Realidad Virtual y Realidad Aumentada

DESARROLLO DE APLICACIONES

UNITY 3D

WINDOWS MIXED REALITY

VUFORIA

ARKIT

ARCORE

www

Desde **www.ra-ma.es** podrá descargar material adicional.

Libro disponible en: eybooks.com

FERNANDO NAVARRO
ANTONIO MARTÍNEZ
JOSÉ M. MARTÍNEZ

Ra-Ma®

Realidad Virtual y Realidad Aumentada

Desarrollo de aplicaciones

Realidad Virtual y Realidad Aumentada

Desarrollo de aplicaciones

Fernando Navarro

Antonio Martínez

José M. Martínez

Ra-Ma®

La ley prohíbe
fotocopiar este libro

MARCAS COMERCIALES. Las designaciones utilizadas por las empresas para distinguir sus productos (hardware, software, sistemas operativos, etc.) suelen ser marcas registradas. RA-MA ha intentado a lo largo de este libro distinguir las marcas comerciales de los términos descriptivos, siguiendo el estilo que utiliza el fabricante, sin intención de infringir la marca y solo en beneficio del propietario de la misma. Los datos de los ejemplos y pantallas son ficticios a no ser que se especifique lo contrario.

RA-MA es marca comercial registrada.

Se ha puesto el máximo empeño en ofrecer al lector una información completa y precisa. Sin embargo, RA-MA Editorial no asume ninguna responsabilidad derivada de su uso ni tampoco de cualquier violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Esta publicación tiene por objeto proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para el editor ninguna forma de asistencia legal, administrativa o de ningún otro tipo. En caso de precisarse asesoría legal u otra forma de ayuda experta, deben buscarse los servicios de un profesional competente. d e s c a r g a do en: e y b o oks. c o m

Editado por:
RA-MA Editorial
Calle Jarama, 3A, Polígono Industrial Igarsa
28860 PARACUELLOS DE JARAMA, Madrid
Teléfono: 91 658 42 80
Fax: 91 662 81 39
Correo electrónico: *editorial@ra-ma.com*
Internet: *www.ra-ma.es* y *www.ra-ma.com*
ISBN: 978-84-9964-739-5
Depósito legal: M-18384-2018
Maquetación: Antonio García Tomé
Diseño de portada: Antonio García Tomé
Filmación e impresión: Safekat
Impreso en España en octubre de 2018

ÍNDICE

AGRADECIMIENTOS

Fernando

A Rosa y Fernando, por su apoyo constante.

A Marcos Rodríguez, de Bravent, por compartir sus impresiones sobre "el continuo de lo virtual". Sigo pensando que es una de esas cosas que nunca se comprenden del todo y que no hay que tenerla demasiado en cuenta, pero, aun así, hablo brevemente de ello en el libro. Espero que te guste como ha quedado ;)

A Ramon Cabezas Navas, de KAPS Management, porque sin su visión particular de la Transformación Digital a través de la metodología PETRA, todo esto sería otra cosa.

José María

A Belén, Julia, María y Esther, por llenar de ilusión mi vida todos los días. A mis padres por sus consejos y apoyo incondicional. Y como no a mi socio por aguantarme todos los días, sobre todos los malos.

Antonio

A mis padres, mis herman@s, mis sobrin@s, mis amig@s por aguantarme siempre, por acompañarme esas noches interminables, por apoyarme incondicionalmente y por estar siempre ahí. Sin vostr@s no hubiese sido posible.

Nota de Fernando y José María: *Tranquilo, te seguiremos aguantando y convenceremos a los demás para que también sigan haciéndolo :)*

Y por parte de todos:

A Federico Peinado, Doctor y Profesor de la Universidad Complutense de Madrid y Director de laboratorios Narratech, por la elaboración del prólogo de este libro y por la participación de Narratech en el capítulo 2.

A todas las personas que han colaborado en la preparación de los contenidos del capítulo 2 (espero no dejarme a ninguna, porque han sido muchas).

- A Juan Rivas y Daniel Rubio, de 6DLab.
- A Giorgio Airoldi y Pepe Carnevale, de ARS VIVA.
- A José Luis Carrascosa, Marcos Rodríguez, Janire Sainz y Roseline Paola, de Bravent
- A Marga Marticorena , Vanessa Lafuente, Esther Olivares y Encarnación Prieto, de EPTISA.
- A José Tomás González y Michiel van Vliet, de Kabel.
- A Pedro Antonio González, de Padaone Games.

SOBRE LOS AUTORES

Fernando Navarro

Especialista en ingeniería informática y desarrollando su actividad en este ámbito desde 1989, se especializa en formación y RRHH y desde 1999 ha participado en el desarrollo de multitud de sistemas de información aplicados a estas áreas para grandes corporaciones como Deloitte, Endesa, Cecabank, la Universidad de Deusto, etc., incluyendo el videojuego y la gamificación como elementos innovadores dentro de los procesos formativos.

Desde el año 2007 ha colaborado con distintos estudios en el desarrollo de varios videojuegos para distintas plataformas.

Actualmente ejerce su labor profesional como director técnico del estudio Digital Dream Factory.

Durante los últimos años, ha participado en el desarrollo de varios sistemas de Realidad Virtual, Aumentada y Mixta aplicados a los sectores de la ingeniería, la construcción, la formación y la museología.

Sobre Digital Dream Factory

Digital Dream Factory es un estudio dedicado a la consultoría y el desarrollo de soluciones, especializado en la integración de videojuegos educativos en sistemas de formación on-line, gamificación de contenidos y desarrollo de aplicaciones para la Transformación Digital, basadas en realidad Virtual, Aumentada y Mixta e IoT. Más información en *http://www.digitaldreamfactory.es*.

José María Martínez

Ingeniero informático en Sistemas por la universidad de Granada con diferentes cursos complementarios en distintas universidades (Universidad de Granada, Universidad de Jaén, Standford University).

Responsable de departamento TIC / Sistemas de distintas empresas en sectores como la Moda, la madera o la construcción.

Desde 2013 desarrolla su labor profesional en Radikal Dreamers, empresa de la que es fundador y donde, además de las labores propias de todo empresario, se dedica al desarrollo e integración de producto.

Antonio Martínez

Diseñador de profesión, cacharrero desde que tengo uso de razón, friki por naturaleza y programador por necesidad.

Diseñador 3D generalista llevado desde el mundo de la animación al del videojuego. Freelance como pistoletazo de salida a mi carrera profesional pasando por profesor y creador de videojuegos hasta llegar a CEO. Cofundador de Blue Shadow Games (padre de videojuegos tales como Naught, Non-Flying Soldiers …) y actual CEO de Radikal Dreamers.

Amante de las nuevas tecnologías y siempre con la ilusión de aprender cosas nuevas.

Sobre Radikal Dreamers

Radikal Dreamers desarrolla soluciones verticales para el sector industrial y son especialistas en nuevas tecnologías como la Realidad Aumentada, Realidad Virtual, Motion tracking, Tours virtuales, Holografía, etc. se dedica a fabricar soluciones artesanales "impactantes" para clientes tan dispares como empresas museísticas, del sector turístico, industrial, marketing o entretenimiento. Con una trayectoria reseñable dispone de varios videojuegos, aplicaciones, interactivos y simuladores en su portfolio. Más información en *http://www.radikaldreamers.com*.

PRÓLOGO

No es casualidad que Jaron Lanier, el pionero que creó y popularizó el mercado de los dispositivos de Realidad Virtual, sea a la vez compositor de música clásica y escritor especializado en cultura digital. Desde la atalaya de la interdisciplinaridad defiende una máxima, "lo más importante de la tecnología es cómo cambia a las personas", con la que resulta fácil estar de acuerdo, aunque a menudo olvidemos.

Precisamente este libro lo que pretende es cambiarnos a todos: alumnos, profesores, clientes, desarrolladores… y también directores de I+D, y responsables de agendas científicas y tecnológicas. Aspira a abrirnos la mente a todos para que sepamos reaccionar y anticiparnos a las consecuencias de una revolución transversal que ya tenemos encima.

Sobre los peligros de reducir el fenómeno de la Realidad Virtual y Aumentada exclusivamente al ámbito tecnológico, a tratarlo como una simple mejora en las interfaces gráficas de usuario, ya escribió a comienzos de los 90 el profesor Joseph Bates. El entonces investigador de la Carnegie Mellon se quejaba del enfoque con el que trabajaban sus colegas, mucho más enfocados -en plena revolución digital multimedia- en desarrollar potentes visores y software de renderizado que en explorar las posibilidades narrativas que ofrecía este medio con la ayuda de los sistemas de Inteligencia Artificial y un adecuado Diseño de Experiencias de Usuario.

Hoy día, arrastrados por esta novedosa oleada de dispositivos VR de consumo, podemos caer en la misma tentación y poner el foco exclusivamente en la carrera tecnológica que HTC, Facebook, Sony, Google, Apple y mucho otros están llevando a cabo, provocando en nuestras redes sociales un aluvión de noticias y rumores casi diarios. Sin embargo, son las aplicaciones reales, que poco a poco toman forma, y las propias estructuras comunicativas del medio, consolidándose

experiencia tras experiencia, las que guían de verdad la transformación social y cultural que la Realidad Virtual ha venido a traernos.

Por otro lado, como ingeniero y profesor en el ámbito de la tecnología sé que la única manera de poder alcanzar la "tierra prometida" de la Realidad Virtual, de crear -como tantas empresas y emprendedores buscan- una experiencia disruptiva que cambie la manera en que nos relacionamos con la realidad, es cruzando el desierto de la investigación, el estudio en profundidad y el aprendizaje de todas estas tecnologías -dispositivos, sistemas, APIs, aplicaciones, etc. que emergen a una velocidad endiablada en este novedoso área de la Informática.

Con esa convicción animo al lector a armarse de valor y enfrentarse a las páginas de este libro con la ilusión de saber que camina en la dirección correcta. Siempre desde el entendimiento de que la historia, los fundamentos y las distintas tecnologías de la Realidad Virtual y la Aumentada son el equipaje que necesitamos para aventurarnos con posibilidades de éxito hacia el verdadero objetivo del progreso tecnológico: mejorar la vida de nuestros semejantes.

Federico Peinado
Director de Narratech Laboratories
Universidad Complutense de Madrid

1

INTRODUCCIÓN

Nota del autor

Lo que ofrecemos a los lectores en este capítulo, es el resultado de un trabajo de investigación sobre los orígenes y la evolución de la Realidad Virtual desde sus inicios hasta la actualidad.

Hemos pretendido ser lo más rigurosos posible, pero en muchas ocasiones, sobre todo en lo referente a proyectos de investigación, no hemos podido tener acceso directo a las fuentes, la mayoría de ellas procedentes de USA y con una antigüedad de hasta sesenta o setenta años.

Esta "**breve historia de lo virtual**", ha sido reconstruida a base de consultar bibliografía, artículos, textos y documentación de fuentes accesibles al público, algunos de organismos oficiales que participaron en las propias investigaciones o de los propios autores de las mismas y otros de publicaciones o webs dedicadas a tecnología.

Una cosa que hemos observado durante este proceso de investigación, es que hay mucho baile en las fechas, a veces es difícil precisar de forma exacta en qué momento determinado apareció un prototipo, durante que periodo se realizó una determinada investigación o incluso nos ha llegado a ser francamente difícil obtener una imagen de alguno de los prototipos de los que hablamos para poder mostrárosla. En ocasiones, hemos encontrado información contradictoria y varias fechas, en estos casos, hemos optado por incluir la información que hemos estimado más fidedigna teniendo en cuenta el espacio temporal, las fuentes consultadas, etc. Lo que no quiere decir necesariamente que no hayamos podido sufrir alguna equivocación.

En algunos casos, hemos podido contrastar que se hicieron varias versiones de algún prototipo determinado, en distintos años y por ello, si contrastáis la información e investigáis más allá de lo que os ofrecemos en este libro, os encontrareis que, bajo el mismo nombre, podréis encontrar dispositivos que pueden ser diferentes o que su aparición data de años que no se corresponde con lo que aquí os contamos. Como norma general, hemos tomado la imagen y la fecha de aparición del primer dispositivo construido.

Otra cosa con la que hemos tenido alguna dificultad es con la identificación de la autoría de algunas imágenes, debido a su antigüedad, no nos ha sido posible averiguar quiénes son los titulares de los derechos, si es que los hay, no obstante, las imágenes solo se ofrecen con propósitos ilustrativos.

En definitiva, creemos que hemos conseguido mostrar a grandes rasgos la evolución de la tecnología de la Realidad Virtual de una forma bastante fidedigna y en cualquier caso, esperamos que disfrutéis tanto leyéndola como nosotros escribiéndola.

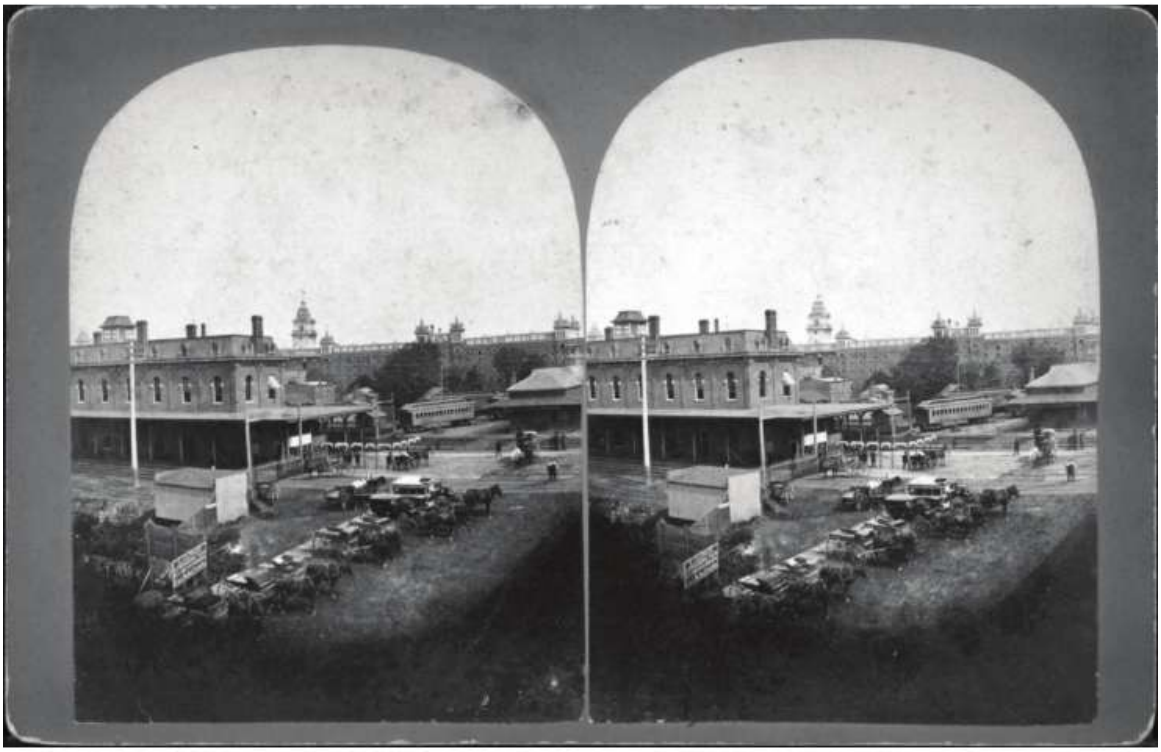
1.1 UNA BREVE HISTORIA DE LO VIRTUAL

Realidad aumentada o Realidad mixta son términos que llevamos oyendo desde hace relativamente poco tiempo, ambos parten desde uno que es bastante más antiguo, el de realidad virtual. Este concepto, aunque se comenzó a llamar así en los años 80, tiene más tiempo del que en principio nos pudiésemos imaginar, pues las primeras investigaciones en este campo se hicieron en los años 50 y las primeras máquinas de realidad virtual estuvieron disponibles a principios de los 60.

Las primeras máquinas estaban basadas, como en la actualidad, en la estereoscopía o visión estereoscópica. Sin entrar por el momento en mucho detalle, diremos que consiste en disponer de dos imágenes desde dos puntos de vista distintos, cada una de los cuales se correspondería al punto de vista de uno de los ojos y que al mostrarle a cada ojo su correspondiente imagen por separado, el cerebro las ordena creando la sensación de tridimensionalidad.

El uso de la estereoscopía es mucho más antiguo que la realidad virtual, en la siguiente imagen podemos ver dos fotografías para ser vistas en 3D mediante un visor estereoscópico, las imágenes datan de 1875 y forman parte de una colección del fotógrafo Robert N. Dennis que está compuesta por algo más de 42.000 imágenes estereoscópicas y que puede consultarse de forma digital en la página web de colecciones digitales de la Biblioteca Pública de New York desde el siguiente enlace:

https://digitalcollections.nypl.org/collections/robert-n-dennis-collection-of-stereoscopic-views



Si observamos detenidamente la imagen y prestamos atención a los detalles de los bordes, podremos apreciar como en una fotografía están ligeramente desplazados con respecto a la otra para que cada ojo tenga un punto de vista diferente cuando mediante el visor se encargue de mostrar a cada ojo la imagen que le corresponde de forma separada.

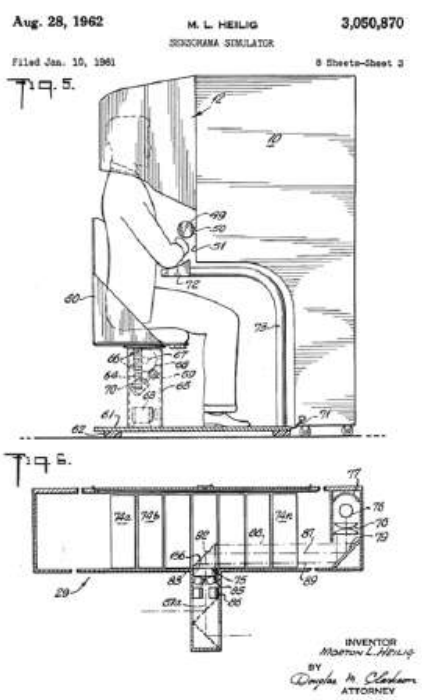
La estereoscopía se utiliza mediante diversas técnicas, una de las cuales hemos utilizado todos nosotros con casi total seguridad, se trata del anaglifo. Un anaglifo es una única imagen que contiene las dos imágenes correspondientes a los dos puntos de vista de cada ojo mezcladas, pero impresas en distinto color y que se puede observar mediante un visor que oculta una de las imágenes mezcladas a uno de los ojos.

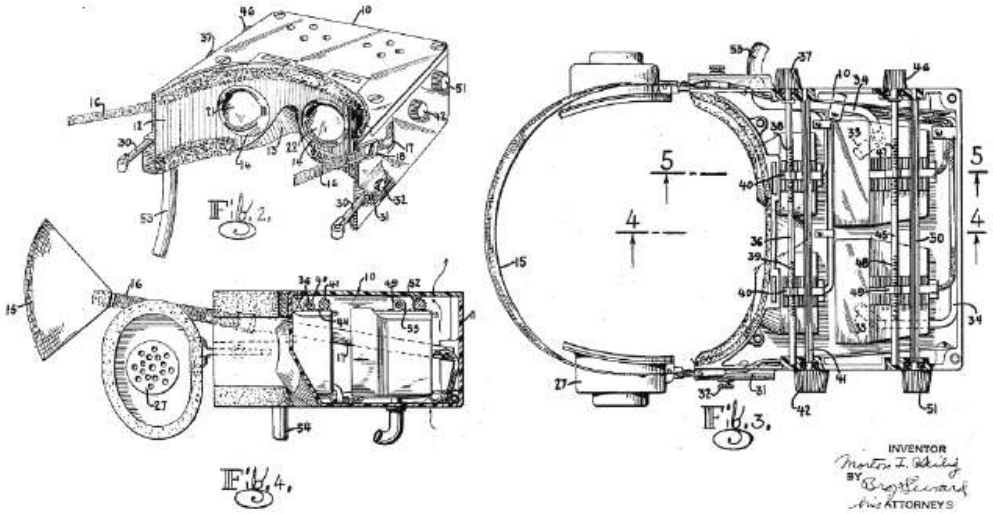
Lo habitual es que el anaglifo se imprima en colores rojo y azul y que el visor consista en unas gafas con un cristal rojo y otro azul, de esta forma el cristal rojo oculta a su ojo la parte roja del dibujo y el cristal azul oculta la parte azul al suyo, viéndose una imagen distinta con cada ojo.

Las imágenes, que están convenientemente desplazadas, son visualizadas de forma independiente por cada ojo y procesadas por nuestro cerebro que las une obteniendo una imagen con sensación de tridimensionalidad, como en una imagen real. Esta es la razón por la que las personas que han perdido la visión en un ojo no pueden percibir adecuadamente la tridimensionalidad y tienen dificultades en el cálculo de las distancias.

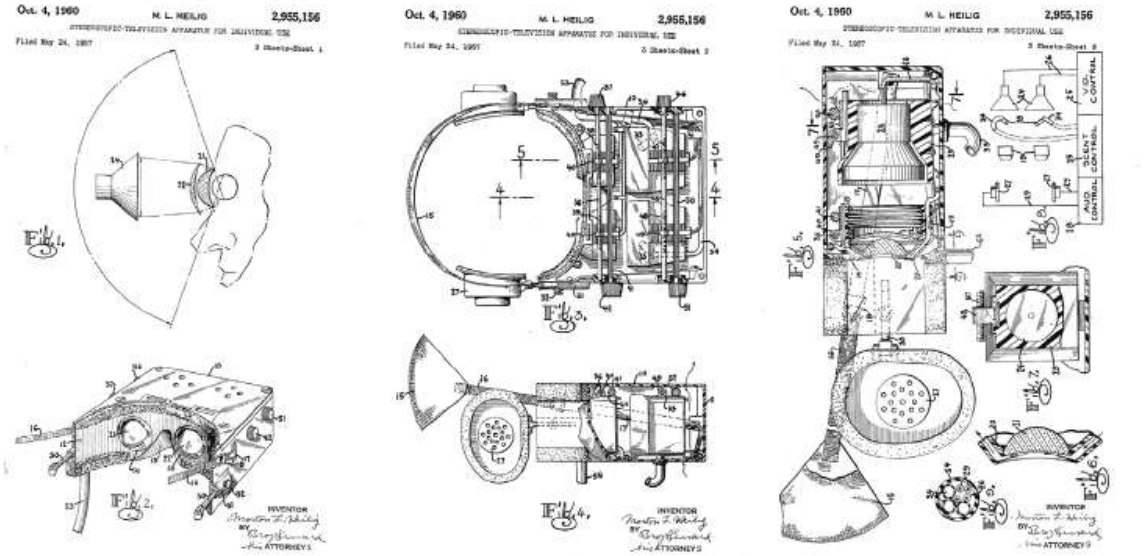
1.1.1 Los inicios de la Realidad Virtual

Partiendo de esta técnica, aparecieron las primeras máquinas de realidad virtual, la primera, inventada en 1957, aunque la patente es de 1962, fue la **Sensorama**. Esta máquina, creada por Morton Heilig, permitía disfrutar de "**Experiencias Teatrales Multisensoriales**" como las denominaba su autor, ya que ofrecía experiencias inmersivas de cine en 3D mediante la proyección estereoscópica en color, acompañadas de sonido estéreo, vibraciones, un sistema de emisión de aire para que el usuario sintiese el viento en la cara e incluso la posibilidad de emitir ciertos olores que aumentasen la experiencia de inmersión. La **Sensorama** era totalmente mecánica.





Posteriormente, el mismo creador fabricó un **HMD**, que no era interactivo y solo permitía la visualización de imágenes, llamado Telesphere Mask. **HMD** es el acrónimo, en inglés, de Head Mounted Display o "visualizador montado en la cabeza", un aparato que normalmente tiene forma de casco o de gafas y que nos permite ver, al menos imágenes en 3D, aunque los actuales disponen de muchísimas más funciones, en este libro nos referiremos a estos aparatos habitualmente como gafas.



Morton Heling realizó otras investigaciones, debido a que su oficio era el de productor, director y cámara de cine, sus investigaciones iban orientadas a conseguir una experiencia lo más real posible en este campo, Creó una versión ampliada de la Sensorama, el Experience Theater, un cine con una pantalla curva, que mostraba imágenes en 3D, con imágenes periféricas, sonido direccional, aromas, viento, temperatura variable e inclinación de los asientos. Ninguno de los inventos de Heling llego a ser comercializado.

En 1965, Ivan Sutherland, escribió un artículo titulado "The Ultimate Display", en el que analizó las interfaces hombre-maquina existentes en aquel momento y teorizó sobre un sistema que permitiría la inmersión del usuario en un mundo virtual generado por computadora con el cual podía interactuar. Hay que considerar que en aquel momento la capacidad gráfica de los ordenadores era muy limitada, en el artículo se mencionan cosas como que sería útil que se pudiesen dibujar curvas simples o que no están desarrolladas para el uso directo el rellenado de figuras, lo que nos da una idea de en qué situación se encontraba la capacidad gráfica de un ordenador en el año 1965. En el artículo también trata la tridimensionalidad

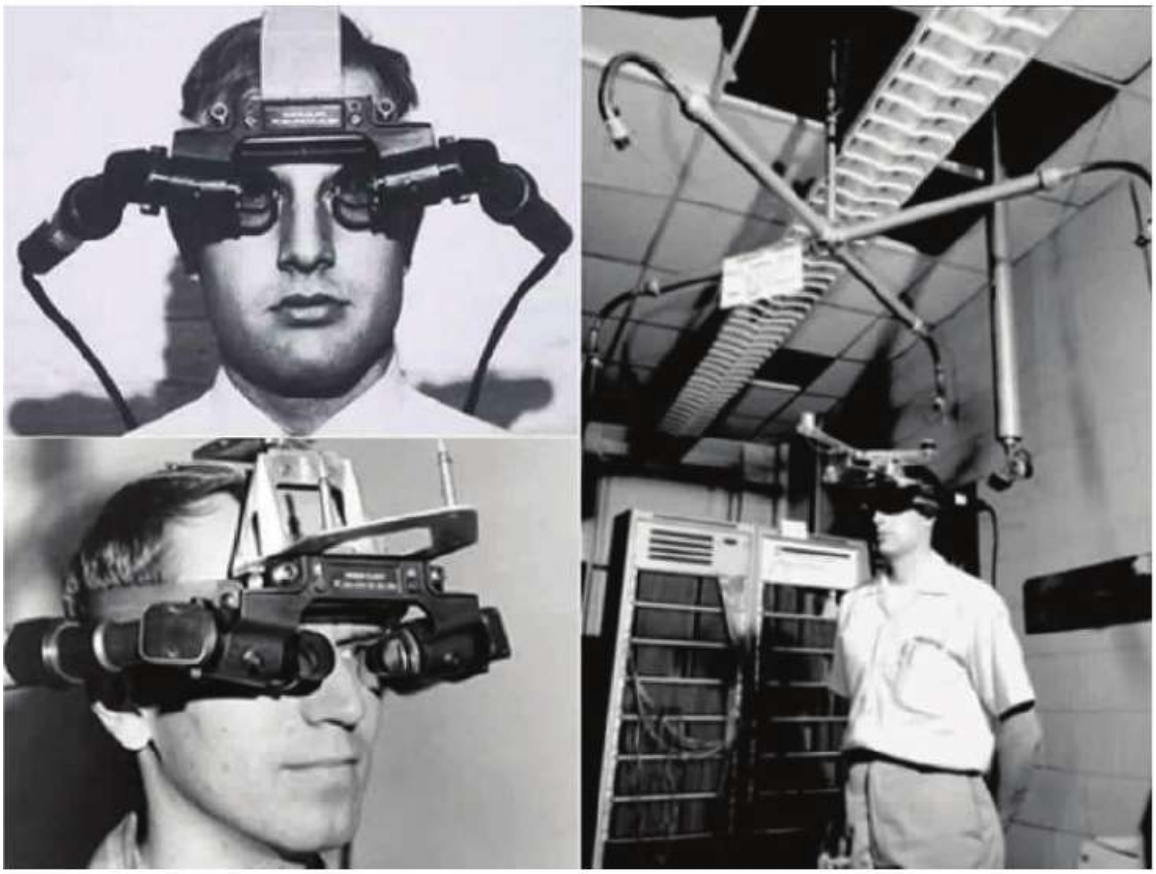
y la diferencia de trabajar con una pantalla en 2D, los gestos y periféricos como los lápices ópticos que en aquella época eran prácticamente experimentales, el uso de mandos y la posibilidad de utilizar la vista para controlar el ordenador.

Todos estos conceptos, son, hoy en día, habituales en cualquier sistema de realidad virtual.

También en el año 1965, la empresa Philco Corporation inventó el primer casco o HMD con visión estereoscópica, al que llamó Headsight. Su utilización estaba orientada a la videovigilancia estando conectado a una cámara de seguridad y tenía la capacidad de detectar los movimientos de la cabeza del operador y controlar la cámara mediante los mismos.

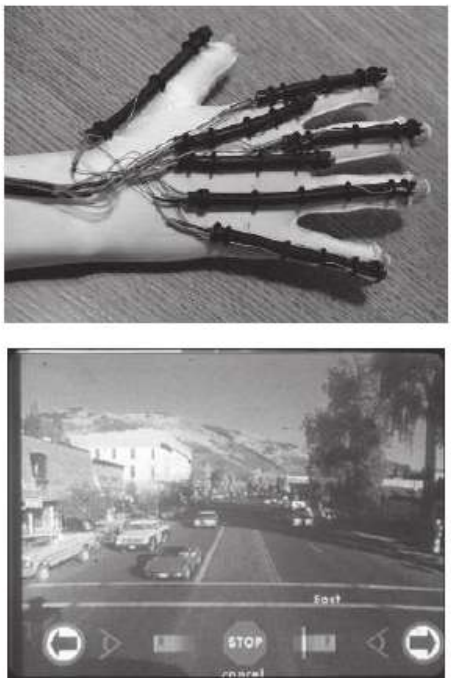
Algunos años después, en 1968, Ivan Sutherland con la ayuda de uno de sus estudiantes, Bob Sproll crean un casco, llamado "The Sword of Damocles" ("La espada de Damocles"), que permitía mover las cámaras rastreando el movimiento de los ojos y la cabeza del usuario. Este casco pone en práctica las bases que estableció en su artículo "The Ultimate Display" tres años antes.

En aquel momento, la tecnología impedía que el casco tuviese un peso lo suficientemente ligero como para que el usuario lo pudiese portar, por lo que debía estar suspendido del techo.



Desde el año 1969 hasta mediados de los años 70, un artista llamado Myron Kruegere, creó varias obras de arte digitales en un laboratorio de la Universidad de Connecticut, consistentes en entornos generados informáticamente que interactuaban con las personas sin utilizar gafas ni guantes. Los proyectos, llamados GLOWFLOW, METAPLAY y PSYCHIC SPACE sirvieron para perfeccionar la tecnología de detección de movimientos a la que llamó VIDEOPLACE y que plasmo en 1983 en el libro "Artificial Reality".

VIDEOPLACE usó proyectores, cámaras de vídeo y hardware específico logrando crear siluetas de los usuarios en las pantallas para que interactuasen entre ellos, aunque estuviesen en salas separadas y que también pudiesen interactuar con objetos, lo que ofrecía una sensación de presencia, aunque los usuarios no estuviesen juntos.



Durante los años 70 y hasta mediados de los 80, se realizan multitud de experimentos; entre ellos destacan algunos como el proyecto **Glove**, de Richard Syre, que monitorizaba el movimiento de las manos y de los dedos mediante sensores de luz, el **Aspen Movie Map**, que realiza un tour virtual en la ciudad de Aspen, en Colorado, de forma bastante parecida a como funciona actualmente **Google StreetView** o el simulador **VCASS** (**Visually Coupled Airborne Systems Simulator**), consistente en una maqueta de una cabina de un avión de combate con un casco, con el que el piloto podía realizar los entrenamientos mediante un simulador de vuelo.

Algunos de estos proyectos o experimentos, tenían como objetivo la virtualización, otros, como por ejemplo el HMD Headsight, no trabajaban con nada que fuese virtual, pero establecieron o definieron conceptos que después se aplicaron en la Realidad Virtual, todo esto que hemos visto y muchos proyectos más, pues hubo multitud de ellos, fueron aportando conceptos y definiendo las características que un sistema de realidad virtual debería tener y entonces, fue cuando se comenzó a hablar de Realidad Virtual.

1.2 LA REALIDAD VIRTUAL

En el año 1987 se utilizó por primera vez el término **Realidad Virtual**, cuando Jaron Lanier, fundador de **Visual Programming Laboratory Research**, una empresa de San Francisco, creó un área de investigación con ese nombre.

Esta empresa creo distintos productos con tecnología de Realidad Virtual: varias gafas, llamadas **EyePhone**, como la **EyePhone 1** o la **EyePhone HRX** y guantes como el **DataGlove**. En las siguientes imágenes se pueden ver las gafas **EyePhone 1** y las **EyePhone HRX**, con los guantes **DataGlove**.

Estos productos fueron comercializados, no únicamente prototipos con fines experimentales o creaciones a medida como, por ejemplo, los simuladores de entrenamiento de pilotos para el ejército del aire. Como es lógico, en aquel momento esta tecnología era extremadamente cara; el precio de las gafas variaba entre 10.000 y 50.000 dólares y el precio de los guantes se situaba en torno a los 9.000, con la extrapolación correspondiente, teniendo en cuenta que corría el año 1987, ahora podríamos estar barajando cifras aproximadas de entre 22.000 y 111.000 dólares para las gafas y 20.000 dólares para los guantes.

En la segunda mitad de los años 80, la informática sufre un crecimiento muy importante, los ordenadores crecen en capacidad de procesamiento, capacidad gráfica y se abaratan a la vez que se empiezan a popularizar los ordenadores personales.

Esto provoca varias cosas, en primer lugar, al disponer de más capacidad de procesamiento, más capacidad gráfica, al comenzar la miniaturización, toda la informática en general comienza a evolucionar ya que es posible desarrollar sistemas más complejos, si pensamos en veinte años atrás, en el artículo "**The Ultimate Display**", Ivan Shutterland estaba teorizando sobre el dibujado de curvas y el rellenado de figuras en el apartado gráfico y su primer HMD estaba sujeto al techo,

mientras que en este momento ya se comercializan sistemas relativamente ligeros y la capacidad gráfica ha evolucionado enormemente.

Además, el que se popularicen los ordenadores personales, implica que aparece en escena un elemento muy importante que tendrá mucho que ver en la evolución de la Realidad Virtual en los siguientes años: los videojuegos. A finales de los 80 y principio de los 90, empresas como Nintendo, Sega o Atari, en competencia directa con los ordenadores personales de 8 bits como el Sinclair ZX Spectrum, el Amstrad CPC o el Commoddore64, entre otros, inundan los hogares de consolas con el objetivo de hacerse un hueco en el mercado del ocio. Como veremos más adelante, todas estas empresas hicieron sus correspondientes aportaciones a la realidad virtual con el objetivo de mejorar la experiencia de juego para los usuarios de sus consolas.

Mientras finaliza la década de los 80, aparecen otro par de prototipos interesantes, como ya hemos comentado, desde mediados de los 80 la capacidad gráfica de los ordenadores aumenta notablemente, lo que hace que se realicen multitud de investigaciones orientadas a definir y mejorar las interfaces hombre-máquina, la NASA llevaba ya unos años investigando en este terreno y entre 1988 y 1990, crea el sistema **VIEW**, acrónimo de **Virtual Interface Environment Workstation**.

El sistema VIEW consiste en unas gafas con visión estereoscópica **EyePhone**, un guante, el **DataGlove** y un traje, llamado **DataSuite**. En la imagen podemos ver un prototipo de gafas del año 1985

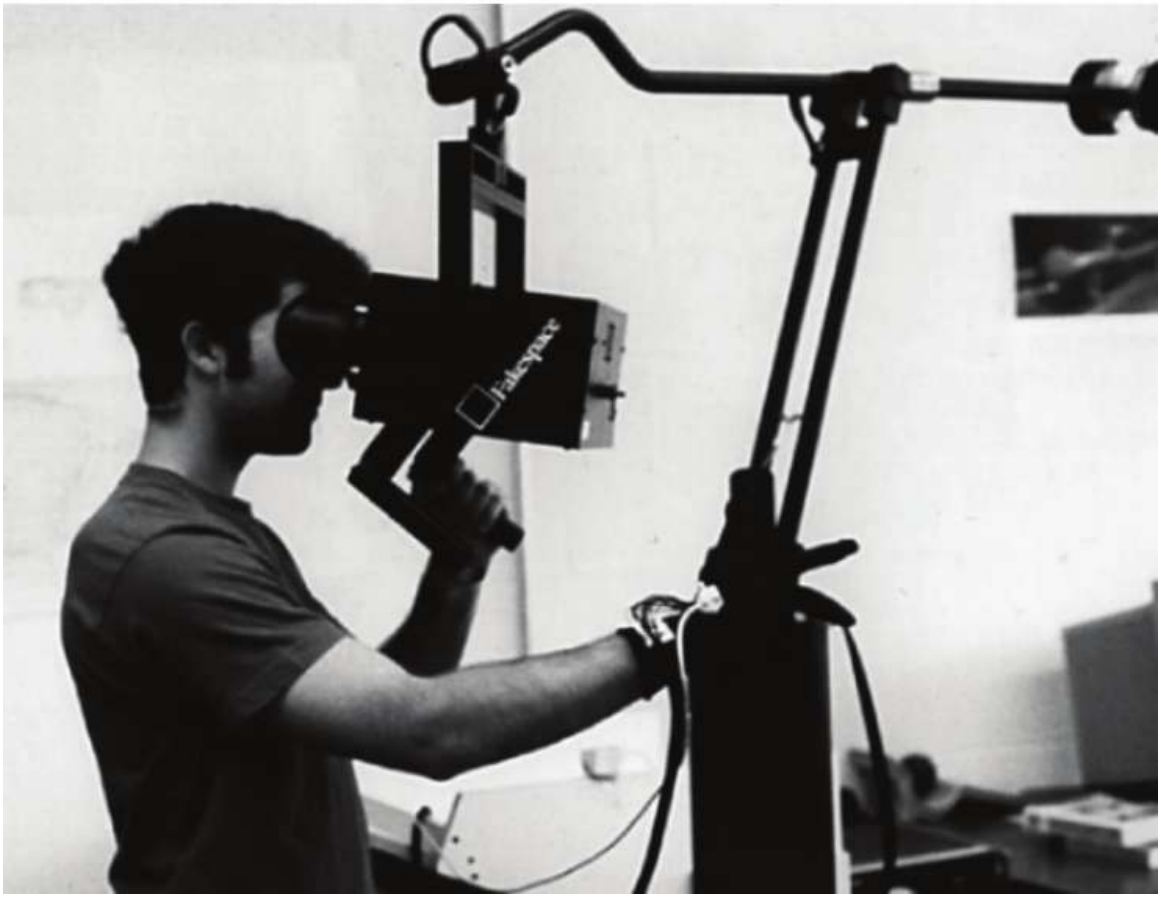
Las gafas, pueden presentar una imagen real retrasmitida por cámaras o un sistema virtual generado por ordenador con el que el usuario puede interactuar. Esta interactuación se realiza por medio del guante o **DataGlove**, que recoge toda la información de los movimientos de la mano del usuario mediante sensores y los transmite a un ordenador que la interpreta y genera una copia de la mano de forma

virtual integrándola en la imagen que se visualiza mediante las gafas. El **DataSuit** es un traje equipado con sensores que informan del movimientos, los gestos y la orientación espacial del usuario.

Quizá el lector se haya dado cuenta de la similitud de los nombres del sistema **VIEW** y de los productos comercializados por la compañía **Visual Programming Laboratory Research**, y es que todos aquellos productos comerciales tuvieron correlación con estos prototipos y de hecho la compañía **Visual Programming Laboratory Research** comercializó un kit similar al del sistema **VIEW** llamado **RB2 Virtual Environment**.

En las imágenes siguientes podemos ver el desarrollo **VIEW** de la NASA y el **RB2** comercial

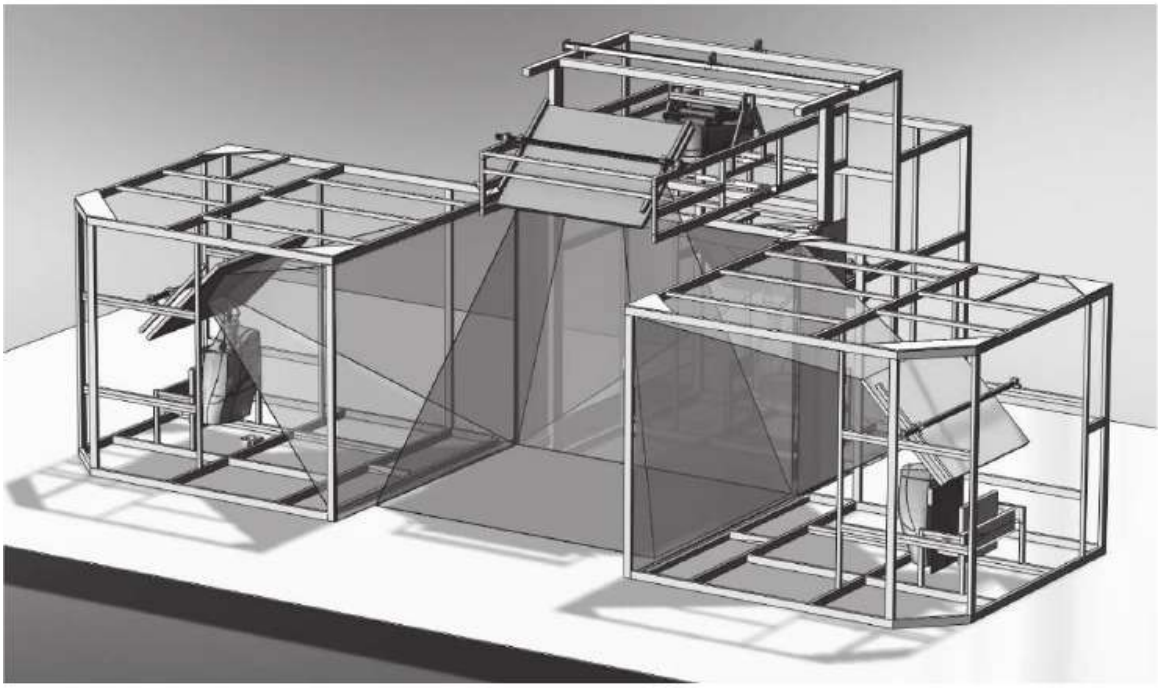
En 1989, la compañía **Fakespace Systems Inc.** crea el dispositivo **BOOM**, acrónimo de **Binocular Omni-Orientation Monitor**, este dispositivo se encuentra acoplado en un chasis, donde el dispositivo cuelga de un brazo, dentro del dispositivo se encuentra un sistema de visión estereoscópica y el usuario observa el mundo virtual a través de unos binoculares que dan acceso visual al sistema estereoscópico.



El brazo del que cuelga el dispositivo dispone de sensores que controlan el movimiento de la cabeza y además, el dispositivo puede ser movido por el usuario en un determinado ángulo para guiarlo por la escena virtual que se representa con él.

En 1991, científicos del **Electronic Visualization Laboratory** de la Universidad de Illinois en Chicago, crean un entorno de Realidad Virtual Inmersiva llamado **C.A.V.E.** o **Cave Assisted Virtual Environment**. Este entorno se basa en una sala oscura con una habitación en su interior en la que hay proyecciones en sus paredes.

Las paredes de esta habitación son en realidad pantallas sobre las que se proyectan imágenes provenientes de proyectos de alta resolución que se reflejan a través de espejos. Además, se pueden proyectar hologramas de objetos que pueden ser rodeados y vistos por todas sus caras. Todas las proyecciones son tridimensionales y el usuario necesita utilizar unas gafas especiales para ver las imágenes, que son de tipo estereográfico. En la siguiente imagen se puede ver la estructura de un sistema C.A.V.E.



En el **Centro de I+D+i en Eficiencia Energética, Realidad Virtual, Ingeniería Óptica y Biometría**, de la **Universidad Politécnica de Madrid**, existe un sistema C.A.V.E. de cinco caras. La información sobre este sistema se puede consultar en la página web le tiene dedicado en *https://www.cedint.upm.es/es/infraestructura/caver-5-caras-i-spacer*. En la siguiente imagen se puede ver dicho sistema:

1.3 LA REALIDAD AUMENTADA

A principios de los 90 se comienza a barajar un nuevo concepto que consiste en enriquecer la realidad con elementos virtuales e información mostrándola simultáneamente a través de las gafas o los dispositivos empleados,

En 1990, **Thomas Caudell** y **David Mizel,** dos ingenieros de la compañía **Boeing**, **diseñan** y crean un prototipo consistente en unas gafas transparentes que combinaban la detección de elementos del mundo real y el posicionamiento de la cabeza. Con esta tecnología superponen un diagrama de cableado en una posición específica en un tablero de instrumentos de un avión en el mundo real para ayudar al montaje del cableado del tablero.

Los resultados de esta tecnología son presentados en la **25 Conferencia Internacional de Ciencia de Sistemas** en Hawái en al año 1992, en la que es llamada **Realidad Aumentada** por primera vez.

Esta tecnología ya se había comenzado a utilizar antes de este año, por ejemplo, en el proyecto **K.A.R.M.A.** o **Knolewdge based Augmented Reality for Maintenance Asistence**, un software realizado por la **Universidad de Columbia**, que utilizaba un **HMD** para presentar el manual de usuario de forma tridimensional sobre las impresoras para ayudar en los procesos de mantenimiento.

Un accesorio virtual es una superposición de información sensorial aumentada sobre la percepción del usuario de un entorno real con el fin de mejorar el rendimiento humano en la operación del sistema.

Virtual Fixtures o Accesorios Virtuales en inglés, fue un proyecto desarrollado por Louis Rosenberg en 1992 en los Laboratorios Armstrong de la Fuerza Aérea de los Estados Unidos, y es considerado el primer sistema de Realidad Aumentada inmersivo.

Las superposiciones sensoriales virtuales se pueden presentar como estructuras reales, de tal forma que el usuario las perciba plenamente en el entorno real del espacio de trabajo. Las superposiciones sensoriales virtuales también pueden ser abstracciones que tienen propiedades y que no es posible presentar como una estructura física

El concepto de superposiciones sensoriales es difícil de visualizar y hablar, como consecuencia se utiliza la metáfora del accesorio virtual.

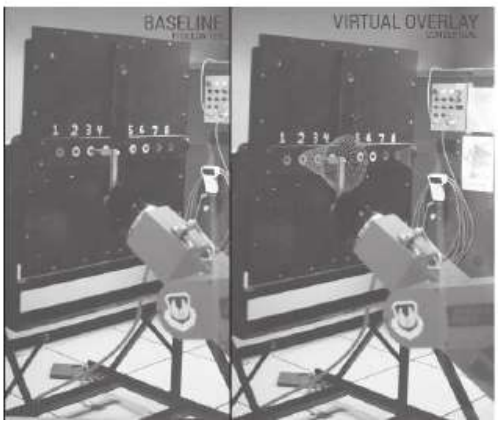
Para entender lo que es un accesorio virtual, es una analogía con un elemento físico real, el ejemplo que normalmente se emplea para explicar esto es el de una

regla: normalmente para una persona el trazar una línea totalmente recta es algo bastante difícil ya que intervienen factores como el movimiento involuntario de la mano, la rigidez de la articulación, la percepción espacial de la persona y varias cosas más que hacen en mayor o menor medida que la línea trazada sea más o menos precisa pero muy difícil que llegue a ser totalmente recta.

Sin embargo, con un accesorio tan simple como una regla, cualquier persona puede trazar una línea recta rápidamente y de forma precisa. En este caso el accesorio sería la regla y si tuviésemos un sistema de Realidad Aumentada en el que precisásemos trazar líneas rectas horizontales, un accesorio virtual equivalente a la regla podría ser un soporte controlado por el sistema, que nos impidiese mover la mano hacia arriba o hacia abajo para evitar que la línea se torciese y por lo tanto lograríamos una mejora en la realización de nuestra tarea de trazar una línea recta.

