivo no proyecta ningún tipo de sombra sobre este.

Figura 3.16: Supuesto mapa de calor generado con ARCore.[44]

Pese a conocer que esta nube de puntos se genera principalmente con las *features* en el contorno de los objetos, si se aumenta considerablemente el número de puntos que se pueden generar por cada frame se consigue una mejor oclusión. Esto sin embargo lleva a una caída bastante considerable del rendimiento de la aplicación así como a un consumo desmesurado de la batería del dispositivo, y por consiguiente también aumenta su temperatura.

A día de hoy realizar una oclusión con un nivel de detalle como el mostrado en los ejemplos de ARCore, desde un dispositivo móvil, es prácticamente imposible. Nos encontramos totalmente limitados por la tecnología de los dispositivos actuales.

Pese a todo esto, ARCore si que ha dado un gran paso dentro de la generación de oclusión en tiempo real. Su tecnología no precisa tener varias cámaras integradas ni sensores ToF para realizar un escaneo de la profundidad. En el futuro, cuando toda la tecnología de dispositivos móviles mejore, se podrá realizar oclusión en tiempo real sin ningún problema, aunque de momento no serviría para realizar fotogrametría en tiempo real.

Resumen

En este capítulo se ha explicado el funcionamiento a fondo de la nube de puntos generada por ARCore para poder ser utilizada en las aplicaciones de Unity.

Por otra parte, una vez que se ha comprendido el funcionamiento de la nube de puntos, se pueden distinguir dos tipos de *shaders*, el de profundidad y el de oclusión. Ambos *shaders* buscan utilizar la nube de puntos de algún modo para generar un efecto de oclusión.

[44]Fuente: https://www.youtube.com/watch?v=1q0-jdknbTs

Por último, se han realizado distintas pruebas con dicha nube de cara a medir la eficiencia y la precisión del efecto de oclusión generado con uno de los *shaders*.

En el próximo capítulo se profundizará en el funcionamiento de distintas tecnologías de sistema de búsqueda de rutas y de cómo se pueden relacionar con ARCore.

Capítulo 4

Mapas

4.1 Introducción

Para poder ofrecer una correcta experiencia de realidad mixta a la hora de realizar la navegación, es necesario conocer la ubicación del usuario. De este modo se puede saber que elementos digitales hay que colocar en el mundo. Además, es necesario conocer la información de la ruta como maniobras y direcciones para poder guiar al usuario.

Existen distintos sistemas de búsqueda de rutas actualmente donde cada cual aporta funcionalidades variadas. Estas tecnologías están relacionadas con la búsqueda del camino más corto entre dos puntos, dicha búsqueda está centrada mayoritariamente en el **Algoritmo de Dijsktra**.

Estas tecnologías ofrecen distintas posibilidades a la hora de utilizarlas. Se puede consultar una ruta desde un punto de origen A hasta un punto de destino B así como calcular el tiempo de dicho trayecto. En el marco de este proyecto es importante la funcionalidad que brindan estos sistemas de búsqueda haciendo uso del GPS (sección 2.2.1).

La ubicación del GPS se obtiene mediante un proceso lento debido a que se calcula mediante la recepción de datos de satélites los cuales dan vueltas a la Tierra. Es por ello que apareció el conocido como **A-GPS** del inglés *Assisted GPS*.

La información de ubicación obtenida gracias al A-GPS[45] es más rápida ya que se envía una señal a un servidor externo con la identificación de la antena obteniendo como respuesta los satélites ubicados encima del usuario. El inconveniente del A-GPS es la necesidad de poseer una tarifa de datos. Aun así, aumenta considerablemente la velocidad a la hora de iniciar una navegación.

[45]Fuente: https://www.xataka.com/moviles/que-es-el-a-gps

4.2 Servicios de mapas en la nube

4.2.1 Google Maps

La empresa Google ha creado una plataforma llamada *Google Maps Platform* [44], esta plataforma ofrece 3 servicios principales:

- ***Maps***: Permite crear experiencias simples y personalizadas de mapas estáticos y dinámicos, imágenes de *Street View* y vistas en 360°.
- ***Routes***: Este servicio permite a los usuarios conocer la mejor forma de ir de un punto a otro a través de rutas, ya sea en transporte público, en bici, en coche o a pie. Además, calcula la duración del trayecto y su distancia.
- ***Places***: Permite a los usuarios conocer información sobre más de 100 millones de ubicaciones. Asimismo, permite conocer la ubicación del dispositivo sin utilizar el GPS (a través de las torres de telefonía y nodos WiFi).

Para poder hacer uso de esta plataforma, es necesario añadir una cuenta de financiación ya que todos sus planes son de pago. *Google Maps Platform* se puede utilizar en Android, iOS e incluso en un entorno web.

4.2.2 HERE

HERE es una aplicación de la mano de *HERE Technologies*, esta aplicación posee distintos SDK para desarrolladores. Pese a ser una aplicación gratuita, existen distintos planes de pago que añaden funcionalidades extra. Las funcionalidades que aporta HERE son las siguientes:

- **Instrucciones en tiempo real:** Aporta señales visuales y auditivas para la navegación en tiempo real, además, transmite información sobre señalización y carriles.
- **Modo *Offline***: Permite a los usuarios la búsqueda de lugares así como la navegación sin necesidad de conexión a internet.
- **Gran disponibilidad:** HERE está disponible en 190 países y 60 idiomas.
- **Visualización del terreno:** Esta funcionalidad añade distintos modos de visualización del entorno: peatón, tráfico, terreno...
- **Plantillas de interfaz de usuario:** Añade una forma rápida de modificar colores, tamaños y cambios en la lógica a través de plantillas.
- **Personalización del mapa:** Personalización del mapa desde cambios de color de los iconos hasta destacar elementos importantes. Además, permite editar propiedades de los elementos como carreteras o edificios.
- **Obtención de información:** Gracias a esta funcionalidad se puede obtener información detallada de las carreteras como señales, límites de velocidad...

Esta aplicación se puede utilizar en móviles (Android e iOS), a través de Javascript o con una REST API.

4.2.3 Waze

Waze es una aplicación desarrollada por Waze Mobile que además de poder ser utilizada por los usuarios finales, aporta funcionalidades a los desarrolladores para crear sistemas de búsqueda de rutas.

Aunque Waze ofrece distintos SDK, en este caso el relevante es ***Waze Transport SDK***. Este SDK de pago ofrece lo siguiente:

- **Progreso y puntos de la ruta:** Esta funcionalidad permite conocer el tiempo estimado de llegada al punto de destino.
- **Navegación:** Aporta a los conductores la funcionalidad de calcular rutas de forma rápida evitando incidencias en carreteras.
- **Obtención de información:** Esta función del SDK permite obtener la información de las rutas.

Waze está enfocado a ser utilizado únicamente en dispositivos Android e iOS.

4.2.4 Navmii

Navmii es otro sistema de búsqueda de rutas que ofrece SDKs para acceder a distintos mapas y a un sistema de navegación. Los requerimientos técnicos de Navmii están enfocados para ser usados en dispositivos Android e iOS. El *Navigation SDK* ofrece las siguientes posibilidades:

- ***Routing***: Permite calcular la ruta entre dos puntos, aporta instrucciones para seguir la ruta y permite descargar los mapas de más de 190 países.
- ***Navigation***: Esta funcionalidad añade comandos de voz a las instrucciones en tiempo real para guiar al usuario en cada paso, muestra el tráfico en tiempo real de las carreteras y permite su uso sin conexión.
- ***Places and Coordinates***: Esta característica del SDK transforma las coordenadas en direcciones y viceversa, es decir, da nombre a unas coordenadas. Permite buscar calles o lugares por nombre.

Navmii se puede utilizar gratuitamente, sin embargo, existen distintos planes para potenciar alguna de sus características.

4.2.5 Mapbox

Mapbox [45] es una herramienta de código abierto de localización para móviles y entornos web. Mediante Mapbox se pueden crear aplicaciones con funciones como mapas, navegación o búsquedas.

Las distintas funciones que ofrece Mapbox son:

- ***Maps***: API que ofrece la visualización de mapas dinámicos y personalizables.

- ***Navigation***: Motor de búsqueda de rutas preciso con actualizaciones en tiempo real de accidentes, dependiente de si el usuario va caminando, en bicicleta o en coche. Esta característica de Mapbox permite la navegación por voz y la navegación sin conexión a internet (a tavés de mapas ya descargados).
- ***Atlas***: *Atlas* es una plataforma de desarrollo de aplicaciones de Mapbox en la nube, permite al usuario guardar los datos de su aplicación en la nube (para poder trabajar en ella desde cualquier lado).
- ***Search***: Esta herramienta ofrece la posibilidad de dar un contexto al usuario final de la aplicación cambiando sus coordenadas GPS por el nombre de su ciudad o localización. Además, permite colocar etiquetas con nombres en el mapa.
- ***Studio***: Herramienta web que permite crear un estilo propio de mapas.
- ***Vision***: Esta herramienta utiliza la cámara del móvil para funcionar como un copiloto al conducir un coche controlado por inteligencia artificial, este SDK aporta funciones como alertar al conductor cuando no cumple la distancia de seguridad o cuando sobrepasa la velocidad permitida.
- ***Data***: Permite obtener un gran conjunto de datos para analizar desde información sobre las fronteras hasta movimiento de coches y tráfico.

Aunque Mapbox es una herramienta de código abierto, los servicios mencionados anteriormente funcionan bajo demanda. Estas funciones tienen un plan gratuito limitado que se puede ampliar pagando. Por ejemplo, se puede aumentar el número de usuarios activos del servicio *Maps* de los 25.000 del plan gratuito a 125.000 comprando el primer plan de pago.

4.3 Mapbox para búsqueda de rutas

Como se tratado en el punto 4.2.5, Mapbox ofrece distintas funciones. En el caso de la búsqueda de rutas se utiliza la función *Navigation*. Esta función aporta de forma muy detallada información sobre las rutas. Debido a esto y a su facilidad de uso (no requiere facturación) se ha elegido este servicio para este trabajo.

La función *Navigation*[46] ofrece distintas APIs:

- ***Mapbox Directions API***: Permite calcular rutas óptimas conduciendo, andando y en bicicleta teniendo en cuenta el tráfico. Además aporta instrucciones de voz paso a paso.
- ***Mapbox Matrix API***: Esta API es utilizada para calcular el tiempo de viaje entre puntos. Se genera una matriz con las distancias entre todos los puntos dados y se escoge la distancia o la duración de trayecto más cortos entre puntos.
- ***Mapbox Optimization API***: Aporta la posibilidad de obtener una optimización de la duración de una ruta dadas unas coordenadas.
- ***Mapbox Map Macthing API***: Con esta API se generan los caminos que se van a visualizar en el mapa.

En este caso la API que es interesante es *Mapbox Directions API*. Se utiliza para obtener la información de ruta entre dos puntos. Estos dos puntos A y B mencionados anteriormente se pueden elegir de diferentes formas. La funcionalidad *search* ofrece la posibilidad de introducir el nombre de una dirección y transformarla a coordenadas de latitud y longitud. Además, se puede introducir directamente coordenadas de latitud y longitud para definir dichos puntos. Del mismo modo, se puede utilizar la ubicación detectada por el GPS para definir uno de los puntos (indicando que se quiere usar como origen o destino las coordenadas actuales del usuario).

Cuando esta API recibe dos puntos A y B para crear una ruta entre ellos, disecciona la ruta en partes más pequeñas de cara a ser utilizadas en el navegador por voz. Gracias a esa disección se puede obtener distancias estimadas entre esas partes más pequeñas y hacia que lado hay que girar al llegar a dichas partes para alcanzar el siguiente punto.

Esto se puede observar más detalladamente en la figura 4.1 donde se puede ver la información que aporta Mapbox sobre una ruta. Esta información se puede dividir en tres partes.

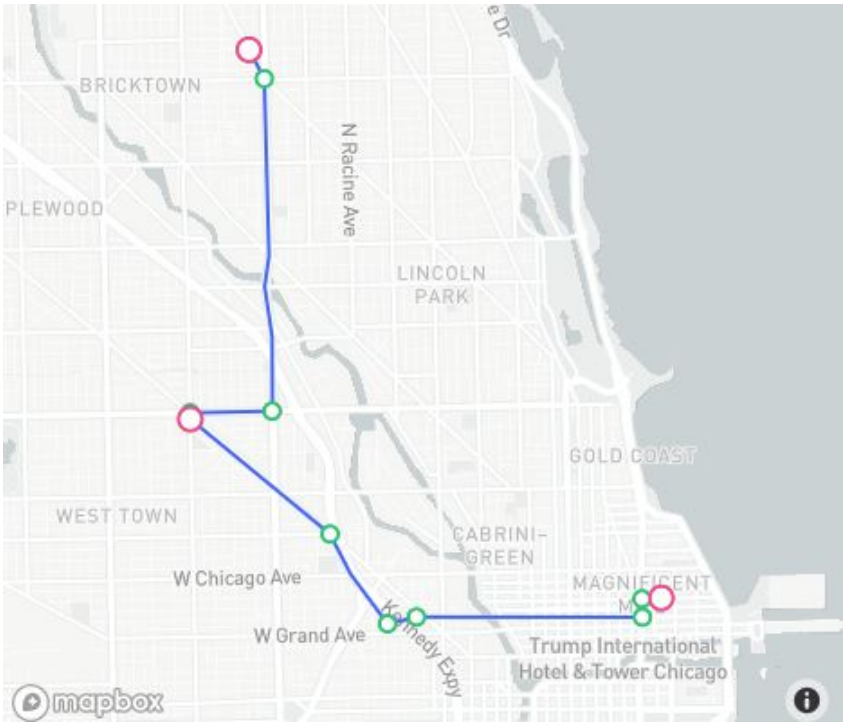


Figura 4.1: Información sobre el progreso de una ruta en Mapbox[47].

En primer lugar, la línea azul representa la ruta entre el punto de origen y destino. Dentro de ese recorrido se pueden ver unos puntos de color rosa. Dichos puntos según la nomenclatura de Mapbox se conocen como *leg*.

El círculo rosa llamado *leg* representa un punto de referencia o punto de parada durante la ruta. Por último, los puntos verdes (de menor tamaño que los puntos *leg*) son conocidos como *step*. Un *step* es un punto intermedio entre dos puntos *leg*. Estos puntos verdes

[46] *Nagivation* de Mapbox: https://docs.mapbox.com/api/navigation/
[47] Fuente:https://docs.mapbox.com/android/navigation/overview/route-progress/

representan maniobras.