En el proyecto realizado por Rosemberg en 1990, se utilizaron dos robots, controlados por un exoesqueleto que el usuario llevaba acoplado en la parte superior del cuerpo. La experiencia de inmersión consistió en incluir un sistema de visualización para que el usuario viese los brazos del robot en la ubicación exacta sus brazos. El resultado fue una experiencia de inmersión en la que el usuario movía sus brazos, mientras veía los brazos del robot en el lugar donde deberían estar los suyos. El sistema utilizaba las superposiciones sensoriales generadas por computadora en forma de barreras físicas simuladas, campos y guías, diseñadas para ayudar al usuario mientras realiza tareas físicas reales.

Las pruebas realizadas por Rosemberg demostraron que se podían lograr mejoras significativas en el rendimiento de las tareas en el mundo real proporcionando superposiciones inmersivas de realidad aumentada a los usuarios.



A lo largo de los años 90, la Realidad Aumentada se utiliza ampliamente en el ámbito del entrenamiento y la formación y especialmente en sistemas de simulación para el entrenamiento militar. En el año 1993, el contratista militar LORAL Corp. conjuntamente con el **United States Army Simulation**, realiza la primera demostración en el **Training Technology Center** de equipos para entrenamiento con sistemas de Realidad Aumentada incorporados para ayudar en el aprendizaje en los simuladores. A partir de este momento, esto será una constante en el entrenamiento militar y la formación mediante Realidad Aumentada será utilizada en otras áreas en las que el aprendizaje es complejos y se realiza mediante simuladores como en la aeronáutica o en ciertos sistemas industriales.

A finales de esta década, aparece un nuevo concepto de Realidad Aumentada, la Realidad Aumentada Espacial o **SAR**, acrónimo de **Space Augmented Reality**, consistente en que los elementos virtuales que complementan a los objetos en el mundo real no están limitados a un usuario en concreto y por lo tanto no se muestran en unas gafas o en un dispositivo independiente, sino que están disponibles para todo el mundo que acceda al entorno real que dispone de capacidades de Realidad Aumentada. Esto tiene como ventaja que el usuario no tiene que llevar equipo encima y que varios usuarios pueden estar trabajando a la vez con el mismo elemento, pero evidentemente la infraestructura necesaria para lograr esto es mucho más costosa y compleja, y crece dependiendo de lo complejo que sea el Sistema de Realidad Aumentada.

Un sistema de este tipo fue desarrollado por primera vez por cinco científicos del departamento de Ciencias de la Computación de la Universidad de Carolina del Norte y su trabajo presentado en el **First International Workshop on Augmented Reality** celebrado en San francisco en 1998, bajo el título **Spatially Augmented Reality**.

1.3.1 La Realidad Virtual y Aumentada en los videojuegos

Hemos querido dedicarle un apartado separado a los videojuegos porque estos han contribuido en gran medida al desarrollo de la realidad virtual, por una parte, han contribuido a que el gran público comenzase a tener acceso a este tipo de tecnología, ciertamente de una forma recortada en sus inicios y por otra parte, debido a que el gran público tiene unas características muy determinadas en cuanto a capacidad económica, espacio disponible, etc. ha obligado a que la realidad virtual sea paulatinamente más económica y más portable.

Vamos a hacer un pequeño recorrido desde finales de los años 80 hasta la primera década del siglo XXI, momento en el que la realidad virtual y aumentada se estandariza y de alguna forma los sistemas comienzan a dejar de estar separados entre sistemas profesionales o de investigación y domésticos, salvo algunas excepciones debido a su elevado coste o al requerimiento de tecnología especializada para una tarea industrial en concreto.

Como ya hemos comentado anteriormente, en los años 80, junto con la popularización de los ordenadores personales, comenzaron a hacerlo también las consolas de videojuegos, destacando principalmente tres empresas: Nintendo, Sega y Atari.

La primera e intentar incorporar la realidad virtual a los videojuegos fue **Nintendo** que conjuntamente con la compañía **Mattel**, lanzó al mercado el guante **PowerGlove** en 1989 para su consola **Nintendo Entertainment System**. Este fue el primer periférico que recreaba los movimientos de la mano en la pantalla en tiempo real orientado al mercado doméstico. Funcionaba colocando tres sensores en el televisor donde estaba conectada la consola

El guante permitía jugar con él a todos los juegos existentes en la consola, ya que incorporaba accesorios típicos de los mandos tradicionales como botones de selección y disparo o una cruceta de movimientos que podía utilizarse desde el guante, pero también se crearon algunos juegos específicos para ser utilizados con él.

El guante fue un fracaso comercial, fundamentalmente porque se hicieron muy pocos juegos para ser usados con él, además de que al parecer, era bastante impreciso y su manejo era complicado, sin embargo, el **PowerGlove** obtuvo cierta fama ya que **Universal Pictures** produjo una película titulada **The Wizard**, que en España se tituló **El campeón del videojuego** y en Latinoamérica **El campeón del videojuego** o **Video genio** según el país.

Curiosamente, el sistema de controles que utilizaba PowerGlove, era, aunque más rudimentario, similar en su funcionamiento al que luego se utilizó en los mandos de las consolas WII y PlayStation 3, por lo que realmente fue pionero en esa tecnología, aunque no obtuviese un éxito comercial.

En 1990, **Sega** presentó el **Activator**, un periférico para la consola **Sega Megadrive** que permitía detectar los movimientos del cuerpo del jugador mediante el uso de rayos infrarrojos.

Este periférico estaba formado por ocho piezas que se montaban en el suelo, formando un octógono, en cuyo interior se sitúa el jugador que interrumpe los rayos infrarrojos con sus movimientos y de esta forma son detectados y enviados a la consola.

Básicamente, el octógono simula un mando y sus ocho lados se corresponden con el movimiento de este en sus cuatro direcciones y las diagonales corresponderían a los botones A, B, C y Start.

Al parecer, el control era impreciso y en algunos juegos era complicado jugar con precisión, además de que la eficacia del uso de los rayos infrarrojos se

podía ver afectada por elementos del entorno como la forma del techo o las lámparas. Debido a esto fue otro fracaso comercial y se retiró de la venta a los pocos meses de su lanzamiento.

En 1991 Sega anunció las gafas Sega VR. Se llegaron a presentar varios prototipos de las mismas, que incluían sonido estéreo, reconocimiento del movimiento de la cabeza para posicionar la cámara y pantallas LCD para cada ojo de 320 x 244 pixel con 64 colores y posibilidad de visualizar juegos en 3D. Estas características, que ahora pueden parecer escasas, eran las habituales de las consolas de juegos en aquella época.

Durante los tres años que estuvieron en desarrollo se llegaron a desarrollar cuatro juegos, que fueron presentados en distintas ferias y llegaron a ser probados por el público asistente, estos cuatro juegos iban a acompañar el lanzamiento de las gafas, previsto para 1994, pero finalmente el lanzamiento fue cancelado debido a los problemas de adaptación que tenían los jugadores, que consistían principalmente en mareos, debido al estado de la tecnología en aquel momento, por esta razón, Sega prefirió cancelar el proyecto antes que tener un más que posible fracaso comercial debido a estos problemas.

En 1995, Nintendo lo vuelve a intentar con la consola Virtual Boy, una consola portátil especialmente diseñada para juegos con gráficos en 3D. Se lanzó en USA y en Japón y no consiguió las ventas esperadas.

Nintendo informó que la falta de ventas fue debido a los gráficos, que solo se muestran en rojo y negro y que hubo falta de soporte de software por lo que no se desarrollaron muchos títulos, solo se hicieron diecisiete juegos, pero también es cierto que hay que jugar con ella sentado con la cabeza apoyada en ella, de forma similar a como cuando se usa un microscopio, al poco tiempo de jugar la visión es bastante molesta y a mucha gente le produce dolor de cabeza y realmente no es tan portátil ya que es incómoda de transportar y requería 6 pilas AA que consumía en pocas horas.

Solo estuvo a la venta un año y nunca llego a Europa ni a Latinoamérica. A pesar de que bajo varias veces de precio fue un fracaso comercial.

En el año 2000, en el **Wearable Computer Lab** de la **South Australia University** se crea un mod del popular juego **Quake** de la empresa **ID Software**. Este mod, permite jugar a este juego en realidad aumentada utilizando un **HMD** con imagen transparente fabricado por **Sony,** el modelo **Glasstron**, con una resolución

de 800 x 600, un portátil con un sistema operativo UNIX guardado en una mochila y un sistema GPS.

El equipo necesario para el juego, pesaba en torno a los 16 kilos aproximadamente y permitía jugar con los personajes virtuales en un entorno real.

Este mod se ha mantenido durante varios años y han existido varias versiones e incluso se podía mejorar considerablemente el rendimiento y la experiencia de juego si el jugador invertía algo de dinero en equipamiento, ya que se llegó a formar una comunidad bastante amplia alrededor del mod.

1.4 REALIDAD VIRTUAL, AUMENTADA Y MIXTA

En los ejemplos que hemos visto a lo largo de los apartados anteriores, nos hemos encontrado con diversas cosas: por una parte, la visión de una imagen estática de forma tridimensional mediante la estereografía, por otra, la visión de

una imagen en movimiento con una experiencia inmersiva total en cuanto a sonido, viento, sensación de movimiento e incluso olores. También hemos visto ejemplos de objetos virtuales moviéndose sobre una perspectiva real a través de unas cámaras que mezclan la visión de nuestro entorno con la estereografía de los objetos y todo esto aplicado a distintos campos, como los videojuegos o el cine.

Aunque muchos de estos ejemplos fueron prototipos experimentales y otros se quedaron en fracasos comerciales, lo cierto es que desde el año 2015, todo lo relativo a este campo ha evolucionado enormemente y se ha comenzado a separar en tres líneas diferenciadas (o en algunas ocasiones no tanto).

Así que finalmente se han establecido tres conceptos: el de realidad virtual, el de realidad mixta y el de realidad aumentada.

El mayor problema está en definirlos de forma separada, porque si bien los principios de algunos están muy claros, los de otros están sujetos a interpretación y a veces no está tan claro lo que puede ser realidad mixta, fundamentalmente.

En cualquier caso, parece que hay bastante consenso en las definiciones de realidad virtual y Realidad aumentada, aunque últimamente han aparecido sistemas que llaman de otra forma a lo que ya existía, creando aún mayor confusión entre los usuarios.

A continuación, vamos a intentar explicar en qué consiste cada uno de ellos, pero el lector tiene que tener en cuenta que esta explicación es fruto de la experiencia y opinión de los autores y que, en un momento dado, lo que nosotros podemos definir en este libro, por ejemplo, como realidad aumentada, puede ser considerado por otras personas como realidad mixta

1.4.1 La realidad virtual

Se define como realidad virtual un entorno que puede ser de apariencia real o no, que da al usuario la sensación de estar inmerso en él. Como norma general, este entorno es generado por un sistema informático y visualizado por el usuario mediante un dispositivo específico como pueden ser un casco o unas gafas y dependiendo del sistema y de lo elaborado e inmersivo que pretenda ser, puede estar acompañado de otros elementos como sensores de posición y movimiento, guantes, sonido 3D, elementos como mandos para desplazarse o manipular los objetos del entorno, etc.

En la siguiente imagen, podemos ver un sistema de realidad virtual de la Agencia Espacial Europea para el entrenamiento en el manejo de robots y vehículos espaciales

Por tanto, si utilizamos un sistema de realidad virtual, perdemos la conexión con el mundo real y todas las referencias de este.

Debido a que los dispositivos no permiten ver la realidad, en el caso de que necesitemos desplazarnos físicamente por el entorno virtual (ya que hay otra opción que es el "teletransporte", es habitual que se establezcan ciertos controles para evitar incidentes, por ejemplo, en el caso de las gafas HTC Vive u Oculus Rift, estas gafas definen un área de seguridad delimitada por dos sensores, uno en cada esquina, por la que nos podremos mover y al aproximarnos a los bordes, nos aparece una rejilla que nos indica que estamos a punto de sobrepasar los límites de dicha área de seguridad. En el caso de otras gafas, como las de la plataforma Windows Mixed Reality, no existen estos sensores, pero se puede definir esta área de seguridad marcándola en el suelo y hacer que las gafas la reconozcan recorriéndola con las gafas apuntando a la pantalla del ordenador mientras este está en un modo especial de reconocimiento. Este perímetro previamente reconocido es detectado por sensores que se encuentran en las gafas en vez de en las esquinas del perímetro.

En la imagen anterior, correspondiente al desarrollo de una sesión de entrenamiento para la extinción de un incendio, podemos apreciar que la persona que maneja el sistema no tiene una visión del entorno real, lo que está viendo se refleja en el monitor que hay a su izquierda, solo para que la gente que no está utilizando el sistema de RV pueda tener una visualización.

En este caso, este modelo de gafas, las HTC Vive, están conectadas a un PC que realiza el trabajo gráfico y de cálculo y todos los sensores de movimiento y posicionamiento se encuentran en las propias gafas, de tal forma que cuando la

persona se gira, mueve la cabeza o se desplaza, suministran la información al PC que calcula la presentación del entorno virtual y lo envía de nuevo a las gafas.

Como el movimiento real es limitado, se pueden utilizar dispositivos adicionales, por ejemplo, plataformas que detectan nuestros pasos y hacen que avancemos por el escenario, pero este tipo de dispositivos pueden afectar a los centros de equilibrio de algunas personas y provocar mareos u otro tipo de inestabilidad, por eso, la opción más habitual es disponer de un mando con el que se señala un punto en el escenario virtual y mediante la pulsación de un botón se nos "teletransporta" a la posición deseada.

En el caso de la imagen, al menos un mando es utilizado para manipular el extintor y la persona se puede mover libremente por el área de seguridad marcada por los sensores.

Si observamos la imagen, vemos que en el fondo hay un trípode que tiene uno de los sensores, el otro estará en la esquina opuesta. El otro mando puede ser utilizado para cualquier otra acción, por ejemplo, el desplazamiento.

1.4.2 La realidad aumentada

Al contrario que en la realidad virtual, en la realidad aumentada, no se está inmerso dentro del entorno, el dispositivo suele ser un teléfono móvil o una tableta desde la que vemos nuestro entorno real a través de la cámara y sobre ella se muestran elementos inexistentes que corresponden al entorno virtual, por lo que podemos definir la realidad aumentada como una combinación visual de elementos reales y virtuales que interaccionan entre ellos.

En la ilustración anterior, podemos ver un ejemplo de una aplicación de realidad aumentada, realizada por la compañía Radikal Dreamers, donde se ve a un visitante del Castillo de Burgalimar en la localidad de Baños de la Encina, en Jaén haciéndose una fotografía vestida con ropa medieval, al enfocar con el teléfono a la persona, que lleva un marcador colgado, la imagen se completa mediante la realidad aumentada con un modelo 3D texturizado que se superpone a la persona, dando la sensación de que lleva una ropa distinta y acorde con el castillo. La aplicación dispone también de unos controles para mover y cambiar el tamaño de las prendas y ajustarlas al cuerpo de la persona que se esté realizando la fotografía.

En este tipo de sistemas no es habitual que se utilicen sensores externos como guantes o plataformas de movimiento, aunque si se utilizan capacidades propias de los dispositivos como, por ejemplo, los sensores de posición o movimiento: el GPS o el giroscopio y a través de la programación se suelen simular situaciones tales como la aproximación a las fuentes de sonido gestionando el volumen o las distintas fuentes en función de la proximidad a las mismas.

Debido a que este sistema no es inmersivo y por tanto no perdemos la conexión con el entorno real ya que lo vemos tanto a través de la cámara del dispositivo como a través de nuestra propia vista si miramos alrededor, no es necesario disponer de un área dedicada por motivos de seguridad ni de un sistema que nos avise de la proximidad a los bordes de esta, como ocurre en los sistemas de realidad virtual en los que nos es necesario desplazarnos de forma física.

1.4.3 La realidad mixta

Y el tercer concepto es el de realidad mixta, este es un poco más complicado de definir, ya que en teoría comprende todo lo que se encuentre entre la realidad virtual, es decir la total inmersión sin contacto con el mundo real y la realidad aumentada, que es la combinación de elementos virtuales con el mundo real.

Aquí empiezan las discrepancias en cuanto a su definición, pues hay quien piensa que todo lo que sea realidad mezclado con virtualidad es realidad mixta, lo que implicaría que el uso de un mando, como por ejemplo el de unas gafas HTC, ya sería realidad mixta y hay quien piensa que la realidad mixta pasa de forma obligada por cumplir con las características de la realidad aumentada y ofrecer algo más.

Sin intentar definir un axioma, desde el punto de vista de este libro, estableceremos una diferencia clara entre la realidad virtual, la aumentada y la mixta: y es que, al tratar de combinar de forma activa, y no solo de que aparezcan juntos, los elementos reales y los virtuales, consideraremos obligado que la realidad mixta sea realidad aumentada con características adicionales.

Quizá la característica más importante, y que más defina la diferencia entre la realidad aumentada y la mixta, sea la capacidad de interactuar con el entorno mediante el mapeado espacial, esta funcionalidad consiste en que el dispositivo, que dispone de los sensores adecuados, es capaz de detectar volúmenes y distancias para obtener una nube de puntos con la que crear un modelo en 3D del entorno que nos rodea y de esta forma distinguir todos los elementos para interactuar con ellos.

Como ejemplo, en la siguiente imagen, se puede ver como la señorita interactúa con una ventana virtual colocada en una pared del mundo real, en la que despliega un menú con distintas aplicaciones realizadas por la compañía Bravent. Esta imagen, es para propósitos ilustrativos, en realidad una tercera persona nunca podría ver la ventana, excepto si llevase puestas otras gafas conectadas a las que esta utilizando la persona que está utilizando la ventana.

Con las tecnologías actuales, esto se puede conseguir de distintas formas, los sistemas de realidad aumentada/mixta incorporados en los sistemas operativos de Apple y Google, lo hacen analizando la imagen de la cámara conjuntamente con la información de posicionamiento del dispositivo, mediante una técnica que se conoce como Odometría Visual y cuyo funcionamiento se explica en el capítulo dedicado al ARKit de Apple, pero el reconocimiento de estos volúmenes solo es efectivo en parte. Hay otros dispositivos, de entre los cuales los más conocidos son las gafas HoloLens de Microsoft o los ya descatalogados dispositivos Tango, que disponen

de sensores de medición y proximidad que permiten elaborar una copia del entorno bastante precisa.

Microsoft ha homologado cinco fabricantes: Acer, Dell, HP, Lenovo y Samsung para que comercialicen gafas con unas determinadas características para que sean compatibles con la tecnología Windows Mixed Reality, la cual también incluye los dispositivos Hololens, pero las gafas de estos fabricantes no tienen ninguna posibilidad de analizar el entorno a excepción del control de la posición de las gafas, por lo que su funcionalidad es similar a otras gafas como las HTC o las Oculus que son considerados dispositivos de realidad Virtual y no de Realidad Mixta.

Sin embargo, Microsoft, considera que la Realidad Mixta es tanto Realidad Virtual utilizada conjuntamente con el uso de sensores para la posición, como Realidad Aumentada con el uso de sensores de medición y profundidad como en el caso de sus gafas HoloLens. Este es un caso claro de que el concepto de Realidad Mixta puede tener muchas apreciaciones.

1.4.4 Otras realidades

Como hemos podido ver se perfilan dos conceptos claramente. El primero, la "Realidad Virtual", donde todo los que nos rodea es ficticio y en el que conseguimos una inmersión total. Y por otro lado el concepto de "Realidad Aumentada", donde somos conscientes del entorno real que nos rodea en todo momento, y sobre el cual superponemos ("aumentamos") el contenido virtual adicional.

Aparte, existe un tercer concepto, el de Realidad Mixta, que está sujeto a interpretación y sobre el que, como autores, ya nos hemos posicionado.

Cuando hemos explicado este último concepto hemos hablado de que la Realidad Mixta puede comprender desde lo que deja de ser totalmente virtual hasta lo que no llega a ser totalmente real, este concepto, se conoce como **continuo de la virtualidad** y fue definido en 1994 por Paul Milgram y Fumio Kishino.

En base a esta teorización, aparecen otros conceptos con los que podemos encontrarnos en un momento dado y que, por tanto, vamos a repasar brevemente:

- **Virtualidad Aumentada**: Si la Realidad Aumentada es estar en un entorno real en el que vamos a incluir elementos virtuales que lo enriquezcan, la Virtualidad Aumentada es el caso inverso, estar en un entorno virtual en el que vamos a incluir elementos reales para su enriquecimiento.
- **Realidad Extendida**: Por Realidad Extendida se entiende el conjunto de entornos reales y virtuales, las interacciones entre los mismos y los dispositivos que se utilizan. Es decir, absolutamente todo lo que interviene en el sistema.

Estos conceptos son muy abstractos, demasiado en realidad, por ejemplo, si en un entorno virtual, supongamos algo como nuestro juego favorito con unas gafas Oculus, estamos introduciendo un elemento real como son los mandos Touch para interactuar en el juego ¿eso quiere decir que no es Realidad Virtual sino Virtualidad Aumentada? Posiblemente es así y la Realidad Virtual totalmente pura ni siquiera debería precisar de un mando, ya que este no es un elemento virtual, pero la verdad, pensar mucho en esto no creo que ni nos aporte, ni nos solucione nada, por lo que yo no me preocuparía demasiado por estas definiciones a no ser que el lector tenga un especial interés en teorizar sobre todo esto, que es una cuestión que tiene más que ver con la filosofía que con la tecnología.

1.5 LA INDUSTRIA 4.0 Y LA TRANSFORMACIÓN DIGITAL

Desde hace algún tiempo, seguramente venimos oyendo hablar, además de las "realidades" que ya hemos vistos de otras cosas como la "impresión 3D" o "impresión aditiva", el "big data" o el "Internet de las cosas" y posiblemente habremos oído que todo esto forma parte de la Industria 4.0 y que son herramientas de transformación digital o cosas similares.

Aunque no es el objetivo de este libro, si es cierto, por otra parte, que la Realidad Virtual, Aumentada y Mixta tienen algo que ver con esas herramientas que son el Big Data o el Internet de las cosas, entre muchas otras, y que también forman parte de los procesos de transformación digital de las empresas hacia la llamada Industria 4.0.

Por eso, vamos a intentar aclarar brevemente que es todo esto y que lugar ocupa la Realidad Virtual, Aumentada y mixta dentro de todo esto, así como la relación que mantiene con el resto de herramientas.

1.5.1 La Industria 4.0

Cuando oímos hablar de Industria 4.0, nos estamos refiriendo a una industria que cumple con los estándares de la cuarta revolución industrial, hasta ahora, en la historia de la Humanidad ha habido tres grandes revoluciones industriales, que son las siguientes:

- La 1ª revolución industrial, se inició en Inglaterra a mediados del siglo XVIII, con el invento de la máquina de vapor y se paso de una economía basada en la agricultura y el comercio a una economía basada en la industria y en la mecanización. Implica cambios demográficos, un gran auge del comercio y grandes cambios en infraestructuras y transportes al aparecer el barco de vapor y el ferrocarril y construirse carreteras y canales.

- La 2ª revolución industrial, se inicia a finales del siglo XIX y principios del siglo XX, aparecen nuevas fuentes de energías como el gas, el petróleo o la electricidad y nuevos sistemas de transporte como el avión y el automóvil y de comunicación, como el teléfono y la radio, también se descubren nuevos materiales y productos químicos y se desarrolla el capitalismo y por tanto cobra auge el sistema bancario, apareciendo a la vez, nuevas potencias económicas.

- La 3ª revolución industrial es mucho menos drástica que las anteriores y se basa en una evolución más pausada, se estima que se comienza a producir hace algo menos de veinticinco años, apoyándose en las tecnologías de la información y la comunicación y el desarrollo de energías renovables. Se puede decir que aún estamos inmersos en ella y de hecho en 2007 el Parlamento Europeo aprobó una declaración formal por la cual esta revolución industrial se sustenta en cinco pilares:

  - El cambio a energías renovables.
  - La conversión de los edificios en plantas de energía.
  - El uso del hidrógeno, las baterías y las tecnologías de almacenamiento de energía.
  - La aplicación de tecnología de distribución de energía eléctrica inteligente.
  - El transporte basado en vehículos eléctricos, híbridos y de baterías de combustible, utilizando como energía de propulsión la electricidad renovable.

Viendo como están las cosas actualmente y lo que se propone como 3ª revolución industrial, no parece que resulte muy coherente hablar de 4ª revolución industrial o Industria 4.0 ya que en la actualidad apenas hemos comenzado con la tercera, pero la realidad es que ha comenzado la cuarta revolución industrial simultaneando con la tercera y la razón es, que mientras que la tercera revolución industrial está más enfocada a la optimización y el uso de la energía, la cuarta está orientada a la optimización de la producción, creando lo que se conoce como "fábricas inteligentes".

1.5.2 Los habilitadores digitales

Si analizamos el apartado anterior, cuando enumerábamos las revoluciones industriales, podemos observar que todas ellas sin excepción, han tenido una serie de herramientas, normalmente en forma de avances tecnológicos, que han impulsado dicha revolución industrial.

Así, en la primera, la más evidente fue la máquina de vapor y lo que derivó de ella, como por ejemplo, el ferrocarril. En la segunda, las fuentes de energía, los nuevos materiales o los sistemas de transporte y comunicación y en la tercera, los cinco que hemos enumerado con anterioridad.

En la cuarta revolución industrial o Industria 4.0 también hay una serie de herramientas que son las que están provocando esta revolución.

Como es lógico pensar, no hay un número fijo, mañana pueden surgir nuevas y sería muy difícil enumerarlas todas, pero si es cierto que hay media docena que están claramente identificadas y de las que, con toda seguridad, hemos oído hablar. Estas herramientas es lo que se conocen, dentro de la Industria 4.0 como habilitadores digitales y son los siguientes:

- Internet de las cosas (IoT)
- Cloud Computing
- Fabricación aditiva o impresión 3D
- Big Data
- Ciberseguridad
- Tecnologías de visión.

1.5.2.1 INTERNET DE LAS COSAS

El Internet de las cosas, también conocido como **IoT**, acrónimo de **Internet of Things**, consiste en integrar en todos los elementos, sensores, circuitería electrónica, capacidad de conectividad y/o software, con el objeto de que el elemento en cuestión tenga la capacidad de comunicarse e intercambiar información utilizando Internet.

1.5.2.2 CLOUD COMPUTING

El cloud computing consiste en utilizar servicios a través de Internet, de forma que toda nuestra información se encuentra en servidores en Internet, sin necesidad de preocuparse por tener que almacenar información en nuestros sistemas.

Como toda la información se localiza dentro de Internet, todo el mundo puede acceder a la información, sin necesidad de disponer de una gran infraestructura.

1.5.2.3 FABRICACIÓN ADITIVA

La fabricación aditiva consiste en la adición de capas de material sobre una base para conseguir una pieza confeccionada de uno o varios materiales. Aunque habitualmente se usa este término como sinónimo de Impresión 3D, no es exactamente lo mismo, la impresión 3D suele referirse a la fabricación de piezas de tamaño limitado con materiales plásticos, mientras que la impresión aditiva se suele utilizar en entornos industriales y los materiales, aparte de no estar limitado a un único material, pueden ser de otro tipo que no sea plástico, por ejemplo, algunos metales.

1.5.2.4 BIG DATA

El BigData es la utilización de cantidades masivas de datos, que normalmente son tan grandes que no pueden ser organizados y gestionados por medios convencionales como una base de datos. La utilización de estos datos estaría orientada al análisis de los mismos y la toma de decisiones a partir de dicho análisis, para poder predecir fallos, realizar mantenimientos predictivos, etc.

1.5.2.5 CIBERSEGURIDAD

La ciberseguridad consiste en la protección de la infraestructura informatica y la información que contiene o que circula a través de las redes. Para garantizar esa protección, se aplican o usan distintos estándares, protocolos, métodos, reglas y herramientas para minimizar en lo posible los riesgos a los que puede estar sometida la infraestructura informática o a la información que esta contiene.

1.5.2.6 TECNOLOGÍAS DE VISIÓN

Las tecnologías de visión son precisamente el objeto de este libro, si bien la Realidad Virtual, Aumentada y Mixta no son la únicas. También hay otras como la Visión artificial o el reconocimiento de imágenes que se aplican en la Industria 4.0

1.5.2.7 VEAMOS UN EJEMPLO

Vamos a poner un ejemplo, totalmente ficticio de un proceso en el que van a intervenir varios habilitadores digitales.

Imaginemos por un momento a un operador de mantenimiento de una planta industrial que se dirige a comenzar su jornada laboral. Este operador es un ingeniero eléctrico encargado de realizar las labores de mantenimiento de los equipos eléctricos de una planta de producción.

Al comenzar su jornada laboral, el ingeniero recoge como parte de su equipamiento una tableta con un software que contiene las ordenes de trabajo que debe realizar en su jornada laboral.

Cuando inicia sesión en la aplicación, comprueba que tiene una orden de trabajo para realizar un mantenimiento preventivo, el sistema de información ha realizado un análisis de toda la información facilitada por todos los dispositivos que tiene almacenada correspondiente a los ocho años que la planta lleva en funcionamiento (**Big Data**) y ha encontrado un patrón repetitivo de tal forma que un determinado componente de un armario eléctrico ha informado diez errores por exceso de temperatura en las últimas ocho horas (**IoT**).

Tras el análisis de la información, el sistema ha comprobado que un 80% de las veces que se han producido esa cantidad de errores en ese espacio de tiempo, el componente se ha averiado, produciendo una parada completa del sistema de producción por lo que ha emitido la orden de trabajo para que sea sustituido de forma preventiva antes de que se produzca el fallo.

El operador realiza la sustitución del componente, cuyo stock es comprobado de forma automática por el sistema y que tras comprobar que las unidades se encuentran por debajo de las unidades mínimas de las que se debe disponer en el almacén, genera una orden de compra que es introducida en el sistema de información de la compañía para que el departamento de compras que está ubicado en la sede central realice el correspondiente tramite administrativo, esta petición se realiza de forma totalmente transparente para el operador mientras su dispositivo se encuentra conectado a internet (**Cloud Computing**). Esta petición que incluye información relativa a la instalación que es confidencial se envía de forma cifrada y debidamente protegida (**Ciberseguridad**)

Para poder realizar la sustitución del componente, el operador previamente ha tenido que acceder al armario eléctrico en el que se encuentra, como en la orden de trabajo tiene la ubicación del armario eléctrico, se dirige al mismo y como este tiene alojados ochenta y tres componentes, ubica el componente que debe ser

sustituido enfocando un marcador que se encuentra pegado en el armario eléctrico, una vez reconocido, el sistema señala el dispositivo a sustituir mediante un icono superpuesto sobre la imagen de la cámara (**Realidad Aumentada**), para comprobar que el componente que ha recogido del almacén para sustituir al que puede averiarse es correcto, el operador selecciona el icono en la aplicación de Realidad Aumentada y en la pantalla del dispositivo se le muestra toda la información técnica del componente, que coteja con el componente que tiene para asegurarse.

Una vez que ha sustituido el componente, accede a la agenda de mantenimiento del armario eléctrico desde la misma aplicación, comprobando que dos días más tarde hay previsto un ciclo de mantenimiento que consiste en comprobar el cableado y realizar una serie de mediciones. Aprovechando que ya ha accedido al armario y que está trabajando con él, que dicho ciclo de mantenimiento tiene un margen de realización de +/- 4 días y que tiene disponible toda la documentación del mismo, accede al esquema eléctrico y realiza las comprobaciones y mediciones prevista para dos días después, dando por realizadas esas tareas de mantenimiento que no serán incluidas en la orden de trabajo que se genere con posterioridad.

En este breve ejemplo, hemos utilizado cinco de los seis habilitadores que hemos visto, solo nos ha faltado la fabricación aditiva. Este seria un ejemplo de un proceso bastante digitalizado ya que todas las operaciones se hacen apoyándose en habilitadores, pero es bastante ilustrativo del uso de los habilitadores digitales en un proceso, en este caso de mantenimiento industrial.

1.5.3 La Transformación Digital

Después de saber que es la Industria 4.0 y que son los habilitadores digitales, parece algo claro que para implementarlos en una compañía debe haber un proceso cuando menos, bien diseñado y meditado que permita a la compañía pasar del estado en el que se encuentra a un estado en el que aplique las herramientas o habilitadores digitales a sus procesos de producción.

Este proceso de cambio es lo que se llama Transformación Digital. Su complejidad puede ser muy variable, casi podríamos afirmar que existen tantas transformaciones como compañías y esto es por una razón muy simple: la transformación no afecta solo a los procesos productivos, sino a todo el funcionamiento de la organización.

Es necesario considerar que modificar el fin ultimo de la compañía: la producción, implica necesariamente cambios, por ejemplo, en el personal que interviene en esa producción, en los procedimientos que utiliza para realizar sus

tareas, en la adquisición de las materias primas, en los departamentos que gestionan las compras o el personal, etc.

Otro aspecto importante, es que los cambios no son solo tecnológicos, la aplicación de la tecnología implica, por ejemplo, distintas formas de comunicarse o cambiar los protocolos de actuación.

Hay muchos más aspectos a considerar y uno que no es nada desdeñable es la diferencia generacional en cuanto a la tecnología, en estos momentos confluyen en las empresas tres generaciones, tecnológicamente hablando: las personas nacidas antes o durante la década de los 60 que tuvieron que adaptarse a la implantación de la informática, los nacidos en la década de los 70, algunos de los cuales comenzaron su relación con los sistemas de información con el boom de los ordenadores personales de los 80, estos, están mucho más acostumbrados a la tecnología, pero aún así hay mucha gente de esa generación que ha tenido que adaptarse, la generación posterior de los años 80, conocida como **millenials**, ya considera normal el uso de dispositivos electrónicos y la automatización de muchas tareas y desde luego, comparada con la primera generación, la adaptación a un cambio del modelo a uno profundamente digitalizado, le será enormemente más fácil. Desde siempre han conocido las compras on-line, las redes sociales, los dispositivos móviles, etc. Y es importante destacar que para 2025 serán más del 70% de la fuerza laboral del mundo.

Esto quiere decir que la Transformación Digital tiene mucho que ver con las relaciones humanas, la interacción entre las personas y el cambio en los modelos de relaciones y de comunicación cambiará el enfoque que se les pueda dar a los distintos procesos en la organización.

Debido a la profundidad de los cambios la transformación digital puede ser muy simple o muy compleja y todas las implicaciones han de ser tenidas en cuenta antes de abordar la transformación.

No obstante, aunque ahora sea el momento en el que la Transformación Digital se está produciendo a lo grande, hace ya un tiempo que estamos inmersos en ella, aunque no lo hayamos notado, incluso no solo en las empresas, sino en nuestro día a día.

Por ejemplo, si tenemos un correo electrónico de Gmail o de Outlook.com, Guardamos nuestras fotos en un servicio en la web y las compartimos mediante una red social, estamos utilizando el Cloud Computing.

Cuando en el año 2009 apareció un juego que con unos patrones en unos cartones nos permitía jugar con unos exóticos animales con poderes llamados Invizimals o hace un par de años, todos los niños andaban como locos cazando

Pokemons con el móvil, en realidad estábamos utilizando Realidad Aumentada. Muchas marcas comerciales como Ikea, Lego o Carrefour han realizado sus catálogos con la posibilidad de visualizar sus artículos en 3D y animados y ahora, la cadena de supermercados DIA, mediante un acuerdo con la productora de la ultima película de la saga Jurassic Park, a lanzado una colección de cromos en al que los dinosaurios cobran vida con un móvil o una Tablet. La Realidad Aumentada es una de las tecnologías que más tiempo lleva en nuestra vida cotidiana.

También hay quien ya tiene su impresora 3D en casa, no se puede decir que eso sea Fabricación Aditiva, claro, pero el abaratamiento de costes ha hecho que ahora mismo una impresora 3D sea asequible, aún es algo compleja su utilización, pero como todo, es una cuestión de tiempo. Como es lógico, otras cosas como el Big Data, tardarán más tiempo en llegar al gran publico debido a que tienen una utilidad más empresarial, o quizá no se utilicen nunca en el ámbito doméstico y otras como el IoT lo harán, pero más lentamente.

Para finalizar este capítulo, me voy a permitir citar un párrafo del libro escrito por mi amigo y colega **Ramón Cabezas**, titulado "**PETRA. Una metodología para transformar humana y digitalmente su empresa**", cuya lectura, en caso de abordar un proyecto de este tipo, recomiendo encarecidamente, es el siguiente:

"Es más que evidente que la pregunta que tengamos que contestar sea algo incomoda: ¿Por qué una empresa que es líder, que arroja unas espléndidas cifras de beneficios y que tiene la competencia bajo control, necesita transformarse? Porque es evidente que, si una empresa va mal, es débil, necesita algún cambio mejor antes que después. Veremos que no es fácil tener una respuesta simplista a esa pregunta, pero antes nos haremos una pregunta más, que aún pareciendo similar es radicalmente distinta: ¿Para qué necesita una empresa transformarse? Así como la primera pregunta habla sobre posibles explicaciones que se identifique en la situación actual que justifiquen el cambio, la segunda pregunta se refiere a escenarios sobre la situación futura tras la transformación que nos convenzan de que se va a cambiar a mejor, puesto que el cambio por el cambio, la transformación precipitada e inconsciente aumenta la probabilidad de encaminar nuestra empresa al desastre.

En resumen, siempre puede ocurrir que, incluso con las mejores intenciones del mundo, algún consultor le proponga realizar transformaciones en su empresa que concluyan en escenarios a los que el responsable no quiera llegar, aún cuando al principio la idea pudiera parecer interesante. Asegúrese el lector de entender el porqué y el para qué, o si no, no se queje luego."

2

EJEMPLOS PRÁCTICOS DE APLICACIONES

Una vez que ya hemos visto en el capítulo anterior en que consiste la Realidad Virtual, Aumentada y Mixta, de que tecnologías hace uso y cuál ha sido su evolución hasta llegar a la situación actual, hemos considerado que sería muy interesante seleccionar distintos ejemplos que ilustrasen el uso de esta tecnología.

Para eso, hemos contactado con las empresas más destacadas dentro del desarrollo de aplicaciones que aplican tecnologías de Transformación Digital, haciendo hincapié en las que utilizan tecnologías de visión y más concretamente en las que utilizan Realidad Virtual, Aumentada y Mixta, que son el objeto de este libro.

Con el objetivo de dar la visión más amplia posible, hemos intentado que las aplicaciones y las empresas seleccionadas abarquen el mayor número de campos, por tanto, hay aplicaciones de simulación, de aprendizaje, que a la vez sirven para recoger datos y analizarlos, juegos que en algunos casos son educativos, algunas de museología, etc.

Como es obvio, nos es imposible abarcar toda la casuística que pueden abarcar este tipo de aplicaciones, pero lo que sí hemos conseguido es mostrar aplicaciones de Realidad Virtual totalmente inmersivas, como el ejemplo de "Realidad Virtual para análisis de conducta humana en situaciones de emergencia" de la empresa 6D Lab, "Baños Roller Coaster" de Radikal Dreamers o el juego "The Overman" de Narratech.

También hemos incluido varios ejemplos de Realidad Aumentada donde se mezcla la parte real y la virtual, como en el juego educativo "Enigma Galdiano" de PadaOne Games, en la aplicación de museología "Museo Sorolla "de ARS VIVA o las implantaciones de las herramienta "Space 1" de la compañía "OverIT" realizadas

por EPTISA para el mantenimiento de líneas de producción, inspección remota o soporte técnico en los procesos de instalación.

Dentro de la Realidad Mixta, tenemos también un gran número de aplicaciones que mostraros, con ellas se puede subir a los coches de Toyota del Rally RACC y correr con ellos en la aplicación desarrollada por Bravent y también podremos ver una aplicación para la localización de personas dentro de un edificio, llamada HoloSitum desarrollada por la empresa Kabel.

Esto es solo una parte de lo que podrás ver en las siguientes páginas, ya que hay muchos más proyectos. Esperamos que con ellos puedas tener una visión completa de todas las posibilidades que te puede ofrecer la Realidad Virtual, Aumentada y Mixta.

Todos los ejemplos mostrados son reales y en la información de cada empresa hay una persona de contacto que te puede informar acerca de los proyectos o de las tecnologías en las que están especializados. No dudes en contactar con ellos para lo que necesites.

- Radikal Dreamers: *http://www.radikaldreamers.com*
- Narratech: *http://www.narratech.com*
- 6D Lab: *http://6dlab.com*
- ARS VIVA: *http://www.arsvivaexperience.com*
- Kabel: *http://www.kabel.es*
- Bravent: *http://www.bravent.net*
- EPTISA: *http://www.eptisa.es*
- Padaone Games: *http://www.padaonegames.com*

2.1 RADIKAL DREAMERS

Radikal DREAMERS	*http://www.radikaldreamers.com* Chema Martínez / Antonio Martínez Iberia,33. La Carolina. E-23200. Jaén *cm@radikaldreamers.com* am@radikaldreamers.com

Radikal Dreamers nace en el 2013 con la idea de aplicar las nuevas tecnologías a la creación de entornos virtuales que ayuden a trasmitir un mensaje

de forma diferente. La aplicación de tecnologías tan novedosas como la Realidad aumentada, Realidad Virtual, Holografía o detección de movimiento dan un abanico de posibilidades enormes y novedosas para nuestros clientes.

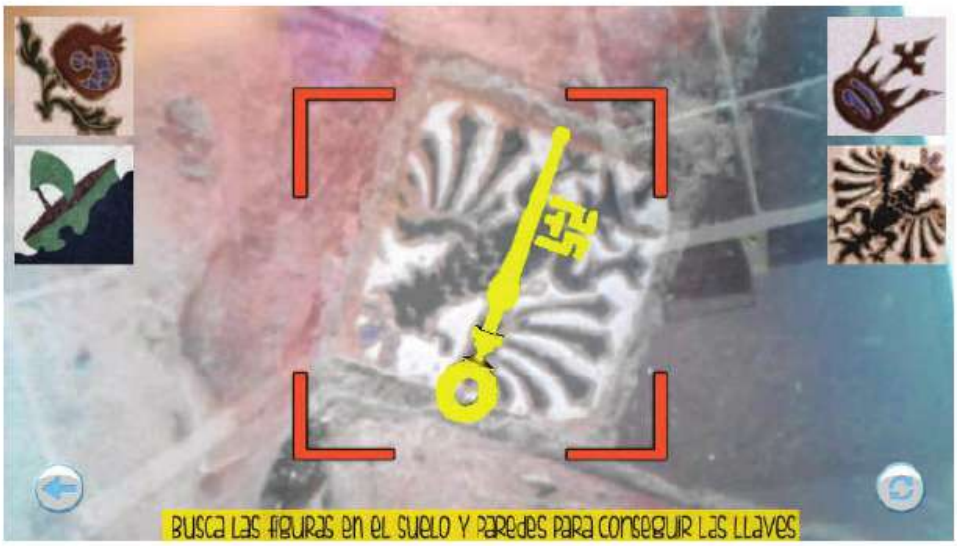
Disponemos de una variada cartera de clientes que apuestan por un producto que no te deja indiferente y que cubre desde el entorno Museológico hasta el industrial, donde actualmente estas tecnologías están tomando un gran protagonismo.

Contamos con productos tan variados como Videojuegos, Aplicaciones de Advergaming, Simuladores, Interactivos o plataformas Web

2.1.1 La Alhambra - El castillo rojo

Desarrollada en colaboración con el promotor turístico Granavisión, es la primera audioguía infantil con realidad aumentada del mundo y propone una gymkana por el monumento de La Alhambra donde el niño consigue una serie de hitos desbloqueando llaves a través de Realidad Aumentada y minijuegos. Publicada en el 2014 fue una APP realmente novedosa en su momento, entre otras cosas, por el uso de la Realidad Aumentada.

En la búsqueda de las llaves escondidas en la ciudadela te irás encontrando con importantes habitantes de la época que, situados en las cuatro zonas más representativas de la Alhambra, serán los encargados de recibirte, guiarte y mostrarte la historia y curiosidades de las diferentes estancias, palacios y jardines.



"La Alhambra. El Castillo Rojo" consiguió el premio "The AppTourism Awards 2016" como mejor APP en turismo cultural y enogastronómico en fitur 2016.

2.1.2 La Carolina Virtual

La Carolina Virtual es una aplicación desarrollada en colaboración con Musaraña Gestión de Museos dentro de un proyecto financiado por Diputación de Jaén, la Junta de Andalucía y el Ayuntamiento de La Carolina. Propone un recorrido por la ciudad de La Carolina. Este recorrido está marcado con 15 hitos de Realidad Aumentada mediante unas baldosas numeradas ubicadas en diferentes puntos de interés de la ciudad.

Mediante tecnología de posicionamiento, la aplicación nos permite ubicar un terminal móvil dentro de un mapa con los 15 hitos marcados. En cada uno de los hitos se podrá disfrutar de la recreación 3D de ese lugar en el pasado a través del giroscopio y acelerómetro. También se incluye la posibilidad de girar mediante gestos táctiles.

Durante el recorrido nos acompañará Joaquinito Steiner, un trabajador de la torre de perdigones de ascendencia colona, ya que sus abuelos vinieron de Alemania en la época de la fundación de La Carolina. Aunque se trata de un personaje ficticio, se basa en una familia real: Los Steiner, original de la región de Constanza.

Además de las escenas de Realidad Aumentada y con objeto de que el usuario tenga información turística accesible se geoposicionarán otros hitos en la aplicación como Restaurantes, Hoteles y otros puntos de interés.

2.1.3 Castillo de Burgalimar

Aplicación desarrollada para la musealización del Castillo de Burgalimar en Baños de la Encina. Como en el caso anterior, el proyecto has sido realizado en colaboración con Musaraña Gestión de Museos y financiado por Diputación de Jaén y el Ayuntamiento de Baños de la Encina.

La aplicación funciona en el castillo con el Bluetooth encendido y va marcando el recorrido a través de diferentes hitos que vienen posicionados mediante iBeacons, unos dispositivos que detectan la proximidad del teléfono y envían información al mismo.

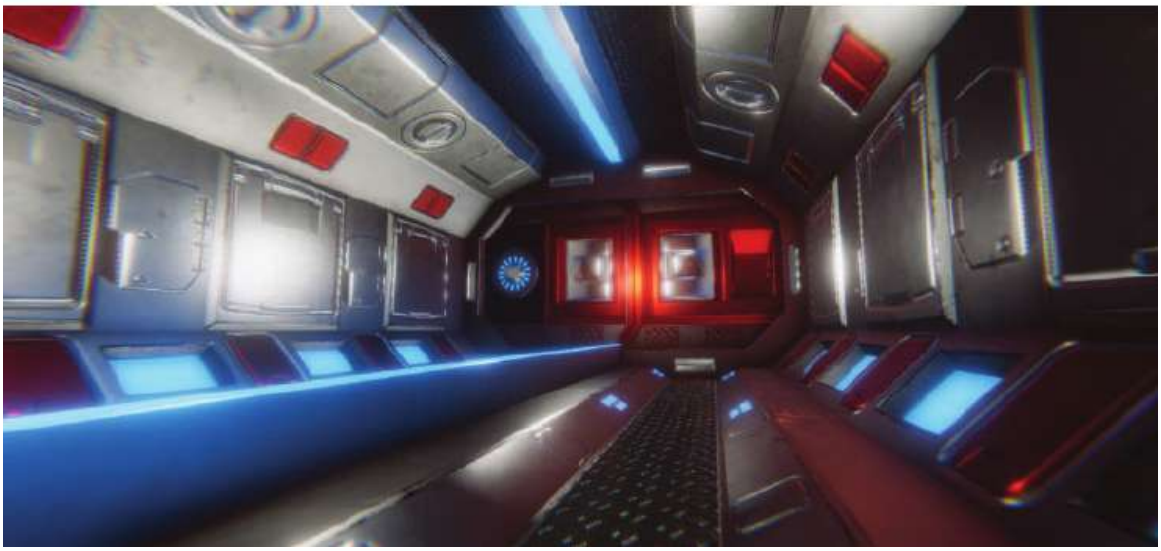
Entre los distintos hitos disponemos de varias escenas recreadas mediante Realidad Virtual, la visualización de un antiguo templo romano usando un marcador cúbico o la posibilidad de hacerte fotos vestido como auténticos guerreros islámicos usando marcadores y una funcionalidad que permite que la vestimenta se pueda ajustar a la persona fotografiada ajuste.

2.1.4 Baños Roller Coaster

Este proyecto es un simulador con Realidad Virtual de una montaña rusa. El proyecto ha sido financiado por la Diputación de Jaén y el Ayuntamiento de Baños de la Encina.

Se trata de conectar mediante una atracción el Castillo de Burgalimar (Monumento emblemático de Baños de la Encina y conocido por todo el mundo) con el yacimiento arqueológico de Peñalosa (Importante yacimiento de la época argárica pero no tan conocido).

Se propone un recorrido entre los dos lugares mediante una atracción que va dando saltos en el tiempo a través de portales dimensionales.



Como hardware para la experiencia se cuenta con:

- Chasis construido a propósito dividido en dos módulos. Uno para el asiento del usuario y otro auxiliar para el PC y otro hardware.
- Gafas Oculus Rift
- PC Gaming
- Vibración en el asiento sincronizada con la experiencia
- Ventilador para simular el viento sincronizada con la experiencia
- Placa electrónica con microcontrolador desarrollada a propósito.
- Iluminación led externa
- Vinilado

La experiencia se puede disfrutar en Baños de la Encina (Jaén)




2.2 NARRATECH

NARRATECH LABORATORIES

http://www.narratech.com

Federico Peinado

Despacho 409 – 411. Facultad de Informática. Universidad complutense de Madrid.
Cl. Prof. José García Santesmases, 9.
E-28040 Madrid.

info@narratech.com
email@federicopeinado.com

+34 91 394 7599

Narratech es una organización dedicada a la investigación y el desarrollo tecnológico de videojuegos narrativos surgida al amparo del grupo de investigación NIL de la Universidad Complutense de Madrid. Su misión principal es contribuir al desarrollo y la formación en tecnologías innovadoras, preferentemente libres y gratuitas, que expandan las posibilidades de los creadores de experiencias interactivas. Entre estas tecnologías se encuentran las relacionadas con la Inteligencia Artificial, el Análisis de Datos y la Realidad Virtual. También realiza formación y consultoría para empresas, proyectos de divulgación en medios de comunicación y, aunque sus miembros son especialistas en Ingeniería Informática, tiene una fuerte vocación interdisciplinar y vínculos con el mundo del Arte y las Humanidades.

Los proyectos que aborda varían mucho en alcance y naturaleza, siendo desde iniciativas propias de colectivos universitarios a otros financiados por entidades como Telefónica Educación Digital o Fundación BBVA. La mayoría de los miembros trabajan en Madrid, como es el caso del director, Federico Peinado, aunque hay colaboradores en otras ciudades españolas e incluso en Tokio, desde donde trabaja Nahum Álvarez, responsable de investigación.

El equipo ha estado, desde los primeros prototipos, muy interesado en los desafíos de diseño que plantea esta nueva ola de dispositivos de Realidad Virtual inmersiva, trabajando principalmente con Oculus Rift y HTC Vive. Se han presentado trabajos y prototipos de investigación tanto en congresos nacionales, como el *Congreso de la Sociedad Española para las Ciencias del Videojuego* (CoSECiVi), como internacionales de reconocido prestigio como el *International Conference on Advances in Computer Entertainment* (ACE). Sin embargo, muchos de sus miembros no provienen del ámbito universitario, sino directamente de la industria de los videojuegos, como es el caso de los desarrolladores Maximiliano Miranda (Wildbit Studios) y Gabriel Peñas (Animatoon Studio), así como del compositor y especialista en sonido Manuel López (Protocol Games). En el equipo también hay perfiles del área de marketing y producción, como Jorge Osorio o Jaume Esteve, así como colaboraciones con otros artistas *freelance*.

2.2.1 The Overman; In her footsteps

Aunque la mayoría son herramientas destinadas para terceros o prototipos de investigación, el proyecto más conocido de Narratech es un videojuego de realidad virtual para Playstation VR llamado The Overman. Se trata de un ambicioso proyecto en desarrollo del que se mostró de forma exclusiva un anticipo jugable en la feria *Barcelona Games World*, The Overman: In Her Footsteps, reconocido como uno de los 20 mejores títulos de los *Premios Playstation 2016*.

En esta aventura inmersiva, breve pero intensa, el jugador despierta aturdido en el lúgubre sótano de una mansión francesa de los años 40. Aunque a duras penas puede recordar lo ocurrido, pronto será el protagonista de una serie de horribles sucesos. Se trata del domicilio de un científico que está siendo utilizado por el ejército nazi para crear una sustancia capaz de transformar a cualquiera de sus hombres en una bestia invencible. La última amenaza de un mundo que se encuentra ya de por sí atrapado en una vorágine de maldad.

Para crear una experiencia de horror en primera persona innovadora se invirtió mucho esfuerzo en cuidar la narrativa, incluyendo un impactante giro argumental, así como en la ambientación audiovisual que debía ser realista y convincente. El ocultismo nazi y la experimentación con seres humanos durante la Segunda Guerra Mundial es un trasfondo estimulante y propicio para una experiencia de horror orientada a jugadores adultos. Para recrear esta atmósfera en un breve espacio de tiempo se optó por trabajar con la tecnología de Unreal Engine. Dos especialistas en modelado de personajes, los hermanos José y Juan Chavarría, decidieron ocuparse íntegramente de todo el apartado visual (escenario, objetos interactuables, iluminación, etc.) para conservar la coherencia, aunque también colaboró la artista conceptual Patricia Liébana. Manuel Garrido compuso una sutil banda sonora, los efectos de sonido, cruciales en un juego de horror, corrieron a cargo de Daniel Nuñez, y la voz del narrador, Julio López, fue grabada en los estudios sevillanos Alta Frecuencia.

El diseño del juego, incluyendo las mecánicas de supervivencia, los eventos especiales y el modelo de navegación por el escenario y de la interacción con objetos fue realizado principalmente por Federico Peinado, con asesoramiento de Carlos L. Hernando. Sufrió muchos cambios, volviéndose más personalizable cada vez debido a lo variado de las convenciones que se iban proponiendo en esos primeros meses de este renacer de la realidad virtual. De la programación, enfocada inicialmente en PC y Oculus Rift, se ocupó un equipo extenso de personas incluyendo a Sergio Calero, Alejandro Rivas, Daniel Novillo, Alejandro Zabala y Guillermo Alonso. Otros, como Yusef Abubakra, Diego Valbuena, Félix Redondo, Miguel Andrés, Javier Fernández, David Martín ayudaron a prototipar, dar soporte y sobre todo testear el juego, dedicando muchas horas de inmersión forzosa en esta prisión virtual del terror.

2.3 6D LAB. SIX DIMENSION

http://www.6dlab.com

Juan Rivas
Daniel Rubio

hola@6dlab.com

Avda. Gregorio Peces Barba, 1.
E-28919. Leganés. Madrid.

Six Dimensions S.L. (en adelante 6D LAB) es una empresa fundada en el año 2015 con el objetivo de desarrollar nuevas tecnologías para su aplicación en el ámbito de la Realidad Virtual (RV), Realidad Aumentada (RA) y Realidad Mixta (RM).

6DLAB transforma los conocimientos de 1+D de los Grupos de Investigación de la UC3M en productos tecnológicos de mercado.

2.3.1 Realidad Virtual para análisis de conducta humana en situaciones de emergencia

En este proyecto, 6D Lab ha desarrollado, junto con Fundación Mapfre y el Colegio Oficial de Ingenieros Técnicos Industriales, una experiencia tecnológica innovadora: la simulación de una emergencia de incendio, empleando los últimos avances y tendencias en Realidad Virtual.

Su objetivo principal es estudiar la conducta y la reacción de diferentes colectivos de población (niños, adultos, ancianos y discapacitados) frente a un incendio en un edificio virtual. Dichos comportamientos son fácilmente extrapolables a las que se producirían en una emergencia real, siendo así una herramienta tecnológica pionera dirigida hacia la innovación en los procesos de training formativo, desarrollando un nuevo producto para la prevención y educación durante la evacuación de edificios en situaciones de emergencia, siendo capaz de identificar, analizar y evaluar las variables o factores que condicionan el éxito o fracaso del sujeto durante el proceso de evacuación.

El proyecto se ha realizado en diferentes fases, que abarcan desde la realización de la planimetría 2D hasta la programación de una base de datos para registrar y almacenar las variables cuantitativas y cualitativas correspondientes.

La primera fase abarcó una extensa recogida de información del edificio institucional escogido para recrear la experiencia: el Parque Científico de la UC3M, en Leganés (Madrid). Se realizó la toma de datos, fotografías y planimetrías, así como un inventario de todos los elementos de mobiliario y materiales existentes en el edificio, para que la reproducción virtual fuera lo más precisa posible.

Tras esto, se comenzó la fase de modelado del edificio y mobiliario, teniendo muy presente lo que se quería reproducir y la normativa legal aplicable en este caso (Código Técnico de la Edificación), poniendo especial hincapié en el cumplimiento del Documento Básico de Seguridad en Caso de Incendio. A continuación, se inició el proceso de texturizado de los modelos 3C), empleando software de 3D-Painting como Substance Painter. El modelo del edificio se texturizó en el software Unreal Engine.

A la hora de reproducir las condiciones que se viven en un incendio real, 6D lab contó con la colaboración de miembros del Cuerpo de Bomberos de Alcorcón, que aportaron su experiencia y conocimiento del comportamiento fuego y el humo para asegurarse el mayor realismo de este incendio virtual.

Una vez modelado el edificio en 3D y obtenido el espacio virtual hiperrealista, el principal reto consistía en desarrollar todo el back-end de la App, con el objetivo de crear una experiencia de usuario sencilla e intuitiva, a la vez capaz de parametrizar todos los datos que resultan claves para la investigación y funcionalidad del proyecto.

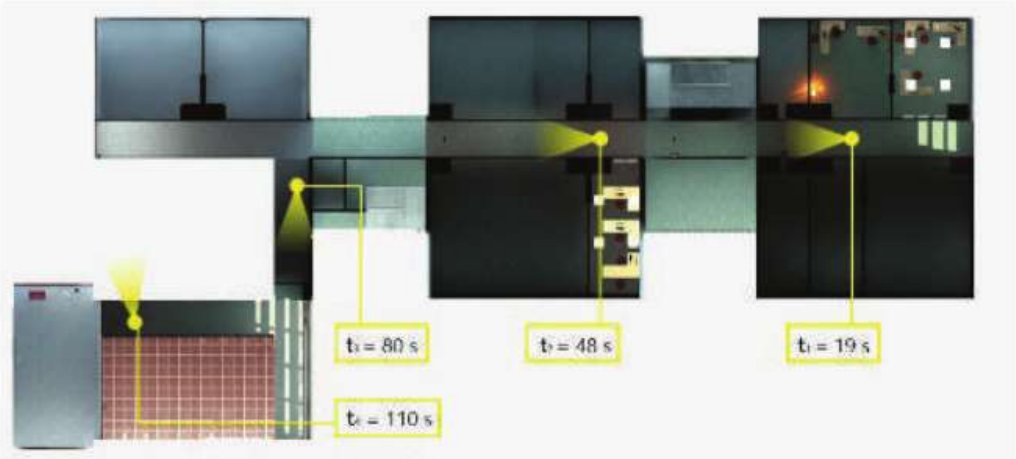
Uno de los hitos que más importancia ha tenido para mejorar la experiencia de usuario ha sido el modo de desplazarse dentro de la aplicación. Tras realizar muchas pruebas y testeos sobre la forma de desplazarse dentro del entorno virtual, se optó por la implementación de una plataforma para los pies: 3D Rudder. Este sistema tiene múltiples ventajas: el usuario está sentado, más cómodo y tiene estabilidad que reduce el mareo, el movimiento es más intuitivo, se dejan las manos libres y, sobre todo, que al provocar el movimiento con los pies y la inclinación del cuerpo, resulta mucho más natural.

Otro requisito funcional fundamental para los objetivos del proyecto es la programación del sistema de Eye-Tracker, para el reconocimiento y detección de la mirada. Gracias al desarrollo de esta función, somos capaces de registrar todos los puntos a los que ha mirado el usuario y, llegado el caso, poder sacar conclusiones científicas del uso que cada usuario hace de la señalética instalada en los edificios oficiales.

De la misma manera, hemos incorporado un pulsómetro como dispositivo accesorio (tecnología iOT) sincronizado con la experiencia virtual, obteniendo así las pulsaciones en tiempo real, con el objetivo de analizar el estado de ansiedad que experimenta el usuario durante la experiencia.

Todos estos datos se exportan a una base de datos, junto con las mediciones de velocidades, USER: lime 3212 s tiempos de reacción, posiciones y otra serie de variables, ayudan de forma integral a comprender el comportamiento del usuario frente al incendio (si ha intentado coger un ascensor, si ha salido por la escalera de evacuación correcta, si decide interactuar con los avatares asistentes de la brigada antiincendios, etc.). Para finalizar la experiencia, el usuario rellena una pequeña encuesta para evaluar los niveles de ansiedad que ha experimentado antes, durante y después de la experiencia.

Los resultados obtenidos en más de 300 usuarios que han testeado la Aplicación son muy prometedores, obteniendo un registro de variables exhaustivo para el tratamiento estadístico sobre los tiempos de evacuación, tiempos de reacción, movimientos, uso de señalética, etc. de todas las personas que han realizado la simulación virtual.



2.4 ARS VIVA

http://www.arsvivaexperience.com
Giorgio Airoldi
giorgio.airoldi@arsvivaexperience.com
+34 696 413 554
arsvivaexperience

ARSVIVA nace de la pasión por el arte compartida por unos amigos. Su objetivo consiste en desarrollar una nueva forma de disfrutar del arte. El nombre ARSVIVA, que significa 'arte viva' en latín, resume la misión de la compañía: 'hacer vivir' la obra de arte. Gracias a la Realidad Aumentada, los productos de ARSVIVA ofrecen una experiencia visual única que une disfrute estético y comprensión artística. Nuestra misión es facilitar el acceso al arte para todo tipo de usuario, independientemente de sus conocimientos previos.

Siempre respetando estrictos criterios de rigor científico y respeto, las APPs de ARSVIVA permiten añadir elementos adicionales a la obra, bien sea para verla como era originalmente o antes de una restauración, o bien para que el autor o uno de los personajes retratados hable de ella. Por ejemplo, apuntando hacia unas ruinas, se podrá ver el aspecto original del edificio; o, apuntando hacia un cuadro, uno de los personajes saldrá del mismo para contar su historia.

ARSVIVA tiene sedes en España (Madrid) e Italia (Milán), dos países con un riquísimo patrimonio cultural distribuido a lo largo de todo el territorio, y que abarca desde la antigüedad hasta la edad moderna.

2.4.1 Museo Sorolla

Por Giorgio Airoldi. giorgio.airoldi@arsvivaexperience.com

La aplicación, desarrollada conjuntamente con nuestros socios tecnológicos de 6D Lab, permite el disfrute de la visita al museo Sorolla a través de una experiencia novedosa de Realidad Aumentada. El Museo Sorolla de Madrid, que fue la casa del pintor, reúne las obras más famosas de Joaquín Sorolla, uno de los artistas españoles

del siglo XX más conocido y amado. Sorolla, junto con su esposa, guía al visitante, a través de su casa y su estudio, mientras revela los secretos de su técnica y cuenta con detalle su vida.

La aplicación se basa en el reconocimiento automático de un cuadro o de un elemento arquitectónico del edificio, que lanza un video del pintor que se superpone a la imagen capturada por la cámara del dispositivo y que está perfectamente dimensionado y proporcionado con la imagen real (al acercarse o alejarse, el tamaño de las imágenes superpuestas se modifica).

Los videos se han realizado con actores en estudio, sobre fondo monocromático, por lo que al no ser una creación virtual, el efecto es totalmente realista y la impresión es de que entre los demás visitantes, se encuentre una persona más.

Para que la experiencia sea más amena, se han realizado varias tipologías de videos. En algunos, el reconocimiento se basa en elementos arquitectónicos del edificio, con los cuales Sorolla interactúa, por ejemplo, apareciendo desde detrás de un busto, como se puede ver en la imagen de arriba.

En otros, el pintor está dentro del cuadro y se mueve entre los personajes retratados, como se puede ver en las siguientes imágenes:

También hay casos en los que se juntan las dos situaciones anteriores: por ejemplo, cuando Sorolla habla al cuadro que retrata a su mujer, y el retrato cobra vida y le responde; o cuando Sorolla se sube a un banco (presente en el museo) para poder explicar mejor un cuadro colgado a cierta altura. Esto se puede apreciar en la imagen siguiente:

La aplicación permite también realizar fotos con los protagonistas de los videos, bien sea a través de capturas de pantallas, o gracias a una funcionalidad incluida en la aplicación, mediante la cual es posible elegir entre varias poses de los actores y modificar su tamaño para que sea proporcional al de las personas que aparecen en la foto, como puede apreciarse en las siguientes imágenes:

La evolución de la APP incluirá el uso con gafas muy económicas de cartón o plástico en las que se puede alojar el móvil, para que la sensación de inmersión sea aún mayor.

2.5 KABEL

http://www.kabel.es
José Tomás González Martínez
Foronda, 4. E-28034. Madrid
tgonzalez@kabel.es
+34 913833224
tgonzalez@kabel.es
https://twitter.com/J_T_GM?lang=en
https://www.linkedin.com/in/jtomasgonzalez/

KABEL es una tecnológica española que se define en su página web de la siguiente manera *"Kabel es el **catalizador tecnológico** de la transformación de las organizaciones. Diseñamos, desarrollamos y nos emocionamos construyendo soluciones para la **revolución digital** basados en nuestro framework "4Ps": People, Proceses, Platforms & Places. Nos diferencia nuestra gente, todos **artesanos digitales** apasionados por la tecnología, por el aprendizaje permanente y por compartir su conocimiento con el mundo. Inspirados a partir de nuestra presencia en Silicon Valley innovamos alrededor de las últimas **tecnologías exponenciales** para acompañar a nuestros clientes en la transformación de sus negocios."*

Kabel es una empresa dinámica y moderna que idea, desarrolla y acompaña en la adopción y gestión de plataformas tecnológicas desarrolladas a la medida de sus clientes. Lo hace usando tecnologías y servicios de fabricantes de referencia como **Microsoft o Google**. Kabel ha logrado convertirse en el Partner con más competencias de Microsoft en España y el número 19 a nivel mundial con apenas 115 consultores en su plantilla. Entre los componentes más habituales en sus proyectos podemos encontrar aplicaciones cloud, desarrollo web y móvil o plataformas de integración basadas en microservicios y *serverless computing* que se combinan entre sí para crear servicios únicos y personalizados para los clientes y trabajadores digitales de las principales firmas españolas.

Kabel también diseña e implementa soluciones Analíticas y de Big Data junto a proyectos de IoT (internet de las cosas) lo que les permiten resolver problemas de negocio en muchas ocasiones con aproximaciones disruptivas a los problemas. Incorporan piezas de Inteligencia Artificial o Realidad Aumentada y Mixta en sus proyectos para resolver problemas concretos.

Kabel lleva alrededor de 2 años trabajando en soluciones de Realidad Mixta con HoloLens de Microsoft que integra en sus desarrollos con servicios *cloud* modernos, servicios cognitivos (IA), Internet de las cosas o plataformas de posicionamiento de interiores.

2.5.1 Antecedentes y Proyectos

Por: José Tomás González Martínez; "***experto en nada y aprendiz incansable***"

@J_T_GM
https://www.linkedin.com/in/jtomasgonzalez/

El mundo y la sociedad que nos rodea está cambiando a un ritmo nunca visto en la historia de nuestra especie. La forma en que nos relacionamos con los

demás, consumimos, disfrutamos de nuestro tiempo libre, pensamos o resolvemos problemas se está reinventando permanentemente. Esto, además, está cambiando cada vez más rápido con un ritmo **exponencial.** La mayor parte de estos cambios vienen inducidos por **olas tecnológicas** que traen con ellas **nuevos modelos de negocio** y **nuevos actores** y están redefiniendo sectores y mercados forzando a las empresas a reinventarse para sobrevivir. Entre dichas tecnologías, parece que en los próximos años veremos mucho relacionado con Inteligencia Artificial, Robótica, Impresión 3D, Computación Cuántica, Biología Digital, *Blockchain*, nuevos materiales o los que se ha venido denominando las **Realidades con adjetivo**: Realidad Aumentada (AR) o Realidad Virtual (VR) que en mi opinión convergen en la **Realidad Mixta (MR)** o en lo que se está denominando recientemente **Realidad Extendida. (XR).**

Con la intención de aportar algo de claridad en este complejo mundo de realidades, intentaré dar mi visión de cada uno de estos conceptos.

- Diremos que la Realidad Virtual (VR) es aquella tecnología capaz de sumergirnos en una experiencia digital que nos hace perder la noción de lo que tenemos a nuestro alrededor engañando a nuestros sentidos.

- Por el contrario, la Realidad Aumentada (AR) es aquella que "aumenta" la información que obtenemos a través de nuestros sentidos añadiendo contexto que mejora nuestro entendimiento de lo que nos rodea. Es decir, es capaz de darnos información por ejemplo de la calle en la que estamos y de la historia de los edificios que estamos mirando en cada momento.

- Por último, la Realidad Mixta (MR) es aquella capaz de incorporar a nuestro entorno sensorial aplicaciones holográficas tridimensionales estereoscópicas que parecen y se comportan como si fueran parte de nuestra realidad. La realidad extendida es el concepto que se maneja actualmente en varios círculos para abarcar cualquiera de las anteriores.

El referente reconocido por buena parte del mercado en el campo de la Realidad Mixta es hoy en día, HoloLens de Microsoft. Si bien ya existían hace algún tiempo otras soluciones basadas en *Headsets* menos conocidas como Meta 2, o Atheer AiR. Por otro lado, los grandes *players* tecnológicos están apostando fuerte por tecnologías que parece van a darle un nuevo impulso al mercado, como parece ser el caso de *Magic Leap* que ya se esperaba hace algún tiempo. Los conceptos anteriores aplican también a móviles y tabletas, donde Google, Microsoft, Facebook y Apple entre otros se están posicionando con soluciones y grandes inversiones en un mercado con una fuerte previsión de crecimiento para los próximos años. Varios analistas hablan ya de un valor combinado del mercado AR/VR/MR/XR de más de 80 mil millones de dólares para 2021.

Salvo algunas excepciones como Pokemon Go o los juegos disponibles a modo de escaparate para HoloLens, las soluciones de AR/MR están aún enfocadas al entorno empresarial (B2B). En concreto, el precio de HoloLens es el principal factor que limita su acceso al consumidor final o al mercado de videojuegos. HoloLens tiene un precio aproximado de 3.000$ + IVA en su edición para el desarrollador y 5.000$ + IVA en su correspondiente comercial y desde finales de 2017 ya se puede adquirir en España a través del store de Microsoft o a través de revendedores homologados. Recientemente se han incorporado al mercado otros dispositivos que incluyen Windows 10 como sistema operativo. Fabricantes de Hardware Partners de Microsoft han presentado recientemente dispositivos de "Realidad Mixta"; entre ellos Acer, Lenovo, Samsung, Dell, HP... En mi opinión, aunque tienen alguna capacidad de reconocimiento espacial se parecen más a otras soluciones como *HTC Vive, Oculus Rift* o *Samsung Gear* clasificadas en la categoría de plataformas de Realidad Virtual. Éstas, en algunos casos sí han salido al mercado a precios "asequibles" para un consumidor final medio.

Algunas de las razones por las que HoloLens es en mi opinión actualmente el referente del mercado XR/MR es:

- En primer lugar, es capaz de generar **modelos 3D (mallas) de los espacios** que visita **en tiempo real** y almacenarlos en un registro del S.O Windows 10 donde guarda el estado de aplicaciones y contenidos usadas en cada uno de dichos espacios.

- Usa estas mallas para posicionar sobre ellas aplicaciones holográficas **tridimensionales estereoscópicas** que permiten al desarrollador crear interacciones entre ambos mundos, el virtual y el real.

- Incorpora **interfaces naturales** para comunicarse con el humano. El ratón y el teclado a los que estamos acostumbrados no son interfaces naturales en la comunicación humana. Sin embargo, el uso de gestos, voz o nuestra mirada sí que lo son. Hemos descubierto escenarios en los que disponer de las manos "libres" supone es una gran ventaja al usar un computador holográfico. Véase una reparación, un montaje de un motor o una cirugía.

- Dispone también de un sistema de **sonido 3D** que aporta realismo a la experiencia ya que sus elementos emiten sonidos en la dirección correspondiente a la ubicación espacial de cada emisor virtual.

- Tiene además la capacidad de **grabar y almacenar** la mezcla generada de lo real y lo virtual a través de lo que Microsoft denomina MRC (*Mixed Reality Capture*).

- Incorpora la capacidad de cómputo y baterías en el propio dispositivo sin necesidad de elementos externos o cables y se comunica con el exterior a través de **Wifi** de última generación (802.11ac) lo que le **convierten en un cliente con sensores incorporados potentísimo** capaz de integrar servicios cloud modernos o conectarse a fuentes de datos y servicios corporativos, representar información de sensores de IoT o actuar sobre actuadores de los Gemelos digitales (Avatares cloud) de cada uno de los elementos de IoT.

Hay quien ya señala que este tipo de soluciones acabará desbancando a móviles, PCs y tabletas **convirtiéndose en la herramienta de trabajo** del usuario del futuro cambiando radicalmente la concepción del puesto de trabajo. Su medio kilo aproximado de peso, su tamaño, su aun limitado campo de visión, la duración de su batería o los 5 metros de máximo descubrimiento del entorno, son aun limitaciones que considerar y se necesitarán nuevas versiones y/o nuevos competidores para hacer una realidad ese escenario.

En lo relativo a Realidad Mixta, Kabel ha estado desarrollando software los últimos 2 años específicamente para HoloLens mediante un pequeño equipo dedicado. A continuación, a modo de anticipo, resumo en una tabla algunos aceleradores y proyectos de los que os iré "hablando" durante las próximas líneas.

Acelerador /Solución	Video	Cliente/ Sector	Casos de Uso	Algún Detalle
Education Shared Experience		engage Independent School Educación	Clase Guiada 3D Colaboración sobre contenido 3D	Shared Experience, MRTK (*Mixed Reality Toolkit*) y Unity Gestion de Permisos en las sesiones Azure como Back-end
HoloShelves		Retail y Salud	Lineales Holográficos personalizables para diseñar y validaren remoto configuraciones de productos y marcas sobre los estantes.	Disponible en Windows Store Unity y Azure Gravedad y Reconocimiento espacial Gaze tracking Conexión Remota

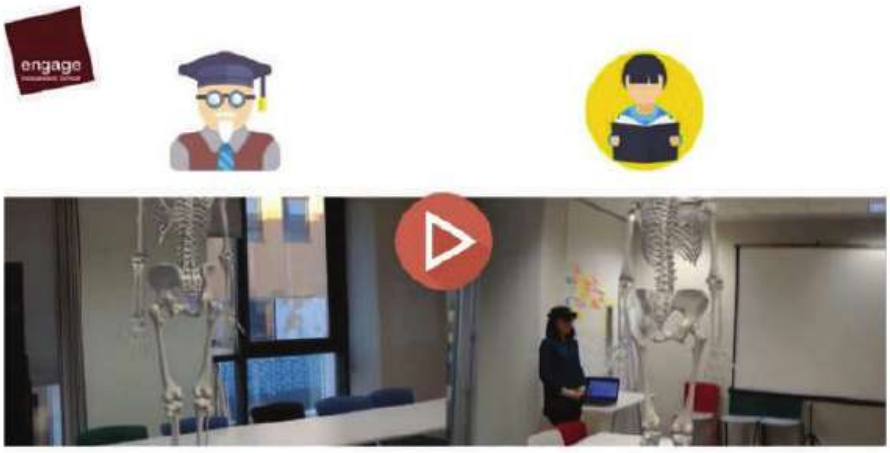
HoloFox		Cualquier sector	Entrenamiento y capacitación	Asistente Holográfico para construir formas en el espacio. Modos; Entrenamiento y Asistencia Azure, BOT Framework, LUIS, Unity
HoloSitum		Logística, Industria, Salud, Telco, Museos	Localización de personas y activos en edificios. Guiar de personas	SITUM. Posicionamiento de interiores sin Balizas (1 metro, 1 segundo). Generación de Modelos 3D de edificios, Reducción Poligonal Calibración & Sincronización
HoloIVI		Salud	Explicar a pacientes distintas patologías en 3D.	Azure, Unity Bots, Servicios cognitivos: de traducción, LUIS. *Speech to text* Unity
Holovision		Industria y Construcción	Reconocimiento de Objetos y personas	Azure, Unity Unity Servicio cognitivo "*custom vision*"
HoloBlind Support			Medición de distancia a un obstáculo. Descripción del entorno con Visión artificial.	Azure, Unity Mediciones en el espacio Servicio cognitivo "*computer vision*"

En cualquier caso, hoy en día hay ya muchos casos de uso y soluciones interesantes para el mundo empresarial. Tal como podemos identificar a primera vista en la tabla anterior, buena parte de ellos integran las capacidades inherentes de realidad mixta con otras tecnologías como computación ***cloud***, **inteligencia artificial** (AI), **internet de las cosas** (IoT), **posicionamiento** de Interiores o **drones** para construir soluciones empresariales de ciencia ficción hace tan sólo una década.

Hemos identificado dos casos de uso que son transversales a cualquier sector que voy a intentar describir a través de 2 proyectos piloto que hemos desarrollado en Kabel; Son la relativa a capacitación y entrenamiento del trabajador y la mejora y personalización de la experiencia de compra.

1. **Engage Education Shared Experience**

   El primero es el relativo a la **capacitación y entrenamiento de trabajadores y estudiantes**. La mayoría de los humanos hemos cimentado nuestro aprendizaje mediante el uso de herramientas de dos dimensiones; primero usando Pizarras y Libros, más adelante Tabletas, Móviles, PCs con monitores, televisiones o pizarras electrónicas todos ellos, elementos 2D que hemos usado y seguimos usando para explicar conceptos que en el mejor de los casos tienen 3 dimensiones. Estudios recientes identifican claros patrones de mejora en los ritmos y persistencia del aprendizaje cuando esté se realiza usando técnicas interactivas y colaborativas en 3D que, además, suele representar mucho mejor el mundo que nos rodea. En concreto, imagínese que podemos compartir un contenido en 3D entre varios alumnos que colaboran para realizar un trabajo de una determinada asignatura y un profesor que guía la experiencia. Cada uno mantiene la perspectiva espacial que desee sobre ese contenido a la vez que va interactuando con él y con las explicaciones animadas del profesor. Se van sincronizando los cambios que producen los distintos colaboradores sobre el contenido en tiempo real.



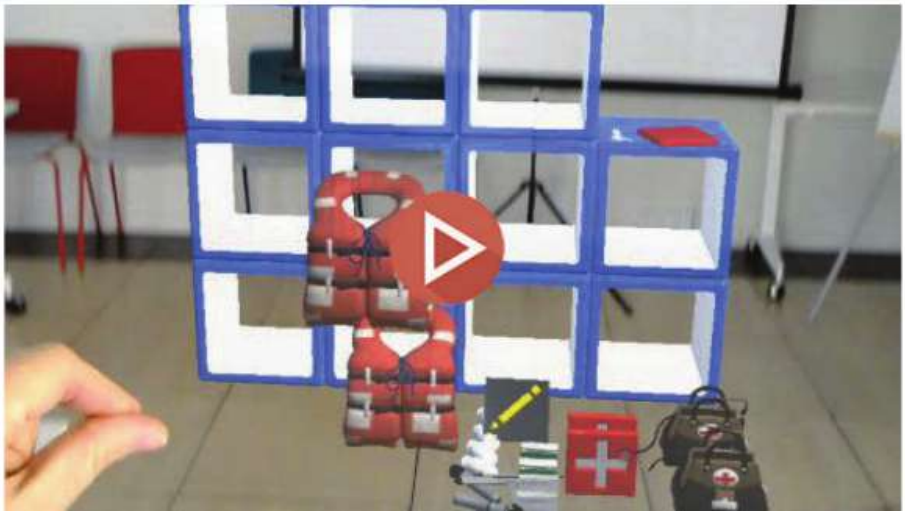
   Esto que os describo lo realizamos en 2017 para el Colegio **Engage de Madrid.** La solución permite que un profesor de acceso a sus alumnos a contenidos y animaciones 3D sobre la que van interactuando para aprender las partes del cuerpo humano o el funcionamiento de un motor de explosión de un coche siendo el propio profesor el que guía la experiencia y va otorgando permisos a los alumnos que pueden probar sus habilidades y

conocimientos con supervisión presencial o remota del profesor. Todo ello, usando comandos de voz y gestuales. Se puede ver un video explicativo de la solución **Education Shared Experience** en *https://youtu.be/FaVwl2-3hNw* que se ha desarrollado a partir de una aplicación UWP (*universal windows platform*) sobre el motor 3D de *Unity* y algunas librerías *open-source* como el *Mixed Reality ToolKit* (MRTK) disponible en GitHub, por cierto, recientemente comprada por Microsoft. Quiero agradecer expresamente desde estas líneas la visión y el apoyo dispensado por el director del **Colegio Engage** durante el proyecto. Su Visión y su clara apuesta por la tecnología y el método científico en la educación son en mi opinión una espectacular referencia para el resto.

2. **HoloShelves**

   El Segundo caso de uso transversal es aquel que **mejora la experiencia de compra** permitiendo la **personalización en tiempo real** del producto o servicio y el guiado remoto de la misma por un experto. De nuevo el concepto de *Shared experience* se muestra como un elemento muy interesante.

   Como ejemplo de este segundo caso de uso en Kabel hemos desarrollado un piloto para un fabricante de material sanitario que permite personalizar los lineales con sus productos en un supermercado y probar distintas configuraciones sobre él sin necesidad de realizar inversión en su montaje ni en los productos que incorpora a la vez que probándolas de forma muy ágil. Hemos denominado a la solución **HoloShelves** cuya versión reducida puede ser descargada desde *store de Windows para HoloLens*. La aplicación está pensada para el sector retaIl (venta por menor) pero estamos también identificando usos en sectores como el de la construcción, diseño de interiores, farmacéutico y sanitario.



La solución permite montar estanterías configurables sobre cualquier recinto y colocar distintos productos y marcas para evaluar las distintas configuraciones del lineal. A la vez, se transmite en remoto la solución al cliente (la gran superficie) que puede ver las distintas configuraciones y discutirlas con el fabricante sin necesidad de desplazarse ni de montar físicamente los estantes ni los productos. Utilizando aplicaciones como *Remote Assist* recientemente publicada por Microsoft se puede mostrar HoloShelves en remoto a los expertos de cualquier gran superficie tomando con ellos las decisiones relativas Al diseño de lineales y la configuración de productos y marcas en los supermercados.

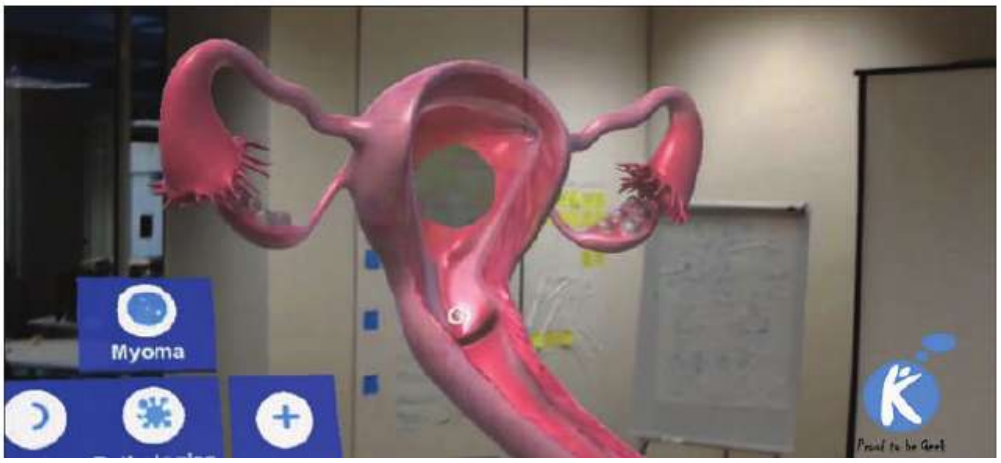
Además, el sistema desarrollado sobre el motor de Unity aporta características inerciales a sus elementos de tal forma que los productos a colocar interactúan tanto con el estante o lineal virtual como con el suelo, mesas o pareces reales del entorno sobre el que se aloja. Todos ellos pueden ser colocados directamente por el propio usuario mediante gestos. Es posible igualmente sacar estadísticas de pruebas de usuario acerca de cuáles son los productos recogidos o más mirados para distintas configuraciones O proponer en el propio diseño aquellas configuraciones que maximicen el valor de compra si ya se ha obtenido un patrón de la analítica registrada en el lineal del supermercado.

Incorpora también la posición de focos de iluminación sobre estantes y productos pudiendo moverlas y observar sus efectos.

Además de estos dos casos de uso transversales, existen algunos sectores para los que se están construyendo soluciones especialmente interesantes. **Salud, Construcción y diseño de interiores, Logística, Energía, Fabricación, Ingeniería, Telecomunicaciones**... A continuación, describiré algunos de los desarrollos en que estamos trabajando para estos sectores.

3. **HoloIVI**

Uno de los sectores en los que más desarrollo está teniendo HoloLens es el del sector salud. Desde visualización de Imagen Medica en 3 Dimensiones hasta llegar a escenarios de Cirugía asistida o incluso guiada por HoloLens. La capacidad para mostrar el interior de nuestro cuerpo o para explicar el efecto de un tratamiento farmacológico o intervención quirúrgica de forma animada es especialmente útil en la colaboración entre profesionales médicos y laboratorios y en las explicaciones que se le dan a un paciente. La conexión remota para pedir ayuda u opinión a otro experto es también un caso de uso muy interesante, así como la visualización interactiva de imagen proveniente de ecógrafos y otros dispositivos.



En el sector salud estamos desarrollando para IVIRMA, firma multinacional de origen español líder mundial en técnicas de reproducción asistida, **HoloIVI**. El objetivo en este caso es explicar a los pacientes las patologías que sufren apoyándonos en la visualización de modelos 3D del sistema reproductor. Se trata de que el paciente pueda entender mejor el problema y los tratamientos que le está planteando su ginecólogo de IVIRMA. La aplicación, aún en fase piloto, incorpora un Bot que asiste a ambos durante el proceso y permite grabar vía voz la explicación personalizada para cada cliente.

La plataforma piloto está compuesta de una aplicación basada en Unity y un back-end en Azure sobre la que se muestran los modelos 3D con explicaciones de las correspondientes patologías de los pacientes. Uno de los elementos claves a considerar en este caso es el tratamiento de modelos e imagen médica y la reducción poligonal adecuada para garantizar una buena experiencia.

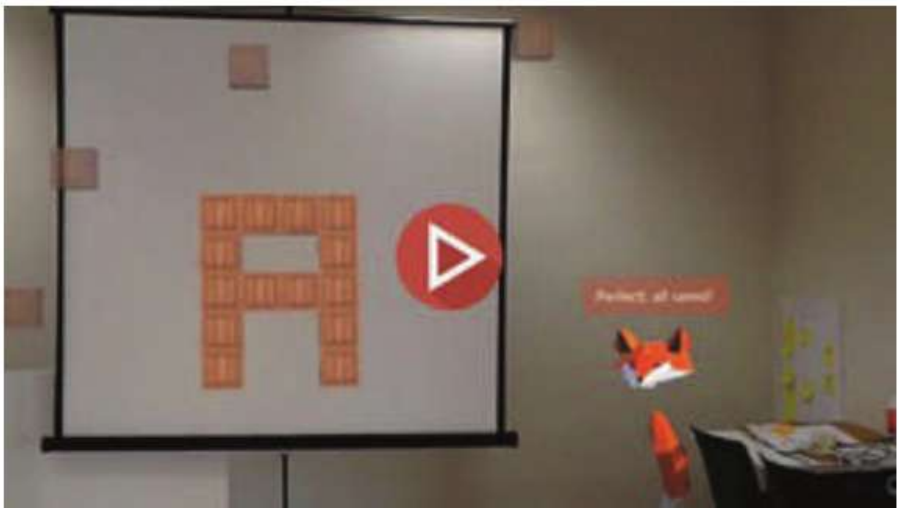
4. **Holofox y Holobot**

Tanto **Holofox** como **Holobot** son aceleradores (o piezas de código reutilizables) que hemos desarrollado para incluir asistentes virtuales en los desarrollos de proyectos MR. Ambos hacen uso de *Bot Framework* disponible en Azure. Generamos un modelo animado 3D que atiende a las peticiones de voz del usuario de HoloLens dentro de una determinada aplicación. A la orden de "Hey Bot" o "Hey Fox" en cada caso, el asistente se acerca y ayuda al usuario a realizar sus tareas estableciendo con él una conversación de voz donde además se muestra el texto de dicha conversación para dar visibilidad al usuario de que el Bot le está entendiendo. Se apoya en el uso de servicios cognitivos cloud como

LUIS (*Language Understanding Intelligent Servic*e) para interpretar la intención del usuario y ejecutar la ayuda o acción que corresponda.

El sector de **retail** para dar explicaciones a clientes o el de **fabricación** para asistir a sus trabajadores o el de **salud** para explicar a pacientes o el de l**ogística** para guiar a los operadores son algunos de los usos que hemos identificado.

Holofox, permite además de esta función de asistente, entrenar un sistema virtual para aprender a colocar o montar piezas virtuales de una determinada manera y a continuación asistir a otros que puedan estar aprendiendo a llevar a cabo su entrenamiento de forma guiada. El escenario más simple es alguien que quiere montar bloques en el espacio para crear una determinada figura. Una vez el sistema ha sido entrenado y otro usuario intenta realizar el montaje, el Bot le irá dando indicaciones de si va bien, cual es la pieza que está mal colocada o cual es la mejor propuesta para colocar la siguiente pieza que le lleve a conseguir su objetivo. El siguiente video muestra su funcionamiento.



5. **Holositum**

Para sectores como el de energía, logística, salud, industrial, turístico, ingeniería o de seguridad tiene mucho sentido para un trabajador disponer de aplicaciones holográficas que tengan la referencia de **su posición actual en un determinado recinto** y una **referencia visual a otros objetos y personas**. Supone una gran ventaja disponer de ayuda o guiado para encontrar activos o personas o para guiarnos caso de que los recintos sean de gran tamaño como una nave logística o industrial… o incluso, ¿por qué no? un museo o estadio.

Holositum es un acelerador para proyectos de localización y guiado. Se basa en un desarrollo de UWP sobre Unity e incorpora llamadas a servicios de una plataforma de posicionamiento indoor de origen español que se llama **Situm** de quien kabel es Partner. A partir de un móvil y sin necesidad de desplegar balizas de posicionamiento (Beacons), la plataforma de SITUM es capaz de posicionar a todo aquel que lleve un cliente móvil de Situm en un recinto previamente calibrado.

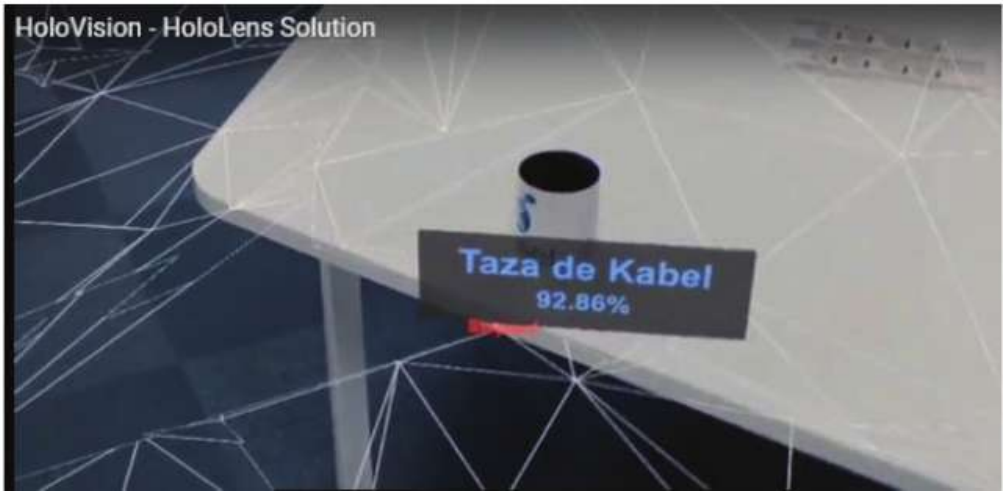
Dicha posición es representada sobre un modelo 3D del recinto correspondiente y le hemos añadido interacciones de voz y gestuales para mostrar al usuario una vista en 3D en tiempo real y con hasta un metro de precisión, de la posición de dichas personas en el recinto (y con respecto a determinadas partes o activos que estén dispuestos dentro de dicho recinto).

Imaginemos estas capacidades en un sistema de control de seguridad, en un estadio o en un hospital para localizar profesionales ante una emergencia o personal de seguridad para dar respuesta a un incidente en un estadio.

6. **Holovision, Holoblind support y Holobot**

La inteligencia artificial es un elemento muy presente en los proyectos de realidad mixta. Algunas personas en Microsoft aseveran que es el producto que han construido que incorpora más elementos de Inteligencia Artificial (IA).

Algunos de los casos de uso más interesantes son los que ayudan al reconocimiento del entorno para identificar anomalías o para ayudar a reconocer o entender mejor lo que nos rodea usando los sensores y cámaras de HoloLens y la potencia de los servicios de IA disponibles en Cloud.



Casos de uso que usan visión artificial para el reconocimiento de objetos, caracterización de patrones o identificación de anomalías en objetos y espacios a la vista de los sensores y cámaras de HoloLens. Estos funcionarían mejor si la latencia necesaria para el cálculo actualmente en cloud de los servicios cognitivos fuese menor.

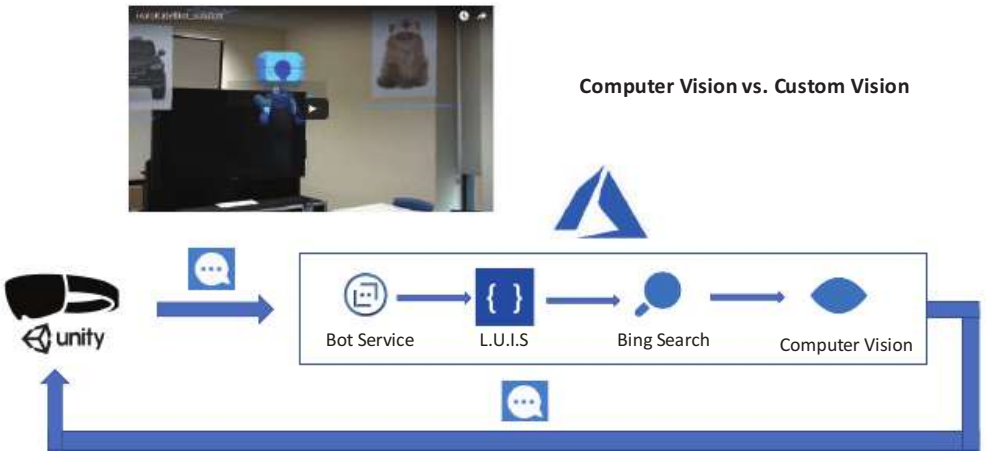
En Kabel hemos desarrollado un acelerador para HoloLens que permite identificar objetos y personas usando el servicio cognitivo de Azure "Custom Vision" al que hemos entrenado para detectar objetos y personas específicas. Hemos llamado a la solución Holovision. En la siguiente imagen se puede identificar como es capaz de detectar el objeto "taza con

el logo de Kabel" y como en función de cuanto se vea el logo de Kabel al girar la taza se va incrementando o decrementando la probabilidad de que el objeto identificado sea el entrenado.

Lo mismo es aplicable si incluimos servicios de **visión artificial** generalistas como *computer vision* para dar una idea del entorno circundante como hace el acelerador **HoloBlindSupport** desarrollado por Kabel como prueba de concepto que es capaz de indicar la distancia al objeto más cercano o dar una descripción del entorno que rodea al usuario adhoc.

También en lo relativo al procesamiento del lenguaje y la voz para incorporar asistentes como bots "virtuales" basados en una ejecución local de DNNs por tanto con menor retardo en la respuesta sería una ventaja.