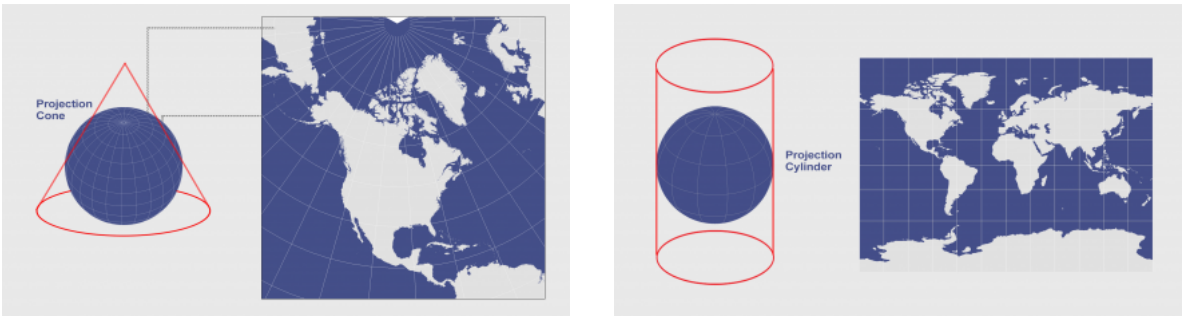
Una vez dados un punto de origen y otro de destino Mapbox devuelve toda esta información a través de un archivo con extensión *.json*. En este archivo se pueden ver datos como la latitud y longitud de los *steps* o las maniobras a realizar (girar, continuar...).

4.4 Transformación de coordenadas GPS a Unity

Para poder guiar al usuario a través de una experiencia de realidad mixta es necesario colocar elementos digitales en ciertos puntos de interés para indicarle el camino a seguir. La información que se obtiene de dichos puntos de interés (en este caso *legs* y *steps*) al crear una ruta vienen dados en coordenadas GPS. Es por ello que es necesario realizar una transformación de dichas coordenadas GPS a las coordenadas del mundo de Unity.

Para comprender la transformación es necesario conocer el concepto proyección de mapas. En cartografía se define una proyección como la forma en la que se transforma la superficie esférica de la Tierra a una superficie plana [46].

Existen distintos tipos de proyecciones en función de la forma geométrica a la que se adapte la transformación de la esfera. Por ejemplo, en la figura se pueden ver dos de estos tipos: proyecciones cónicas y cilíndricas.



(a) Proyección cónica. (b) Proyección cilíndrica.

Figura 4.2: Dos ejemplos de proyecciones cartográficas[48].

En este caso, Mapbox utiliza la proyección conocida como ***Spherical Mercator***[49]. Hay que tener en cuenta que las proyecciones distorsionan la información. Esto se puede ver en la figura de la proyección esférica utilizada por Mapbox donde Groenlandia (2,166 millones km^2) ocupa la misma superficie en la proyección que África (30,37 millones km^2).

Es importante tener en cuenta esta distorsión ya que cuanto más alejado del Ecuador se encuentre el usuario mayor distorsión habrá en las mediciones.

[48]Fuente: https://gisgeography.com/map-projections/

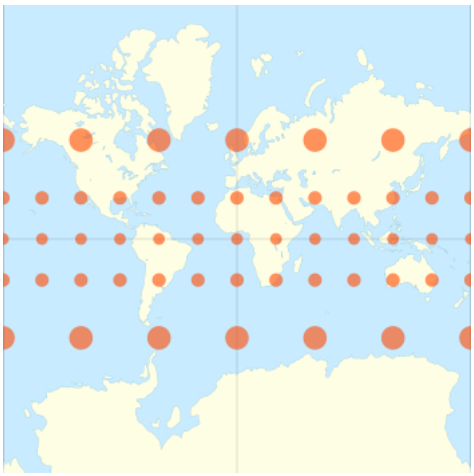


Figura 4.3: Distorsión de la proyección *mercator*[50].

Al igual que se distorsiona la información al hacer la proyección de las coordenadas, se pierde precisión a la hora de colocar las coordenadas transformadas de la proyección en los *transform* de Unity. Esto ocurre porque estos *transforms* utilizan valores de 32 bits. Para corregir este error se realiza una transformación más para utilizar posiciones relativas al usuario (evitando perder tanta precisión).

En este caso, para calcular la proyección se han utilizado las siguientes fórmulas, donde en primer lugar hay que tener en cuenta la constante *OriginShift*:

$$OriginShift = \frac{2\pi \cdot EarthRadius}{2}$$

Teniendo en cuenta un valor del radio de la tierra de 6.378.137km, para calcular la posición en el eje x se utiliza la siguiente fórmula:

$$x = \frac{longitud \cdot OriginShift}{180}$$

En cambio, el valor de la posición en el eje y se obtiene a través de la fórmula:

$$y = \frac{\log(\tan \frac{(90+latitud)\cdot\pi}{360})}{\frac{\pi}{180}}$$

Una vez que se ha aplicado la fórmula anterior, para ajustar la posición y respecto al origen igual que se ha hecho con la posición x se realiza el calculo de:

[50] *Mercator projection:* https://en.wikipedia.org/wiki/Mercator_projection

$$y = \frac{y \cdot OriginShift}{180}$$

Por último, hay que realizar los siguientes cálculos para que las posiciones sean relativas a un punto de referencia (en este caso el usuario):

$$x = x - refPoint.x$$

$$y = y - refPoint.y$$

Estos dos valores son la transformación final de coordenadas del mundo real (GPS) al sistema de coordenadas de Unity. En este caso correspondería a las coordenadas en los ejes x y z.

4.5 Pruebas de geolocalización sobre ARCore

Se desarrollaron dos aplicaciones utilizando Mapbox y ARCore. La primera, una aplicación para comprobar la precisión de la transformación de coordenadas GPS a coordenadas del mundo de Unity. En segundo lugar una aplicación que obtuviese la información de una ruta desde un punto de origen hasta un punto de destino. Utilizando esa información se colocarían elementos digitales en puntos característicos de la ruta para guiar al usuario.

Figura 4.4: Pruebas de la transformación de coordenadas en interiores.

Lo primero de todo fue crear un proyecto en Unity con Mapbox y ARCore integrados como se explica en el apartado A para desarrollar las aplicaciones.

El objetivo de la primera aplicación era comprobar la precisión de la transformación de coordenadas de latitud y longitud a una posición en el entorno de ARCore. Para ello, se utilizó las coordenadas GPS del dispositivo. Esas coordenadas de latitud y longitud son transformadas como se cuenta en la sección 4.4 para obtener coordenadas de Unity.

En esa posición se colocó un cubo de color gris para comprobar que se posicionaba en la ubicación del dispositivo. Se comprobó que en la aplicación en interiores se colocaba el elemento virtual cercano a las coordenadas como se puede ver en la figura 4.4. Sin embargo, dicho posicionamiento de los elementos virtuales tenía una desviación en las coordenadas de entre 1 y 2 metros.

Dado que las coordenadas en exteriores suelen ser más precisas, se pasó a probar el funcionamiento de la aplicación en exteriores. El resultado obtenido fue muy similar al resultado en interiores como se aprecia en la figura 4.5.