Holobot, el acelerador que hemos mencionado antes es un buen ejemplo. Se trata de un **asistente virtual** que establece una conversación de voz con el usuario de HoloLens, la interpreta usando el servicio cognitivo LUIS y lleva a cabo ciertas tareas como la búsqueda de una imagen de una determinada temática en internet usando el servicio cognitivo cloud bing search. El resultado de las distintas búsquedas se va presentando en el espacio 3D que rodea al usuario. Mediante una nueva conexión a un servicio cognitivo de visión artificial de azure (*computer vision*) proporcionamos una descripción al usuario del propio contenido descargado.



Me gustaría terminar mencionando algunas de las evoluciones anunciadas por Microsoft para inicio de 2019 en HoloLens 2, (también denominado proyecto Sídney). Aparte de mejoras generales en el rendimiento y cambios en el hardware que incluyen por ejemplo el de un nuevo procesador ARM Snapdragon en lugar de Intel tenemos cosas como:

- Mayor duración de la batería actualmente limitada a unas 3 horas.
- Aunque aún no se han emitido menciones oficiales acerca del campo de visión (FoV) ni del precio del dispositivo, se esperan mejoras en ambos aspectos. Veremos
- Incorporará Soporte LTE lo que permitirá desarrollo de aplicaciones siempre conectadas y una mayor velocidad en las comunicaciones.
- Se ejecutará en Windows 10 core OS (Oasis).
- Por último, se espera la Incorporación de un **Coprocesador en la HPU** (*Holographic processing unit*) para ejecutar algoritmos de Inteligencia Artificial (AI) asistidos y actualizados desde la nube en lo que se denomina *Mixed Reality Cloud*. Actualmente cada dispositivo incorpora un Procesador de propósito general (CPU), un procesador gráfico (GPU) y un procesador holográfico (HPU) encargado de procesar la entrada-salida de sus sensores y cámaras. Con este nuevo coprocesador se podrán realizar cálculos de redes neuronales profundas (DNNs) en local para disminuir la latencia.

Este último punto es interesante para mejorar los tiempos de respuesta a usuarios por ejemplo en soluciones como las descritas de **Holovision, HoloBlindSupport y Holobot** actualmente resueltas con servicios ejecutados en cloud.

A todo el que tenga interés le recomiendo que se una al grupo de LinkedIn de Mixed Reality que hemos creado 3 integrantes de la IAMCP. Ahí estamos publicando información de casos de uso y soluciones del ecosistema de Partners europeos de Mixed Reality donde ya hay más de 260 participantes. *https://www.linkedin.com/groups/8642483*

2.6 BRAVENT

Bravent es una consultora IT especializada en tecnologías Microsoft como .NET, Sharepoint, Xamarin y Azure aplicadas a las áreas de movilidad, web, ALM y diseño y UX.

Como partner de Microsoft y Xamarin, cuenta con un equipo de líderes de la comunidad técnica que establecen soluciones creativas y eficaces, basadas en la aplicación de las tecnologías Microsoft.

Con diferentes sedes en España, como Madrid y Málaga, se llevan a cabo proyectos con clientes de una amplia variedad de sectores como el deporte, la salud, derecho, banca… de la mano de sus más de cincuenta empleados.

En definitiva, Bravent trabaja con grandes empresas de la mano, ayudándoles en su proceso de transformación digital.

Una de las líneas de trabajo más características es su área de innovación, con proyectos de Realidad Virtual, Inteligencia Artificial e IOT.

Dentro del campo de la realidad virtual, Bravent trabaja con tecnologías de Microsoft Hololens, un dispositivo de Realidad Mixta por medio del cual los usuarios pueden ver hologramas superpuestos en la realidad e interactuar con ellos.

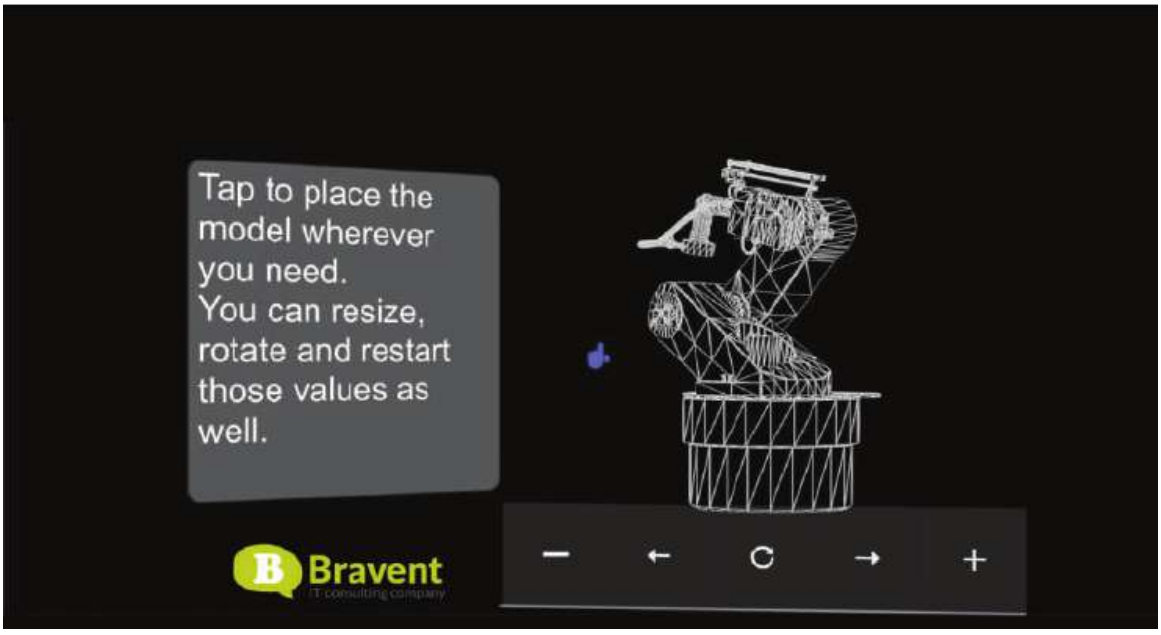
Sus sensores integrados permiten que funcione de manera inalámbrica; y cuenta con CPU y GPU propias. A continuación, os presentamos dos proyectos muy interesantes en los que Bravent ha trabajado:

2.6.1 HoloLearning

Con esta tecnología, Bravent ha desarrollado HoloLearning, un framework con cursos para aprender a desarrollar determinada acción por medio de la interactividad que ofrece la Realidad Mixta.

El usuario no tiene más que ponerse las Microsoft HoloLens y seguir los vídeos, audios, tooltips o interactuar con los elementos virtuales para su aprendizaje, sin necesidad de acudir presencialmente a ningún lugar.

Además, HoloLearning ofrece la posibilidad de realizar estos cursos con teleasistencia conectando el dispositivo a internet. De esta manera, otra persona podrá ver los movimientos que el alumno está realizando y hacer un seguimiento en vivo a través de chat de voz, aportando las recomendaciones y mejoras necesarias.



Los cursos se diseñan bajo demanda, de este modo se pueden abordar todo tipo de formaciones, desde cambiar un embrague a un coche a un curso teórico de vuelo o el montaje de un simulador de carreras. ¡Todo es posible!

Esta plataforma permite reducir al máximo los costes e incrementar la seguridad de los trabajadores durante su proceso de aprendizaje, además de conseguir una mejor experiencia de usuario, generando así mayor interés en la formación. También ofrece un tiempo de aprendizaje ilimitado y accesible en cualquier momento. Además, al ofrecer una inmersión total, facilita el aprendizaje y el asentamiento de los conocimientos.

HoloLearning ha planteado distintos retos, uno de ellos es la organización de una interfaz dinámica basada en capas y que consume servicios web a través de Unity. Tanto para generar los layouts como para cargar los datos como modelos, audios etc, hay que ser extremadamente cuidadoso a la hora de crear la interfaz responsiva al contenido y compartiendo una base técnica que pueda ser usada por cualquiera de los cursos pero a la vez permita cierto grado de libertad a la hora de plantearlos.

Otro reto que se presentó fue la funcionalidad online, implementada con Photon. Al plantearse la UI dinámica por capas, esta se planteó animada para facilitar una mejor UX. Esto supone que a la hora de compartir datos online, se está cargando una misma escena en los dos dispositivos, en la que, el master levanta eventos relativos a datos, posición, escala y triggers de animaciones, y el slave, recibe y replica al instante.

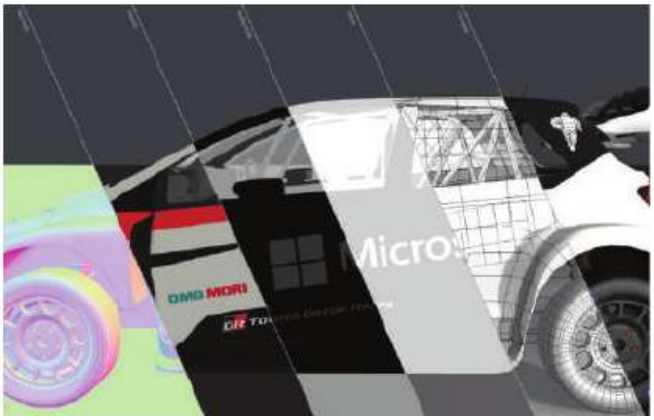
También digno de mención es el trabajo gráfico en este proyecto, debido a la limitada capacidad de Hololens a la hora de hacer entornos gráficos cargados de polígonos, siendo la recomendación de máximo 100k bien optimizados, el equilibrio que se debe obtener entre número de polígonos, calidad y rendimiento puede ser una cuerda floja en la que caer en cualquier momento.

Además, al ser una interfaz basada en paneles holográficos y por motivos de capacidades del dispositivo, el AA entra en juego y con un papel primordial. Sin una configuración adecuada de AA, el aliasing sufrido en este entorno puede llegar a ser incluso molesto, no solo antiestético.

2.6.2 HoloRacing

El objetivo de Holoracing era crear una aplicación que recrease en tres dimensiones un auténtico juego de carreras en el que el usuario sienta la adrenalina de un auténtico piloto y ofrecer no solo una experiencia única, original y novedosa; sino también revolucionar el mundo del automovilismo y el entretenimiento.

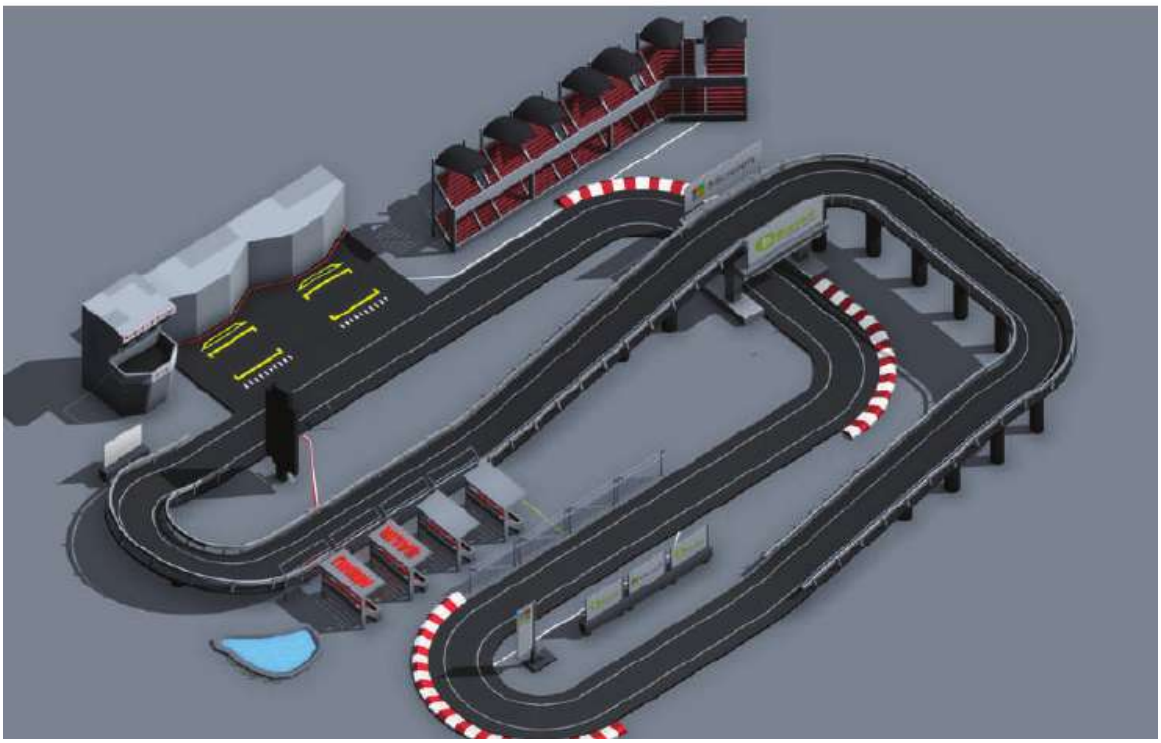
Las gafas Microsoft Hololens ofrecen una inmersión total en la app y una completa interactuación, y pasar así por cinco circuitos diferentes, cada uno con su ambientación y dificultad, dirigiendo un auténtico coche de rally.



El juego dispone de dos submenús donde se puede elegir tanto modo libre como modo circuito. En el modo libre, el usuario puede conducir el vehículo a través de su propio entorno real, esquivando baches reales como sillas, mesas, cajas, etc.

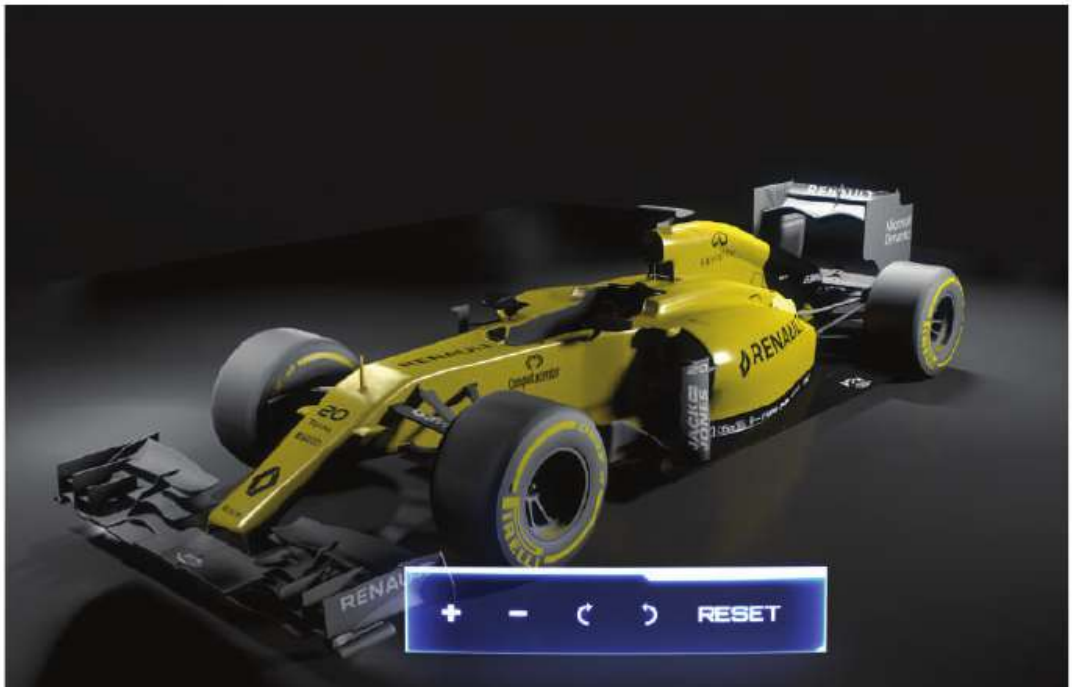
Seleccionando el modo circuito, el jugador puede elegir entre varios circuitos con diferente ambientación y dificultad, y hacer sus mejores tiempos.

El visor de las Hololens ofrece al usuario la posibilidad de ver el vehículo en detalle e interactuar con él, ampliarlo o reducirlo, e incluso utilizar una visión 360º; además de meterse en su interior para ver las carreras desde dentro.



La Realidad Mixta tiene el potencial de ayudar a hacer cosas que, hasta ahora, parecían imposibles. Estas experiencias ayudan a realizar tareas de una forma diferente, más dinámica e interactiva.

El principal reto que supuso este proyecto fue el uso del mando de la Xbox one para el manejo de los vehículos, además de las típicas piedras que se encuentran en este tipo de proyectos, diseño, físicas de vehículos de motor, etc.



2.7 EPTISA

http://www.eptisa.es

Esther Olivares Briones
Gerente Sector Privado

esther.olivares@ti.eptisa.com

+34 626 229 083

Encarnación Prieto Salas
Consultora experta en Geocall WFM

encarnacion.prieto@ti.eptisa.com

+34 91 594 95 00 Ext. 3722

Emilio Muñoz, 35. E-28037. Madrid

Somos una compañía multinacional de ingeniería, consultoría, tecnologías de la información y desarrollo institucional, económico y social con más de 60 años de experiencia.

La calidad de nuestros servicios nos convierte en referencia en los sectores de Transporte, Agua, Medioambiente, Industria, Energía, Edificación y Desarrollo Institucional, Económico y Social.

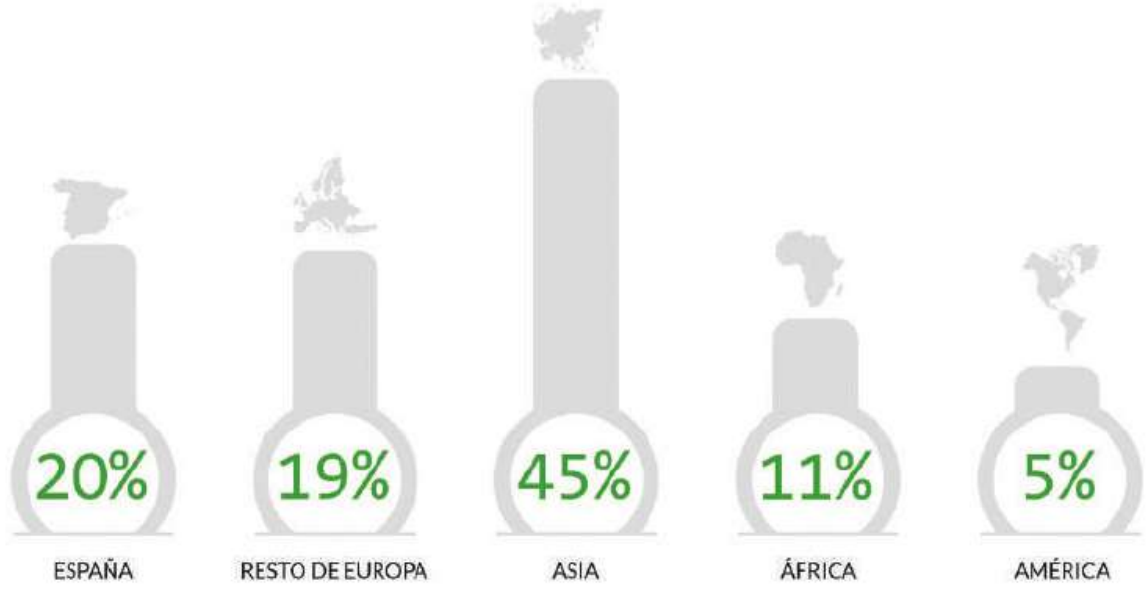
Aplicamos el conocimiento técnico, la innovación y nuestra capacidad de gestión de proyectos en beneficio del cliente y el desarrollo sostenible de la sociedad.

Con presencia en más de 55 países y una red de 28 oficinas locales consolidadas, Eptisa es una empresa global que actúa en local, cercana al cliente y a la sociedad civil en la que opera.

La adaptación local y la calidad de nuestros servicios son los elementos clave de nuestra estrategia, permitiendo crecer y expandir el alcance de nuestros proyectos en respuesta a la evolución de las necesidades de nuestros clientes.

ACTIVIDAD POR SECTORES

DISTRIBUCIÓN GEOGRÁFICA



2.7.1 OverIT

Eptisa es distribuidor autorizado de la solución Geocall WFM de OverIT , una empresa que diseña e implementa soluciones para la gestión de las actividades de campo en el ámbito técnico, de ventas y transportes.

Experta en la realización de soluciones móviles de trabajos de campo e innovadoras soluciones TI, OverIT optimiza los procesos de campo con innovadoras tecnologías (Realidad Aumentada, Mixta, Virtual y dispositivos wearables), con el objetivo de ayudar a las empresas a reducir costes, aumentar la productividad, reducir el tiempo de comercialización y dar al cliente el mejor servicio posible.

Fundada a finales de los años 90 como un subproducto del departamento de TI de una empresa multinacional líder en distribución de alimentos, actualmente OverIT cuenta con más de 380 profesionales y opera a nivel mundial, colaborando con las empresas más importantes ayudándolas a alcanzar una excelente optimización del rendimiento de los procesos de trabajos de campo.

Los clientes de OverIT pertenecen a diferentes sectores, como Energía y Servicios Públicos, Petróleo y Gas, Industria, Servicios y Transportes. Todos ellos consideran la gestión de actividades de campo como una prioridad.

La compañía invierte alrededor del 10% de su facturación anual a investigación y desarrollo. Gracias a ello y a la colaboración con prestigiosas universidades, permite a OverIT poner a prueba constantemente innovadores servicios y soluciones avanzadas, con el fin de mantener una gran ventaja competitiva en el mercado.

Gracias a la combinación de más de 20 años de experiencia de campo con los conocimientos adquiridos a través de la constante investigación, actividades de prueba, y la competencia de sus recursos, la compañía es capaz de satisfacer las exigencias de los clientes con eficiencia y calidad.

2.7.2 Geocall WorkForce Managment

Geocall WFM (en adelante Geocall) es una plataforma de software innovadora y totalmente parametrizable para la gestión completa de los trabajos de campo.

Geocall es el fruto de más de dos décadas de experiencia en la planificación y desarrollo de aplicaciones para las principales compañías internacionales, dedicándose a apoyar a los técnicos y al personal de mantenimiento mientras realizan actividades en el campo.

Grandes compañías, principalmente las enmarcadas en el negocio de las utilities y las organizaciones públicas, pueden beneficiarse del uso de Geocall. Es una plataforma de aplicaciones innovadora y completamente parametrizable, que proporciona diversos módulos verticales integrando distintos contenidos funcionales y tecnológicos orientados a la eficiencia, la mejora de la calidad del servicio y reducción de costes.

Gracias a Geocall, OverIT está incluido en el cuadrante Mágico de Gartner para la Gestión de Trabajos de Campo desde el año 2014.

Geocall proporciona el mantenimiento planificado y no planificado, optimización y programación de actividades, administración de contratos y la integración de las plataformas GIS más comunes (gracias a la alianza con Esri y a la integración con el paquete de productos ArcGIS), así como la habilitación e informes de la fuerza de trabajo móvil. Geocall está integrado con los sistemas SAP y puede implementarse en las propias instalaciones del cliente o como SaaS (software como servicio).

El crecimiento constante e internacional de OverIT, especialmente en mercados de FSM (Field Services Management), ha sido posible gracias a las inversiones continuas en innovación (Realidad Aumentada y Virtual, optimización de programación, bases de datos en memoria, wearables como HoloLens, etc.), lo que le permite estar a la vanguardia del mercado cuando las innovaciones tecnológicas presentan nuevas oportunidades para equipar a las organizaciones de FSM.

A lo largo de los años, OverIT siempre se ha caracterizado por su contenido innovador y ha tratado constantemente de anticiparse a las necesidades del mercado y del cliente.

El centro de investigación y desarrollo de OverIT está completamente dedicado a diseñar, probar y desarrollar las soluciones del futuro.

La innovación en OverIT nace en el departamento de investigación, pero crece en el terreno, en colaboración con nuestros clientes. Para nosotros, la innovación solamente agrega valor si ofrece beneficios reales para el usuario.

2.7.3 Space 1: Realidad Aumentada, Mixta y Virtual.

La Realidad Aumentada, Mixta y Virtual aplicada en los procesos de mantenimiento, mejora la ejecución de las actividades de campo en lo que a seguridad, eficiencia y calidad respecta. Los técnicos cuentan con un conjunto de herramientas que dependen de los procedimientos interactivos guiados, en las funciones

multimedia avanzadas y la cooperación con los usuarios desde un dispositivo móvil: tableta, teléfono inteligente, gafas inteligentes y dispositivos alojados en la cabeza.

OverIT aprovecha al máximo estas innovadoras tecnologías a través del producto Space1 para soluciones de Realidad Aumentada, Mixta y Virtual.

La Realidad Aumentada, Mixta y Virtual son tecnologías capaces de cambiar la forma en las que las personas interactúan y perciben el mundo, llevando la experiencia del usuario a un nivel completamente nuevo. Cuando se aplica a las empresas, se mejora la experiencia al remodelar los procesos industriales mejorando la eficiencia general aumentando la percepción del entorno y así la satisfacción del cliente.

Aprovechando la amplia experiencia de los profesionales de OverIT en la creación de aplicaciones de Realidad Aumentada para empresas, Space 1 es una solución innovadora para administrar las ventas y actividades de campo, colaboración virtual y capacitación de aprendizaje.

Space 1 es rápido y fácil de implementar gracias a la amplia gama de parametrizaciones y conectores para la integración con cualquier sistema empresarial. Además, es multiplataforma y multidispositivo para garantizar una gran flexibilidad y facilidad de uso fuera de línea.

Space 1 es un paquete de aplicaciones compuesto por cuatro productos:

- **Space 1 Virtual Collaboration**, que permite a los usuarios compartir un campo de visión prospectivo con otros operadores y la colaboración e intercambio de información entre ellos en tiempo real.
- **Space 1 Maintenance** para mejorar el rendimiento de los técnicos con innovadoras funcionalidades para el soporte y la colaboración a través de procedimientos interactivos basados en flujos de trabajo.
- **Space 1 Sales**, mejorando las actividades de venta y presentación a través de catálogos de productos virtuales que son más interactivos y visuales.
- **Space 1 GIS**, dando la posibilidad al usuario de interactuar con los datos geoespaciales que se muestran en el mundo real a través de la Realidad Aumentada.

Space 1 es un producto multiplataforma diseñado y compatible con múltiples sistemas operativos. Está disponible en iOS, Android y Windows, dependiendo del dispositivo hardware elegido.

El principal beneficio de usar una solución multiplataforma radica en la amplitud del segmento de mercado, objetivo que puede ampliarse sin comprometer la calidad general del producto.

Space 1 no es sólo multiplataforma, es mucho más que eso. De hecho, la realidad virtual se puede ejecutar en dispositivos portátiles (Tablets y Smartphones), tabletas acopladas, gafas inteligentes y auriculares. Gracias a este enfoque multiplataforma, destaca frente a la competencia y ofrece a los clientes las características que siempre están en sintonía con las últimas tecnologías para proteger su inversión, esto les permite:

- Usar Space 1 en un nuevo dispositivo tan pronto como éste aparece en el mercado.
- Usar Space 1 como concepto de BYOD (Bring Your Own Device), "trae tu propio dispositivo".
- Probar múltiples dispositivos y elegir el que mejor se adapte a los requerimientos y entorno operativo.

Space 1 está compuesto por un servidor con soporte multinavegador y un cliente. La primera, conocida como la parte back-end, es una aplicación web disponible en la mayoría de los navegadores más comunes y que permite la parametrización y configuración de procedimientos a alto nivel, realizada en tiempo real y visible al cliente.

La aplicación cliente se ejecuta en el dispositivo del usuario y moviliza los contenidos configurados previamente en el back-end.

La configuración con Space 1 es rápida. Con sólo agregar algunos clics en las plantillas predeterminadas que hay disponibles en la parte del back-end, nuestros clientes pueden fácilmente cumplir con sus requerimientos. También los usuarios de back-end pueden ajustar rápida y fácilmente los procedimientos guiados, las instrucciones operativas y la información virtual relacionada con sus procesos comerciales. Los usuarios de móviles también reciben esta información, y por lo tanto pueden mejorar sus habilidades y completar sus tareas de una manera más rápida, segura y precisa.

Space 1 es una solución diseñada para funcionar tanto en línea como sin conexión (online/offline). Todos los datos necesarios para ejecutar la aplicación cliente (es decir, el flujo de trabajo, contenido multimedia, modelos 3D, GIS,…) se descargan cada vez que se accede y se almacenan en el dispositivo, manteniendo así a los técnicos productivos incluso fuera del alcance de la red.

2.7.3.1 CASOS DE ÉXITO SPACE 1

Space 1 Virtual Collaboration es una herramienta eficaz para todas las situaciones en las que se establece un contacto entre dos usuarios que operan desde diferentes sitios de trabajo.

En los siguientes ejemplos podremos ver varios casos de uso típicos.

2.7.3.1.1 Inspección remota

Una empresa líder de suministro de energía confía en *Space1 Virtual Collaboration* para inspecciones remotas. Los materiales son probados por proveedores externos por todo el mundo antes del envío y se aplican procedimientos estrictos para cumplir con los estándares internacionales. Los inspectores internos deben moverse hacia y desde diferentes sitios de trabajo para realizar pruebas y verificaciones de cumplimiento.

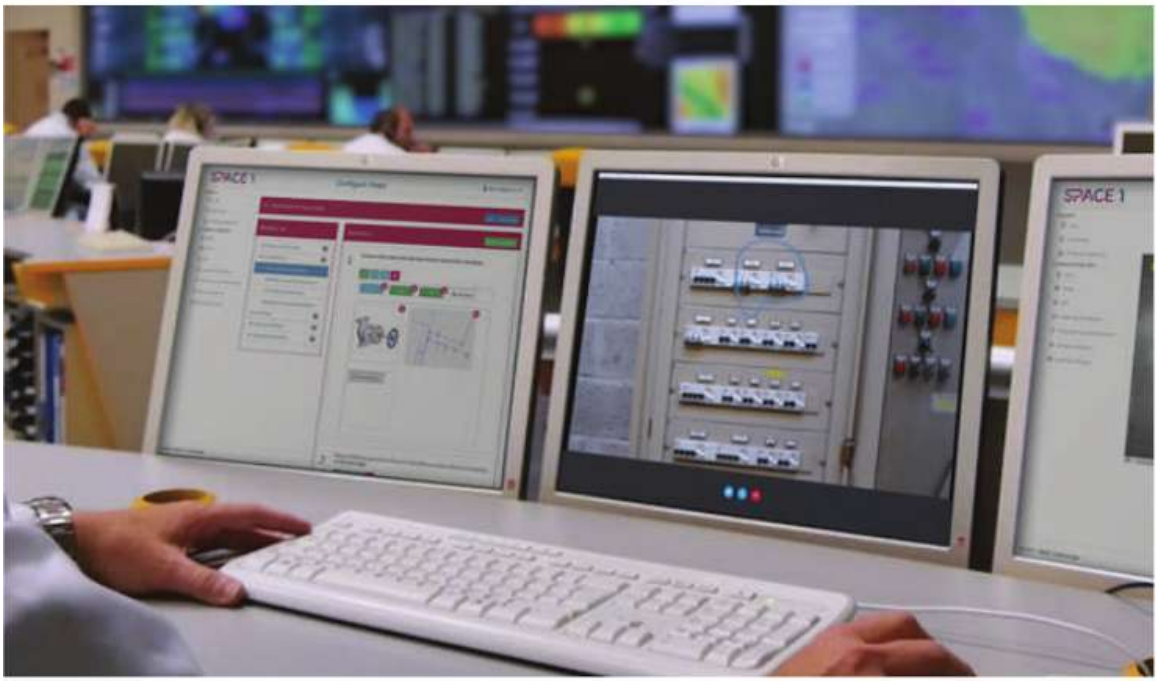
La compañía pretendía reducir costes y los tiempos necesarios para completar las actividades de inspección, dejando así más tiempo para realizar las mismas pruebas. Además, la inspección en modo remoto debe llevarse a cabo como si los inspectores estuvieran físicamente presentes.

Space 1 Virtual Collaboration ha cumplido con todos los requisitos establecidos por la empresa al proporcionar ODG R-7 a sus proveedores para ayudarlos durante las pruebas remotas.

Beneficios

Las ventajas de la solución de *Space 1 Virtual Collaboration* son los siguientes:

- Mayor impacto en la primera vista.
- Resolución de problemas en remoto.
- Reducción del tiempo medio por intervención.
- Mejor servicio y satisfacción del cliente.
- Intercambio de conocimiento entre el personal menos calificado y el más calificado.
- Requerida menor experiencia para llevar a cabo las actividades de campo.
- Confirmación de la correcta ejecución de los procedimientos operativos.
- Creación e intercambio de conocimiento y contenido.



Space 1 Maintenance generalmente se usa para las plantas y también para las cadenas de fabricación y redes de distribución para respaldar la línea de producción en los procesos de administración e instalación como se explica en los casos de uso aquí provistos.

2.7.3.1.2 Mantenimiento en la línea de producción

Una gran multinacional de alimentos y bebidas ha optado por *Space 1 Maintenance* para apoyar las actividades de mantenimiento a lo largo de la línea de producción a través de procedimientos digitalizados, y proporcionó a su personal dispositivos portátiles y gafas inteligentes HoloLens de Microsoft.

El primer procedimiento implementado se refería a la revisión de la válvula de la máquina de llenado, esencial para el proceso de producción. Hasta ahora, la actividad había sido realizada por una empresa externa, que a menudo no podía resolver el problema en un corto periodo de tiempo, lo que obligaba a esta multinacional a detener temporalmente su línea de producción.

Gracias a la solución Space 1, la empresa finalmente pudo digitalizar sus procedimientos y habilitar incluso a los operadores menos capacitados para reemplazar la válvula con éxito y de forma segura, ahorrando tiempo y dinero. Ahora, la cantidad de flujos de trabajo ha aumentado y las competencias del personal han mejorado.

2.7.3.1.3 Soporte técnico en los procesos de instalación

Una empresa italiana de domótica ha elegido la solución *Space 1 Maintenance* para el apoyo del servicio técnico durante la instalación de los sistemas de domótica. Con la ayuda de dispositivos portátiles HMD, ahora pueden crear asistentes interactivos tanto para los procesos de instalación y configuración como para la resolución de problemas.

Antes de introducir Space 1, las instrucciones de instalación y configuración se recogieron en un manual, mientras que ahora todos los pasos se describen a través del flujo de trabajo y la ejecución de cada tarea se informa debidamente y se asocia con la fase individual correspondiente. Los objetos físicos se muestran en 3D pudiendo interactuar y manipularlos. Su diagrama permite visualizar no solo componentes externos sino también los internos que, a su vez se pueden unir a los archivos para proporcionar a los usuarios la información necesaria para ejecutar las tareas de la manera más sencilla posible.

Además, la compañía introdujo *Space 1 Virtual Collaboration* para asistir a usuarios cuando tuvieran un nuevo problema. Si se requiere asistencia desde un colaborador virtual, la interacción se registra para crear nuevos flujos de trabajo de solución de problemas y tener una cobertura completa de tareas y problemas relacionados.

Beneficios

Las ventajas de la solución de *Space 1 Maintenance* son los siguientes:

- Mejora de la eficiencia y la productividad.
- Reducción de errores.
- Mejora de la seguridad del usuario y disminución de lesiones.
- Reducción de los altos costes para la capacitación de los técnicos.
- Se requieren menos habilidades para que el operador pueda llevar a cabo las tareas.
- Disminuir el tiempo medio para realizar el mantenimiento.

2.8 PADAONE GAMES

http://www.padaonegames.com

Pedro Antonio González Calero

Prof. José García Santesmases, 9.
E-28040 Madrid

pedro@padaonegames.com

PadaOne Games surge como una empresa de desarrollo de videojuegos por iniciativa de tres profesores de la Facultad de Informática de la Universidad Complutense de Madrid (UCM), Pedro González, Marco Antonio Gómez y Pedro Pablo Gómez. Estos tres profesores son, además, los responsables y parte del equipo docente de las tres modalidades del máster de videojuegos de la misma universidad y miembros del equipo de investigación que lleva más de 10 años centrando su trabajo en aspectos relacionados con los videojuegos, su arquitectura e inteligencia artificial. En definitiva, un grupo de pioneros en la Universidad española que fueron de los primeros en apreciar desde el mundo académico el gran potencial que había en lo que hasta ese momento se había desarrollado fundamentalmente a espaldas de la Academia.

La idea de esta empresa surge en primer lugar para transferir los resultados de investigación en herramientas de autoría para videojuegos desarrollados en el Grupo de aplicaciones de inteligencia artificial (GAIA) de la UCM. El desarrollo de esta línea de I+D confiere un gran potencial de crecimiento a la empresa a medio y largo plazo. El primer resultado en esta línea es "Behavior Bricks" (http://www.padaonegames.com/bb/) un middleware para Unity 3D que facilita la creación de contenido para videojuegos sin necesidad de programar.

Una segunda línea de investigación que también está dando lugar a transferencia es la aplicación de la tecnología de los videojuegos a otros ámbitos más allá del entretenimiento, lo que se conoce como juegos serios, donde miembros de GAIA llevan investigando desde principios de siglo. Relacionado con este trabajo está el desarrollo del juego oficial de la serie de RTVE "Carlos Rey Emperador" (http://juegocarlos.rtve.es/), que, asesorado por historiadores, recrea la política del reinado del rey Carlos I de España, y el desarrollo de nuestra primera aventura cultural, "Enigma Galdiano" (http://www.padaonegames.com/enigma/), desarrollado en colaboración con la Fundación Lázaro Galdiano de Madrid y financiado en parte por el Ministerio de Cultura.

PadaOne Games busca también convertirse en un partner tecnológico para estudios españoles e internacionales que necesiten un equipo con experiencia en el desarrollo de videojuegos para consola. Esta colaboración puede ser en forma de co-producción, como es el caso del videojuego "Zombeer" (http://zombeerthegame.com) desarrollado para PlayStation 3 por PadaOne en colaboración con la empresa Moonbite, o puede consistir en el desarrollo de una versión para consola de un videojuego desarrollado previamente para otra plataforma, como es el caso del videojuego "Shiny the Firefly" (http://www.stageclearstudios.com/games/shinythefirefly/) que hemos desarrollado en PadaOne para Wii U a partir de una versión previa desarrollada por la empresa Stage Clear para dispositivos móviles.

Por último, PadaOne Games busca servir de lanzadera de los estudiantes del Máster de videojuegos de la Universidad Complutense, un programa de posgrado que ya va por la decimotercera edición y es una referencia para la industria del videojuego en España. Esto se puede concretar en la publicación de juegos desarrollados por los estudiantes, como es el caso de los videojuegos "90Blox" (http://www.padaonegames.com/90blox/) y "Germbusters" (http://www.padaonegames.com/germbusters/), en el desarrollo de una versión para consola de videojuegos desarrollados por grupos de estudiantes, como es el caso del videojuego "Roving Rogue" (*http://www.padaonegames.com/rr/*).

2.8.1 Enigma Galdiano

El primer resultado de PadaOne Games en esta línea es "Enigma Galdiano" un juego para dispositivos móviles que convierte la visita al museo Lázaro Galdiano de Madrid en una aventura cultural. El juego está pensado para ser jugado por un adulto y un niño. El niño lleva un dispositivo móvil con el juego y el adulto un mapa en papel con pistas adicionales. El juego consiste en ir encontrando ciertas obras del museo a través de las pistas que se van proporcionando, utilizando el móvil para reconocer las obras en cuestión. Con algunas de las obras se asocian mini-juegos que es necesario completar para seguir avanzando y que en algunos casos requieren utilizar información sobre las obras.

El objetivo principal del juego es proporcionar al adulto una herramienta para interactuar con el niño y despertar su curiosidad e interés sobre el contenido del museo. Además de la app, que es gratuita, se invita a los aprendices de pirata a que compren el mapa del tesoro que incluye pistas que les hará un poco más sencilla la búsqueda. En el mapa se incluyen indicaciones sobre las salas del museo donde se encuentran los cuadros junto con otras pistas visuales:

Nuestros principales acompañantes serán la capitana del Enigma, Anne Jack y su inseparable loro el Sr. Mendoza:

Ellos nos irán guiando en la búsqueda, donde nos iremos encontrando con otros personajes, como el malvado "Gobernador Galdiano", que son ilustraciones creadas a partir de cuadros del museo:

Para avanzar por las pruebas de la aventura es necesario encontrar ciertas obras de la colección y apuntar a ellas con el móvil para que este reconozca la imagen en cuestión y nos permita acceder al contenido asociado a esa pista:

El contenido puede ser un mini-juego, contenido en 360º u otra pista para avanzar en el juego. El juego contiene dos "salas mágicas", una al principio y otra al final, que son escenas en 3D que el aventurero puede visualizar en 360º utilizando su móvil apuntando en cualquier dirección para explorar la sala:

En estas escenas usamos el giroscopio del móvil para mover la cámara según se mueve el usuario, o bien ofrecemos un joystick en la pantalla si detectamos que el móvil en cuestión no tiene giroscopio.

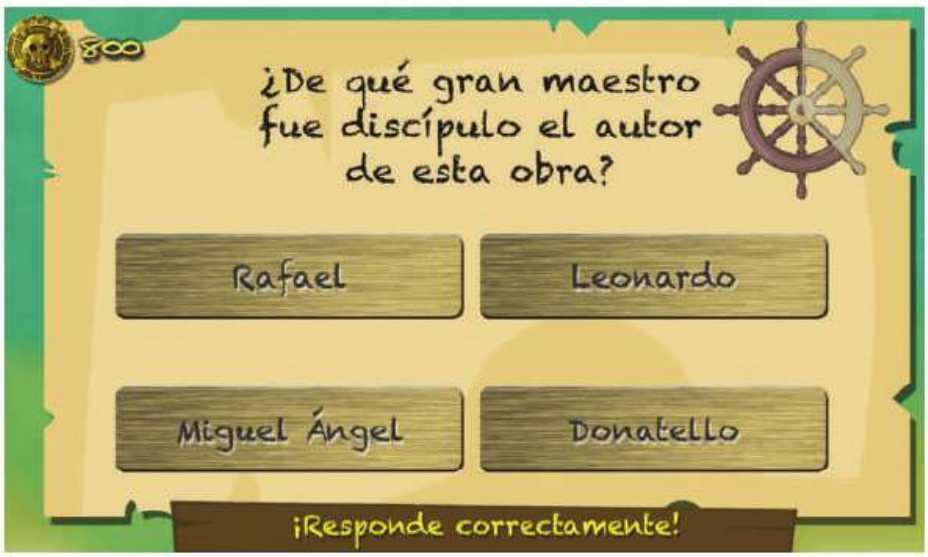
Entre los mini-juegos que ofrecemos al aventurero, están encontrar el orden correcto en que se deben pulsar los cajones de un mueble:

encontrar las diferencias entre la imagen del móvil y el cuadro que tenemos delante:

o capturar a un duende que se mueve mágicamente entre las figuras del cuadro al que apuntamos con el móvil:

Existe el riesgo de que la aventura se convierta en una carrera donde el niño se precipita de una obra a la siguiente sin prestarles mayor atención. Para evitarlo,

cada tres o cuatro pruebas de la aventura incluimos una pregunta sobre la obra frente a lo que nos encontramos. La respuesta a la pregunta está en el cartel explicativo de la obra, y hay un tiempo máximo para responder. De esta forma buscamos un equilibrio entre invitar al aventurero a aprender algo sobre las obras, pero sin sacarle abruptamente de la dinámica del juego:



Según vaya avanzando en la aventura el jugador irá recopilando algunos objetos virtuales, hasta llegar a la última prueba donde, en la sala mágica del tesoro, se le entregará la espada del Conde de Tendilla, uno de los mayores tesoros escondidos en las Islas Galdianas:

Una vez terminada la aventura, que supone una visita de aproximadamente una hora, se invita a los aventureros a que sigan utilizando el catalejo mágico para descubrir otros secretos del museo. De esta forma, una vez que ya hemos conseguido el interés del niño, le pedimos que siga de una forma más reposada buscando las historias y secretos escondidos entre las obras del museo.

Hemos realizado una selección de obras especialmente representativas del museo, y recopilado una breve anécdota acerca de ella, de forma que el jugador cuando encuentre la obra en cuestión y la reconozca con su móvil, podrá acceder al contenido. Para encontrar las obras los jugadores disponen de estas indicaciones visuales:

y este es el tipo de anécdotas o datos curiosos que es posible desvelar:

En definitiva, buscamos que el niño comprenda que un museo esconde historias y tesoros que le pueden interesar y animarle a seguir investigando ese mundo.

3

CREACIÓN DE UNA APLICACIÓN DE REALIDAD VIRTUAL Y REALIDAD AUMENTADA BÁSICA

3.1 INTRODUCCIÓN

En este capítulo aprenderemos a configurar nuestro sistema Unity 3D para poder generar una aplicación de Realidad Aumentada y Realidad Virtual Básicas. La idea es conseguir generar dos aplicaciones para Android en formato .apk que podamos probar en nuestro Smartphone a la finalización del mismo.

Los requerimientos de software externo y su correcta configuración son imprescindibles para llevar a buen puerto nuestro objetivo, por lo que conviene repasar la sección donde detallamos esta parte en caso de producirse algún tipo de error en la compilación.

Aunque incluiremos los proyectos completos en la sección de "Contenido descargable" del libro, también se incorporarán diferentes paquetes "unity" con distintos elementos 2D, 3D y Scripts que nos facilitarán la construcción del proyecto desde 0.

Con objeto de no hacer demasiado pesado el contenido y no perdernos en detalles alejados del objeto principal de este libro, solamente se comentará el código fuente que tenga que ver con el uso de la Realidad Aumentada o Realidad Virtual. Se dejará como "Contenido descargable" la explicación del resto de código fuente para los lectores menos expertos que quieran ver como se han desarrollado algunos de los scripts que no explicamos.

Pasamos ya a la primera parte del capítulo creando una aplicación de Realidad Aumentada básica.

3.2 APLICACIÓN DE REALIDAD AUMENTADA BÁSICA

Para crear una APP de Realidad Aumentada básica de la manera más sencilla posible hemos elegido el SDK de Vuforia por su sencillez y la reciente integración dentro de Unity 3D como un tipo de componente más integrado de serie. Hay que reseñar que esta integración y el método descrito en este libro solamente está disponible desde la versión 2017.2 de Unity 3d. En versiones anteriores del motor la integración del SDK se debe de hacer de manera manual integrando un paquete Unity ("*.unitypackage*") dentro del proyecto.

Vuforia es una plataforma de software que nos permite crear Apps de Realidad Aumentada de manera muy sencilla e intuitiva. Permite a los desarrolladores añadir funciones de visión artificial para reconocer imágenes y objetos del mundo real, pudiendo añadir objetos virtuales asociados a posteriori e interactuar con ellos.

En este ejemplo trataremos una APP básica con un marcador simple (imagen) dejando la explicación del resto de características y funcionalidades de Vuforia para el siguiente capítulo.

Pero antes de empezar a desarrollar nuestra APP debemos de configurar nuestro entorno de trabajo adecuadamente. Esta configuración servirá para poder trabajar los dos ejemplos básicos, tanto el de Realidad Aumentada, como el de Realidad Virtual.

Configuración del Sistema de Desarrollo

Para preparar nuestro sistema de desarrollo y poder crear Aplicaciones con Vuforia, necesitamos tener instalado diferentes paquetes de software. Haremos una breve descripción de estos paquetes de software necesario y mostraremos algunas capturas orientativas para la correcta instalación.

3.2.1 El sistema Operativo

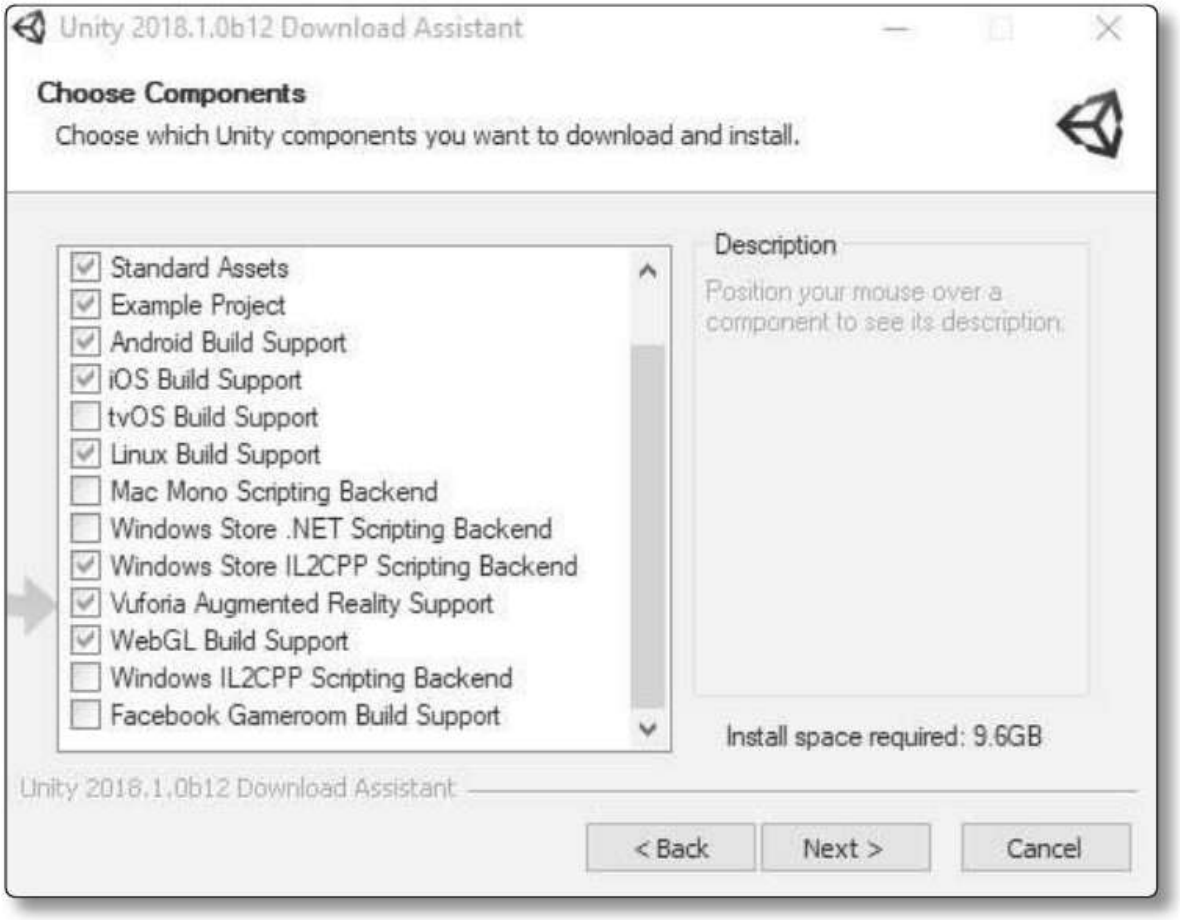
El motor de gráficos Unity 3D dispone de diferentes versiones para Windows, Mac y Linux. La información reflejada en este libro se ha desarrollado usando un PC con Windows 10. En el caso de querer compilar para iOS necesitaremos un Mac con el sistema operativo y *Xcode* actualizado.

3.2.2 Unity 3d

Para la descarga de Unity3D debemos de ir a *https://unity3d.com/es* y descargar la última versión del motor. En nuestro caso será la versión 2018. Dependiendo del sistema operativo debemos elegir un instalador u otro. Unity dispone de un sistema de licencias con pago mensual dependiendo del nivel de ingresos que tenga nuestra empresa e incorpora una serie de paquetes adicionales dependiendo la licencia que elijamos.

Para los fines didácticos de este libro existe una versión "Personal" de uso gratuito con todas las prestaciones básicas del motor que, para lo que vamos a ver, es más que suficiente.

Como indicamos en el capítulo 3 debemos de incluir en la instalación los componentes necesarios para compilar nuestra APP. En el caso de querer generar una APP para Android, **Android Build Support**, para iOS, **iOS Build Support** y por supuesto el componente de Vuforia, **Vuforia Augmented Reality Support**.



3.2.3 Java SDK (JDK)

El SDK de Java es necesario para la compilación de nuestra APP para Android. Unity usa la API Android instalada con el *Android SDK* y este a su vez la API de Java. Para descargarlo debemos ir a la web *http://www.oracle.com/technetwork/java/javase/downloads/index.html*

3.2.4 Android SDK

El *Android SDK* debemos de tenerlo instalado para usar la API de Android en sus distintas versiones. Lo podemos descargar en:

https://developer.android.com/studio/index.html?hl=es-419

Normalmente viene con la última API Android disponible instalada y con las opciones por defecto es suficiente para los ejemplos que vamos a tratar en el libro. Después veremos cómo acceder a él para instalar algún paquete extra o una versión diferente de la API si es necesario.

3.2.5 Android NDK

También es necesario para compilar para Android si usamos IL2CPP. La última versión está disponible en el siguiente enlace:

https://developer.android.com/ndk/downloads/index.html

La versión de Unity 3d que hemos usado nos indicaba que era compatible con la r13b, aunque *Android NDK* ya va por la r16. Es posible en esto cambie en revisiones futuras por lo que debemos estar atentos en caso de que tengamos algún error. Actualmente Unity 3D te avisa si la versión no es compatible cuando enlazas con *Android NDK* (Ver apartado "Enlazar todo con Unity 3D" un poco más adelante).

Para la descarga de versiones antiguas (En nuestro caso la r13b) se debe de seguir este enlace:

https://developer.android.com/ndk/downloads/older_releases.html

Tras la descarga del .zip se descomprime en una carpeta de nuestra elección.

3.2.6 Unity Remote

Esta instalación es opcional, aunque muy recomendable.

Nosotros recomendamos que para determinados ejemplos donde se usen funciones sobre el hardware del dispositivo se use para la depuración del software esta APP.

Unity Remote es una APP (Existe tanto para Android como iOS) que nos permite conectar un dispositivo físico (Smartphone o Tablet) a Unity 3D permitiendo usar el hardware del mismo (pantalla multitáctil, acelerómetro, brújula o giroscopio) en la depuración de la APP. Para usar *Unity Remote* debemos seguir los siguientes pasos:

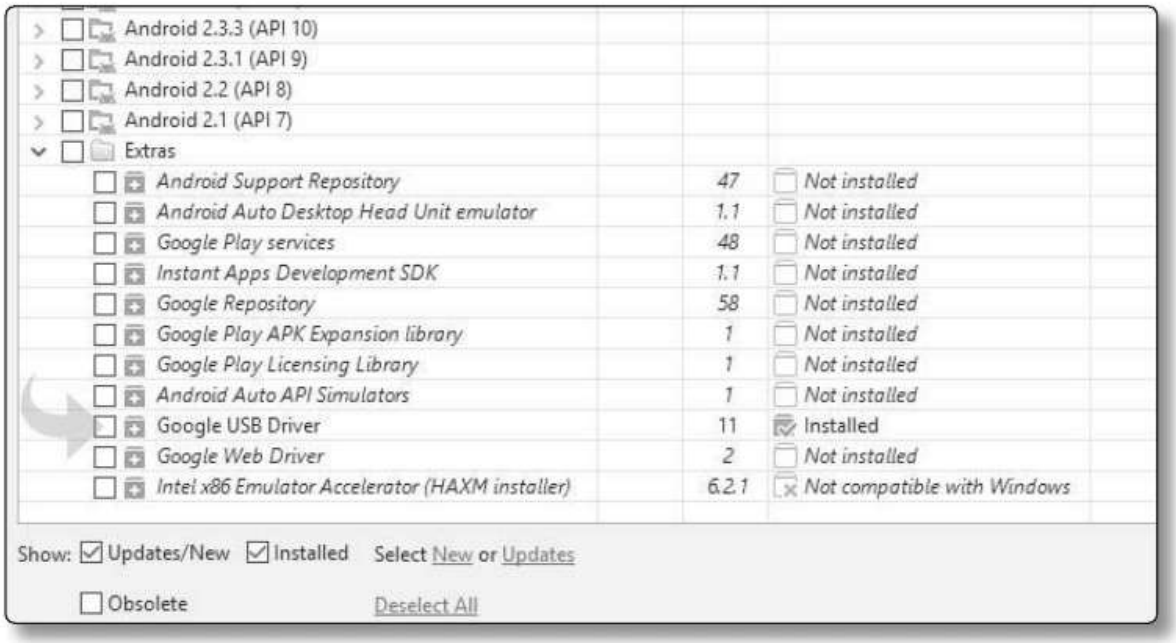
- ⇨ Para dispositivos Android debemos de activar las opciones de desarrollo y depuración USB en el dispositivo. Para activar las opciones de desarrollo basta con irnos a **Ajustes→Información del teléfono** y pulsar 7 veces encima del "número de compilación". Una vez activadas ya nos aparecerá el menú "**opciones de Desarrollo**" en Ajustes y podremos activar "**Depuración por USB**".

- ⇨ Después de esto debemos instalar *Unity Remote* en nuestro dispositivo.

  Está disponible para su descarga tanto en *Google Play* como en *iTunes*

⇨ Una vez instalado en nuestro dispositivo tenemos que asegurarnos que en *Android SDK* tenemos instalado el driver *Google USB*.

Para acceder a *Android SDK* (En el caso de Windows) debemos de irnos a **c:\usuarios\<<Carpeta de Usuario>>\AppData\Local\Android\android-sdk\SDK Manager.exe** e instalar **Extras→Google USB Driver**



ⓘ NOTAS

Normalmente *Android SDK* instala la última API de Android que es la más actualizada, pero para determinadas pruebas de compatibilidad se puede usar otra API inferior desde Unity 3D y se debería instalar también con *el SDK Manager*.

3.2.7 Xcode

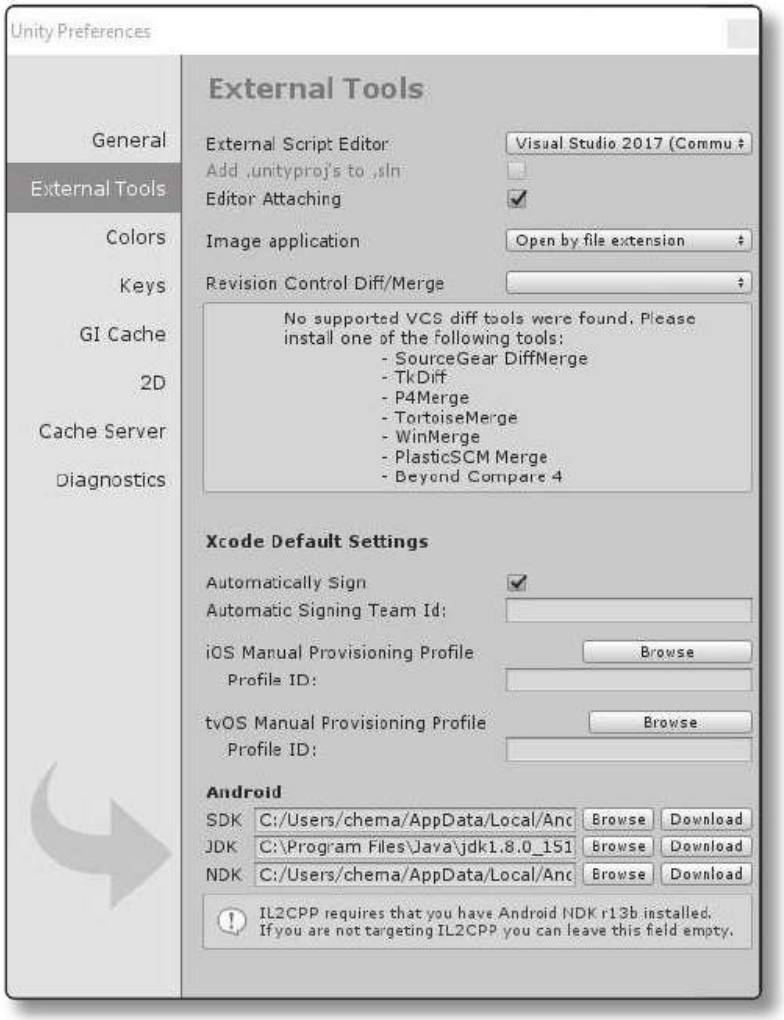
Para poder desarrollar Apps para iOS es necesario tener instalado este compilador en un Mac. La configuración de *Vuforia* requiere que tanto el Sistema operático como el *Xcode* estén actualizados a la última versión.

3.2.7.1 CONFIGURAR TODO EN UNITY 3D

Lo primero que haremos es abrir **Edit→Preferences** en Unity 3D. Con esto mostramos el panel de preferencias del motor. En la pestaña **External Tools**, en la parte inferior, disponemos de varias rutas que debemos especificar:

- ⇨ SDK: Ruta a *Android SDK*.
- ⇨ JDK: Ruta a *Java SDK*.
- ⇨ NDK: Ruta a *Android NDK*.

Estas rutas son necesarias para poder compilar.



ⓘ NOTAS

En el caso del NDK, como indicamos antes, la versión de Unity con la que hemos trabajado admite solamente la versión r13b como se puede ver en la imagen anterior. Esto puede cambiar para futuras versiones.

Finalmente, para configurar *Unity Remote* debemos de irnos a **Edit→Proyect Settings→Editor** y elegir **Any Android Device** o **Any iOS Device** según si el dispositivo que estemos usando para depurar es Android o iOS.

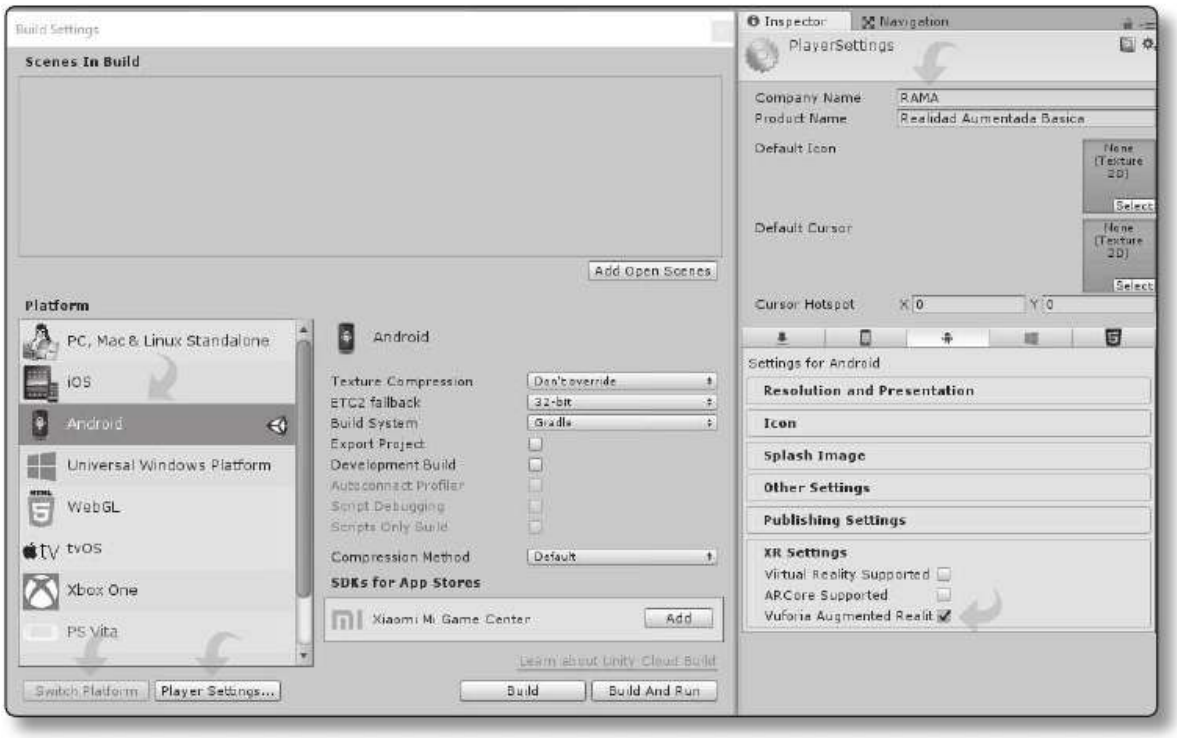
Con esto ya tenemos todo nuestro entorno de desarrollo preparado con todo lo necesario para comenzar una APP con Vuforia.

Si, lo sabemos, esto es un rollo, pero es necesario para que luego todo funcione correctamente.

¡¡Ya pasamos a la parte divertida!

Una vez instalado el motor y teniendo claro la plataforma destino, en nuestro caso Android, debemos de ir a **File→Build Settings**. Una vez allí, seleccionamos **Android** en *Platform* y pulsamos en el botón abajo a la izquierda *Switch Platform*.

Después de esto, pulsamos en *Player Settings* y aquí aparece una serie de campos en la ventana *Inspector* que debemos de rellenar. En este momento solamente rellenaremos el nombre de la compañía (*Company name)* con el nombre que queramos (Nosotros hemos puesto RAMA) y en *XR Settings* debemos de marcar *Vuforia Augmented Reality*. Este último paso es muy importante realizarlo para que Vuforia funcione correctamente.

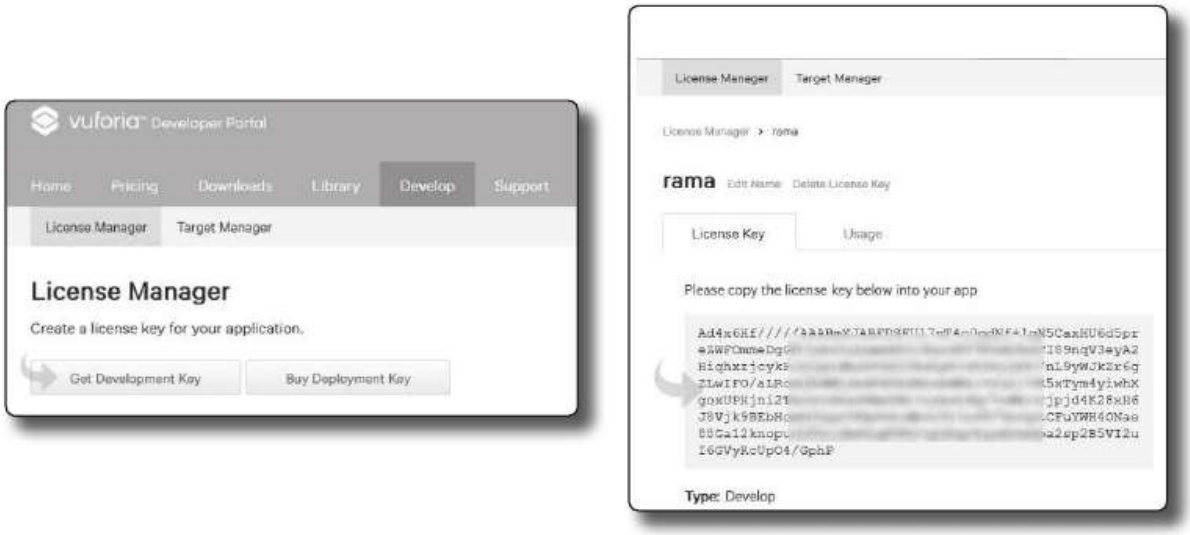


Vuforia nos permite crear Apps de manera gratuita con una marca de agua en el visor. Para esto usamos una licencia que debemos de obtener en su Web (*https://developer.vuforia.com/*). Como en cualquier otro servicio de internet nos debemos de registrar aportando una serie de datos básicos, un email y una contraseña.

Tras el proceso de registro pertinente podemos acceder a la sección *Develop* y pulsar en *Get Development Key*.

Nos piden que proporcionemos un nombre a nuestra APP y ¡Ya tenemos una licencia disponible!

Esta licencia básica nos permite usar hasta 100 marcadores diferentes de manera gratuita para hacer nuestras pruebas (Siempre con la marca de agua).



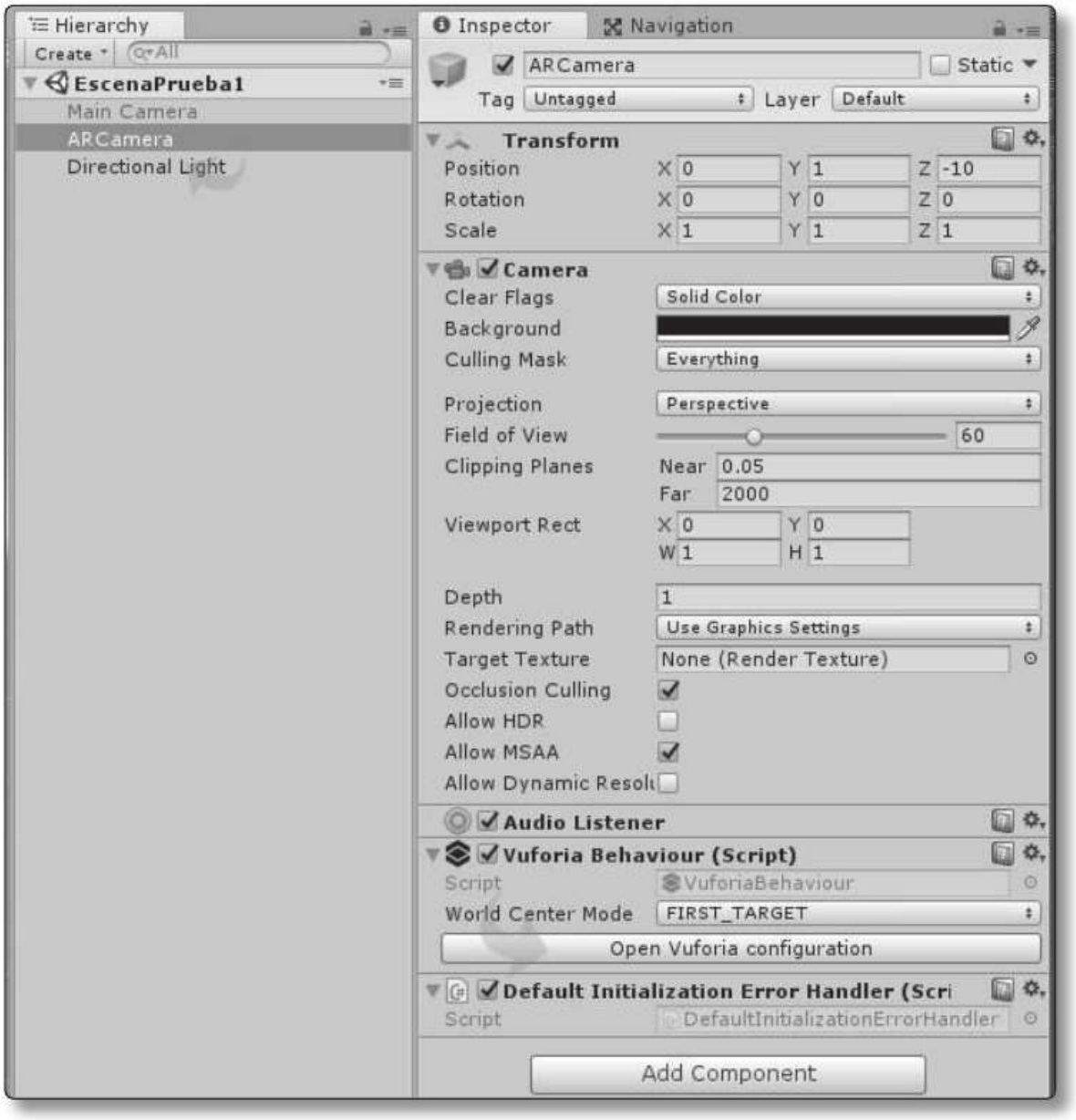
Existen diferentes tipos de licencias con sus limitaciones que se explicarán más detenidamente en el capítulo siguiente. Todas son un código alfanumérico como podemos ver en la imagen anterior que debemos de copiar y tener controlado para insertarlo en nuestro proyecto.

Una vez dispongamos de la licencia debemos de crear una escena (**File→New Scene**) en Unity 3D. Por defecto una escena se crea con un objeto *Main Camera* en la pestaña *Hierarchy*. Esta cámara no nos servirá para nada por lo que recomendamos quitarla o simplemente desactivarla seleccionándola y desmarcando el Tag al lado del nombre en la pestaña *Inspector*.

Para nuestra escena de Realidad Aumentada debemos de incluir una cámara especial llamada **AR Camera**. Este *Asset* está disponible junto al resto de *GameObjects* standard de Unity 3D en el menú **GameObject→Vuforia→AR Camera**.

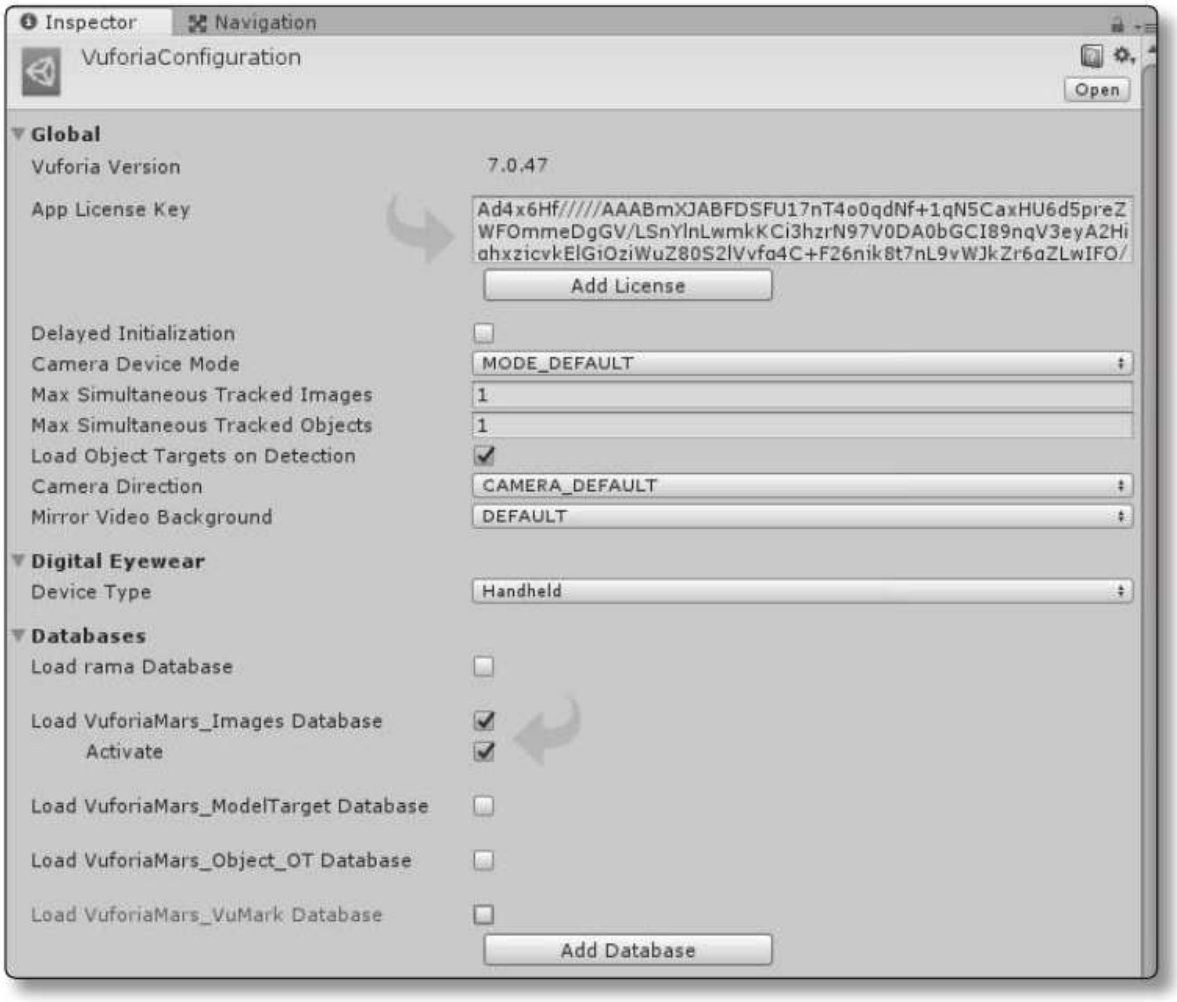
Al insertarla nos aparecerá un mensaje que nos pide permiso para importar varios elementos necesarios para que todo funcione. Pulsamos en el botón *Import* y vemos como aparecen una serie de carpetas nuevas colgadas de la Carpeta *Assets* en la pestaña *Proyect*.

Para configurar nuestra recién adquirida licencia nos vamos a la pestaña *Hierarchy* y seleccionamos el nuevo objeto **ARCamera**. En la pestaña *inspector* pulsaremos en el botón *Open Vuforia configuration* donde definiremos varios parámetros necesarios para nuestro ejemplo.



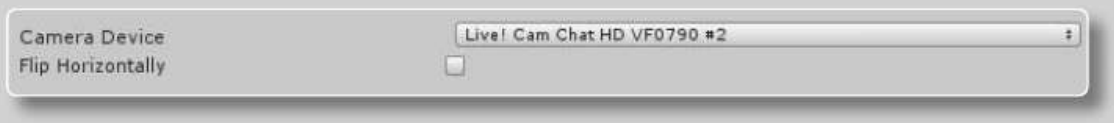
Esta ventana de configuración de Vuforia se puede llamar también desde el menú **Windows→Vuforia Configuration**.

Aquí lo más importante que debemos saber para nuestro ejemplo es que debemos incluir nuestra licencia en *App License Key* y que debemos activar la base de datos que viene de ejemplo con Vuforia (*Vuforia Mars*). El marcar esta base de datos no permitirá usar uno de sus marcadores asociados.



ⓘ NOTA

En caso de disponer de una Webcam es muy importante configurarla aquí también ya que nos permitirá usarla para probar nuestra APP en el motor sin necesidad de compilarla. Nosotros por nuestra experiencia recomendamos usar al menos un cámara con resolución 720P ya que con resoluciones inferiores Vuforia no es capaz de reconocer en muchos casos los marcadores. La cámara debe de estar enchufada en el PC antes de arrancar Unity 3D. Si no es así a veces no es capaz de reconocerla.



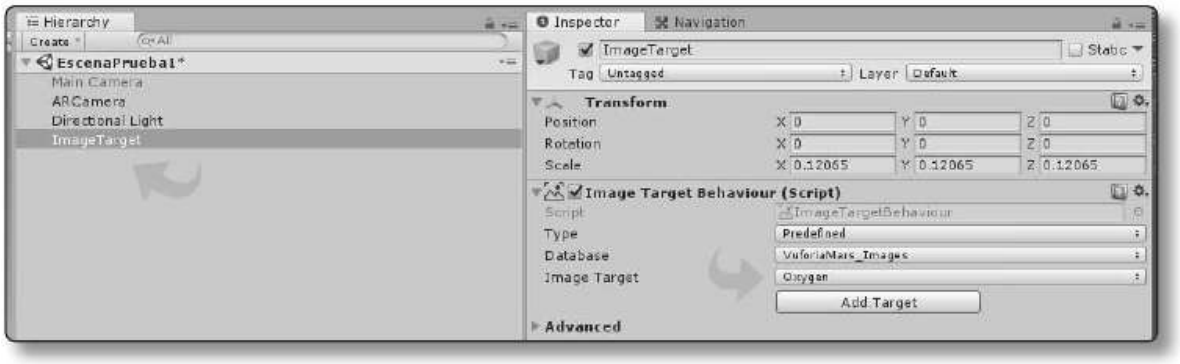
Una vez configurada la cámara, insertada licencia y activada la base de datos de ejemplo, insertamos en escena el marcador. El marcador es el objeto que Vuforia detectará y con respecto al cual nosotros posicionaremos todos los elementos virtuales que necesitemos.

En el próximo capitulo aprenderemos a crear y configurar nuestra propia base de datos con marcadores personalizados, pero en este ejemplo usaremos la base de datos de ejemplo (*Vuforia Mars*) que es la que hemos activado en configuración.

Lo primero que debemos de hacer es localizar el marcador de ejemplo que trae Vuforia e imprimirlo para nuestras pruebas. Este marcador está dentro de la carpeta **Assets\Editor\Vuforia\ForPrint\ImageTargets**. Si encima de esta carpeta pulsamos en el botón derecho del ratón y elegimos *Show in explorer* nos aparecerá una ventana del explorador y podremos abrir el fichero **target_images_ A4.pdf** e imprimirlo.

Con el marcador en nuestras manos debemos de insertar el objeto **ImageTarget** en escena. Para esto nos vamos a **GameObject→Vuforia→Image** y ya tendremos el objeto disponible en *Hierachy*. Todos los *GameObjects* que cuelguen de **ImageTarget** se activarán o no dependiendo de si el objeto **AR Camera** lo detecta.

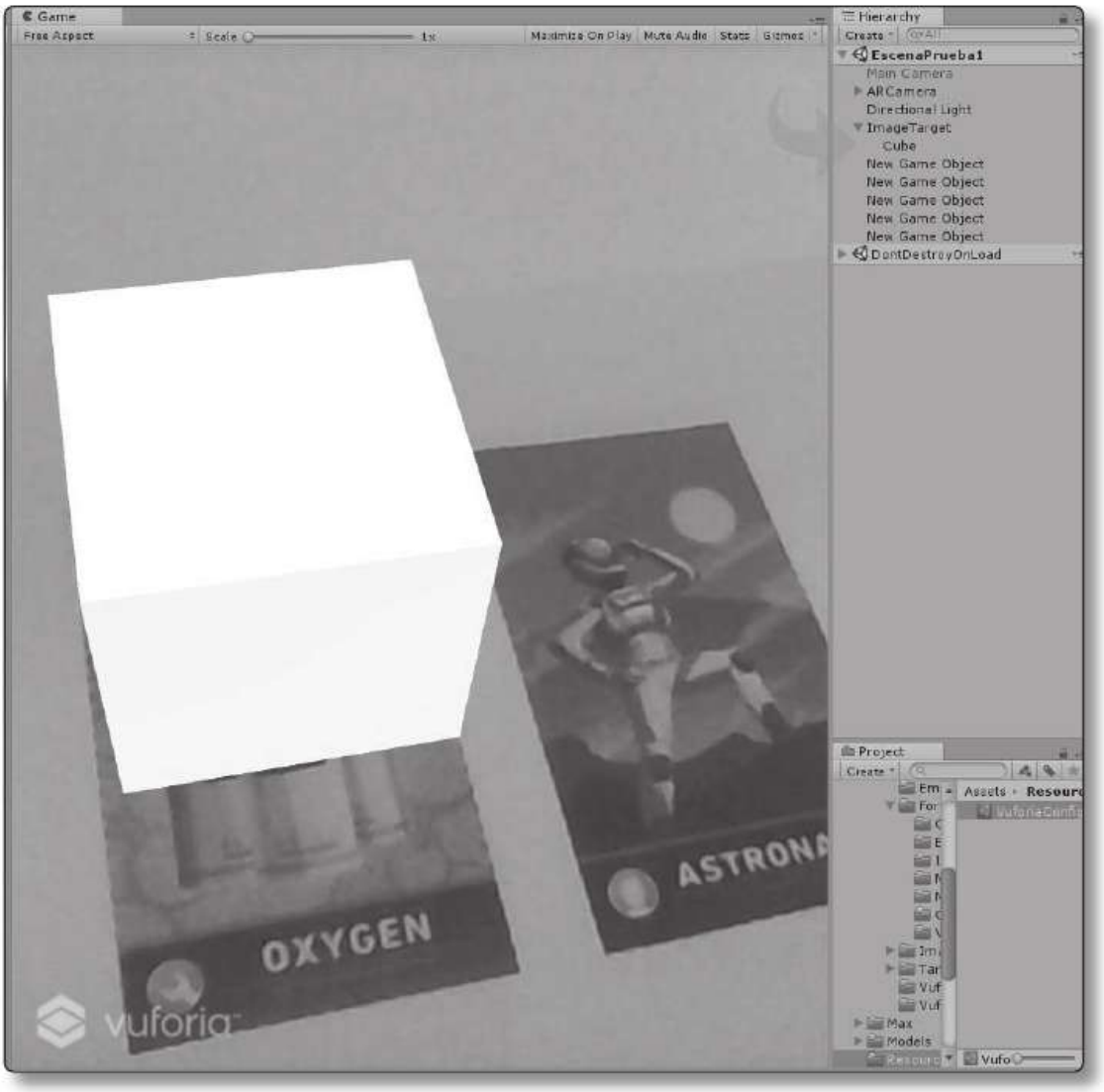
Dentro del objeto, en el *Script* **Image Target Behaviour**, configuramos la base de datos que vamos a usar (Podemos tener varias instaladas) y el marcador que vamos a elegir. En nuestro caso la base de datos es *VuforiaMars_Images* y el marcador es *Oxygen*.



Con objeto de hacer una prueba intermedia para ver si el motor de Realidad Aumentada funciona asociamos un objeto simple al marcador. En nuestro caso usaremos un CUBO. Nuevamente en **GameObject→3D Object→Cube**. El objeto cubo lo arrastramos para que "dependa" del **ImageTarget** como hemos indicado antes.

Debemos posicionarlo y escalarlo de la manera que queramos que se quede con respecto a nuestro marcador (**ImageTarget**). De esta manera, al reconocer el sistema

el marcador, se superpone el objeto virtual (En este caso el cubo) sobre la imagen capturada por la cámara según lo hayamos definido en escena. En esta imagen vemos como debería de mostrárnoslo el motor una vez le demos al Play en la escena.



Con esto ya tendríamos un objeto simple asociado al marcador que se mostraría cuando la cámara del dispositivo lo detecta y que se oculta cuando el marcador se pierde.

Una vez que comprobamos que la cámara de realidad aumentada detecta el marcador y muestra el objeto asociado ya sabemos que Vuforia funciona correctamente y podemos continuar añadiendo funcionalidades a nuestro ejemplo.

Para continuar con el ejemplo práctico vamos a posicionar en escena 3 cubos con los logos de la editorial (Ra-Ma), y de los dos estudios creativos que han participado en este libro (Radikal Dreamers y Digital Dream Factory).

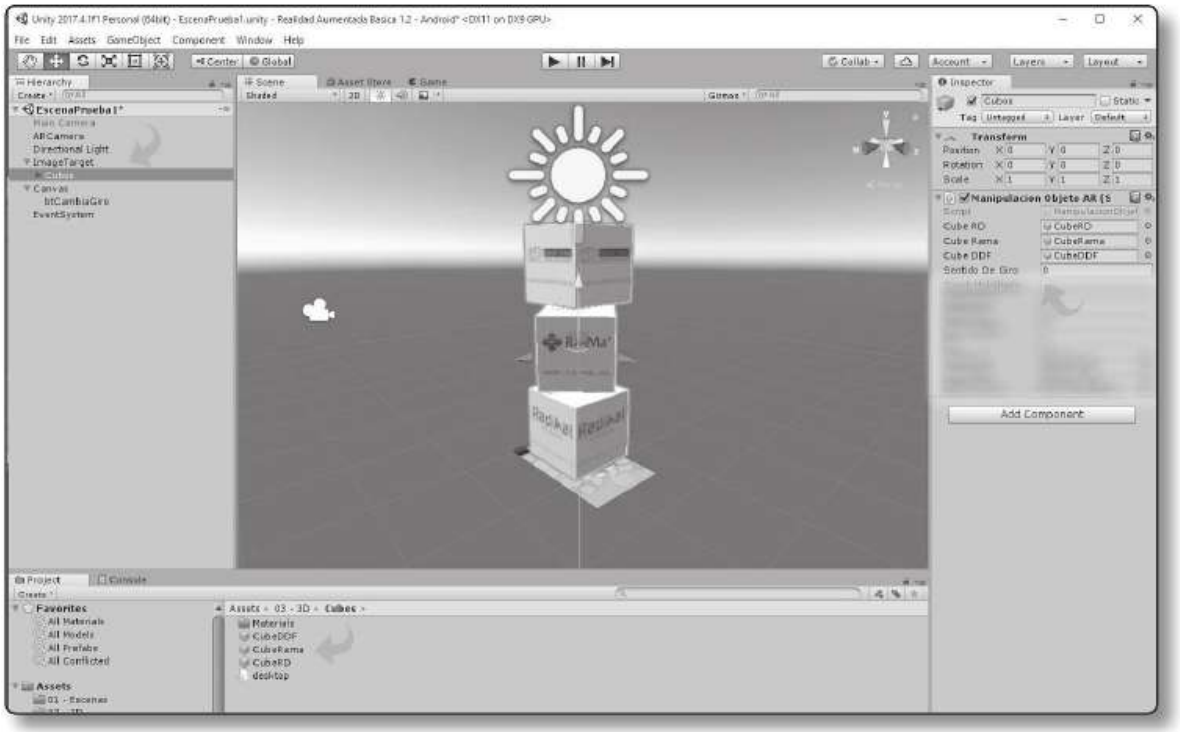
Los tres cubos están girando a una velocidad constante y en sentidos enfrentados.

Hemos preparado un paquete con los *assets* necesarios para este ejemplo en el contenido descargable. Se puede importar el fichero **MaterialesRealidadAumentadaBasica.unitypackage** en **Assets→Import Package→Custom Package**.

Al importar tenemos varias carpetas con los *assets 2D*, los *assets 3D* y los *scripts* necesarios.

Creamos un objeto llamado "Cubos" que contendrá los tres cubos previamente preparados con sus materiales, un componente *Rigidbody*, un *Box Collider* y un *script* llamado **OpenURL**. Este script llama a la URL que configuremos pulsando sobre el cubo individual. Este objeto "Cubos" debe colgar del **ImageTarget** borrando el cubo inicial que hemos usado para la prueba intermedia. Los cubos están disponibles como *Prefabs* en la carpeta **Assets\03 - 3D** y los *Scripts* en **Assets\04 – Scripts**.

Debemos asociar a este objeto "Cubos" el script **ManipulacionObjetos** que contiene el código que hace que los cubos giren y algunas funciones más que iremos explicando para que sirven. No debemos olvidar arrastrar los *GameObjets* de los cubos en su variable correspondiente del *Script*. El cubo central lo hemos girado 45 grados con respecto a los otros dos. La estructura quedaría de esta manera:



ⓘ NOTAS

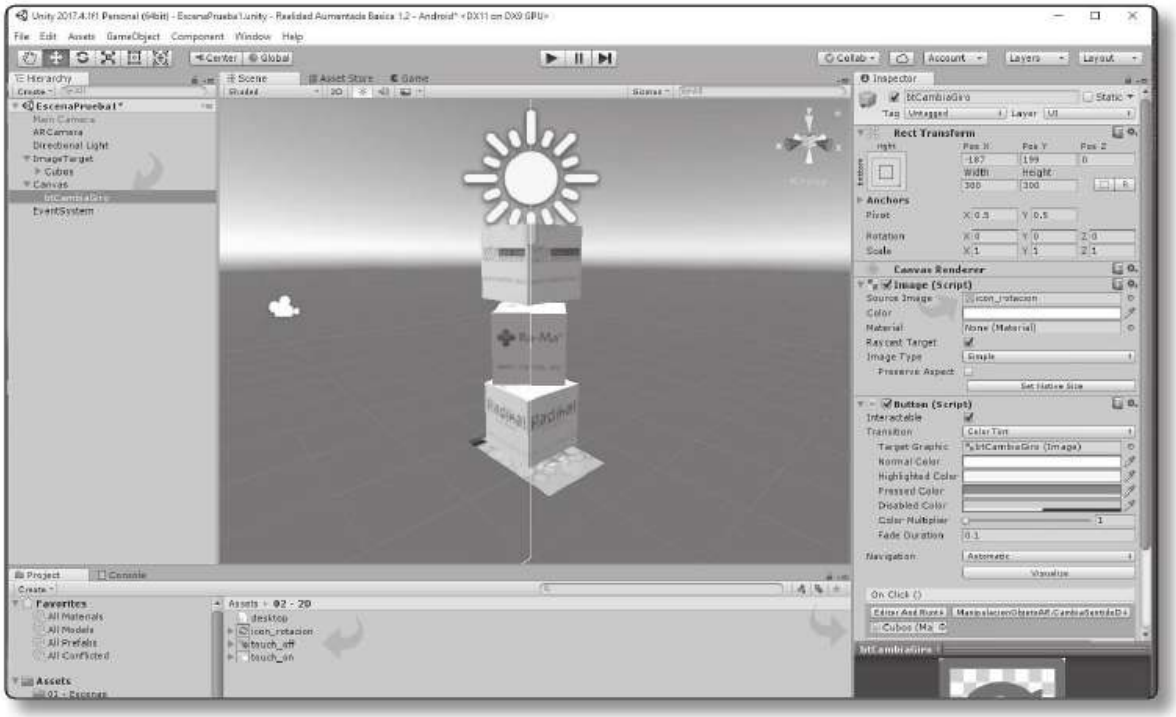
Vemos como existen otras variables en el Script (En la imagen emborronadas) que usaremos en la segunda fase del ejemplo y que iremos explicando más adelante. Ahora mismo no debemos prestarles atención.

Si ejecutamos el proyecto vemos como los cubos giran sobre el marcador una vez enfocamos con la cámara.

Para meter un primer grado de interacción hemos creado una función básica llamada *CambiaSentidoDeGiro()* en el script **ManipulacionObjetos** que nos permitirá cambiar el sentido de giro de los cubos.

Para llamar a la función integramos un *Canvas* nuevo creando un `UI.Button`. Esto lo hacemos llamando a **GameObject→UI→Button**.

Las imágenes para el *Sprite* de los botones las tenemos en la carpeta **Assets\02-2D** del ejemplo. Finalmente asociamos al botón *CambiaSentidoDeGiro()* y vemos como pulsando efectivamente cambia el sentido de giro.



Resumiendo, hasta ahora, tenemos implementadas dos tipos de interacciones con los objetos virtuales:

- ⇨ Pulsamos sobre los cubos individuales y abrimos una URL.
- ⇨ Podemos cambiar el sentido de giro.

Para pasar a un grado de interacción mayor vamos a usar el sistema de entrada (*Input*) *Touch*.

Unity 3D nos permite controlar el número de dedos (*Touches*) que tenemos en la pantalla táctil del smartphone. Para poder usar este sistema sin necesidad de compilar debemos de usar *Unity Remote* en nuestro smartphone y tenerlo conectado mediante USB al PC.

> ⓘ **NOTAS**
>
> Para que no te vuelvas loco, para que funcione correctamente debemos de habilitar "Unity Remote" en el Sistema Android y después arrancar Unity3D. Si arrancamos "Unity Remote" después de arrancar Unity3D no funciona correctamente.

Para aprovechar las capacidades del sistema *multi-touch* en la manipulación de nuestros cubos vamos a implementar las 3 funciones básicas en nuestro script **ManipulacionObjetoAR**:

- ⇨ *Rotación*: Con un solo dedo podremos rotar nuestro conjunto de Cubos en la dirección que arrastremos el dedo.

- ⇨ *Escalado*: Con dos dedos podremos aumentar o disminuir el tamaño del objeto en la escena. A esto se le llama *Pinch Zoom* o "Zoom por pellizco" ya que juntando los dedos disminuimos el tamaño y alejándolos lo aumentamos.

- ⇨ *Translación*: Con tres dedos sobre pantalla podemos trasladar el objeto con respecto a nuestro marcador.

Una vez explicadas las funciones vamos a activarlas en nuestro Script con la función *Activa_Desactiva_Touch()*. Sin entrar en demasiados detalles (En el contenido adicional comentaremos el código del sistema *multiTouch* en más detalle) esta función hace lo siguiente:

Si el sistema *multiTouch* está inactivo (Está así por defecto en el arranque) hace lo siguiente:

- Activa el sistema *Touch*.
- Desactiva la apertura de una URL.
- Desactiva el giro.