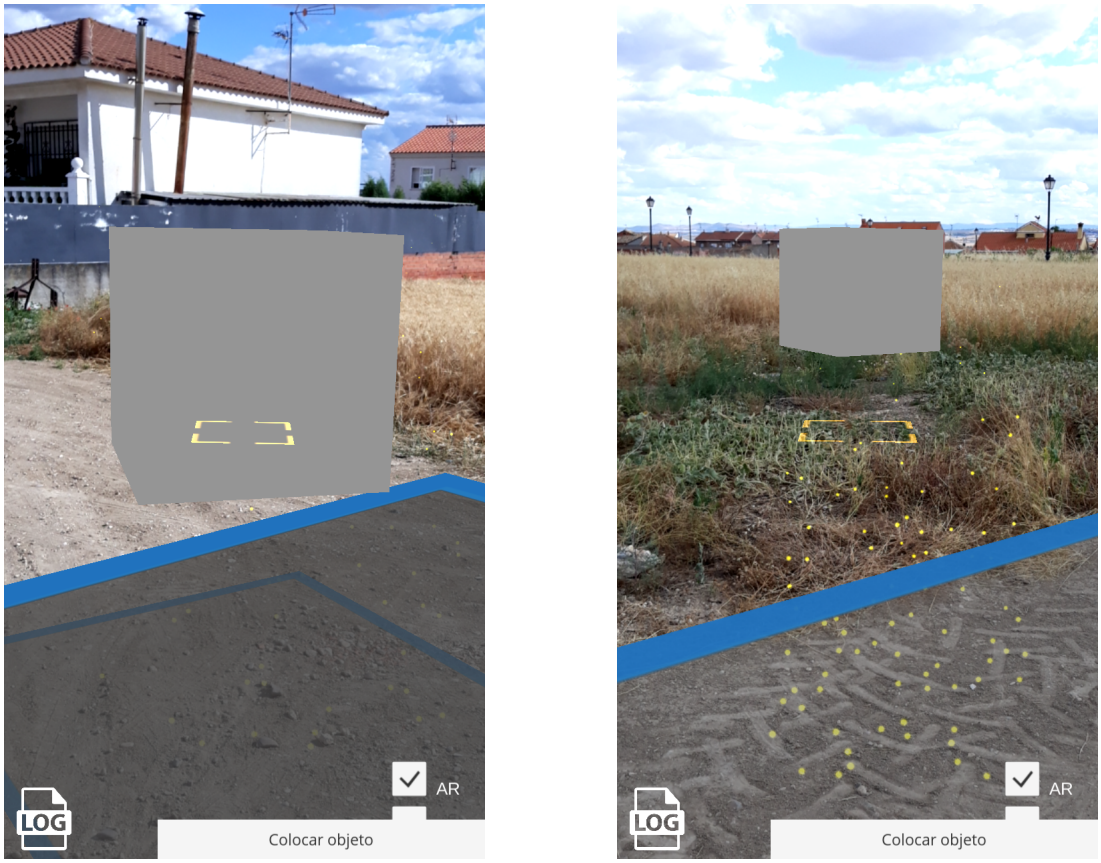


Figura 4.5: Pruebas de la transformación de coordenadas en exteriores.

Como se puede observar, no existen grandes cambios en las pruebas en exteriores respecto a las pruebas en interiores. Se llegó a la conclusión de que esto se debe a utilizar únicamente la información del GPS y no tener la triangulación de las torres de telefonía que aportaría el A-GPS.

Una vez obtenidos estos resultados se pasó a desarrollar la segunda aplicación. El primer paso fue obtener la información de la ruta entre dos puntos. Esto se consiguió haciendo uso de la API *Directions* de Mapbox (sección 4.3).

Al hacer una llamada a esta API desde Unity con un punto de origen (la ubicación del dispositivo) y un punto de destino (introducido por el usuario) se puede obtener la información de los *steps* y los *legs*. Sin embargo, tras realizar las pruebas concluimos que para poder realizar esta llamada a la API es necesario conexión a internet en el dispositivo (WiFi o a través de datos móviles). Esto se debe a que Mapbox no calcula las posiciones de destino ni de los *steps* o las *legs* dentro de la aplicación, sino que realiza una petición a través de la red para ofrecer dicha información.

El siguiente paso consistió en colocar elementos digitales (cubos de color gris) en la posición transformada a Unity de los *steps*. Además, se añadió una línea de color fucsia que une los cubos para poder seguir el camino hacia el siguiente cubo (figura 4.6).

Figura 4.6: Ejemplo de ejecución de búsqueda de rutas en AR.

[50]Aplicación: https://docs.mapbox.com/help/demos/how-mapbox-works/how-directions-works.html

Tras realizar las pruebas, hemos podido comprobar que se produce un error de localización por GPS de entre 2 y 5 metros desde la posición esperada. También hemos tenido en cuenta a la hora de realizar las pruebas que la ruta a seguir por la aplicación estaba establecida para vehículos. Por este motivo a la hora de realizar un seguimiento de la ruta la aplicación genera trayectos poco intuitivos para un viandante (figura 4.7).

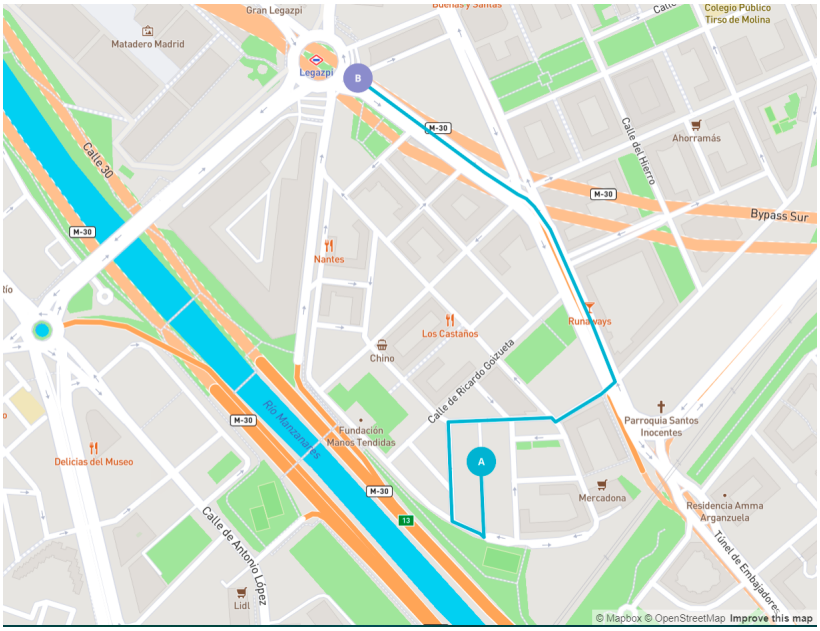


Figura 4.7: Comprobación de la ruta generada por nuestra aplicación con la demostración de Mapbox.[51]

Capítulo 5

Conclusiones y trabajo futuro

El objetivo del trabajo era la realización de una aplicación de búsqueda de rutas para dispositivos móviles con realidad mixta. Para ello se ha llevado a cabo un estudio sobre la realidad mixta, la realidad aumentada y la realidad virtual, tanto su funcionamiento como sus requisitos y características principales. Como el punto crítico de la realidad mixta es la oclusión, se puso a prueba su funcionamiento, comprobando tanto su tasa de acierto como su rendimiento. En lo referente a la búsqueda de rutas, se compararon distintas aplicaciones que ofrecían funcionalidades para la búsqueda de rutas en dispositivos móviles. El resultado de dicha comparación desembocó en generar una aplicación de búsqueda de rutas con *Mapbox* para dispositivos móviles.
Después de la investigación de las distintas realidades abarcadas en las XR, centrándonos en la realidad mixta y realizando distintas pruebas sobre este entorno se han obtenido los siguientes resultados:

La realidad mixta a día de hoy es una tecnología poco madura y que necesita un desarrollo más prolongado. Esta cuenta con una gran variedad de dispositivos compatibles, aunque las grandes industrias se centran más en el desarrollo de equipos para realidad virtual que en realidad aumentada o mixta. Esto hace que los costes de los HMDs de realidad mixta sean más elevados que los de realidad virtual y se encuentren menos en el mercado, exceptuando los dispositivos móviles. Además, el número de aplicaciones desarrolladas en HMDs de realidad mixta es mucho mayor que el desarrollo de estas para teléfonos móviles.