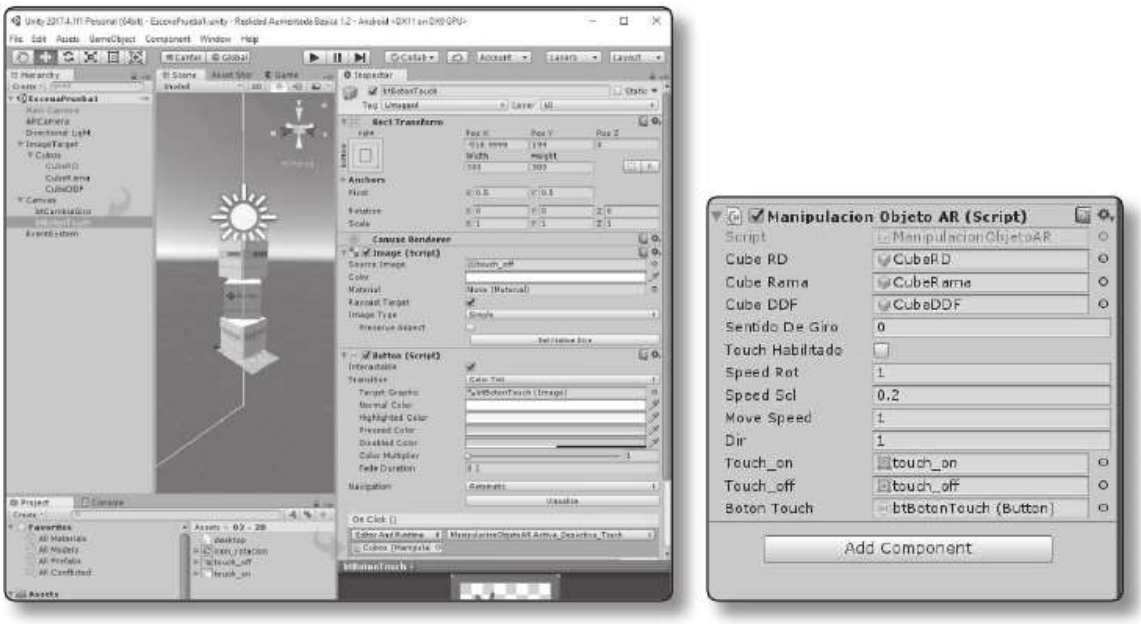
Si el sistema está activo

- Lo desactiva.
- Activa la apertura de la URL por parte de los cubos.
- Activa el giro.

Para poder llamar a la función crearemos un **UI.Button**. En nuestra escena se llamará *btBotonTouch*.

Para que todo funcione correctamente debemos de crear el nuevo botón en el *Canvas* del proyecto y asignarle la función *Activa_Desactiva_Touch()*.

Hemos preparado unas variables en el script **ManipulacionObjetoAR** para que además cambie el *Sprite* (Imagen) asociada dependiendo de si tenemos o no activado el *Touch*. En *Boton_Touch* debemos de arrastrar el **UI.Button** y en *Touch_on* y *Touch_off* los *sprites* correspondientes que encontraremos en la carpeta **Assets\02–2D**.



Existen otras variables que podemos dejar con el valor por defecto o cambiarlas al gusto. Estás son:

- ⇨ *SpeedRot*: Velocidad de rotación.
- ⇨ *SpeedScl*: Velocidad de escalado.
- ⇨ *MoveSpeed*: Velocidad de traslación.

Finalmente, para ver nuestro proyecto funcionando en el Smartphone debemos de compilar el proyecto.

Si está todo configurado correctamente bastaría con irnos a **File→Build Settings** y pulsar *Build*. De esta manera generamos un ".apk" instalable en cualquier dispositivo Android.

3.3 APLICACIÓN DE REALIDAD VIRTUAL BÁSICA

Para crear una aplicación lo más sencilla posible de realidad virtual usaremos el SDK de *Google Card Board* para Unity 3d.

Hemos elegido este Kit por varias razones:

- ⇨ Sencillez de uso y configuración.
- ⇨ Está integrado con el motor Unity 3D a partir de las versiones de 2017.
- ⇨ Y la principal razón es que está desvinculado de un hardware específico.

La adquisición de un casco de realidad virtual puede ser un hándicap difícil de superar para algunos lectores, pero un smartphone es algo que todos tenemos de una manera u otra a mano.

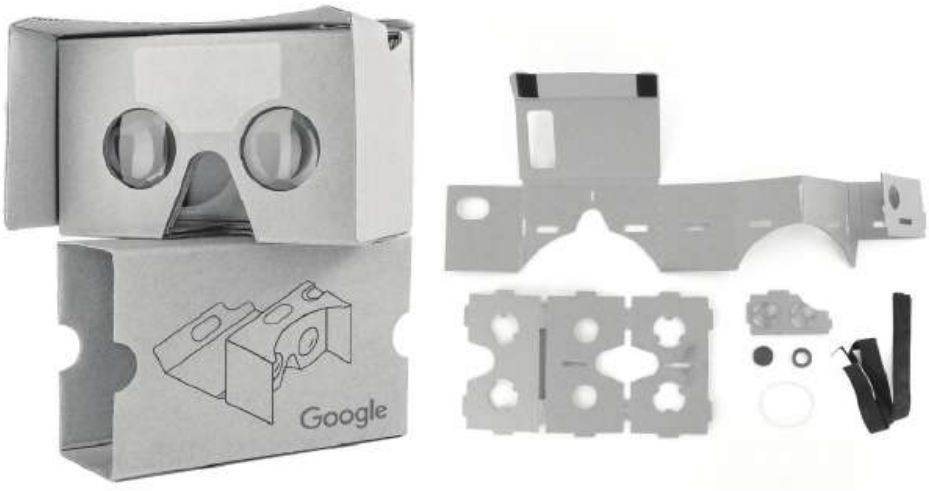


Figura 3.1. Imagen de una versión de Google Card Board

Google Card Board fue desarrollada en 2014 por Google y nació como una plataforma VR (*Virtual Reality*) que permite usar un soporte de cartón plegable y un smartphone para construir un casco de realidad virtual.

La premisa de la idea original era que con un smartphone y algo menos de 5$ ya podías disfrutar de la Realidad Virtual.

Básicamente las *Google Card Board* están compuestas por varias piezas de cartón autoensamblable, un par de lentes, una cinta para sujetar en la cabeza y un imán que hace de botón. Google ha creado varias versiones de su Kit. Actualmente existen empresas especializadas en crear chasis de cartón personalizados, usados principalmente como vehículos publicitarios y de promoción por la facilidad de personalización del material.

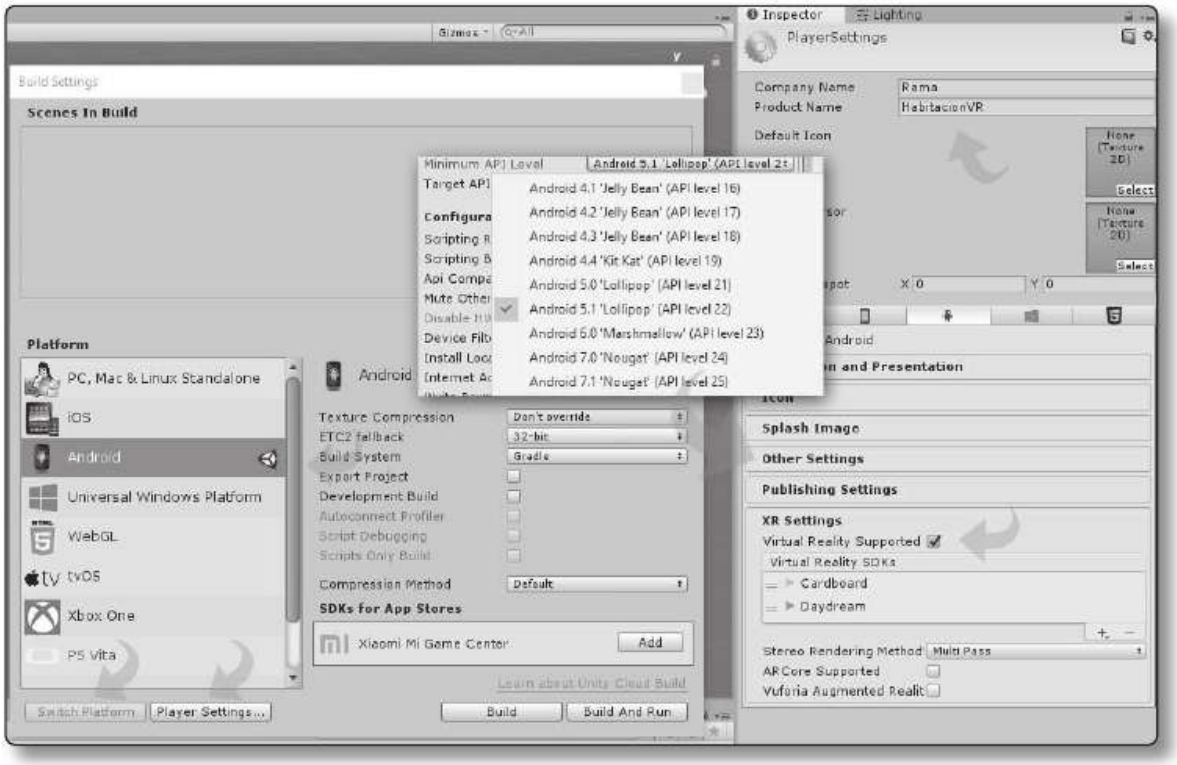
Gracias a este desarrollo surgen multitud de chasis de plástico para Smartphones más robustos, que, a diferentes precios, mejoran la experiencia de las *Card Board* originales.

Usaremos Android como plataforma destino de nuestro ejemplo, así que lo primero que haremos será crear un nuevo proyecto en Unity 3D y guardar la escena actual (Aunque esté vacía). Esto lo haremos en **File→Save Scenes**.

Después de esto vamos a definir Android como plataforma de desarrollo y a configurar el proyecto.

Para esto nos vamos a **File→Build Settings**, pulsamos en `Android` y realizamos las siguientes acciones:

1. Elegimos `Android` en *Platform y* pulsamos el botón *Switch Platform*.
2. Pasamos a configurar el proyecto pulsando en *Player Settings*.
3. Introducimos el nombre de la compañía y el nombre de la APP, en la cabecera.
4. En la pestaña *Other Settings* debemos marcar la API mínima para nuestro proyecto (*Minimum API Level*) que este caso es *Android 4.4 Kit Kat (Api level 19)*. *CardBoard* no funciona con APIs inferiores a la 19.
5. Finalmente debemos de irnos a la pestaña *XR Settings* donde marcaremos la casilla *Virtual Reality Supported*. Pulsamos el "+" y añadimos *Cardboard* a la lista.
6. No debemos de olvidar añadir la escena a *Scenes to Build* pulsando el botón *Add Open Scenes*.



Para nuestro ejemplo práctico vamos a crear una habitación virtual con mobiliario. Aunque se va a proporcionar el proyecto completo montado y funcionando en material adicional, hemos creado diferentes *assets* en paquetes "unity3d" con el objeto de que el lector pueda seguir fácilmente el proceso de creación de escena centrándonos en lo verdaderamente importante en este capítulo: "Crear una APP de Realidad virtual". El primero que vamos a usar es un paquete con todos los *Assets 3D* de la habitación a crear.

En la sección de este capítulo del material adicional existe un paquete llamado **HabitacionVR3D.unitypackage** donde están todos los elementos 3D que conforman la habitación, los elementos de iluminación necesarios y los *scripts* que se van a usar en la escena. Para integrarlo en nuestra escena primero debemos irnos a **Assets→Import package→Custom package** y lo elegimos para importar.

Dentro de la carpeta **Assets\Assets3D** disponemos de un *Prefab* llamado **HabitaciónVR3D** que podemos arrastrar a la escena directamente y ya tendríamos todos los objetos y una iluminación básica. Para no distorsionar con sombras extras conviene eliminar cualquier luz en la escena antes de integrarlo.

El paso siguiente es descargar el SDK de *CardBoard* e integrarlo.

Para la descarga debemos de irnos a la siguiente URL: *https://github.com/googlevr/gvr-unity-sdk/releases*

Elegimos el fichero acabado en "*.unitypackage*" y lo descargamos.

Al igual que con el paquete de *Assets* lo importamos en **Assets→Import package→Custom package** creando una carpeta en el proyecto llamada **Assets\GoogleVR**.

Ahora vamos a incorporar a nuestra escena varios *Prefabs* y realizar diferentes modificaciones que incorporarán diferentes funcionalidades a nuestro proyecto. Vamos definiendo y comentando cada paso:

En **Assets\GoogleVR\Prefabs** tenemos **GvrEditorEmulator**. Este *prefab* se encarga de emular los movimientos de la cabeza de una persona en el editor y nos va a permitir realizar pruebas previas a la compilación.

1. Debemos arrastrarlo a la escena. Ahora, si ejecutamos el proyecto, podemos mover la cámara de dos maneras.
   - Si mantenemos pulsado *ALT* + movimiento del ratón la cámara gira.
   - Si pulsamos *CTRL* + movimiento del ratón la cámara cabecea lateralmente.

   Veremos que si intentamos ejecutar el proyecto la cámara aparece a "ras" de suelo. Si intentamos modificar los parámetros de posición de esta al ejecutar de nuevo vuelve a hacer lo mismo. La solución a este problema la damos en el siguiente paso.

2. Creamos un *GameObject Empty* en la escena. Lo renombramos a "Persona" y cambiamos el *transform.position* en la pestaña Inspector a (x=0, y=1.7, z=0).

3. Una vez hecho esto hacemos que **MainCamera** cuelgue de él.

4. De esta manera incorporamos una altura de 1.7 con respecto al suelo y solucionamos el problema de la cámara a "ras" que comentábamos en el punto anterior. En este objeto "Persona" es donde marcamos la posición inicial que debe tener la cámara en el arranque.

5. Para gestionar los *inputs* de los controladores VR vamos a introducir el *prefab* **GvrControllerMain**. Lo encontraremos en **Assets\GoogleVR\Prefabs\Controller**.

6. El siguiente *Prefab* a incorporar a escena es **GvrEventSystem**. Es el encargado de gestionar los eventos. Está localizado en **Assets\ GoogleVR\ Prefabs\EventSystem**.

7. El último *prefab* a agregar es **GvrReticlePointer**, pero no lo haremos en raíz de la escena, sino que lo haremos hijo de **Main Camera**. La función de este objeto es mostrar el puntero en pantalla y lo podemos encontrar en **Assets\GoogleVR\Prefabs\Cardboard**.

Con esto ya tenemos una demo funcional tanto en editor como si decidimos compilar. Así debería quedar la pestaña *Hierarchy* con los *prefabs* y objetos que hemos colocado.

¡¡¡Ahora sería el momento de generar nuestra APK y probar nuestra *CardBoard*!!!

Para los neófitos bastaría con irnos a **File→Build Settings** y pulsar a *Build* para generar la APK si tenemos configurados todos los SDKs necesarios (*Java JDK, Android SDK* y *Android NDK*) al igual que hemos hecho con el ejemplo de Realidad Aumentada.

Con objeto de incorporar algo de interacción en nuestra escena vamos a usar una técnica llamada *Ray casting*. El *Ray Casting* consiste en "lanzar un rayo" (De ahí lo de Ray) que nos permite interaccionar con objetos cuando este colisiona con ellos. El símil más parecido es el del puntero laser. Cuando el rayo (*ray*) alcanza a los objetos podemos programar diferentes comportamientos.

Vamos a usar nuestro puntero asociado a **Main Camera** para señalar objetos y programar comportamientos.

Para esto necesitamos añadir un script llamado **GvrPointerPhysicsRaycaster** a nuestra **Main Camera**. Este script está ubicado en **Assets\GoogleVR\Scripts\ EventSystem**.

Al añadir este *script* conseguimos que **Main Camera** "emita" el rayo, pero ahora faltaría programar los comportamientos en cada objeto físico.

Para este ejemplo concreto mostraremos unos títulos en 3D dependiendo de a que mueble apuntemos con nuestro puntero. Lo primero que haremos será crear los *gameobjects* necesarios en escena con los distintos carteles 3D.

Creamos en nuestra escena un nuevo *gameobject empty* llamado "Textos3D" que agrupará a todos los textos de los diferentes muebles.

Empezamos con el texto de la estantería y el resto de los muebles se harían igual. Creamos un objeto **3d Text** que cuelgue de "Textos 3D" y lo posicionamos encima de la estantería en la escena.

Ahora incorporaremos a este texto 3D un script llamado **textoEscala** y que lo tenemos disponible en **Assets\Scripts**.

En este script tenemos programados en diferentes funciones los comportamientos del texto.

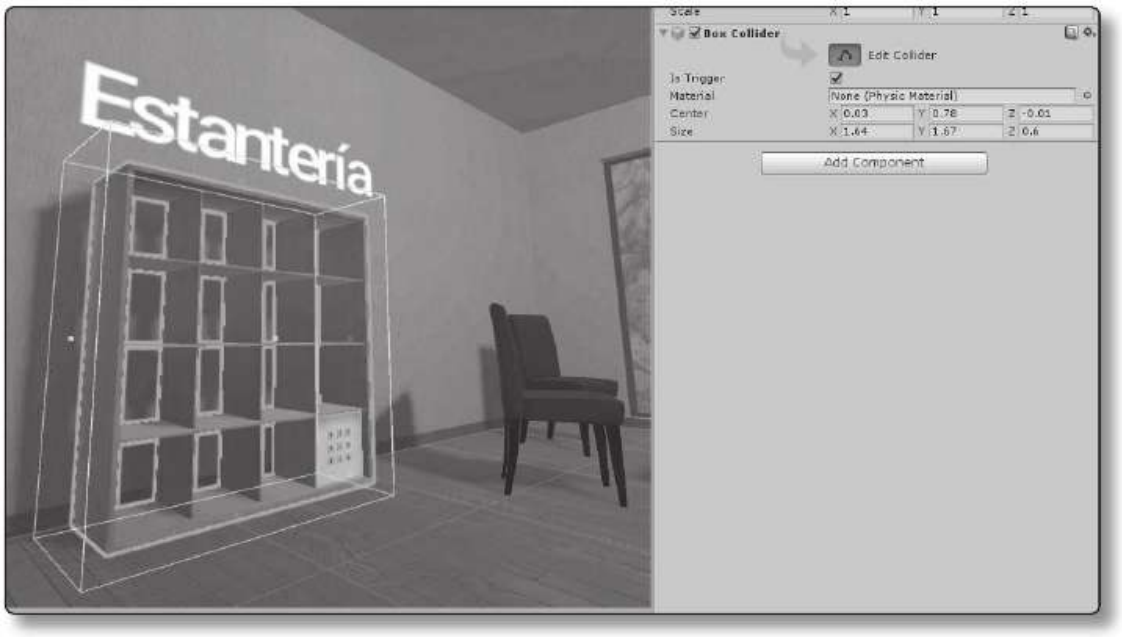
En la función *Start()* escalamos el texto a 0 para que desaparezca de la escena en el comienzo. Después disponemos de *TextoEscalaIn()* que escala el texto hasta mostrarlo a la escala original y *TextoEscalaOut()* que lo vuelve a escalar hasta ocultarlo.

Una vez definidas las funciones y asignadas al texto debemos de irnos al objeto 3D a definir un par de componentes necesarios para gestionar el comportamiento ante el *Ray Casting*.

El primer componente que debemos asignar al mueble, en este caso la estantería, es un *Box Collider*. Para esto desplegamos el objeto **HabitaciónVR3D** y localizamos el objeto *estanteriaCajon*.

En la pestaña inspector le asociamos el nuevo componente pulsando en el botón *Add Component*. Elegimos *Physics* y pulsamos en *Box Collider*. El *Box Collider* delimita el espacio en el que somos capaces de detectar colisiones con otros objetos, en nuestro caso concreto no es un objeto el que impacta sino el rayo emitido por nuestra **Main Camera**.

Una vez definido el componente debemos de ajustarlo al mueble editando los límites. Para esto pulsamos en el botón *Edit Collider* y ajustamos los límites que vienen definidos por una línea verde. También debemos marcar *Is Trigger* en el componente.



El segundo componente que debemos de añadir se encargará de llamar a las distintas funciones del *Script* del texto dependiendo de los eventos que sucedan.

En este caso usaremos un **Event Trigger** y para esto volvemos a pulsar en **Add Component→Event→Event Trigger**.

Este componente nos permite definir comportamientos cuando algo interacciona con nuestro objeto (O con el *Box Collider* que lleva incorporado más exactamente).

En nuestro caso queremos que cuando el puntero lo señale muestre el texto y cuando lo deje de señalar lo oculte. Para esto usaremos los eventos preconfigurados *PointerEnter* y *PointerExit* del componente.

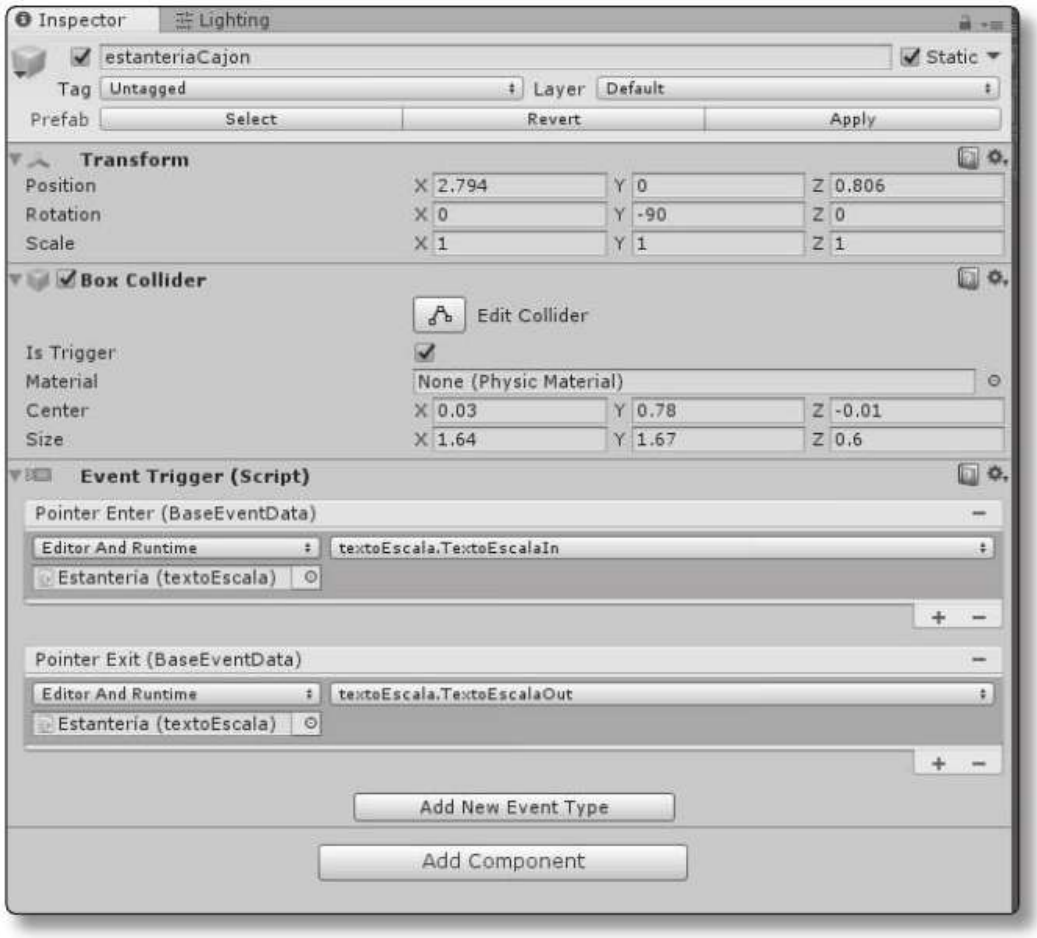
Pulsamos en *Add New Event Type*, en *Pointer Enter* y nos aparece una lista de acciones que podemos lanzar con este evento. En nuestro caso solamente es una, pero podrían ser varias.

Para añadir nuestra acción pulsamos en el "+" y arrastramos el objeto "Estantería" que contiene el texto 3D y que tiene el script **textoEscala** al casillero de *Gameobject*.

Después en el desplegable de la derecha definimos que función usar de los componentes asociados a este *gameobject*. En nuestro caso debemos elegir el componente **textoEscala** y la función *TextoEscalaIn()*.

Ya tenemos definido el evento de entrada y nos faltaría el de salida. Para esto volvemos a pulsar en *Add New Event Type*, elegimos *PointerExit* y en este caso usamos la función *TextoEscalaOut()* de **textoEscala**.

La configuración del *Inspector* debe quedar así.



Si probamos nuestro proyecto en el editor (Recordamos que manteniendo pulsado ALT y con el ratón podemos mover la cámara) al apuntar a la estantería se muestra el texto y al dejar de apuntar se oculta.

Invitamos al lector a seguir definiendo textos para el resto de muebles.

Al igual que el ejemplo de Realidad Aumentada para compilar y, si está todo configurado correctamente, bastaría con irnos a **File→Build Settings** y pulsar *Build* para generar él fichero ".apk".

4

VUFORIA

Vuforia es un producto de Qualcomm Technologies, empresa fundada en 1985 por el Dr. Irwin M. Jacobs, Dr. Andrew Viterbi, Harvey White, Franklin Antonio, Andrew Cohen, Klein Gilhousen y Adelia Coffman. Qualcomm empezó trabajando sobre una tecnología inalámbrica, única para la época, que fue usada por parte del ejército norte americano. También fueron pioneros en sistemas de comunicación, conectividad y en internet móvil. Llegaron a comercializar un terminal móvil propio, crearon el primer chipset que integraba GPS, procesadores para móviles, etc.

En la actualidad, Qualcomm desarrolla todo tipo de chipsets para dispositivos móviles, además de tener uno de los procesadores más usados en terminales móviles, la familia Snapdragon. En 2016 los propietarios de Vuforia vendieron la división a PTC, la cual, sigue desarrollando esta tecnología con la misma filosofía que Qualcomm.

Pero ¿Qué es Vuforia?. Básicamente, Vuforia es un SDK (Kit de desarrollo) para dispositivos móviles creado por Qualcomm en 2013, que permite la creación de aplicaciones de realidad aumentada. Utiliza tecnología de visión artificial para rastrear imágenes planas **(Image Targets)** y en sus últimas versiones, también reconoce Objetos 3D simples y superficies.

Hay que decir que esta tecnología no fue pionera en Realidad aumentada (Ya existían otros SDKs anteriores, como *ARKit* o *Wikitude*), pero sí que lo fue en el uso de marcadores. Hasta la aparición de Vuforia los marcadores de realidad aumentada eran imágenes con composiciones de formas negras sobre blanco (Similares a los códigos QR) para facilitar la tarea a los algoritmos de reconocimiento de la imagen existentes. Vuforia cambio todo esto e introdujo la posibilidad de usar una imagen cualquiera como marcador.

Es necesario tener en cuenta que Vuforia no tiene en cuenta los colores por lo que no es posible utilizar dos marcadores iguales, pero de distinto color, ya que para Vuforia serán considerados como un único marcador.

Marcador usando patrón bicolor

Marcador de Vuforia usando imágenes reales (Ejemplo de APP La Alhambra, El Castillo Rojo)

Marcador usando patrón bicolor Marcador de Vuforia usando imágenes reales (Ejemplo de APP La Alhambra, El Castillo Rojo)

El poder usar cualquier imagen como marcador permite que una aplicación funcione por si misma sin marcadores específicos que deban ser pegados o instalados en los elementos que van a ser objeto de aplicación de la realidad aumentada, ya que se pueden usar objetos, partes distintivas, imágenes o lugares del mundo real a modo de marcador.

ⓘ NOTAS

En el capítulo 2 de este libro hablamos de “La Alhambra, el castillo Rojo”, la primera guía de realidad aumentada para niños del mundo, que, usando esta tecnología, propone pequeños retos a los más pequeños en un recorrido por el monumento de La Alhambra. Incluye varios minijuegos de Realidad Aumentada donde los pequeños deben de buscar llaves usando lugares concretos del monumento.

Esta característica de marcadores “del mundo real”, junto a la posibilidad de integrar esta tecnología sobre smartphones o tablets (Android, iOS, Windows-UWP), fue toda una revolución en la creación de aplicaciones de Realidad Aumentada. Todo esto marcó un antes y un después en el uso de esta tecnología y hace que Vuforia sea el SDK más usado para la creación de aplicaciones de Realidad Aumentada en la actualidad.

El SDK de Vuforia es compatible con una gran variedad de tipos de marcadores, además de los *Image Target* o marcadores simples. Permite el reconocimiento de marcadores múltiples, objetos 3D simples, superficies planas (*Ground Planes*) y un marcador con metadatos conocido como *VuMark*.

Entre las características adicionales del SDK se incluye la detección de oclusión localizada en imágenes, o lo que es lo mismo, la capacidad de detectar cuando tapamos partes del marcador a la cámara (bien con un dedo o con otro objeto). Esto nos permite generar eventos cuando tapamos partes concretas del marcador y nos permite crear 'Botones virtuales' que se pueden programar con cualquier interacción que necesitemos.

Tenemos disponibles diferentes SDK para construir aplicaciones para Android (*Android Studio*), iOS (*XCode*), Windows (UWP) (*Microsoft visual Studio*) y una extensión para el motor Unity 3D, que es la que usaremos en los ejemplos de este libro.

Podemos usar diferentes lenguajes de programación siempre que sean utilizables con estas plataformas para acceder a su API como son: *C++*, *Java*, *Objetive-C++* (un lenguaje que utiliza una combinación de *C++* y sintaxis *Objective-C*) y *C#* que es lo que nosotros utilizamos en este libro conjuntamente con el motor Unity 3D.

4.1 CONFIGURACIÓN DE VUFORIA

Para abordar la configuración de Vuforia partimos de la base de la aplicación de Realidad Aumentada Básica explicada en el capítulo 3.

En este capítulo iniciamos una App usando el SDK de Vuforia con un marcador de la base de datos de ejemplo que trae el propio Kit. Si ha llegado hasta aquí sin leer esa parte conviene que la repase antes de seguir. También conviene que repase la parte de configuración del sistema de desarrollo para disponer de toda la configuración previa antes de abordar los siguientes ejemplos

Hasta ahora hemos usado la base de datos de marcadores por defecto que trae el propio SDK. Para los siguientes ejemplos vamos a personalizar nuestros marcadores y crear nuestra propia base de datos de marcadores para personalizar la experiencia. Para esto debemos de estar registrados en el portal de desarrolladores de Vuforia. Esto es https://developer.vuforia.com/.

Como indicamos en el capítulo 3 debemos de registrarnos, logarnos y pulsando en *Get Development Key* obtenemos una licencia gratuita (Ya veremos más delante las limitaciones que tiene).

Además de la licencia gratuita podemos comprar otro tipo de licencias con diferentes alcances dependiendo de las necesidades de nuestro proyecto. Si pulsamos en *Buy Deployment Key* nos aparecen las diferencias entre ellas y los precios. Vamos a comentar las diferencias principales entre ellas:

⇨ **Free**

Es totalmente gratuita como hemos dicho. Nos permite hasta 100 marcadores fijos predefinidos (Imágenes, Cubos, Cilindros o 3D). Permite también un máximo de 100 *VuMark*. Vuforia la ha pensado para los desarrolladores y superpone una marca de agua en la cámara por lo que solamente sirve para demos técnicas y aprender (Según las condiciones de uso de Vuforia no está permitido el uso de este tipo de licencia para Aplicaciones profesionales). Es la que vamos a usar para los ejemplos de este libro. En las últimas modificaciones la han restringido al uso exclusivo con Unity 3D no permitiendo desarrollos nativos en iOS / Android o UWP.

⇨ **Classic**

Esta licencia mantiene las mismas restricciones que la anterior en el número de marcadores y tipos (Son todos estáticos), pero elimina la marca de agua y ya nos permite generar aplicaciones profesionales sin problema. Tiene un precio único que se paga una vez y dispones de actualizaciones hasta la versión "mayor" del SDK (Esto es, si compras la licencia con un SDK de Vuforia 7.2 por ejemplo, dispones de actualizaciones hasta que el SDK cambie a la versión 8). Es la más usada para pequeños proyectos profesionales.

⇨ **Cloud**

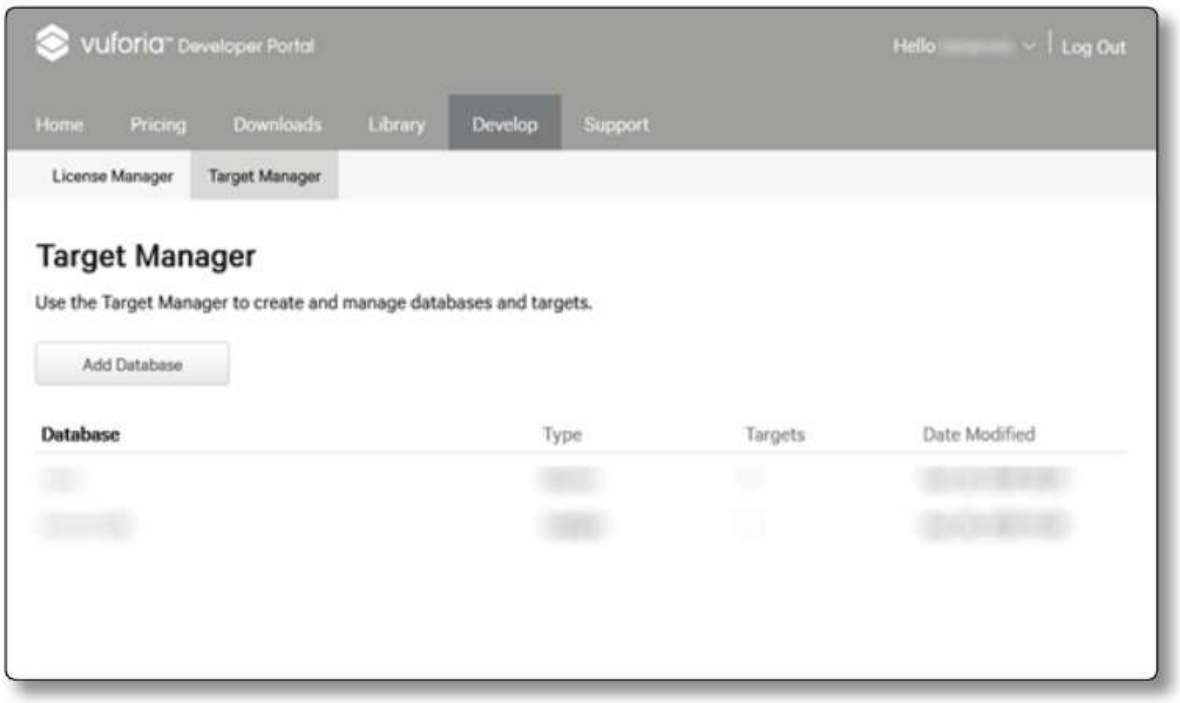
Si necesitamos más de 100 marcadores o queremos que estos se comporten de manera dinámica esta es la licencia que debemos de usar. Nos permite marcadores dinámicos residentes en la nube y un número de reconocimientos mensuales de 10.000. Si sobrepasamos este número de reconocimientos cobran una pequeña cantidad extra por cada uno. Esta licencia se paga mensualmente.

➪ **Pro**

Es una licencia donde las restricciones las acuerdas directamente con el departamento comercial de Vuforia y donde pagas un precio dependiendo del alcance personalizado que acuerdes con ellos. Permite algunos extras como la personalización vía API de nuestro servicio, así como el acceso a las características avanzadas de cámara. Es una licencia pensada para proyectos industriales grandes y con necesidades muy concretas.

A continuación, vamos a proceder a crear una base de datos personalizada para construir los distintos ejemplos que utilizaremos en este capítulo.

Dentro del portal de Vuforia, y una vez logado accedemos a la opción *Develop* en el menú principal y después a *Target Manager*. Para añadir nuestra propia base de datos debemos de pulsar en *Add Database*.



Ahora nos pide el nombre que debemos de poner y el tipo. Existen 3 tipos. Nosotros elegiremos el tipo general o *Device*.

Antes de crear marcadores vamos a hacer un pequeño inciso y comentar algo más sobre ellos.

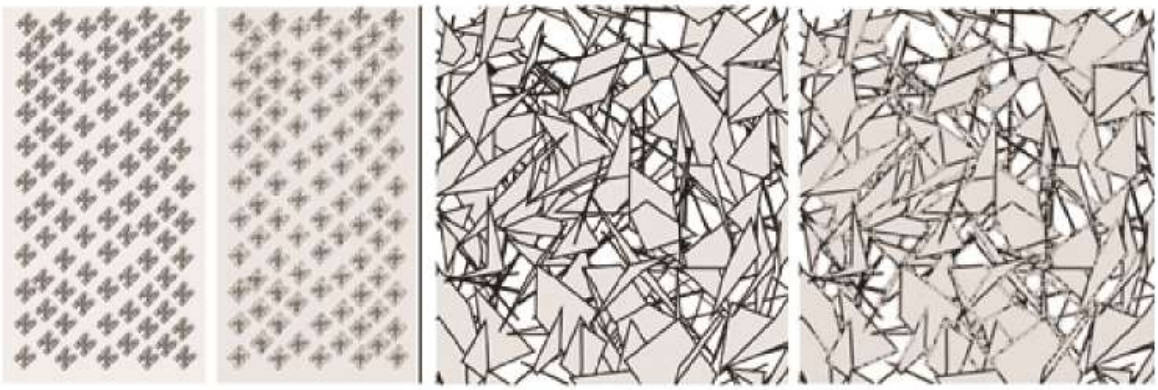
Nuestra base de datos nos permite distintos tipos de marcadores que iremos viendo conforme avancemos en el capítulo. Para empezar, hablaremos de

los marcadores *Simple Image* o **ImageTarget**, que son una imagen que sirve como marcador. Esta imagen, y esta es la característica especial de Vuforia, puede ser cualquier imagen del mundo real.

Dependiendo de los atributos de la imagen, Vuforia nos dará un *rating* con hasta 5 estrellas. Este *rating* nos indica lo "buena" que es nuestra imagen a la hora de detectarla con su algoritmo de visión artificial. En la práctica vemos que obtener estas 5 estrellas no es tan sencillo. Un **ImageTarget** debe de cumplir una serie de reglas para ser un "buen marcador":

- ⇨ *Rico en detalles*: escenas de calle, grupos de gente, collages, mezclas de objetos o escenas deportivas pueden dar estos detalles fácilmente.
- ⇨ *Buen contraste*: Que disponga de regiones brillantes o iluminadas frente a otras más oscuras.
- ⇨ *Sin patrones repetitivos*: Debemos de huir de los patrones repetitivos, ya que el algoritmo necesita un cierto grado de aleatoriedad para funcionar. De nada sirve una imagen con muchos detalles y alto contraste si sigue un patrón definido.

En la siguiente imagen ponemos un ejemplo de marcador con una gran cantidad de detalles reconocibles y un buen contraste pero que siguen un patrón claro (A) y otro que además suma un alto grado de aleatoriedad (Lo ideal para nosotros) que lo hacen el marcador ideal (B).



Dependiendo del tipo de aplicación que necesitemos hacer podremos usar imágenes reales o una serie de marcadores construidos exprofeso.

Un ejemplo de aplicación con marcadores basados en imágenes reales es **La Carolina Virtual** (Ver más información en el capítulo 2).

En esta aplicación se usan baldosas de granito reales instaladas en el acerado que sirven como marcadores reconocibles en la APP.

La aplicación reconoce el marcador (o Baldosa), identifica el lugar donde está el usuario y muestra una reconstrucción virtual del mismo espacio de la ciudad en una época pasada. Permite guardar la misma perspectiva que tiene el usuario desde la baldosa usando giroscopio y acelerómetro para recrear el movimiento en la escena.

En este caso concreto el contraste y la aleatoriedad del granito ya proporciona un marcador de 5 estrellas por sí solo.

Si el marcador lo vamos a generar nosotros, como en nuestros ejemplos, existen diferentes webs que nos ayudarán a generar patrones aleatorios que servirán como base para nuestros marcadores y que nos darán un excelente *rating*. Aquí dejamos un par de ejemplos:

https://shawnlehner.github.io/ARMaker/
http://www.brosvision.com/ar-marker-generator/

Para la serie de ejemplos que vamos a desarrollar en el libro hemos dejado un PDF con los marcadores necesarios listos para imprimir con las proporciones exactas en los contenidos descargables del capítulo, los encontrarás en la carpeta **Capitulo 4 → Marcadores Cámara** , así como las distintas imágenes en ".jpg" para subirlas a la base de datos de Vuforia, estas podrás encontrarlas en la carpeta **Capítulo 4 → Marcadores BD**. El lector puede intentar crear sus propios marcadores personalizados si así lo desea, bien mediante una herramienta de diseño gráfico o mediante una de las aplicaciones web que hemos indicado anteriormente.

Para seguir los dos primeros ejemplos necesitaremos de 5 marcadores diferentes. En el PDF que se encuentra en los contenidos descargables, están marcados como **ImageTarget**. Como se pueden observar en los proporcionados en el PDF tenemos un marcador un poco más grande (de 10 x 10cm) y 4 extras más pequeños (7x7 cm). El más grande corresponde a una habitación y los más pequeños serán para diferentes muebles que vamos a colocar en ella en el ejemplo de *multitarget*.

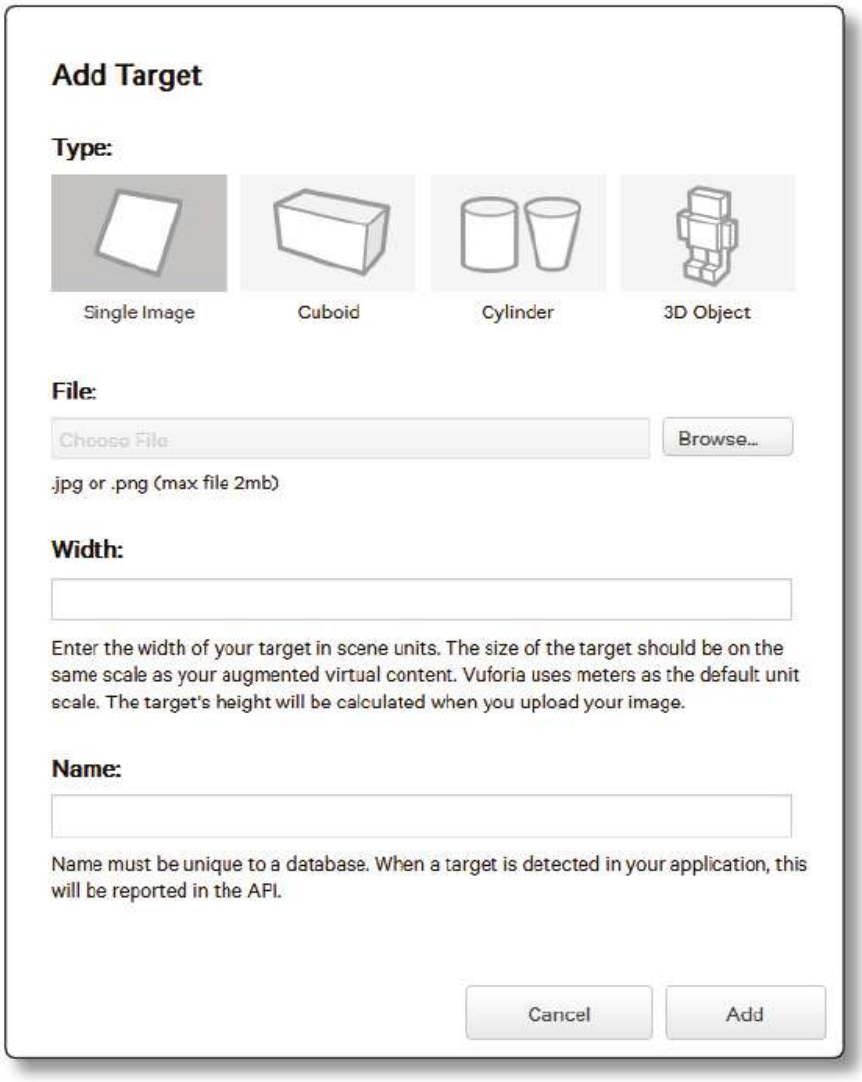
Cuando tengamos nuestra base de datos creada, añadiremos el primer marcador. Para esto pulsamos en *Add Target* y debemos de rellenar los siguientes campos:

- ⇨ *Type*: En este primer caso vamos a incorporar **ImageTarget**, así que elegimos *Simple image*.
- ⇨ *File*: Aquí subimos el fichero con la imagen del marcador. Vuforia tiene una limitación de 2mb por imagen y solo admite *.jpg* y *.png* en escala de grises o color RGB.
- ⇨ *Width*: Aquí marcamos la anchura que tendrá el marcador en nuestra escena virtual. Para el caso de "Habitación" elegiremos 10, ya que es de 10 cm y para el resto 7, ya que son de 7 cm. Esta dimensión es importante si los marcadores van a estar en una escena juntos (*MultiTargets*). En el caso de Unity al incorporarlos se escalarán a unidades Unity dependiendo del valor que marquemos aquí.
- ⇨ *Name*: Será el nombre por el que se les reconocerá en la base de datos.

ⓘ NOTAS

Es importante que el tamaño o escalas relativas entre los marcadores estén bien definidas. Por ejemplo, si usamos un marcador con un ancho de 10 y un diseño de 10cmx10cm, y el objeto virtual relativo asociado está correctamente dimensionado, debemos de imprimirlo en 10 cm x 10 cm. Si imprimiésemos el marcador en 5cm x 5cm, el objeto virtual relativo se escalará y ocupará la mitad del espacio que debería en la escena. En escenas en las que solamente usamos un marcador no es decisivo porque el acercamiento de la cámara puede compensarlo, pero si usamos más de un marcador, y no ajustamos las escalas de los objetos virtuales correctamente al tamaño su marcador físico, los objetos virtuales en la escena representada no guardarán la correspondencia lógica.

Como se puede observar en la imagen siguiente existen otros tipos diferentes, además del *Simple Image*. Estos tipos los iremos abordando más adelante.



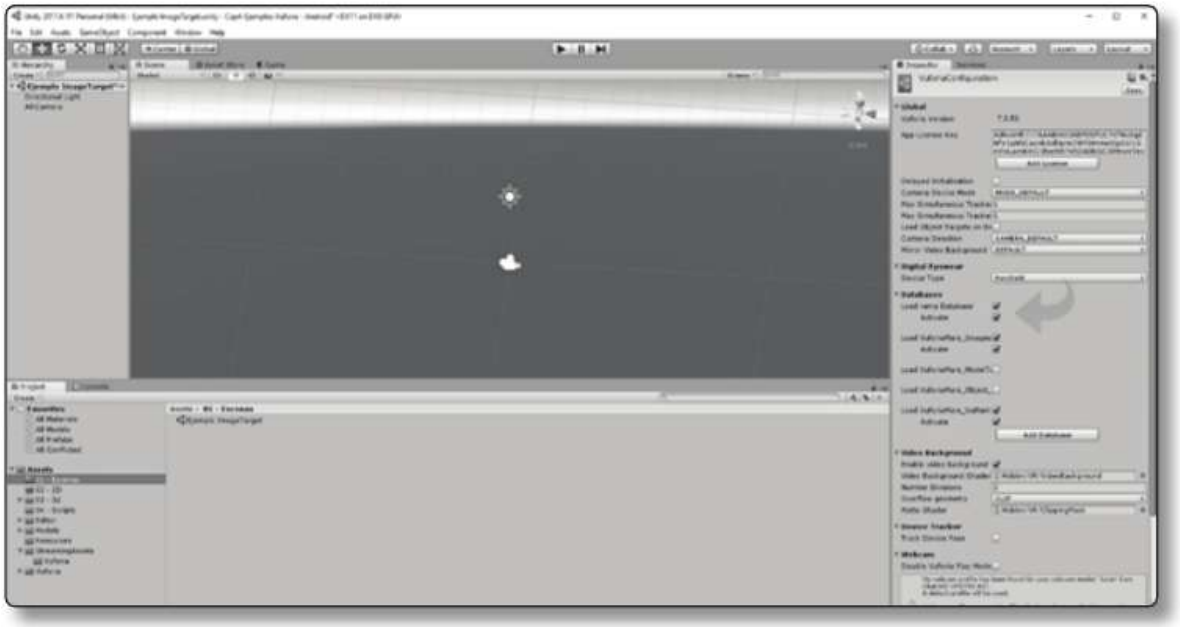
Una vez creados nuestros 5 marcadores podemos ver la valoración de Vuforia en la columna *Rating*, donde veremos el número de estrellas que nos asigna. Nuestra recomendación es que al menos debe de tener 4 estrellas porque, si no es así, el marcador será difícilmente reconocible.

En la esquina superior derecha vemos el botón *Download Database (All)* que si lo pulsamos descarga toda la base de datos en un fichero. Existen dos formatos de descarga, el usado por *Android Studio*, *Xcode* y *Visual Studio* y el que nosotros necesitamos, ósea, el usado por Unity 3D. Elegimos el segundo y tendremos un fichero "*.unitypackage*" listo para integrar en nuestro proyecto.