Sin embargo, la tecnología de realidad aumentada se ve destinada a su uso principal en dispositivos móviles. Gracias a tecnologías como ARCore que innova en el apartado de generación de nube de puntos y permite que esta se genere exclusivamente con una cámara, facilita el acceso a esta tecnología a un mayor número de personas.

En el entorno del videojuego, la realidad mixta está destinada a ser principalmente generada para dispositivos móviles. A día de hoy no existen demasiadas referencias dentro de este ámbito y las pocas que se pueden encontrar no son de realidad mixta, sino que su foco principal se centra en la realidad aumentada.

Gracias a las pruebas que se han realizado con la nube de puntos, se puede afirmar que esta novedosa tecnología de cálculo de profundidad es insuficiente para detección de planos o de superficies que no contengan bordes. También se ha llegado a la conclusión de que la nube de puntos funciona correctamente para la detección de los bordes. Aun así,

debido a las características de los dispositivos móviles actuales, su rendimiento queda bastante por debajo del esperado para una aplicación estable.

Por otro lado, después de documentarnos sobre distintos servicios de búsqueda de rutas, podemos afirmar que Mapbox es la herramienta que mejor se adapta, teniendo en cuenta que presta servicios sin necesidad de invertir dinero, de una forma sencilla y accesible en teléfonos móviles. A través de Mapbox se puede obtener la información de las rutas de forma muy detallada. Además, la extensa documentación de esta tecnología facilita el desarrollo de aplicaciones.

Pese a ello, hemos podido observar dos problemas a la hora de la utilización de esta aplicación:

- En primer lugar, la imprecisión de la posición proporcionada por el GPS puede llegar a alcanzar los cinco metros de desviación, lo que dificulta bastante el entendimiento de la ruta en realidad aumentada.
- En segundo lugar, el hecho de que el dispositivo móvil tenga que estar permanentemente conectado a la red para trazar la ruta. Al no utilizar exclusivamente el GPS se obliga a tener una conexión a internet permanente fuera del domicilio, lo que no está a disposición de todos los usuarios.

Como trabajo futuro a realizar, en primer lugar habría que centrarse en una investigación exhaustiva sobre los dispositivos móviles actuales, sus características a nivel hardware (CPU, memoria principal, procesamiento de imágenes...) y en base a ello realizar una predicción sobre cómo serán estos en el futuro.

Dicha predicción serviría para determinar en qué momento la tecnología de realidad mixta puede llegar a ser viable en dispositivos móviles y ejecutada en tiempo real en estos.

No se debe olvidar tampoco la existencia de los sensores *ToF*, aunque la integración de estos en los dispositivos móviles encarece bastante su producción y venta, por lo que no lo consideramos óptimo para su comercialización a día de hoy.

Mientras se alcanza este punto tecnológico, se debería optimizar al máximo la forma en la que se generan nubes de puntos. Para ello, se debería incorporar la detección de superficies (planos horizontales y verticales) a la generación de la nube de puntos y que esta no sea una funcionalidad aparte de la aplicación.

A su vez, también se debería investigar distintas formas de generar mallas 3D en base a una nube de puntos, las cuales presenten coherencia con esta y se apliquen a las superficies de los objetos.

Respecto a los sistemas de búsqueda de rutas, se podrían realizar las mismas pruebas que en el apartado 4.5 pero en este caso, utilizando la tecnología A-GPS que se ha tratado en el mismo apartado. De esta forma se podría comprobar la exactitud de la ubicación del usuario, lo cual permitiría generar un entorno de realidad mixta más preciso.

Finalmente cabe destacar que el trabajo esperado de este proyecto, una aplicación que unifique la tecnología de realidad mixta con el sistema de búsqueda de rutas no ha sido completamente realizado debido a dos factores:

- La situación ocasionada por el COVID-19[52], que no ha permitido poder realizar las pruebas pertinentes para la búsqueda de rutas.
- Los malos resultados obtenidos con ARCore que impiden una oclusión en tiempo real del entorno.

Sin embargo, si que se ha conseguido crear por separado aplicaciones que generan oclusión en tiempo real y sistemas de búsqueda rutas. El resultado final, cuando la tecnología esté disponible, consistirá únicamente en aunar ambas aplicaciones.

[52]Fuente: https://es.wikipedia.org/wiki/COVID-19

Chapter 5

Conclusions and future work

The goal of the project was to carry out a route searching application for mobile devices with mixed reality. For this, a study on mixed, augmented and virtual realities, its operation, main requirements and characteristics, has been carried out. Since the critical point of mixed reality is the occlusion, its operation was tested, checking both its hit rate and its performance. Regarding route search, different applications were compared that offered functionalities for route search on mobile devices. The result of this comparison resulted in generating a route search application with *Mapbox* for mobile devices.
After investigating the different realities covered in the XR, focusing on mixed reality and conducting different tests on this environment, the following results have been obtained:

Nowadays' mixed reality is a not very mature technology, and needs a longer development. It has a wide variety of compatible devices, although large industries focus more on the development of equipment for virtual reality than for augmented or mixed reality. This makes the costs of mixed reality HMDs higher than those for virtual reality, and there is a lower number on the market except for mobile devices. In addition, the number of applications developed in mixed reality HMDs is much greater than the development of these for mobile phones.

However, augmented reality technology is intended for primary use on mobile devices. Thanks to technologies such as ARCore, that innovates in the point cloud generation section and allows it to be generated exclusively with a camera, it facilitates access to this technology for a greater number of people.

In the video game environment, mixed reality is intended to be primarily generated for mobile devices. Today there are not too many references in this area and the few that can be found are not mixed reality focused, as their main focus is on augmented reality.

Thanks to the tests that have been carried out with the point cloud, it can be affirmed that this new depth calculation technology is insufficient for detecting planes or surfaces that do not contain edges. It has also been concluded that the point cloud works correctly for edge detection. Still, due to the characteristics of current mobile devices, its performance is well below what is expected for a stable application.

On the other hand, after researching about different route search services, we can affirm that Mapbox is the tool that best adapts, taking into account that it provides services

without the need to invest money, in a simple and accessible way on mobile phones. Through Mapbox you can obtain the information of the routes in a very detailed way. In addition, the extensive documentation of this technology facilitates the development of applications.

Despite this, we have been able to observe two problems when using this application:

- Firstly, the imprecision of the position provided by the GPS can reach five meters deviation, which makes it difficult to understand the route in augmented reality.
- Second, the fact that the mobile device has to be permanently connected to the network to trace the route. By not exclusively using GPS, you are forced to have a permanent internet connection outside home, which is not available for all users.

As future work to be carried out, in the first place it would be necessary to focus on an exhaustive research on current mobile devices, their characteristics at the hardware level (CPU, main memory, image processing...) and based on this, make a prediction on how will be these in the future.

This prediction would serve to determine at what moment the mixed reality technology can become viable on mobile devices and run in real time on them.

The existence of *ToF* sensors should not be forgotten either, although the integration of these in mobile devices makes their production and sale more expensive, so we do not consider it optimal for their commercialization today.

While this technological point is reached, the way in which point clouds are generated should be optimized as much as possible. For this, the detection of surfaces (horizontal and vertical planes) should be incorporated into the generation of the point cloud, so this is not a functionality apart from the application.

In turn, you should also investigate different ways to generate 3D meshes based on a point cloud, which are consistent with it and applied to the surfaces of objects.

Regarding route search systems, the same tests could be carried out as in the **??** but in this case, using the A-GPS technology that has been discussed in the same section. In this way, the accuracy of the user's location could be verified, which would allow a more accurate mixed reality environment to be generated.

Finally, it should be acknowledged that the expected work of this project, an application that unifies mixed reality technology with the route search system, has not been completely carried out due to two factors:

- The situation caused by the COVID-19 [53], which has not allowed to carry out the pertinent tests to find routes.
- The bad results obtained with ARCore that prevent a real-time occlusion of the environment.

[53]Source: https://en.wikipedia.org/wiki/COVID-19

However, it has been possible to create separate applications that generate occlusion in real time and route search systems. The end result, when the technology is available, will consist solely of combining both applications.

Capítulo 6

Contribuciones

6.1 Sergio Gavilán Fernández

Mi primer contacto con las XR fue cuando realicé una aplicación de realidad aumentada para medir la accesibilidad. Realicé este trabajo para la Fundación ONCE utilizando la tecnología de ARCore, usando técnicas de realidad aumentada sin marcadores.

Antes de empezar con el desarrollo investigué sobre las distintas tecnologías que permitían el uso de realidad aumentada en móviles. Descubrí distintas tecnologías aparte de ARCore (la cual ya conocíamos) como Kudan o Cordova. Después de esta investigación, y debido también a la experiencia que ya poseíamos de los años anteriores del grado, decidimos que lo mejor era utilizar ARCore con Unity.

Como en los momentos iniciales del desarrollo no disponíamos de teléfonos móviles compatibles con ARCore, decidí investigar sobre la parte que no tenía relación con ARCore, los mapas. En este caso el primero que investigué fue Google Maps. Desarrollé un proyecto de Google Maps en Unity el cual no funcionaba debido a que aunque poseía una *API Key*, era necesario añadir una dirección de facturación.

Ya que también conocía Mapbox y ambos parecían ofrecer las mismas funcionalidades, aunque Mapbox sin necesidad de direcciones de facturación ni pagos, continué dejando de lado el proyecto de Google Maps y cambiándolo por un proyecto de Unity con Mapbox. En este punto conseguí hacer funcionar Mapbox pero dejé ese proyecto parado ya que en ese momento la facultad nos proporcionó un teléfono que soportaba ARCore.

Mi compañero y yo decidimos que lo más importante del proyecto era generar un correcto efecto de oclusión, es por eso que me puse a colaborar con él en las pruebas del *shader* de profundidad (sección 3.2.1) dejando de lado el proyecto de Mapbox como he dicho anteriormente.

Como no conseguíamos grandes avances, empecé a documentarme a través de libros y artículos sobre las XR, principalmente sobre la realidad mixta. Mientras Javier continuaba investigando el *shader* de profundidad comencé a plasmar los resultados obtenidos en la fase de investigación mostrando las definiciones de las tres XR y los dispositivos

compatibles con ellas en el capítulo 2.

Una vez probado el *shader* de profundidad me documenté sobre *shaders* en Unity y desarrollé el *shader* de oclusión. Una vez hecho esto, implementé una aplicación que utilizaba dicho *shader* y posteriormente se utilizó para las pruebas del punto 3.2.2.

Al tener funcionando el *shader* de oclusión y mi compañero trabajando con él, pude continuar el desarrollo sobre el proyecto de Mapbox con Unity. En este proyecto desarrollé una aplicación que permitía captar las coordenadas del teléfono móvil y transformarlas a coordenadas de ARCore. Esta aplicación se utilizó para colocar elementos virtuales en unas coordenadas dadas como se cuenta en la sección 4.5.

Continué plasmando en el capítulo 2 las tecnologías necesarias para utilizar las XR así como escribiendo sobre los usos de dichas tecnologías en la actualidad. Todo esto después de una ardua investigación a través de artículos y libros (los cuales no existen en abundancia actualmente sobre realidad mixta). Finalmente, escribí mis conocimientos obtenidos a través de las pruebas con mapas en las distintas secciones sobre Google Maps y Mapbox.

Por último, desarrollé con Javier la aplicación de navegación que colocaba elementos digitales en coordenadas GPS obtenidas de Mapbox 4.5.

6.2 Javier Cordero Calvo

La primera vez que interactué con la realidad extendida como desarrollador y no como usuario fue algunas semanas antes de proponer el tema de este trabajo a mi compañero Sergio y al tutor. Para comprobar la viabilidad de un proyecto como este, comencé a investigar toda la tecnología hasta la fecha existente, tanto de realidad virtual, realidad aumentada y realidad mixta.

Una vez aprobado el tema del trabajo, comencé a realizar pequeños proyectos a modo de prueba con *Vuforia*, pero debido a la falta de un dispositivo compatible con dicha tecnología, me fue imposible continuar con las pruebas. Sin embargo, decidí generar una aplicación de realidad aumentada sin aplicaciones XR de terceros.

Dado que el trabajo a realizar contenía la idea de mezclar esta tecnología con mapas, creé un objeto a modo de brújula, que utilizaba la brújula del dispositivo para apuntar siempre al norte y situar esta con realidad aumentada dentro de la aplicación.

Tras tener funcionando la aplicación de la brújula, también investigué la tecnología ofrecida por Mapbox para la geolocalización y generé una aplicación con las escenas de prueba que estos ofrecían en Unity. Esta aplicación sería usada en el futuro para facilitar la integración de ARCore con Mapbox.

Una vez dispusimos de los teléfonos móviles compatibles con ARCore, mi labor principal fue la de integrar esta tecnología con Unity y poder generar una aplicación que funcionase en el dispositivo. Para ello, dentro de Unity, investigué todos los paquetes que podrían servir de ayuda para o bien hacer que la tecnología funcione o facilitar su uso. También

investigué cuales eran los requisitos mínimos para la aplicación de ARCore que un móvil debía tener así como el tipo de renderizado con el que debía contar. Estas pruebas las realicé sobre uno de los ejemplos que incorpora ARCore, llamado *HelloAR*.

El siguiente paso que realicé fue conocer en profundidad el funcionamiento de la API de ARCore dentro de Unity para poder modificarla en un futuro si fuese necesario.