Para nuestro ejemplo partimos de una escena ya configurada con el objeto *AR Camera* con la licencia de Vuforia introducida y una **Directional Light** (Ver el capítulo 3 donde explicamos todos los pasos a seguir).

Después de esto importamos la base de datos que acabamos de descargar en **Assets→Import Package→Custom Package** y nos vamos al *GameObject* **AR Camera** y en *Inspector* abrimos la configuración de Vuforia *Open Vuforia Configuration*.

Aquí debe de aparecer la nueva base de datos, en nuestro caso "rama", que debemos de cargar y activar. Para esto marcamos *Load xxxx Database* y *Activate*.



Con esto ya tenemos la base de datos de marcadores creada y configurada en Unity, a partir de aquí ya podemos utilizar estos marcadores. Veamos como:

4.2 USO DE LOS MARCADORES

Para poder usar los *assets* definidos en este ejemplo debemos de cargar el paquete Unity3D llamado **muebles.unitypackage** desde contenidos adicionales. Igual que con el anterior paquete debemos de irnos a **Assets→Import Package→Custom Package** para hacerlo.

Ahora tenemos una carpeta llamada **Assets\03–3D** donde tenemos los diferentes *prefabs* que vamos a usar. También importará algunos scripts básicos que van a ser necesarios para incorporar alguna funcionalidad a los objetos.

El siguiente paso es incorporar un **ImageTarget** a la escena. Esto lo haremos pulsando en **GameObject→Vuforia→Image**. En la pestaña *Hierarchy* tendremos el nuevo objeto que ahora vamos a configurar con detalle. Lo primero que haremos es cambiar el nombre por *IT_Habitacion*.

Seguimos con el proceso de creación e incorporamos un objeto 3D que cuelgue del objeto **ImageTarget**. En este caso usaremos el *prefab* de la habitación que tenemos en **Assets\03– D\Muebles\Habitacion\prefab_habitacion**. Basta con arrastrarlo a la pestaña *hierachy*.

Y ahora pasamos a configurar el marcador propiamente dicho. Todo esto lo haremos en el *Script* **Image Target Behaviour** que tenemos en el objeto *IT_Habitación*. Vamos a explicar cada uno de los campos a rellenar:

- ⇨ *Type*: Lo dejamos en *Predefined* ya que es un marcador normal. El resto de opciones son para marcadores personalizados y marcadores *cloud*.

- ⇨ *Database*: Elegimos la base de datos que acabamos de crear. Podemos tener diferentes bases de datos en el proyecto. En nuestro caso elegimos "rama".

- ⇨ *ImageTarget*: En este desplegable aparecen los distintos marcadores que hemos incorporado a nuestra base de datos. Como es el marcador de la Habitación elegimos *habitación-marker* o como se ha nombrado en la definición de la base de datos.

Aquí tenemos la configuración básica del marcador, pero podemos personalizar el mismo con los parámetros *Advanced*.

- ⇨ *Width y Height*: Anchura del marcador. En este caso vemos que viene ya definido 10 unidades en ambos campos. Esto es así porque especificamos en la base de datos, en el campo *Width*, el valor 10 para este marcador. Al ser una imagen cuadrada la que subimos se calculó la altura al mismo valor (Dependiendo de la resolución de la imagen que carguemos calcula el *Height*). De igual manera vemos que el *Transform* del objeto la Escala también está a 10 unidades. Con un solo marcador no es tan relevante tener esto configurado correctamente, pero si usamos varios marcadores en la escena veremos que toma más importancia ya que si no se producen desajustes en las escalas. Si el marcador va a ir impreso en 10x10cm debe de estar todo acorde a estas medidas.

  Luego veremos que los marcadores de 7x7 cm van a ir con un valor 7 tanto en *Width/Height* como en los valores de escala del *Transform*.

- ⇨ *Preserve Child Size:* Si esta opción está seleccionada, los hijos de este marcador mantienen su tamaño inalterado cuando se cambia el del marcador. Si no se selecciona, los objetos hijo cambiarán de tamaño para que coincidan con la nueva escala marcada. Nosotros lo dejamos desmarcado.

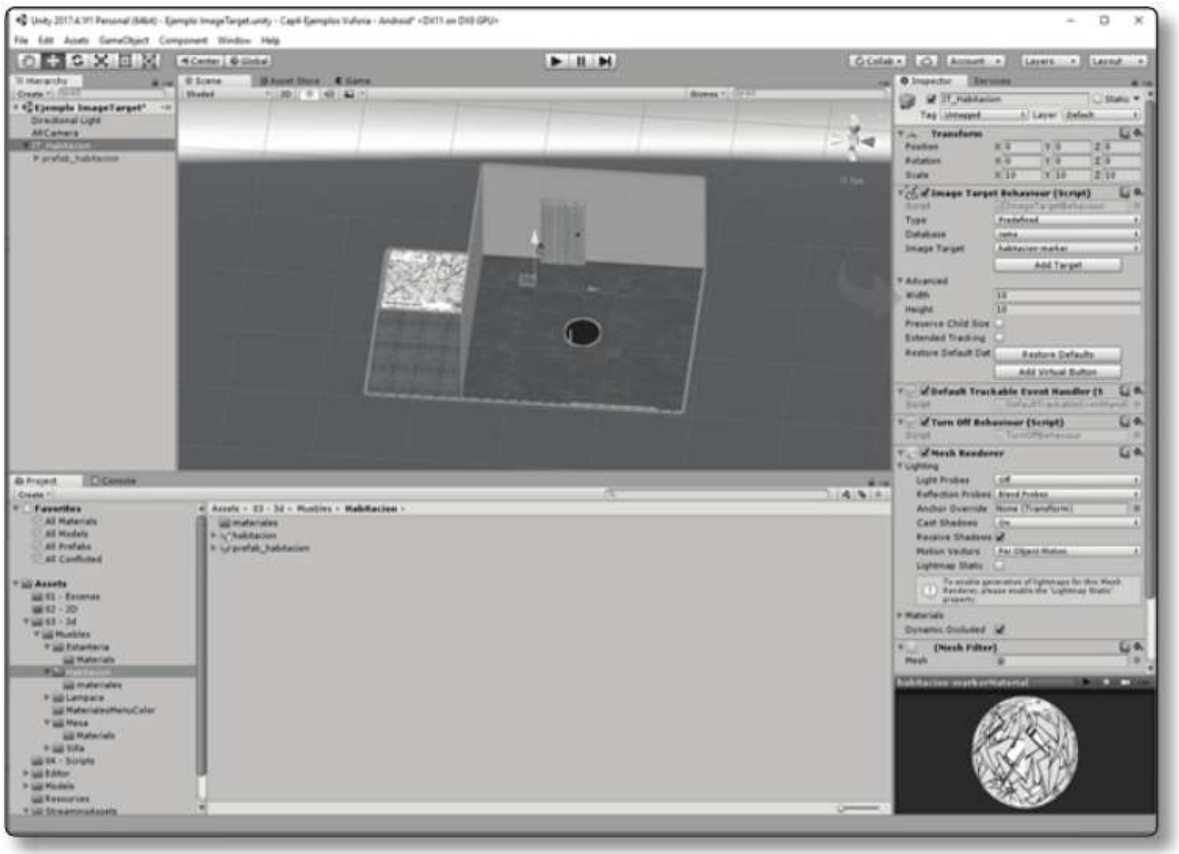
⇨ *Extended Tracking:* El rastreo extendido utiliza características del entorno para mejorar el rendimiento del rastreo y mantener el rastreo incluso cuando el marcador ya no está a la vista. Cuando el marcador se pierde de vista Vuforia usa elementos del entorno para inferir la posición del marcador. Es por esta razón por la que funciona mejor en entornos ricos en detalles.

Se recomienda marcar esta opción si el marcador es estático o se mueve muy poco y sobre todo para la visualización de elementos grandes (que cueste abarcar con el objetivo de la cámara).

En nuestro caso lo marcaremos, ya que va a ser un marcador que vamos a mover poco.

- *Restore Default*: Restaura los valores que hemos marcado en la base de datos.
- *Add Virtual Button*: Este botón lo usaremos más adelante en nuestro ejemplo.

Si todo ha sido configurado correctamente la imagen debe de quedar algo parecido a esto:

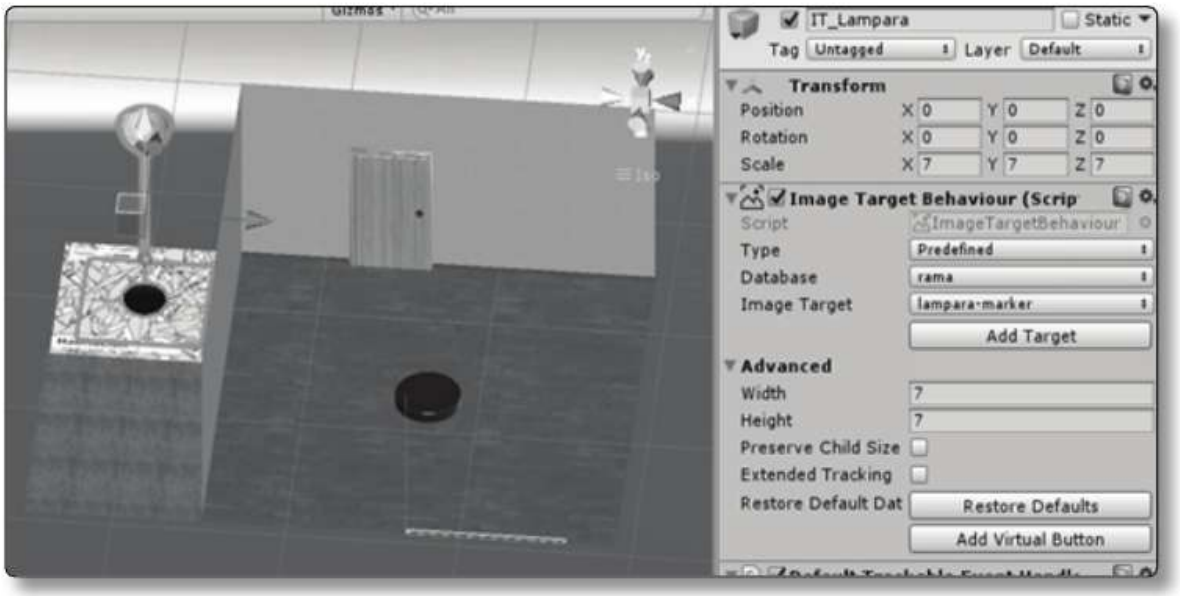


En este momento ya estamos en condiciones de comprobar si Vuforia funciona correctamente. Para esto ejecutamos la escena y enfocamos a nuestro marcador. Si todo está correcto el marcador mostrará una habitación vacía.

4.3 MULTITARGET O VARIOS MARCADORES

Continuando con el ejemplo vamos a incluir muebles vinculados a su respectivo marcador para probar como funciona Vuforia con varios marcadores en la escena. Para esto repetimos la operación con la lampara, esto es:

- ⇨ Creamos el marcador en **GameObject→Vuforia→Image**.
- ⇨ Renombramos el objeto a *IT_Lampara*.
- ⇨ Arrastramos el *prefab* de la lámpara encima del marcador. El *prefab* está situado en **Assets\03–3D\Muebles\Habitacion\prefab_habitacion**. Para que la lámpara quede un poco elevada con respecto al suelo debemos de marcar en *Inspector* la posición Y del *transform* a 0.05. Esto elevará mínimamente la lámpara y no quedará incrustada en el suelo de la habitación.
- ⇨ Configuramos el *Script* **Image Target Behabiour** con la base de datos y el marcador correcto, pero esta vez no marcamos la opción *Extended tracking* para que podamos mover la lampara libremente por la escena moviendo el marcador.



Si ahora intentamos arrancar la escena, veremos que no es capaz de detectar los dos marcadores simultáneamente. O ves la lampara o ves la habitación.

Esto sucede porque tenemos una restricción en el objeto **ARCamera** que debemos de cambiar. Para esto nos vamos al objeto y abrimos la configuración de Vuforia pulsando en *Open Vuforia configuration*. Existen dos campos que debemos de cambiar. En *Max Simultaneous Tracked Images* y *Max Simultaneous Tracked Objects* debemos de poner 5. Aquí marcamos el número de marcadores y Objetos máximos que debe de controlar la cámara. El valor por defecto era 1, y por eso no funcionaba. Ahora al probar ya podremos visualizar la lampara dentro de la habitación".

En la imagen tenemos una visualización de los objetos usando la Webcam y la posición relativa de los dos marcadores con distinto tamaño.

Vemos como moviendo el marcador de la Lampara podemos situarla en cualquier sitio de la habitación.

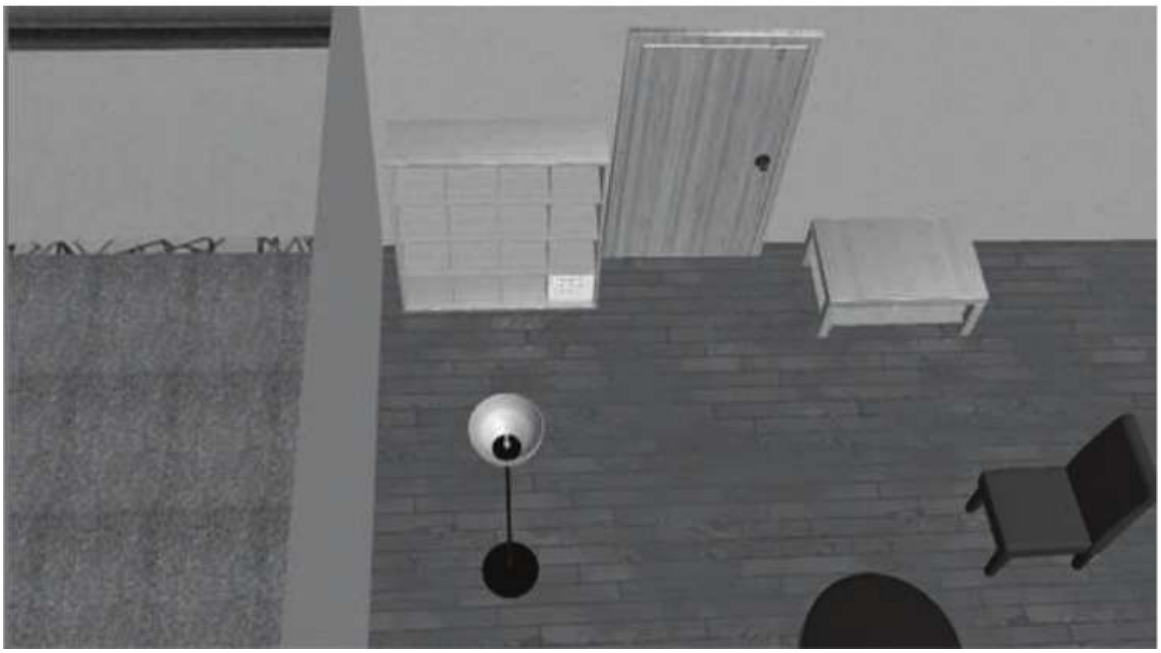
Ahora solo faltaría añadir el resto de mobiliario. Seguimos el proceso que hemos hecho con la lampara para el resto de muebles. Los *Prefabs* los tenemos en:

- ⇨ *Estantería*:
  Assets\03–3D\Muebles\Estanteria\prefab_estanteriaCajon.

- ⇨ *Mesa*:
  Assets\03–3D\Muebles\Mesa\prefab_mesa.

- ⇨ *Silla*:
  Assets\03 – 3D\Muebles\Silla\prefab_silla.

Ya podemos visualizar todos los marcadores juntos y debe de aparecer algo parecido a esto.



Después de esto vamos a añadir una nueva funcionalidad a los muebles que nos permitirá cambiar el color de estos. Para ello necesitamos varios scripts que ahora detallaremos y una estructura de menú sencilla para la selección de color.

Dentro del *prefab* de la estantería hemos incluido todo lo necesario y podemos verlo funcionando pulsando directamente en ella. Ahora vamos a integrar los mismos *scripts* y menú en la mesa para entender cómo funciona.

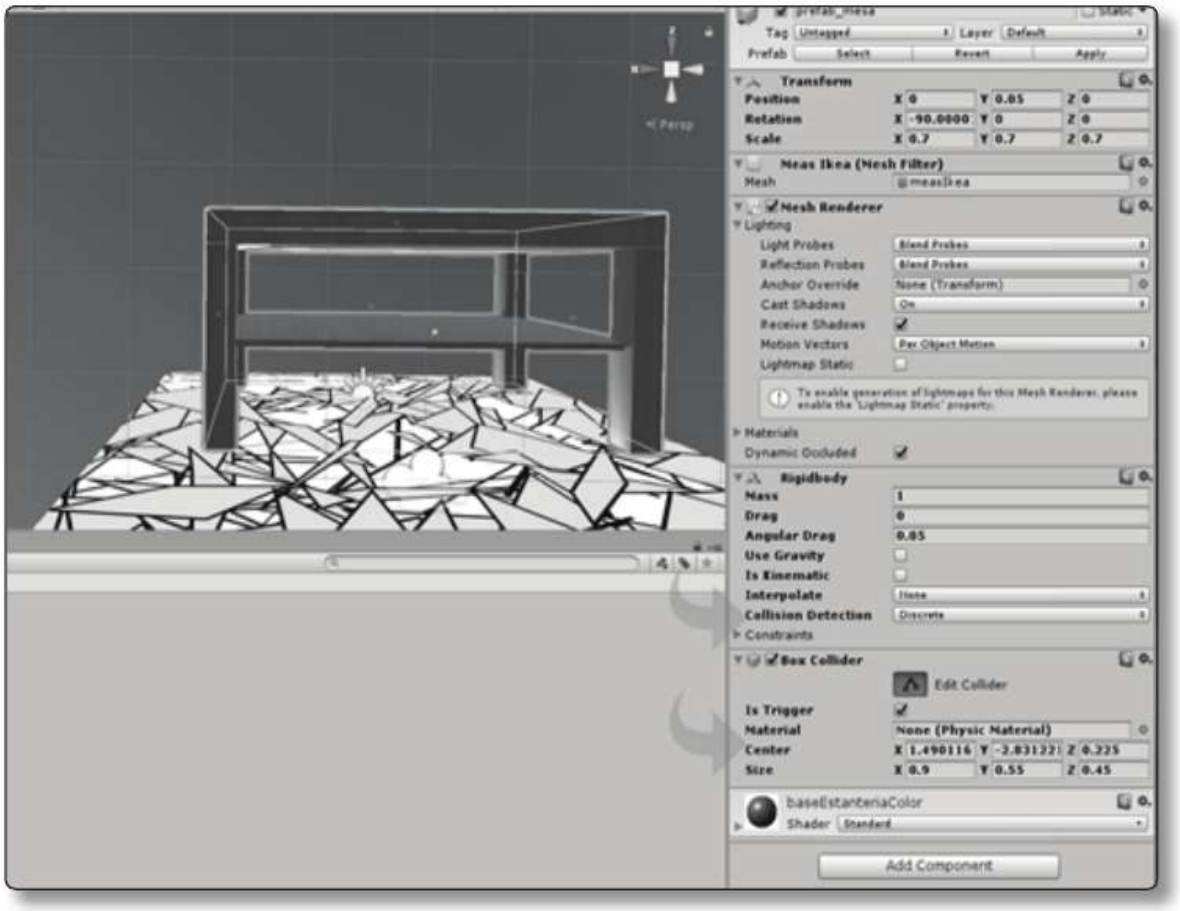
Hemos dejado toda la estructura montada en la estantería para que sirva de guía y consulta al lector en caso de perderse u obtener algún fallo en el funcionamiento.

Como en ejemplos anteriores utilizaremos la técnica *Ray Casting* para la detección de elementos en escena. Para poder detectar la colisión con nuestra mesa, esta debe de llevar una serie de componentes asociados. En primer lugar, añadiremos un componente *Rigidbody*.

Para esto, con la mesa seleccionada en *Hierachy*, pulsamos en el botón *Add Component* y seleccionamos el menú **Physics→Rigidbody**. Dejamos todas las opciones por defecto a excepción de *Use Gravity* que debemos de desmarcar (Si esta opción estuviese marcada nuestro objeto caería, al afectarle la gravedad, hasta que se encontrase con otro objeto con un *rigidbody* incorporado).

El siguiente componente que debemos de añadir es un *Box Collider*. Esto recordamos que lo podemos hacer en *Add Component* y luego en **Physics→Box Collider**. La configuración del *Box collider*, al igual que en los ejemplos del

capítulo 3, la realizamos pulsando *Edit Collider* y ajustando las líneas verdes al objeto. Finalmente debemos de activar la opción *Is Trigger* para que todo funcione correctamente.



Ahora debemos de construir nuestro menú selector de color y fabricarnos una serie de botones que nos permitan el cambio de color del objeto. Para esto creamos un plano que pondremos delante de la mesa y que llamaremos *MenuColor*. Le asignamos el material *BaseEstanteriaColor* que es el color base de la madera (Lo podemos encontrar en **Assets\03–3D\Muebles\Mesa\materials**). Finalmente lo desactivamos en el Inspector para que no se muestre en el inicio.

Dentro de este *GameObject* incluiremos 4 botones (3 como planos 3D y uno como texto 3D, esto es **GameObject→3d Object→Plane** y **GameObject→3d Object→3d Text**), 3 para los colores y uno para ocultar el Menú. En la parte superior colocaremos también un **3D Text** con el texto "Color".

Para cada uno de los botones (Planos 3D) debemos de añadir un *Mesh Collider* (*Add Component*: **Physics→Mesh Collider**) y en el botón salir usaremos

un *Box Collider* (*Add Component:* **Physics→Box Collider**). En todos los botones debemos de marcar la opción *Is Trigger* tanto en los componentes *Mesh Collider* como en el *Box Collider*.

El material de cada botón debe de ser acorde a lo que representa, así que no debemos de olvidar asignarle un material.

Y ya solo nos faltarían los *scripts*. Para los botones de color asignamos el script **botonColor** y para el de salida el *script* **SalirBoton**.

En la variable *Mueble* del *script* **botonColor** arrastramos el objeto de la mesa y seleccionamos el color. Para el color base usaremos el blanco y para los otros dos el color "Rojo" y "Negro".

A

B

C

En el botón de salida debemos de asignar a *Menu* el *gameobject MenuColor* y a mueble el *GameObject* de la mesa. Finalmente, la configuración de los distintos botones quedaría así:

1. Estructura en *Hierarchy* desde el marcador *IT_Mesa*.
2. Inspector de un botón de cambio de color.
3. Inspector del botón de salida.

Para que todo funcione correctamente ya solamente nos queda asignar el *script* **cambiaColor** al *GameObject* mesa.

Aquí asignamos el material *baseEstanteriaColor* a la variable *Mueble Material* y el *GameObject MenuColor* a la variable *Menu*.

De esta manera podremos visualizar el menú pulsando encima de la mesa y cambiar el color pulsando en los botones de color.

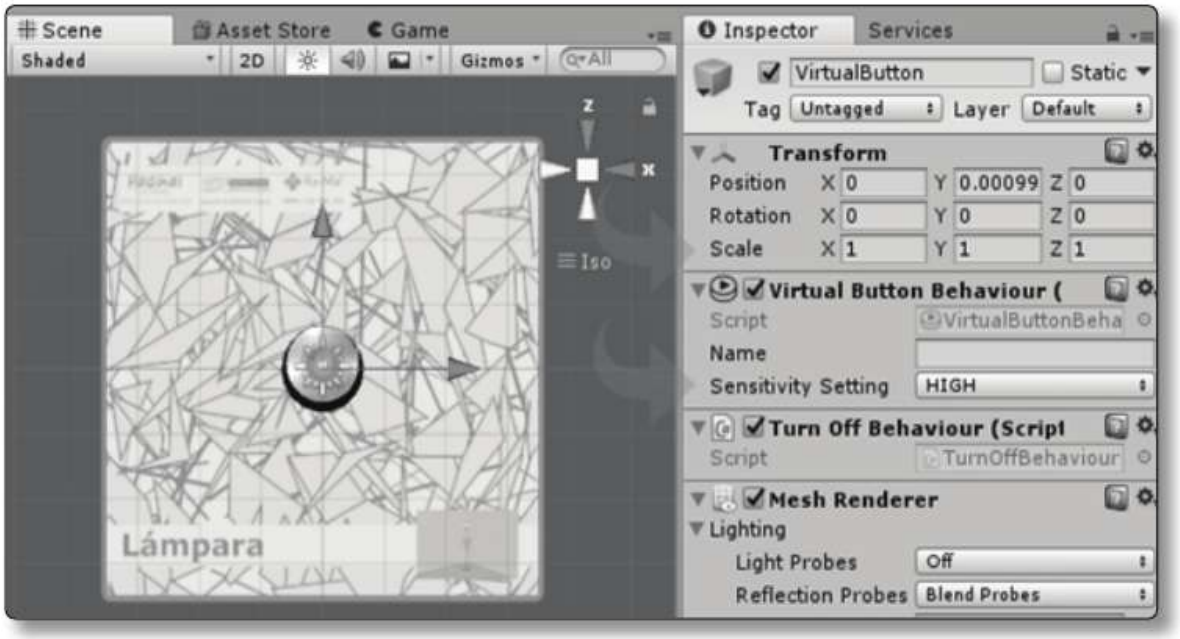
Vemos que de una manera relativamente sencilla podemos manipular los objetos de realidad aumentada con pequeños *scripts* de comportamiento y la técnica *Ray Casting*.

Vuforia nos lo pone todavía más fácil incorporando un sistema propio de botones que podemos programar para realizar determinadas acciones. Los botones son zonas simples encima del marcador que el algoritmo de visión artificial es capaz de manejar. Vuforia es capaz de detectar cuando interponemos un objeto entre el marcador y la cámara en la zona concreta del botón (Por ejemplo, un dedo) y cuando retiramos este objeto.

Para ilustrar esta funcionalidad vamos a crear un botón virtual que encenderá una luz al acercar el dedo y la apagará al retirarlo. Como no puede ser de otra manera usaremos el marcador con la lampara de pie para este menester.

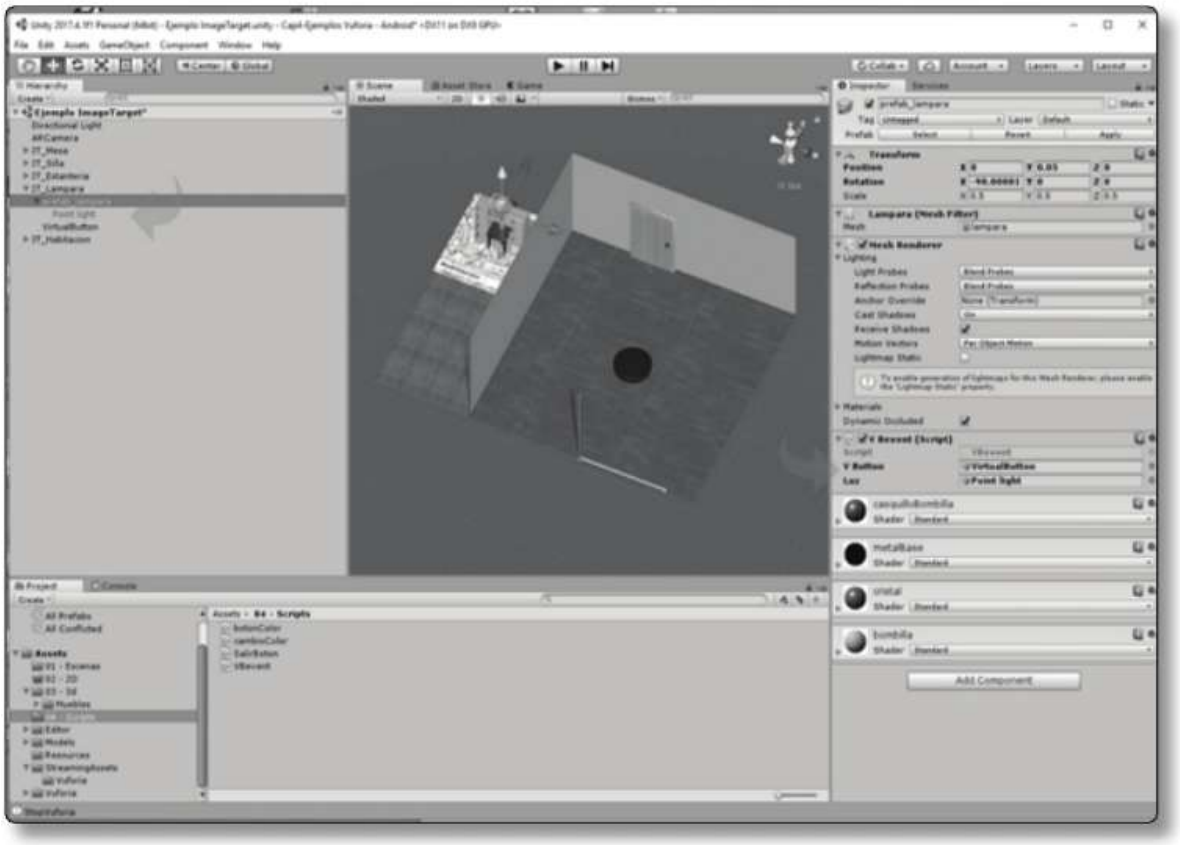
Lo primero que haremos es colgar una *Point Light* del *GameObject* de la lámpara en *Hierarchy*. En principio las únicas modificaciones que haremos en el *Inspector* será dejarla desactivada y en *Shadow Type* elegiremos *hard Shadows* (De esta manera veremos como la luz refleja sombras sobre las paredes de la habitación). También conviene subir la luz para hacerla coincidir con la bombilla de la lampara y que el efecto sea más real.

Después de esto nos iremos al marcador de la lámpara (*IT_Lampara*) y en el *Inspector* desplegaremos la sección *Advanced* del *script* **Image Target Behaviour**. Al final pulsamos el botón *Add Virtual Button* y vemos como se añade un *GameObject* nuevo a *IT_Lampara*. Ahora nos vamos a este nuevo *gameobject* llamado *VirtualButton* y cambiamos *Transform.Scale.X* y *Transform.Scale.Z* al valor 1. Vemos ahora que el botón ocupa todo el marcador. En este caso ocuparemos todo el marcador, pero se puede usar solamente una parte de este y definir diferentes botones para las acciones. Para acabar con este objeto cambiaremos el valor del script **VirtualButton Behaviour→Sensitivity Setting** de LOW a HIGH. Con esto hacemos que el botón sea más sensible a los cambios.

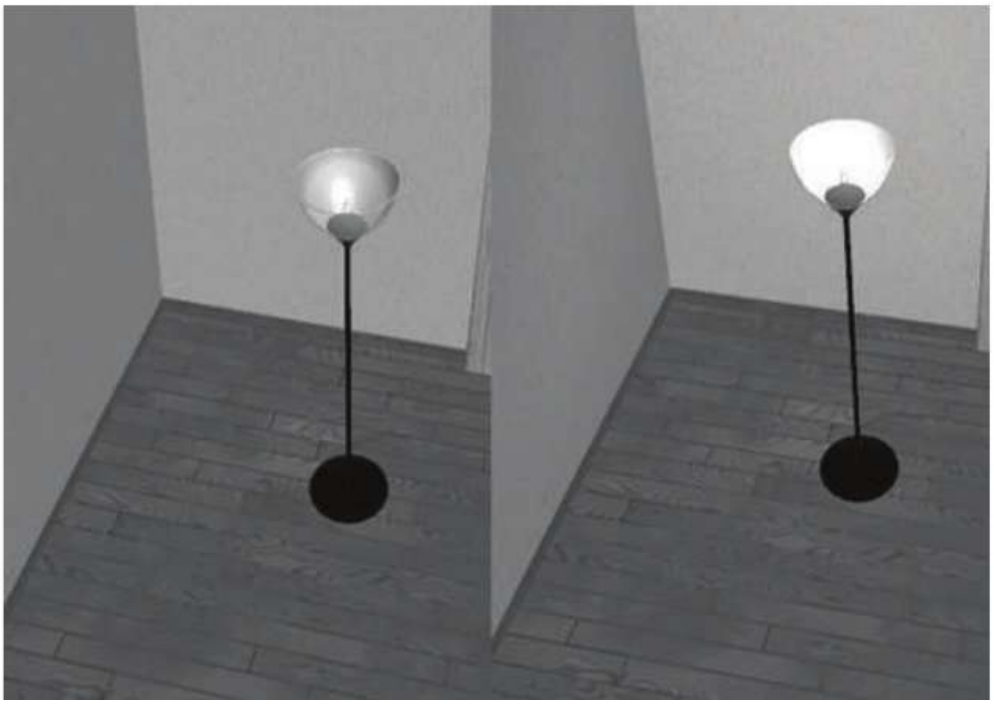


Finalmente arrastramos encima del *gameobject* de la lámpara el *script* **VBevent** que será el encargado de hacer que todo funcione. Deberemos rellenar la variable *VButton* con el *GameObject VirtualButton* y la variable *Luz* con la nueva luz que hemos definido.

Si lo hemos hecho todo bien debe de quedar una escena similar a esta:



Si ahora probamos el ejemplo veremos que con poner un dedo encima del marcador la luz se encenderá. El efecto debe de ser parecido a este en la escena:



4.4 MARCADORES CÚBICOS Y CILÍNDRICOS

Ya que hemos visto cómo funcionan varios marcadores en la escena vamos a explicar otras opciones distintas, además de una imagen simple, para la creación de un marcador.

Vuforia permite crear marcadores con formas en 3D, concretamente con forma cúbica y cilíndrica que nos dan mucho juego a la hora de crear efectos visuales con ellos.

Empezamos con el cubo. Para que la creación de este marcador sea sencilla para los lectores y fácil de implementar vamos a crear un *Papercraft* cúbico con diferentes imágenes.

Lo primero que debemos de hacer es acceder a la web de *Vuforia developer* y en nuestra base de datos (en nuestro caso "Rama") creamos un nuevo marcador.

Ahora, en *Type* elegimos la opción *Cuboid*. En dimensiones marcamos 5 en *Width*, *Height* y *Length*. El nombre de este marcador será *trofeo-marker*. Después de crearlo debemos de seleccionarlo para añadir las 6 imágenes que conformarán las caras del cubo.

Las imágenes de las 6 caras del cubo las podemos personalizar como queramos. En los contenidos adicionales del libro proporcionamos una carpeta en el capítulo 4 llamada **Marcadores cúbico y cilíndrico** que contiene los ficheros *.jpg* necesarios para crear nuestro marcador en el portal de Vuforia, así como los *.pdf* listos para imprimir en cualquier impresora (tanto del cubo como del cilindro).

Como podemos ver en la imagen vamos pulsando en las distintas partes del cubo (*Back,Top,Left,Front,Right* y *Botton*) y subiendo los ficheros (*.jpg*) correspondientes. Con esto tendríamos configurado el marcador cúbico.

Ya que estamos aquí vamos a configurar el marcador cilíndrico también que vamos a usar en el siguiente ejemplo. En nuestro caso usaremos un cilindro perfecto, aunque Vuforia te permite generar marcadores con distintos diámetros en base y tapa.

Para este ejemplo volvemos a pulsar *Add Target* y elegimos *Cylinder* y configuramos los valores de *Botton Diameter*, *Top Diameter* y *Side Length* a 6.36,6.36 y 8 respectivamente. El nombre será *radikalcola-marker*. De igual manera arrastramos el fichero *.jpg* correspondiente al marcador cilíndrico y ya tendremos configurado nuestro marcador.

NOTAS

Las dimensiones del cilindro debemos de calcularlas en base al desarrollo que necesitemos. Para nuestro ejemplo vamos a usar una botella de vino como soporte y las dimensiones de la etiqueta será de 20 cm de Ancho X 8 cm de alto. El cálculo del diámetro lo sacamos de la formula L = 2R. Como la longitud es 20 cm, sacamos el radio de la siguiente manera R = L / (2), o R= 20/(2) = 3.18 cm. El cálculo del diámetro ya es sencillo D = 2R = 2 x 3.18 = 6.36 cm.

Una vez definidos los dos nuevos marcadores volvemos a descargar la base de datos pulsando en el botón *Download Database (All)* y elegimos el formato *Unity Editor*. El fichero generado lo importamos en nuestro proyecto como siempre en **Assets→Import Package→Custom Package**.

Para los ejemplos que vamos a trabajar vamos a necesitar algunos modelos 3D que encontraremos en el fichero **CuboYCilindro.unitypackage** en la sección

del capítulo 4 de contenidos adicionales. Al igual que el anterior lo importamos en nuestro proyecto en **Assets→Import Package→Custom Package**.

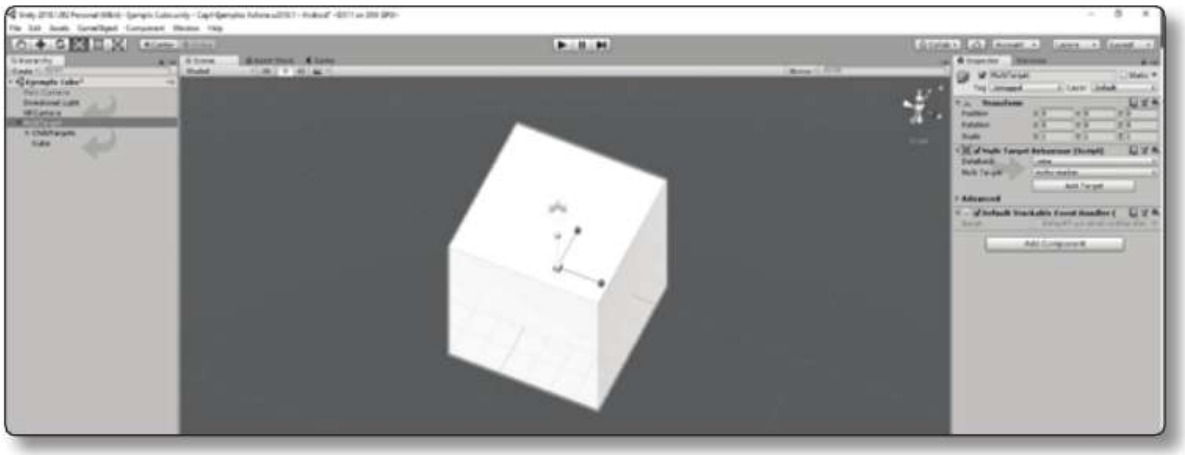
Antes de continuar debemos de tener nuestros dos marcadores físicos impresos y montados. En nuestro caso el cubo es muy fácil de montar con un poco de pegamento. Para el cilíndrico nos basta usar de base una botella de vino, cerveza o lata. Recordamos que los ficheros para impresión (*.pdf*) está disponibles en el material adicional del capítulo para descarga.

Ya tenemos todo lo necesario para configurar nuestra nueva escena. Vamos a empezar por el marcador cúbico. Creamos una nueva escena (En nuestro caso se llamará "Ejemplo Cubo") y anulamos el *GameObject* **Main Camera** como hemos viniendo haciendo en los ejemplos anteriores. El siguiente paso es incluir nuestra **AR Camera** en **GameObject→Vuforia→AR Camera**.

Como en ejemplos anteriores, abrimos nuestra configuración de Vuforia (Pulsando en **ARCamera**) y Open *Vuforia Configuration* en la pestaña *Inspector*. Debemos de comprobar que está incluida nuestra licencia de uso (si no es así la incluimos) y que nuestra base de datos está activa.

Una vez configurada la cámara debemos de incluir nuestro marcador. En este caso el *GameObject* a insertar en escena se llama **Multi Image** y lo encontramos en **GameObject→Vuforia→Multi Image**. En *Hierachy* el objeto aparece como *MultiTarget*. Si nos vamos al Inspector en este nuevo *GameObject* debemos de configurar los parámetros *database* y *Multi Target* del *script* **Muti Target Behaviour**. En nuestro caso, al no existir nada más que un *multitarget* en la base de datos se ha configurado solo.

Finalmente creamos un cubo que cuelgue de este nuevo objeto marcador y lo escalamos para que sea ligeramente más grande que nuestro marcador real. La escena debe de quedar muy parecida esto:



Podemos probar que está todo correcto arrancando la escena. Vemos que girando nuestro cubo físico giramos el virtual sin problema.

Para acabar este ejemplo usaremos un objeto 3D con base cúbica, concretamente un trofeo. Lo encontraremos en **Assets\03 – 3D\trofeo\Prefab_ Trofeo**. Basta con arrastrar el *prefab* para que cuelgue de nuestro **MultiTarget** y escalarlo para que coincida la base del trofeo con nuestro marcador cúbico.

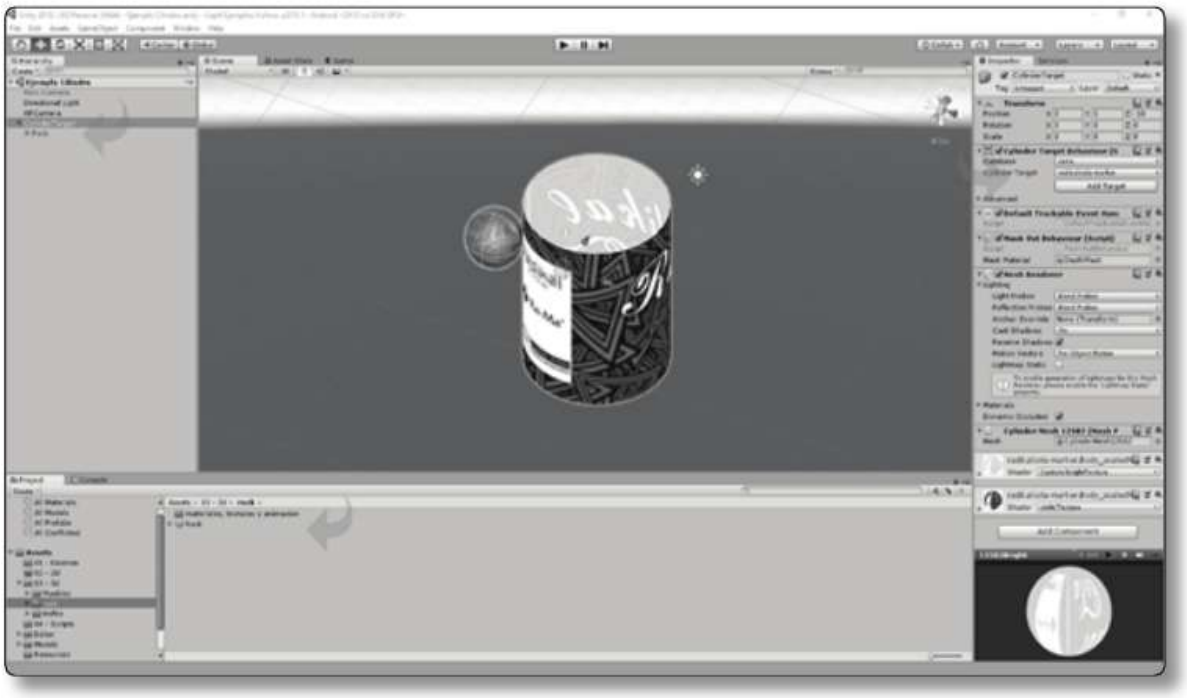
Si ejecutamos de nuevo la escena deberíamos ver algo tal que así.

Para el marcador cilíndrico creamos una nueva escena y volvemos a realizar los pasos comunes, esto es, anulamos el *gameobject* **Main camera** y añadimos la **AR Camera** (**GameObject→Vuforia→ARCamera**). Si estamos dentro del mismo proyecto no debe de haber cambiado la configuración de Vuforia pero siempre conviene revisar la configuración dentro del Inspector en **ARCamera** (*Open Vuforia Configuration*).

Una vez configurada la cámara debemos de incluir nuestro marcador. En este caso el *GameObject* a insertar en escena se llama **Cylinder Target** y lo encontramos en **GameObject→Vuforia→ Cylindrical Image**. En *Hierachy* el objeto aparece como *Cylinder Target*. Si nos vamos al Inspector en este nuevo *GameObject* debemos de configurar los parámetros *database* y *Cylinder Target* del *script* **Cylinder Target Behaviour**. Al igual que nos pasaba con el cubo, al no existir nada más que un **Cylinder Target** en la base de datos, se ha configurado solo.

En este caso hemos añadido una esfera con textura que gira alrededor de nuestro target. El *prefab* de la esfera lo tenemos disponible en **Assets\03–3D\rock\ rock**. Si arrastramos este *prefab* para que cuelgue del objeto marcador.

Si lo hemos hecho todo bien la pantalla debería quedar tal que así:



Si todo ha ido correctamente y ejecutamos la escena veremos como la esfera con textura de roca gira alrededor de nuestra botella dejando una estela.

4.5 VUMARKS

Según los define Vuforia, los marcadores *Vumark* son la "Nueva generación de códigos de barras". Estos marcadores nos permiten definir un diseño personalizado como los **Image Target** pero incluyendo a la vez una inserción extra de datos en ellos a través de una codificación binaria visual reconocible por el algoritmo de visión artificial de Vuforia.

El *Vumark* nos permite ofrecer una experiencia única con realidad Aumentada basándonos en los datos codificados del marcador. Al igual que sucede con los códigos de barras, *datamatrix* o códigos QR, los *Vumark* nos permiten codificar una URL, número de serie de un producto o cualquier otro dato que necesitemos dentro de nuestro marcador. La ventaja del *Vumark* sobre los códigos anteriormente citados está en que incorpora, además de los datos codificados, la experiencia de Realidad aumentada.

Las principales ventajas que supone el usar *Vumark* frente a otros tipos de codificación son:

- ⇨ Podemos representar millones de instancias "reconocibles" diferentes con un "apariencia" similar. Esto es especialmente útil para entornos corporativos e industriales al poder integrar nuestra imagen de marca en la experiencia que desarrollemos para nuestros clientes.
- ⇨ Podemos codificar una gran variedad de formatos de datos usando el mismo diseño.
- ⇨ Establecemos una diferenciación "cualitativa" con respecto al resto de formatos de codificación existentes incorporando la Realidad Aumentada.

Seguro que el lector puede imaginar muchos casos de uso para entornos corporativos y/o Industriales, pero nosotros proponemos algunos aquí:

- ⇨ Identificación de piezas o equipos en almacenes.
- ⇨ Registro preciso de operaciones.
- ⇨ Mostrar instrucciones de uso de producto.
- ⇨ Desbloqueo de contenido digital con una etiqueta física (Por ejemplo, como contenido extra en la compra de producto como juguetes).
- ⇨ Etc.

En primer lugar, y como hemos hecho con los anteriores marcadores, nos vamos al portal de vuforia para generar la base de datos. En este caso vamos a generar una base de datos diferente a la generadas anteriormente. Seguimos los siguientes pasos:

1. En *https://developer.vuforia.com*, nos vamos a *Develop* en el menú y posteriormente a *Target manager*.

2. Pulsamos en el botón *Add Database* y aquí ahora seleccionamos el tipo como *Vumark* y le damos un nombre (En nuestro caso será "RamaVuMk".

3. Ahora tenemos que añadir un marcador. Para diseñar un marcador necesitamos una licencia de *Adobe Ilustrator* junto a un plugin llamado *Vuforia Vumark Designer*. Este plugin está disponible en *https://developer.vuforia.com/downloads/tool* junto al resto de herramientas que proporciona Vuforia. Si descargamos el Kit viene una guía de como diseñar nuestros propios marcadores Vumark (También disponible aquí: *https://developer.vuforia.com/sites/default/files/VuMark%20Design%20Guide.pdf*), los scripts del *plugin* y un par de ejemplos (*Morton Tuxedos* y *Chateau*). En nuestro caso vamos a coger uno de los ejemplos que nos proporcionan, concretamente *Chateau*, para usarlo como marcador Base. Este marcador está configurado con un ID alfanumérico de longitud 10. El fichero **Chateau.svg** está en la carpeta ejemplos del plugin descargado o también en la carpeta **Marcador Vumark** de los contenidos adicionales de este capítulo. Para añadir el marcador pulsamos *Add Target* y subimos el diseño de *Vumark* elegido, en nuestro caso **Chateau.svg.** Configuramos *width* a 10.

4. Para generar los distintos marcadores codificados con el ID que necesitamos debemos de pulsar el *Generate Vumark*. En la pantalla siguiente debemos de introducir el ID y elegir el formato de exportación del marcador.