Una vez comprendido el funcionamiento de esta, comencé a modificar varios factores que la misma aplicación ofrecía para obtener distintos resultados. Dado que ARCore crea una nube de puntos con una malla, procedí a eliminarla y generar puntos propios para comprobar el funcionamiento sin malla.

Tras este punto y dado que ARCore me permitía únicamente una serie de acciones públicas a realizar en el editor, generé un script propio para poder modificar todas las variables y factores que nos conviniesen.

Dentro de esta ahora modifiqué todos los valores que tenía ARCore, cambiando por ejemplo las formas de objetos que instanciaba, eligiendo cuándo quería instanciarlos y cuando no.
Una vez tenía todos los puntos y podía modificar sus formas desde el editor, intenté mediante el Line Renderer proporcionado por Unity generar una oclusión mediante la unión de todos los puntos de la nube.

Cuando ya estaban todos los puntos, la siguiente acción fue comprobar si existía verdaderamente una profundidad con estos. Investigué el *shader* de profundidad explicado en el apartado 3.2.1.

Generé un proyecto común que contenía Mapbox y ARCore, pero debido a la imposibilidad de realizar pruebas por la situación excepcional del COVID-19, la aplicación se descartó y se continuó con las pruebas en aplicaciones individuales.

Junto a mi compañero Sergio realicé la aplicación y las pruebas de Mapbox unida con realidad aumentada para el seguimiento de rutas. La aplicación consiste en establecer el punto al que se desea ir y establecer una ruta que se vea reflejada en realidad aumentada. Las pruebas consistieron en salir al exterior tras el estado de alarma y comprobar el funcionamiento de esta aplicación.

En último lugar, creé y optimicé la aplicación que genera todas las pruebas dentro de Unity así como la aplicación en Python que compara las imágenes de una serie de carpetas dadas y devuelve el resultado de estas comparaciones en archivos *CSV*. La primera aplicación consistía en una ampliación de la que generaba la nube de puntos, incluyendo toda la información de rendimiento de la aplicación así como las funcionalidades de capturar la pantalla, activar y desactivar la nube de puntos, o todo el proceso de generar las pruebas y guardar los resultados. También añadí cierta prevención de errores por parte del usuario para hacerla un poco mas fácil e intuitiva.

En la aplicación de Python, partí de la base de *OpenCV* para la comparación de imágenes. Desde Photoshop, creé todas las imágenes de referencia uniendo las imágenes generadas por la aplicación.

Finalmente, realicé las pruebas con el dispositivo móvil para comprobar las diferencias de las nubes de puntos y poder anotar los resultados de ellas. Tanto las pruebas de oclusión como las de tiempo y la primera prueba de concepto. Por último realicé las pruebas que necesitaban de un ordenador portátil para conectar el dispositivo móvil.

Bibliografía

[1] Rãzvan-George Mihalyi, Kaustubh Pathak, Narunas Vaskevicius, Tobias Doernbach, and Andreas Birk. Robust 3D object modeling with a low-cost rgbd-sensor and AR-markers for applications with untrained end-users. *Robotics and Autonomous Systems*, 66, 01 2015.

[2] Jens Mueller, Simon Butscher, and Harald Reiterer. Immersive analysis of health-related data with mixed reality interfaces: Potentials and open question. pages 71–76, 11 2016.

[3] Akira Utsumi Fumio Kishino Paul Milgram, Haruo Takemura. Augmented reality: A class of displays on the reality-virtuality continuum. *Proceedings of SPIE - The International Society for Optical Engineering (Proceedings of SPIE)*, 1994.

[4] Ivan E. Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), page 757–764, New York, NY, USA, 1968. Association for Computing Machinery.

[5] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf. Consistent video depth estimation. 39(4), 2020.

[6] G. A. Giraldi R. Silva, J. C. Oliveira. Introduction to augmented reality. *National Laboratory for Scientific Computation, Brazil*.

[7] Jack Cheng, Keyu Chen, and Weiwei Chen. Comparison of marker-based AR and markerless AR: A case study on indoor decoration system. 07 2017.

[8] Kundalakesi Mathivanan, Swathi T, Ashapriya B, and Sruthi R. A study of virtual reality. *International Journal of Trend in Research and Development*, 4:2394–9333, 06 2017.

[9] Stephanie Chuah. Why and who will adopt extended reality technology? literature review, synthesis, and future research agenda. 12 2018.

[10] Alan B. Craig. *Understanding Augmented Reality*. Morgan Kaufmann, 1 edition, 2013.

[11] Feng Zhou, Henry Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, 2:193–202, 09 2008.

[12] Thomas AuerStefan BrantnerAxel Pinz. The integration of optical and magnetic tracking for multi-user augmented reality. pages Gervautz M., Schmalstieg D., Hildebrand A. (eds) Virtual Enviroments '99. Eurographics. Springer, Vienna, 1999.

[13] Andrei State Mark A. Livingston. *Department of Computer Science University of North Carolina at Chapel Hill*, 2001.

[14] GPSGOV. Información oficial del gobierno de los estados unidos relativa al sistema de posicionamiento global y temas afines. URL: https://www.gps.gov/systems/gps/spanish.php.

[15] Adrian Kaehler and Gary R. Bradski. Learning opencv 3: Computer vision in C++ with the OpenCV library. 2016.

[16] G. Reitmayr and T. W. Drummond. Going out: robust model-based tracking for outdoor augmented reality. In *2006 IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 109–118, 2006.

[17] B. Jiang, U. Neumann, and Suya You. A robust hybrid tracking system for outdoor augmented reality. In *IEEE Virtual Reality 2004*, pages 3–275, 2004.

[18] G. Reitmayr, T. Langlotz, D. Wagner, A. Mulloni, G. Schall, D. Schmalstieg, and Q. Pan. Simultaneous localization and mapping for augmented reality. In *2010 International Symposium on Ubiquitous Virtual Reality*, pages 5–8, 2010.

[19] Tian Yuan, Guan Tao, and Wang Cheng. Real-time occlusion handling in augmented reality based on an object tracking approach. *Sensors*, 10, 04 2010.

[20] Neil Mathew. Why is occlusion in augmented reality so hard? URL: https://hackernoon.com/why-is-occlusion-in-augmented-reality-so-hard-7bc8041607f9.

[21] Toby Sharp, Yichen Wei, Daniel Freedman, Pushmeet Kohli, Eyal Krupka, Andrew Fitzgibbon, Shahram Izadi, Cem Keskin, Duncan Robertson, Jamie Shotton, David Kim, Christoph Rhemann, Ido Leichter, and Alon Vinnikov. Accurate, robust, and flexible real-time hand tracking. pages 3633–3642, 04 2015.

[22] Viviane Clay, Peter König, and Sabine Koenig. Eye tracking in virtual reality. *Journal of Eye Movement Research*, 12, 04 2019.

[23] S. Best. Perceptual and oculomotor implications of interpupillary distance settings on a head-mounted virtual display. In *Proceedings of the IEEE 1996 National Aerospace and Electronics Conference NAECON 1996*, volume 1, pages 429–434 vol.1, 1996.

[24] Sam Barker. Augmented mixed reality: Impact assessments, sector analysis forecasts 2019-2024. *Juniper Research*, 2019.

[25] Wazir Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97, 02 2019.

[26] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. Cloud computing: An overview. volume 5931, pages 626–631, 01 2009.

[27] Harry Pence. Smartphones, smart objects, and augmented reality. *The Reference Librarian*, 52:136–145, 01 2011.

[28] Alice Bonasio. Artículo "immersive experiences in education". *Microsoft*.

[29] M. Fiorentino, R. de Amicis, G. Monno, and A. Stork. Spacedesign: a mixed reality workspace for aesthetic industrial design. In *Proceedings. International Symposium on Mixed and Augmented Reality*, pages 86–318, 2002.

[30] Google. Arcore overview. URL: https://developers.google.com/ar/discover.

[31] Apple. Arkit - augmented reality. URL: https://developer.apple.com/augmented-reality/arkit/.

[32] Vuforia. Vuforia developer portal. URL: https://developer.vuforia.com/.

[33] XRGO. Xrgo indoor oclussion. URL: https://xrgo.io/en/product/ci-inplace/.

[34] IDEA ingenieria. Definición de nube de puntos. URL: https://ideaingenieria.es/nube-de-puntos/que-es-nube-de-puntos.

[35] IDEA ingenieria. Definición de nube de puntos. URL: https://ideaingenieria.es/servicios/transformacion-digital-4-0/nube-de-puntos.

[36] Wikipedia. Definición de nube de puntos. URL: https://es.wikipedia.org/wiki/Nube_de_puntos.

[37] Wikipedia. Visual odometry. URL: https://en.wikipedia.org/wiki/Visual_odometry.

[38] Apple. Understanding world tracking. URL: https://developer.apple.com/documentation/arkit/world_tracking/understanding_world_tracking.

[39] Wikipedia. Nube de puntos en la exploración de marte. URL: https://en.wikipedia.org/wiki/Mars_Exploration_Rover.

[40] Google. How arcore works. URL: https://developers.google.com/ar/discover.

[41] *OpenGL*. *OpenGL* wiki. URL: https://www.khronos.org/opengl/wiki/Shader.

[42] Unity. Unity - manual: Meshes, materials and shaders. URL: https://docs.unity3d.com/Manual/Shaders.html.

[43] Tayx. Graphy. URL: https://assetstore.unity.com/packages/tools/gui/graphy-ultimate-fps-counter-stats-monitor-debugger-105778.

[44] Google. Google maps platform. URL: https://cloud.google.com/maps-platform/.

[45] Mapbox. Mapbox main page. URL: https://www.mapbox.com/.

[46] John P. Snyder Philip M. Voxland. An album of map projections.

Apéndice A

Integración de los frameworks usados en Unity

Integración de ARCore con Unity

La integración de ARCore con Unity se realiza de manera sencilla mediante la instalación de los paquetes (*packages* en inglés) de Unity.

Un paquete de Unity, según su propia definición, es *"un contenedor que guarda varios tipos de características o Assets"*. Estas características o *Assets* pueden ser herramientas y librerías del editor, gráficas, físicas... O colecciones de Assets como texturas o animaciones [54].

Pese a la facilidad que estos paquetes ofrecen, conllevan una serie de dependencias internas por los proyectos de prueba que integran, por lo que hay que realizar algunos ajustes dentro del proyecto para que todo funcione correctamente.

Dependiendo del tipo de efecto que queramos generar dentro de nuestra aplicación se dará uso de un paquete u otro por las diversas características que presentan.

Para comenzar, en Unity (*versión 2019.3.1f1*), hay que crear un proyecto vacío en 3D, sin ningún añadido o característica especial.

Después, descargar la SDK de ARCore desde su página oficial para Unity. Una vez descargada, con el proyecto vacío de Unity abierto, simplemente tendremos que hacer doble click sobre esta SDK para que se instale dentro de nuestro proyecto.

Debido a las dependencias de ARCore, hay que instalar los paquetes *XR Legacy Input helpers* (versión 2.1.4) y *Multiplayer HLAPI* (versión 1.0.4). Estos se encuentran dentro de la pestaña *Window ->Package Manager*.

Una vez estén todas las dependencias instaladas, hay que realizar toda la configuración del proyecto. Dentro de las opciones de *Build* en Unity, hay que establecer la plataforma

[54]Fuente:https://docs.unity3d.com/Manual/Packages.html

objetivo de la aplicación a Android. Hecho esto, hay que añadir la escena a nuestro proyecto. Esta escena puede estar hecha por nosotros con los elementos de ARCore o, para probar su funcionamiento, podremos partir de la escena que facilita ARCore llamada ***HelloAR*** *(GoogleARCore ->Examples ->HelloAR ->Scenes ->HelloAR).*

Añadida la escena, en la opción *"Player Settings" (File ->Build Settings ->Player Settings)*, en la pestaña *Player ->Other Settings*, hay que eliminar el tipo de renderizado "**Vulkan**", que se encuentra dentro del apartado *Graphics API*. Esto simplemente se debe a que a día de hoy, ARCore no es compatible con dicho tipo de renderizado. Internamente, *"SurfaceTexture"* (textura usada por ARCore) es una textura mutable. Esto quiere decir que cada textura está vinculada a un único objeto de textura. Vulkan no es compatible con este tipo de almacenamiento por lo que es incompatible a día de hoy con ARCore.

En *API Compatibility Level*, establecer ".NET 4x". Esto es debido a que a partir de la versión 2019.1 de Unity las aplicaciones deben ser compatibles con esta configuración. El mínimo *API Level* requerido por ARCore es *API Level 24* (Android 7.0 Nougat), por lo que hay que indicarlo dentro de la misma pestaña. Este es la mínima compatibilidad que presentan los móviles con ARCore.

Finalmente, en el apartado *XR Settings*, habilitar la opción *"ARCore supported*" para que nuestra aplicación sea compatible con esta tecnología.

Integración de Mapbox con Unity

La integración de Mapbox en Unity se realiza utilizando un *package* que ofrece la página de Mapbox[55]. Una vez descargado el paquete se deberá añadir a Unity. En este caso, viene con ARCore incluido.

Para poder utilizar los distintos servicios que ofrece Mapbox es necesario crear una *API Key* (identificador único para autenticar a un usuario al hacer una llamada a una API). Para crear esta *API Key* es necesario iniciar sesión en la página de Mapbox. Una vez que se ha hecho el *login* hay que dirigirse al icono del usuario en la parte superior derecha y pinchar en "*Account*". En esta pestaña se puede crear una *API Key* presionando el botón "*Create token*".

Al añadir el paquete a Unity se nos creará un menú en la parte superior del programa llamado "Mapbox". Si se selecciona se pueden ver distintas opciones de las cuales se deberá seleccionar "*Setup*". Al hacerlo se abrirá una ventana en la que se deberá introducir el *token* generado anteriormente. De esta manera se podrán utilizar los distintos servicios de Mapbox.

Ya que este *package* viene con ARCore incluido, si se quieren utilizar los proyectos de Mapbox con AR es necesario seguir los pasos para integrar ARCore en Unity.

[55] *Package* de Mapbox para Unity: https://www.mapbox.com/install/unity/

Apéndice B

Material adicional

Enlace al repositorio: https://github.com/JavierCordero/TFG

Enlace a carpeta con material adicional: https://drive.google.com/drive/folders/1zhz9I86usgwGc_ZiUkE20Z_CA3oy9eZs?usp=sharing [56]

[56]Todo el material encontrado dentro de esta página está únicamente destinado a miembros de la Universidad Complutense de Madrid, por lo que su distribución queda totalmente prohibida a miembros externos a esta comunidad.