5. Finalmente descargamos la base de datos.

Como podemos observar los marcadores generados guardan el diseño original con pequeños cambios (En nuestro caso en las ventanas de la casa) que son las diferencias entre los distintos ID que contienen. En los contenidos adicionales del capítulo también disponemos de un PDF para impresión con varios marcadores con ID diferente para nuestro ejemplo, aunque animamos al lector a que genere los suyos propios.

El siguiente paso a seguir es importar la base de datos generada en formato *unitypackage* en **Assets→Import package→Custom package**. Nuestra nueva escena como siempre debe de tener anulada el *gameobject* **Main Camera** y sustituirlo por el **AR Camera** de Vuforia (**Gameobject→Vuforia→AR Camera**).

Ahora incluimos el *gameobject* **Vumark** en **Gameobject→Vuforia→ Vumark** y en el *Inspector* nos vamos a **VuMark Behabiour** y seleccionamos la base de datos y la plantilla *Vumark* (En nuestro caso *RamaVuMk* y *Chateau*.

Finalmente vamos a colgar un "Texto 3D" (**GameObject→3D Object→3D Text**) de este objeto *VuMark* y configuramos un texto de inicio como "*ID:*". La escena debe de quedar algo tal que así:

Si probamos vemos que los marcadores impresos nos muestran el texto "*ID:*" cuando los detecta al igual que si fuese un **ImageTarget** normal.

Lo siguiente es "obtener" el ID asociado a cada especificación del *Vumark* que hemos generado antes. Para esto necesitamos manipular algo de código de la API de Vuforia. Vamos a crear un *script* que será la base de cualquier aplicativo que se nos ocurra.

Creamos un nuevo script llamado **RamaVumarkEvents** y se lo asignamos a un nuevo *Gameobject* en escena (nosotros le hemos llamado *VumarkEvents*).

El *Script* es el siguiente (Existe una copia en los contenidos adicionales del capítulo):

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Vuforia; //(1)
5  public class RamaVumarkEvents : MonoBehaviour {
6      public TextMesh Texto3D; // (2)
7      private VuMarkManager vuMarkManager;
8      // Use this for initialization
9      void Start () {
10         // Creamos una instancia de VuMarkManager (3)
11         vuMarkManager = TrackerManager.Instance.GetStateManager().
12 GetVuMarkManager();
13         // Registramos nuestro método personalizado "onVuMarkDetected"
14         //como el método a llamar
15        //cuando detectamos un Vumark (4)
16         vuMarkManager.RegisterVuMarkDetectedCallback(onVuMarkDetected);
17 // Registramos nuestro método personalizado "onVuMarkLost" como
18 // el método a llamar cuando perdemos un VuMark (4)
19         vuMarkManager.RegisterVuMarkLostCallback(onVuMarkLost);
20     }
21
22
23 //Método encargado de obtener el ID integrado dentro de un marcador VuMark detectado (7)
24     private string getVuMarkID(VuMarkTarget vuMark)
25
26     {
27         switch (vuMark.InstanceId.DataType)
28         {
29             case InstanceIdType.BYTES:
30                 return vuMark.InstanceId.HexStringValue;
31             case InstanceIdType.STRING:
32                 return vuMark.InstanceId.StringValue;
33             case InstanceIdType.NUMERIC:
34                 return vuMark.InstanceId.NumericValue.ToString();
35         }
36         return null;
37     }
38     //Método que vamos a llamar cuando detectamos un Vumark (5)
39     public void onVuMarkDetected(VuMarkTarget target)
40     {
```

```
41        Texto3D.text = “ID: “ + getVuMarkID(target);
42    }
43     //Método que vamos a llamar cuando dejamos de detectar un Vumark (6)
44    public void onVuMarkLost(VuMarkTarget target)
45    {
46        Texto3D.text = “”;
47    }
48 }
49
```

Como vemos en la cabecera incluimos una llamada a la librería de Vuforia para poder usar los objetos y métodos de la API (1).

También incluimos una variable pública llamada “Texto3D” que nos servirá para vincular el *GameObject* “Texto3D” que va a recibir el ID del *Vumark* y nos lo va a mostrar visualmente (2).

La variable privada *VuMarkManager* es la que recogerá la instancia del gestor de *Vumark* que como vemos creamos en la primera instrucción de la función *Start()* (3). En la función *Start()* también registramos los métodos personalizados que se ejecutarán al “Detectar” un nuevo *Vumark* y cuando “se pierda la detección” de este (4). En nuestro caso proyectaremos el ID al objeto **3D Text** que tenemos en escena, pero podemos modificarlos para hacer cualquier cosa que se nos ocurra (Consulta a una base de datos, llamada a una URL, etc. …) al ya disponer del “ID” oculto en el marcador. Como se puede ver disponemos de un método que llamamos al detectar el marcador (*onVuMarkDetected*)(5) y otro que se llama al perderlo (*onVuMarkLost*)(6).

Finalmente existe otra función que devuelve un *String* con el contenido del “ID” del marcador *Vumark* que le pasamos como parámetro (7).

Con esto ya tenemos controlado el funcionamiento básico de este tipo de marcadores que están pensados claramente para aplicaciones corporativas e industriales.

Si al ejemplo sencillo que hemos implementado le unimos una imagen corporativa (Diseño personalizado) y un acceso a la base de datos de la empresa podemos imaginar una infinidad de aplicaciones:

- ⇨ Control visual de almacén.
- ⇨ Mantenimiento de maquinaria.
- ⇨ Información extra de producto en exposiciones.
- ⇨ Contenido desbloqueable mediante *tockens*.
- ⇨ Etc.

4.6 GROUND PLANES

Hasta ahora hemos usado marcadores para posicionar objetos virtuales en el mundo real. *Vuforia Ground Plane* permite que el contenido digital se pueda colocar sobre superficies horizontales del entorno como el suelo, mesas, etc. … También permite otros anclajes sobre el nivel del suelo ("*on Air*"), aunque no deja de posicionarlos sobre la superficie detectada.

Ground Plane es totalmente compatible con el resto de capacidades de Vuforia sobre marcadores por lo que el abanico de usos se abre ostensiblemente. Ahora ya podemos usar cualquier escena del mundo real para interactuar con el contenido digital generado.

Las posibilidades de los aplicativos son muy variadas, pero ponemos aquí algunos de los posibles usos que se le está dando en la actualidad:

- ⇨ *Juegos*: Crear videojuegos en los que se pueda posicionar personajes en cualquier lugar sobre superficies es un uso que ya hemos visto con ejemplos como "*Pokemon Go*" y que han tenido un éxito considerable.
- ⇨ *Revisión de diseños 3D*: Poder colocar una representación del modelo 3D y poder rodearlo, ver su interior o manipularlo virtualmente se ha convertido por ejemplo en una herramienta en auge en el entorno del automóvil. Los cambios en cualquier diseño de automóvil son costosos y una herramienta de manipulación de este tipo ayuda enormemente a los diseñadores.
- ⇨ *Posicionamiento de objetos en general en el entorno real*: La colocación de objetos como muebles o electrodomésticos en nuestro hogar antes de adquirirlos nos proporciona una mejor experiencia de compra.

Esta tecnología no es compatible con todos los dispositivos por lo que necesitaríamos uno compatible para poder ejecutar los ejemplos. La lista de dispositivos está disponible en el portal de Vuforia, concretamente en esta URL:

https://library.vuforia.com/articles/Solution/ground-plane-supported-devices.html

Pasamos a explicar cómo montar un ejemplo básico para entender cómo funciona y se integra con Unity 3D.

Nuestro ejemplo básico va a consistir en posicionar un mueble, de los que tenemos ya entre nuestros *Assets* (Nosotros hemos elegido la silla), sobre una superficie.

Para poder hacer la simulación desde el editor debemos de imprimir un marcador y así poder hacer el ajuste. Este marcador ya no será necesario en la versión

compilada para Android, iOS o UWP, lógicamente, siempre y cuando, usemos un dispositivo compatible.

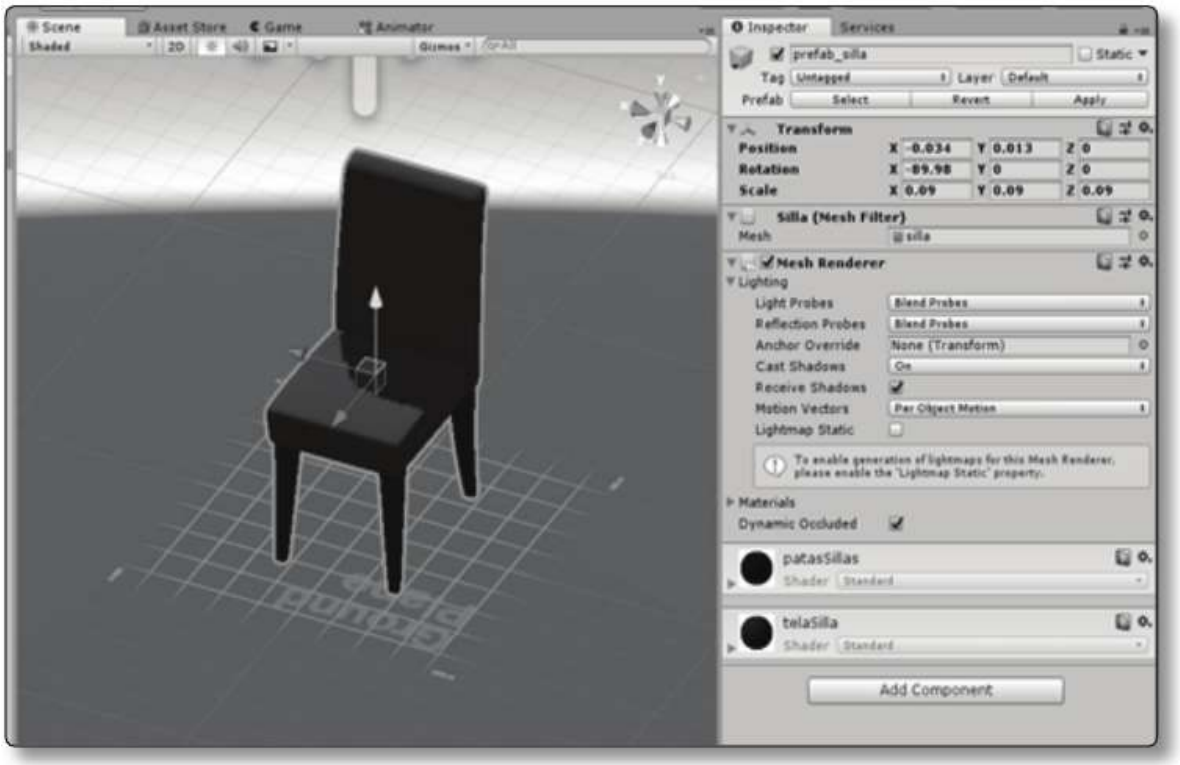
En nuestro proyecto abrimos una nueva escena y localizamos el marcador (físico) en la carpeta **Assets\Editor\Vuforia\ForPrint\Emulator**. Este marcador dispone de unas medidas de 20x20 cm y nos servirá para jugar desde el editor y hacer las pruebas.

En nuestra nueva escena agregamos nuestra **AR Camera** y eliminamos la **Main Camera** como hemos hecho en los ejemplos anteriores.

En *Hierachy* añadimos un nuevo objeto **Ground Plane Stage** bien pulsando al clic derecho del ratón o mediante **GameObject→Vuforia→Ground Plane→Ground Plane Stage**.

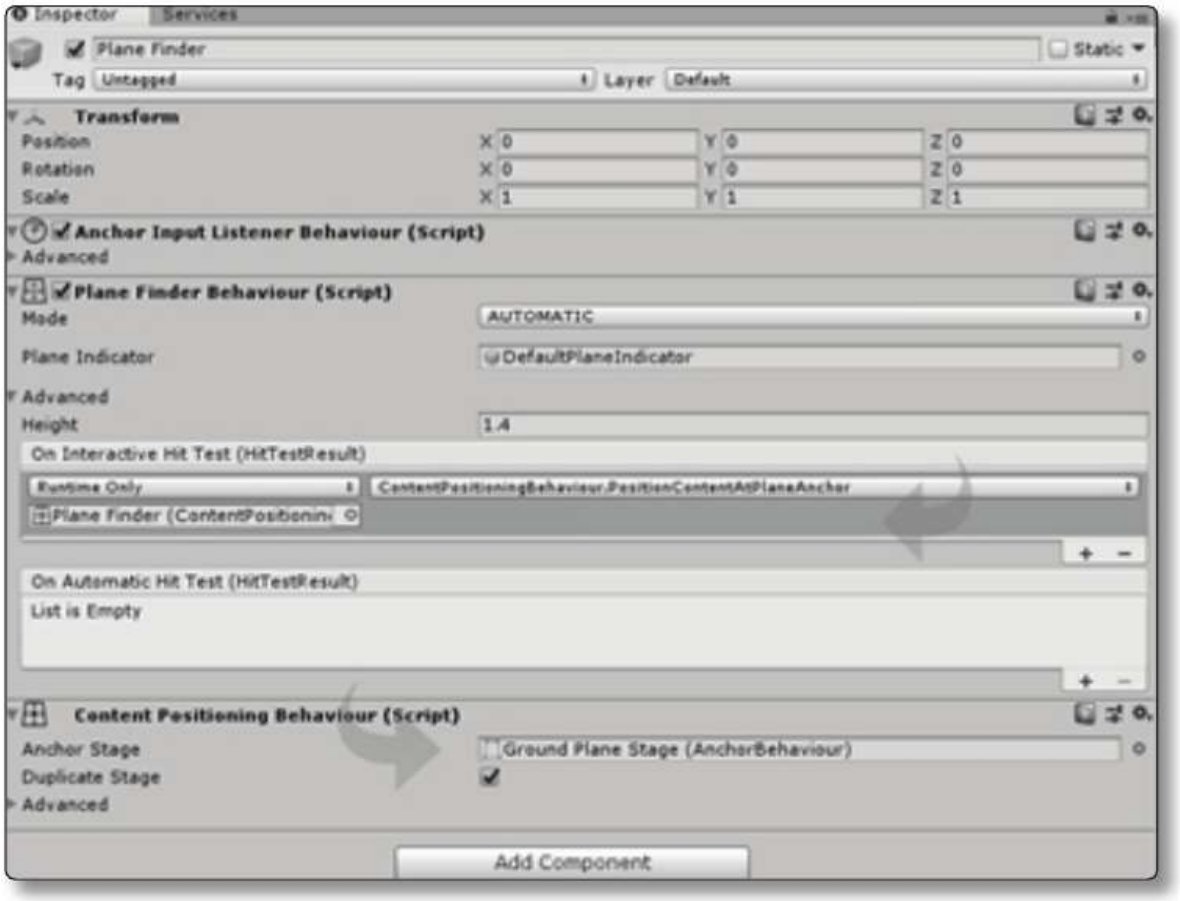
Ahora buscamos nuestra silla en **Assets\03-3d\Muebles\Silla\prefab_silla** y la añadimos como objeto hijo del **Ground Plane Stage**.

Como vemos en el editor la silla se posiciona sobre una superficie de 100x100cm. Es aquí donde debemos de manipular la escala del objeto (el *prefab* de la silla) para adaptarlo a las dimensiones que tendría en el mundo real. Usamos esta plantilla de 100x100 cm para guiarnos y nosotros, como se puede ver en la imagen, hemos ajustado a una escala de 0.09 en los tres ejes.



El siguiente objeto necesario para nuestra escena es el **Plane Finder**, que lo encontramos en **GameObject→Vuforia→Ground Plane→Plane Finder**. Este objeto es el encargado, como su nombre indica, de detectar el suelo y luego posicionar el objeto que hemos definido en **Ground Plane Stage**. Para definir que **Ground Plane Stage** debe de posicionar debemos de seleccionar el **Plane Finder** e irnos a *Inspector*. Debemos arrastrar nuestro objeto **Ground Plane Stage** sobre el campo *Anchor Stage* en el *Script* **Content Posicioning Behaviour.**

Si no tocamos nada, la escena quedaría lista para, que al detectar el plano del suelo y hacer un *touch* en pantalla (Un clic en el editor), posicione la silla en el tamaño a escala sobre el suelo detectado. Cualquier otro comportamiento que queramos definir podemos añadirlo en el script **Plane Finder Behaviour** desplegando la sección *Advanced*. Si hacemos esto veremos cómo queda definido el comportamiento por defecto y pulsando en el "+" podremos añadir cualquier otro que necesitemos.



Si ejecutamos la escena podemos comprobar como aparece previamente un recuadro blanco indicando que hemos detectado suelo y luego al tocar la pantalla o realizar un clic (En editor) aparece la silla. Recordamos que para el posicionamiento usando el editor debemos de usar el marcador de suelo que hemos impreso.

Para los anclajes en el aire Vuforia sigue basándose en la detección del suelo por lo que bastaría con elevar el objeto "*prefab Silla*" sobre el plano del **Ground Plane Stage** para mostrarlo elevado sobre el suelo.

Vuforia dispone de otros *gameobjects* para esto, pero el comportamiento es el mismo que los que hemos explicado, aunque invitamos al lector a que experimente también con ellos. En vez de **Ground Plane Stage** existe un *gameobject* llamado **Mid Air Stage** y para el comportamiento en vez de **Plane Finder** tenemos el **Mid Air Positioner**, pero en esencia, son prácticamente lo mismo. La única diferencia es el gráfico que se muestra como referencia en el editor que cambia a un eje cartesiano en vez de la cuadricula. De hecho, si queremos transformar un **Ground Plane Stage** en un **Mid Air Stage,** bastaría con irnos al *Inspector* y en *Anchor Behaviour* cambiamos de *Plane* a *Mid Air* en *Stage Type*.

Las posibilidades de esta tecnología de uso de superficies y anclajes como soporte a nuestros elementos virtuales entrarían dentro de la denominada "Realidad Mixta". Desarrollaremos este concepto mucho más en profundidad en capítulos posteriores con tecnologías más potentes como ArCore y ArKit.

4.7 CLOUD RECOGNITION

El servicio *Cloud recognition* que proporciona Vuforia es una solución que te permite incluir marcadores en tu base de datos online sin necesidad de volver a importar la base de datos a tu aplicación y por tanto tener que volver a compilarla.

Básicamente la base de datos de marcadores está online, y no en la aplicación, y, por tanto, tienes muchísima más flexibilidad a la hora de gestionar los marcadores.

El uso de este servicio está indicado para aplicativos que requieren de un gran número de marcadores (Puedes usar más de un millón de marcadores en una misma aplicación) o que necesitan que estos marcadores se puedan actualizar con frecuencia.

Como ventaja adicional sobre la base de datos *device* es que puedes incluir metadatos en los marcadores que se capturan mediante la API una vez reconocido el marcador. Esto lo explicaremos más detenidamente más adelante.

Como desventaja implícita (al tener que usar servidores on-line), es que es necesario una conexión a internet activa en el momento del reconocimiento y, que la velocidad de esta conexión, será decisiva en el tiempo que tarde nuestra aplicación en detectar el marcador.

Para poder usar el servicio debemos de crear una nueva base de datos en nuestra cuenta. Con la licencia Free, o de desarrollo, el número de reconocimientos por marcador se limita a 100 al mes, pero es más que suficiente para probar la tecnología y ver cómo funciona todo.

Como hemos hecho con las bases de datos anteriores nos vamos al portal de desarrollador de Vuforia y en *Target Manager* creamos una nueva base de datos. Esta vez marcamos la opción *Cloud*. En nuestro caso la vamos a llamar "ramaCloud".

En principio los marcadores de esta base de datos están limitados a **Image Target** o simples. Nosotros vamos a subir un par de imágenes de las caras del cubo para posicionar el trofeo en 3D y la lampara.

Para crear el primer marcador pulsamos en *Add Target* y al igual que hicimos anteriormente debemos de especificar una imagen, un ancho (*Width*) y un nombre (*Name*). En este caso además nos da la posibilidad (Opcional) de subir un fichero de metadatos que no debe de superar los 2mb.

En este momento hacemos un inciso y explicamos varias cosas sobre los metadatos asociados al marcador:

- ⇨ Una vez reconocido el marcador por la API de Vuforia nos devuelve estos metadatos y los podemos capturar como hacíamos en el caso de los marcadores VUMARK. El uso que le demos posteriormente es algo que se deja a la imaginación del lector, pero las posibilidades son enormes.

- ⇨ El fichero donde introduzcamos los metadatos debe de ser de texto ASCII.
- ⇨ El formato puede ser cualquier tipo de estructura de texto que luego podamos interpretar en la aplicación. Para aplicaciones con gran cantidad de datos se suele usar algún formato transaccional de datos como XML o JSON donde la información queda estructurada de una manera clara y fácilmente "interpretable" o "parseable" por la aplicación. En nuestro caso usaremos cadenas de texto simple para no complicar demasiado el ejemplo, aunque para una aplicación profesional con varios campos de datos por marcador es casi imprescindible manejarse con este tipo de formatos.
- ⇨ Nosotros vamos a explicar cómo realizar este proceso a través del portal Web porque es la manera más sencilla pero la API de Vuforia o *Vuforia Web Services* nos permite gestionar nuestros marcadores sin usar el portal y desde un aplicativo propio. Esta posibilidad abre otro gran abanico de opciones a desarrolladores ya que se pueden generar marcadores en tiempo real, asignarle metadatos y luego que estos sean reconocibles por la misma APP, o por otra diferente, de manera totalmente dinámica. Para profundizar en las posibilidades de *Vuforia Web Services* recomendamos visitar el siguiente enlace:

  https://library.vuforia.com/articles/Solution/How-To-Use-the-VuforiaWeb-Services-API.html

El marcador veremos que, una vez definido, tardará un breve tiempo en procesarse y asignarle un *rating*.

Para nuestro ejemplo vamos a subir el marcador usado para la mesa y la lampara en ejemplos anteriores. Recordamos que el valor de *Width* debe de ser 7 en ambos casos. Para los metadatos debemos de tener creados un par de ficheros de texto con una cadena descriptiva de cada objeto para cargarlo en el campo *Metadata Package*. En nuestro caso las cadenas de texto que hemos puesto son:

⇨ Mesa: "001 - Trofeo"
⇨ Lampara: "002 - Lampara de pie"

Una vez creada la base de datos debemos de tener a mano las *Client Access Keys* y *Server Access Keys* en la pestaña *Database Access Keys*. Como podemos observar ya no existe el botón *Download Database (All)* que teníamos en los otros tipos de base de datos ya que los datos estarán siempre disponibles en la nube.

Pasamos a Unity 3D y creamos una nueva escena anulando **Main Camera** y con nuestro *gameobject* **ARCamera** activo como hemos venido haciendo en los otros ejemplos.

El *gameobject* encargado de soportar los marcadores *cloud* se llama **Cloud Image Target** y lo tenemos disponible en **GameObject→Vuforia→Cloud Image→Cloud Image Target.**

También debemos de añadir un *gameobject* **Cloud Provider**. Al igual que el anterior lo podemos añadir en **GameObject→Vuforia→Cloud Image→ Cloud Provider**.

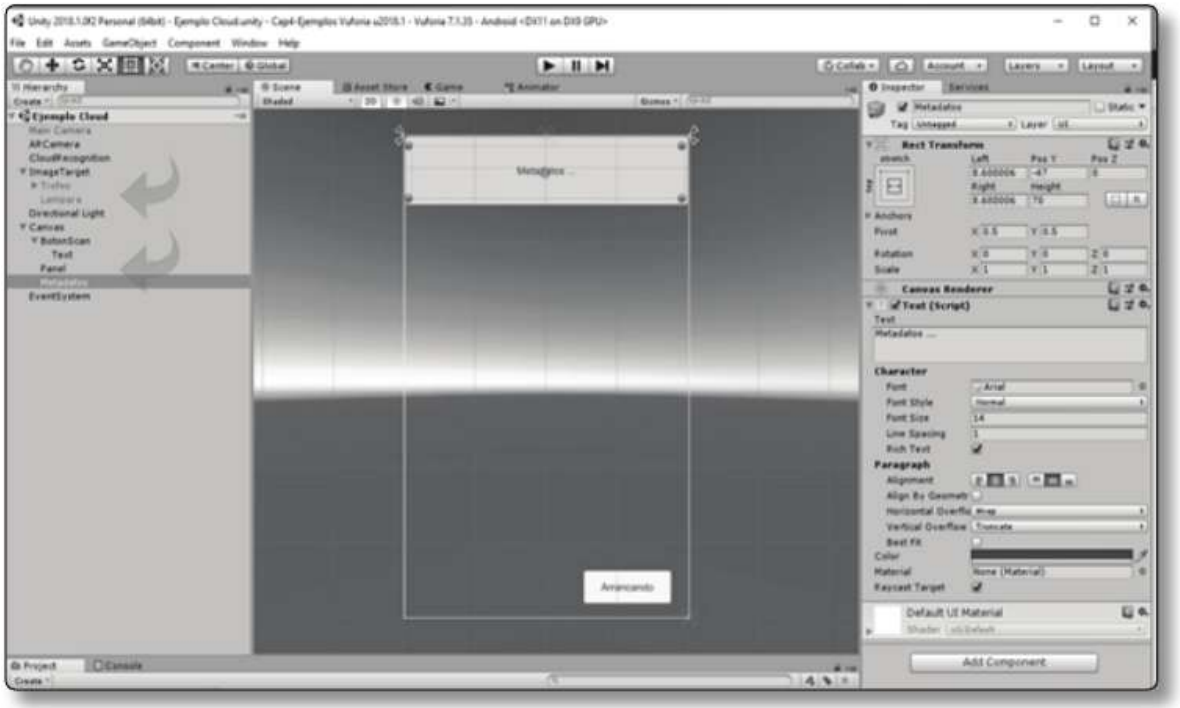
En este último *gameobject*, dentro del script **Cloud Reco Behaviour** en *Inspector* es donde introduciremos la *Access Key* y *Secret Key* que teníamos localizadas en nuestra base de datos. En nuestro caso, como la aplicación es "Cliente" de la base de datos (no vamos a realizar modificaciones sobre los marcadores) elegimos el par de llaves llamado *Client Access Keys*. El otro par de llaves sirven para establecer comunicación con la API mediante la interfaz HTTP REST que dispone Vuforia y que nos permite generar, actualizar y borrar marcadores mediante llamadas al servidor (*Vuforia Web Services*).

Vamos a colgar nuestro trofeo 3D y nuestra lámpara 3D del **Image Target** que tenemos en escena y escalamos ambos sobre el marcador para que salgan con el tamaño que necesitamos. Podemos observar que ya no aparece la imagen del marcador que hemos metido sino una imagen genérica con fondo negro que pone *Cloud Reco*.

Como el lector podrá suponer, el funcionamiento de Vuforia según lo escrito hasta ahora, consiste en, enviar capturas de imágenes de la cámara a sus servidores y contrastarlas con lo que existe en la base de datos. En el momento que existe coincidencia nos envía un objeto con los metadatos asociados al marcador. Si este procedimiento se hiciese constantemente los recursos que se consumirían serían ingentes, por lo que hay que decirle "Cuando" debe de escanear para detectar el marcador.

Para poder "Disparar" el escaneo vamos a generar un **UI.Button** que se encargará de lanzar la función que hace la llamada a la API y un **UI.Text** que nos mostrará los metadatos del objeto.

Si no nos hemos equivocado ni dejado nada atrás la escena tiene que quedar tal que así:



Ahora toca introducir el *script* que le da vida a todo esto. Para este ejemplo hemos modificado el que propone Vuforia en su Web para adaptarlo al ejemplo y lo hemos comentado para que el lector pueda asimilarlo más fácilmente. El script lo podemos encontrar en **Assets\04–Scripts** y se llama **SimpleCloudManagerRama.cs**.

Debemos arrastrarlo encima del *gameobject CloudRecognition*.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using Vuforia;
6  public class SimpleCloudManagerRama : MonoBehaviour,
7  ICloudRecoEventHandler {
8      // (1) Variables públicas
9  public ImageTargetBehaviour ImageTargetTemplate; //El ImageTarget que hemos usado
   en escena
10 public Text TextoMetadatos;  //Texto donde volcamos los metadatos.
11 public Text TextoBotonScan;  //Texto del Botón de escaneo
12 public GameObject o001;  //Gameobject a Activar si los metadatos empiezan por 001
13 public GameObject o002;  //Gameobject a Activar si los metadatos empiezan por
14 //Objeto que controla el reconocimiento y comunicación con la API de Vuforia
15 private CloudRecoBehaviour mCloudRecoBehaviour;
16 private bool mEstaEscaneando = false; //Estado que nos indica si estamos escaneando
   o no
17     //(2) Script para inicialización
18  void Start () {
19         mCloudRecoBehaviour = GetComponent<CloudRecoBehaviour>();
20         if (mCloudRecoBehaviour)
21         {
22             mCloudRecoBehaviour.RegisterEventHandler(this);
23         }
24  }
25     //Función que se ejecuta al inicializar
26 public void OnInitialized()
27 {
28         Debug.Log("Cloud Reco initializing ..");
29 }
30     //Función que gestiona el evento de Error al iniciar un reconocimiento
31 public void OnInitError(TargetFinder.InitState initError)
32 {
33      Debug.Log("Cloud Reco init error " + initError.ToString());
34 }
35       //Función que gestiona el evento de Error al actualizar un reconocimiento
36 public void OnUpdateError(TargetFinder.UpdateState updateError)
37 {
38         Debug.Log("Cloud Reco update error " + updateError.ToString());
39 }
40 // (3) Función que se llama cuando cambiamos de estado (Escaneando o  Parada)
41 public void OnStateChanged(bool scanning)
42 {
43 mEstaEscaneando = scanning;
44 if (scanning)
45         {
```

```
46            TextoBotonScan.text = “Escaneando”;
47                    // Limpia todos los marcadores conocidos
48            var tracker = TrackerManager.Instance.GetTracker<ObjectTracker>();
49            tracker.TargetFinder.ClearTrackables(false);
50        }
51        else
52        {
53            TextoBotonScan.text = “Escanear”;
54        }
55    }
56 //Aquí es donde manejamos el evento que reconoce el marcador Cloud
57 //Se lanza al detectar un patrón o marcador reconocible
58  public void OnNewSearchResult(TargetFinder.TargetSearchResult
59  targetSearchResult)
60    {
61        // Capturo los metadatos del marcador detectado
62        string metadata = targetSearchResult.MetaData;
63        TextoMetadatos.text = metadata.ToString();
64        // Para el reconocimiento de patrones
65        mCloudRecoBehaviour.CloudRecoEnabled = false;
66        if (ImageTargetTemplate)
67        {
68            // Habilitamos el nuevo resultado con el mismo ImageTargetBehaviour:
69            ObjectTracker tracker =
70  TrackerManager.Instance.GetTracker<ObjectTracker>();
71            ImageTargetBehaviour imageTargetBehaviour =
72             (ImageTargetBehaviour)tracker.TargetFinder.EnableTracking(
73             targetSearchResult, ImageTargetTemplate.gameObject);
74        }
75  //Comprobamos los metadatos y activamos un objeto u otro dependiendo de los 3
    primeros caracteres.
76 //Aquí es donde se procesarían los metadatos (Acceso a base de datos, carga de modelos
    3D de manera dinámica, etc. ...
77
78        if (metadata.Substring(0,3) == “001”)
79        {
80            o001.gameObject.SetActive(true);
81            o002.gameObject.SetActive(false);
82        }
83        else
84        {
85            o001.gameObject.SetActive(false);
86            o002.gameObject.SetActive(true);
87        }
88    }
89  // (5) Función que llamamos desde el botón de escaneo y que activa la comunicación con
    la API de Vuforia
```

```
90    public void Escanear()
91    {
92        if (!mEstaEscaneando)
93        {
94            mCloudRecoBehaviour.CloudRecoEnabled = true;
95        }
96    }
97 }
98
```

Como podemos ver nuestra clase *SimpleCloudManagerRama* hereda de la clase *MonoBehaviour* (como todos los *scripts* de Unity 3D) y de *ICloudRecoEventHandler*. Esta última clase es una interfaz para manejar todos los eventos de reconocimiento *Cloud* de Vuforia. La clase dispone de varias funciones que nosotros podemos sobrecargar en nuestro script y que pasaremos a describir a continuación.

Antes de nada, definimos una serie de variables públicas (1) que serán necesarias para las funciones.

- ⇨ *ImageTargetTemplate*: Es la variable que contendrá el **ImageTarget** que hemos usado en la escena, así que debemos de arrastrar el *gameobject* correspondiente aquí.
- ⇨ *TextoMetadatos*: Es el **UI.Text** que mostrará los metadatos detectados en pantalla
- ⇨ *TextoBotonScan*: El **UI.Text** del texto del botón que nos mostrará el estado en el que estamos ("Escanear" / "Escaneando").
- ⇨ "o001": *Gameobject* a activar cuando los metadatos empiecen por 001, en nuestro caso el trofeo.
- ⇨ "o002": *Gameobject* a activar cuando los metadatos empiecen por 002, en nuestro caso la lampara.

En la función "*Start()*"(2) capturamos el componente *CloudRecoBehaviour* de nuestro *gameobject CloudRecognition* y le asignamos nuestro script como manejador de eventos.

Los eventos *OnInitialized*, *OnInitError* y *OnUpdateError* están bastante claros con los comentarios en código.

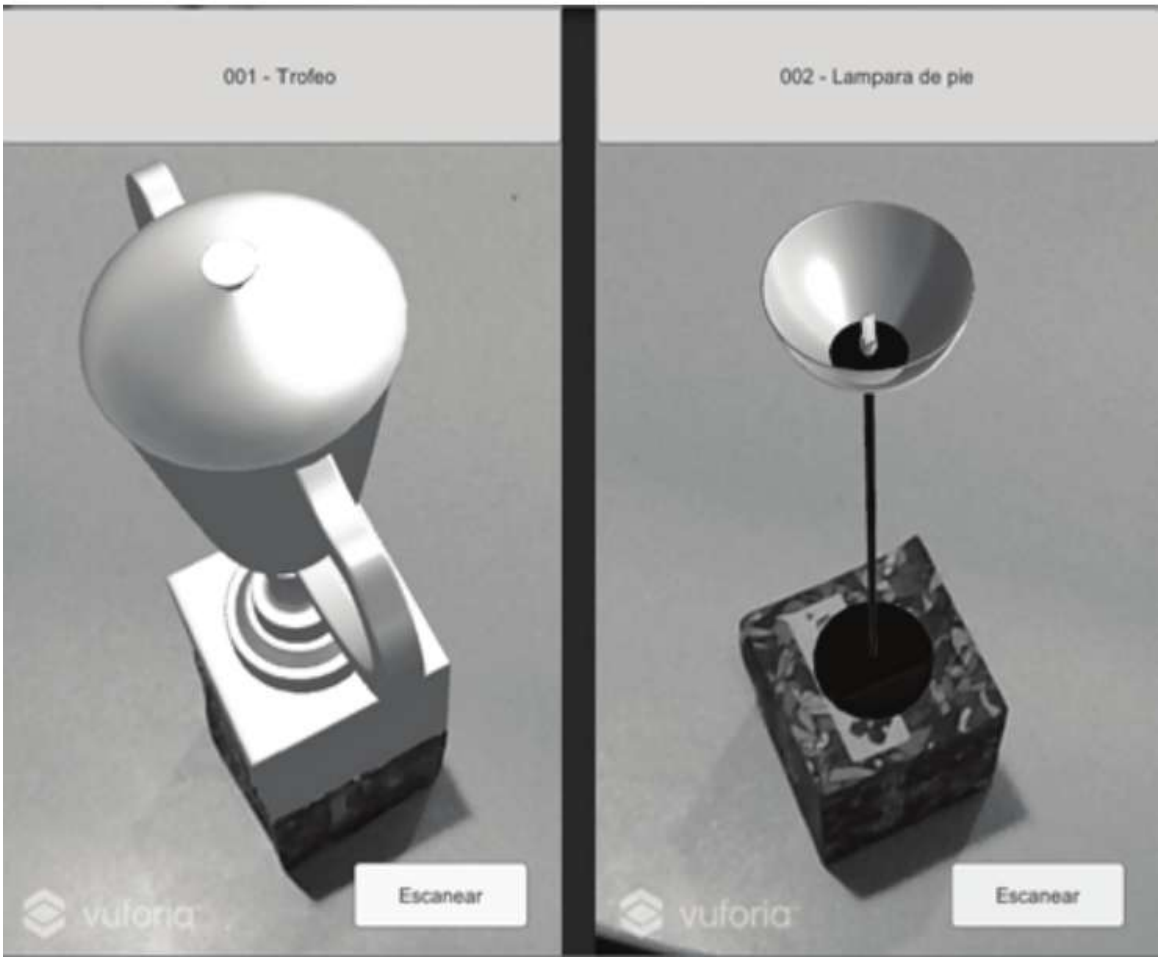
OnStateChange (3) es el evento que gestionamos cuando el estado "*Scanning*" cambia. Básicamente lo que hacemos es limpiar todos los marcadores conocidos detectados previamente y actualizar el texto de nuestro botón al estado en el que estamos.

OnNewSearchResult (4) es el evento que gestionamos cuando el servidor de Vuforia nos devuelve un positivo en la detección. El objeto que devuelve es un *TargetSearchResult* y en la variable *Metadata* tenemos los metadatos asociados al marcador detectado. Tras esto habilitamos el *trackeo* del objeto en el marcador para que el algoritmo de visión artificial de Vuforia haga el seguimiento al marcador. Lo más interesante de este código son sin duda los metadatos. En nuestro caso, solamente los usamos para activar un *GameObject* u otro, pero el abanico de posibilidades es enorme como hemos comentado. En muchas de las aplicaciones existentes en este momento se carga, desde un servidor propio, los contenidos asociados a los metadatos de manera totalmente dinámica.

Finalmente nos queda "Escanear" (5) que es la función que vamos a llamar desde el **UI.Button** de la escena y que habilitará el escaneo del motor.

Si rellenamos las variables de este script y asignamos la función "Escanear" a nuestro **UI.Button** tendremos todo listo para probar nuestro ejemplo.

Si todo ha ido bien el resultado debería ser parecido a esto:



5

ARKIT

5.1 INTRODUCCIÓN

La característica fundamental que define a ARKit o a otras tecnologías similares, como ARCore, es la capacidad de crear y mantener una correspondencia entre el mundo real en el que se encuentra el usuario y un espacio virtual donde se pueden presentar objetos virtuales modelados en 3D. Cuando una aplicación muestra ese contenido junto con una imagen de la cámara en vivo, se aprecia el efecto de que el contenido virtual es parte del mundo real, que es lo que se conoce como realidad aumentada.

Estas tecnologías, no solo ARKit, tienen la capacidad de detectar en mayor o menor medida, superficies del mundo real, por lo que algunas personas la consideran Realidad Mixta, una vez más, la frontera se diluye y pasa a ser un tema de apreciación, como se comentaba en el capítulo 1 cuando se hablaba del "continuo de lo virtual".

ARKit se basa en una tecnología llamada *Visual Inertial Odometry*. Esta técnica combina información del hardware de detección de movimiento del dispositivo iOS con el análisis de la imagen capturada por la cámara del dispositivo, es decir, funciona buscando y rastreando puntos mediante la cámara mientras el dispositivo (iPhone o iPad) se mueve.

ARKit puede extraer planos, puntos de referencias e información de la luz de la escena. ARKit reconoce todas esas características de la imagen y comprueba las diferencias de posición de cada una en cada fotograma, y compara esa información con los datos del sensor de movimiento.

Conjuntamente con la información de estos elementos y con los datos del sensor de movimiento del dispositivo, puede mantener una aproximación de su posición en la estancia.

Para identificar puntos de referencia, debe haber suficiente información visual en la imagen recogida por la cámara y para ello lo más importante es el contraste. Cuanto más contraste hay en una imagen, más fácil es encontrar puntos. Por ejemplo, una pared blanca no tendrá puntos característicos, mientras que una malla blanca y negra tendrá múltiples puntos.

ARKit no tiene ninguna tecnología de detección de profundidad. No puede ocluir objetos, lo que significa que los objetos siempre se dibujan en la parte superior de la imagen de video. Los dispositivos de Apple no tienen sensores de detección de profundidad como los teléfonos Tango o las gafas *Hololens*. Ambos crean una malla espacial del entorno que se utiliza para la oclusión y detección de colisión para efectos realmente inmersivos. Este no es el caso de ARKit, por lo que en nuestras aplicaciones nos pueden pasar cosas como que coloquemos un modelo virtual encima de una mesa que tenga objetos que no hayan sido detectados de forma que veamos un objeto real encima de una mesa y que un objeto virtual se coloque encima. Esto dependerá siempre de dos cosas, la iluminación de la estancia y el contraste entre las distintas superficies.

Como ocurre en todos los sistemas que utilizan la tecnología VIO, como hace también, por ejemplo, ARCore de Google, el objetivo es rastrear una porción del mundo real donde se encuentra el usuario y establecer una correspondencia con un entorno virtual donde poder ubicar los elementos virtuales que necesitemos para lograr la experiencia de realidad aumentada. Debido a que ARKit también realiza un análisis de la luz disponible en la escena, podremos iluminar dichos elementos virtuales de forma que logremos una mayor integración y crear una mayor ilusión de que los elementos virtuales pertenecen al mundo real.

En cada sesión se define el sistema de coordenadas local, ya que la posición 0,0,0 es la posición de la cámara cuando se inicia la aplicación, este es distinto en cada sesión por lo que no es posible almacenar pociones de objetos ya que no se reconoce el mapeado espacial y no se puede ubicar el dispositivo dentro de la escena, aunque una vez posicionado si se puede conocer su posición gracias al seguimientos de las formas obtenidas mediante los puntos escaneados.

Las coordenadas del sistema virtual se utilizan siguiendo la convención de la mano derecha: el eje y apunta hacia arriba, el eje z apunta hacia el espectador y el eje x apunta hacia la derecha del espectador.

Para ver el funcionamiento en vivo de un sistema VIO, existe un excelente vídeo en el siguiente enlace: *https://www.youtube.com/watch?v=3R-YRucej6Q*

En este vídeo se puede ver como la cámara detecta varios puntos distintivos en la imagen y guarda una posición de inicio con la información del acelerómetro, según el usuario va avanzando y los puntos reconocidos se mueven, se calcula la nueva posición, de esta forma haciendo el seguimiento de los distintos puntos y de la información del acelerómetro se puede saber dónde está cada punto reconocido y, por lo tanto, obtener una correspondencia con un entorno virtual para posicionar objetos.

Este vídeo ha sido realizado por Simón Burkard, en su web *http://www.simonburkard.de* puedes encontrar varios artículos dedicados a este tema.

5.2 CONFIGURACIÓN PARA EL DESARROLLO DE APLICACIONES IOS

En primer lugar, es necesario disponer de un ordenador MAC. Desde Windows se pueden generar ejecutables para otros sistemas operativos, incluso paquetes *.dmg* de instalación para MAC OS, pero no es el caso de iOS, en este sentido, Apple tiene políticas muy estrictas y no permite utilizar el MAC OS instalado en PCs para construir ejecutables para iOS. También existen *assets* que permiten generar aplicaciones iOS desde Windows y diversos *hacks*, pero tarde o temprano habrá problemas, por ejemplo, si se quieren desplegar betas para el testeo de la aplicación o si se quiere publicar una aplicación en la *Apple Store*.

En cualquier caso, en este libro se contempla únicamente el proceso oficial, con un ordenador Apple, una cuenta de desarrollador y las aplicaciones oficiales de Apple.

Además, también es necesario disponer de una cuenta de desarrollador de Apple, con esta cuenta tendremos acceso a distintas herramientas para la revisión y publicación de aplicaciones en la tienda, el despliegue de betas a otros usuarios a través de la aplicación *TestFlight* y las betas de iOS o *XCode* para testear nuevas funcionalidades antes de que se pongan a disposición del público en general.

A diferencia de la cuenta de usuario de Apple, la de desarrollador es de pago, costando 99 € anuales y es absolutamente necesaria para poder compilar las aplicaciones ya que con ella se generan las firmas y los certificados que requieren las aplicaciones y ofrece todos los servicios de publicación.

Otra cosa necesaria para el desarrollo de aplicaciones con ARKit es un dispositivo iOS que sea compatible con esta tecnología, pero no valen todos los dispositivos, ya que no todos soportan ARKit, aunque soporten la versión de iOS necesaria.

La versión de ARKit que utilizaremos es la 1.5, aunque en el momento de La publicación de este libro la versión 2.0 se encuentra en fase beta, aún no es oficial.

En resumen, lo que necesitamos para poder comenzar a desarrollar aplicaciones con ARKit 1.5 es lo siguiente:

- ⇨ Dispositivo IPhone o iPad con iOS 11.3 o superior.
- ⇨ Ordenador MAC con MacOS *High Sierra* 10.13 o superior.
- ⇨ Unity 2017.1 o superior.
- ⇨ *XCode* 9.3 o superior.
- ⇨ La última versión del brazo *"spring2018_update"* del *plugin* Unity ARKit *Plugin* (más delante explicaremos que es esto y como obtenerlo.
- ⇨ Una cuenta de desarrollador de Apple.

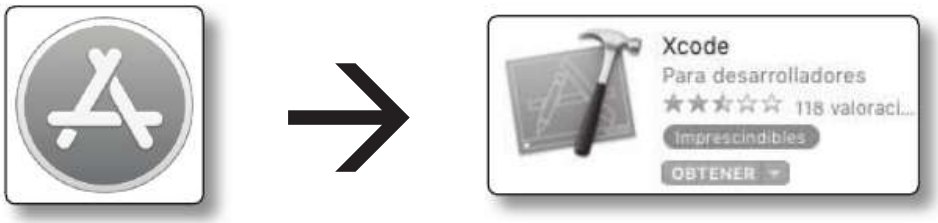
Si disponemos de todo esto, entonces ya estamos en condiciones de comenzar a hacer nuestra primera aplicación, pero antes, debemos configurar todo lo necesario.

5.2.1 Instalación del software necesario

Para crear una aplicación con ARKit necesitamos instalar la herramienta *XCode*, *XCode* es un IDE de Apple que nos permite compilar para este sistema operativo, cuando realizamos nuestro proyecto con Unity y decidimos crear la aplicación, al haber establecido que es una aplicación para iOS, Unity nos generará un proyecto para *XCode*, que abriremos con esta herramientas y compilaremos y ejecutaremos en el iPad o iPhone que tengamos conectados al MAC o enviaremos a la Apple Store para que sea distribuido como una beta a otros dispositivos o sometido al proceso de publicación.

Lo primero que debemos de hacer es comprobar si la versión del sistema operativo MAC OS es al menos la versión 11.3. La versión de iOS se puede ver haciendo clic en el icono de la manzana del menú superior y seleccionar la opción "Acerca de este Mac". Si no tenemos la versión 11.3 disponible, deberemos actualizarla mediante el botón Actualización de software

Una vez comprobada la versión de MAC OS y actualizado este, si fuese necesario, debemos instalar *XCode*. Para instalar *XCode*, hay que ir a la App Store con la aplicación que hay instalada en el MAC, buscarlo y hacer clic en el botón obtener. *XCode* no tiene ningún coste adicional.



El dispositivo con el que vamos a probar las aplicaciones y por supuesto, todos en los que las aplicaciones vayan a ser testeadas o utilizadas, también tienen que estar actualizados a la versión 11 de iOS.

> ⓘ **AVISO**
>
> La versión 12 de iOS se encuentra, en el momento de publicación de este libro, en fase Beta, por lo que solo será posible su instalación en dispositivos en los que el ID de Apple se corresponda con una cuenta de desarrollador, ya que una de las ventajas de este tipo de cuenta, es el acceso a las betas disponibles.
> Esta versión, la 12, es la que incluye la versión 2, también en beta de ARKit.

También debemos asegurarnos de que la versión de iOS es correcta. La versión de iOS se puede ver en la herramienta de configuración del dispositivo, en la opción **Ajustes → Actualización de software**, donde nos mostrará la versión de iOS que hay instalada y en caso de que haya actualizaciones disponibles, también las mostrará y desde aquí tendremos la posibilidad de instalarlas.

Otra cosa que debemos hacer, es tener la cuenta de desarrollador de Apple debidamente configurada, con al menos, los certificados de desarrollo generados y los APP ID creados.

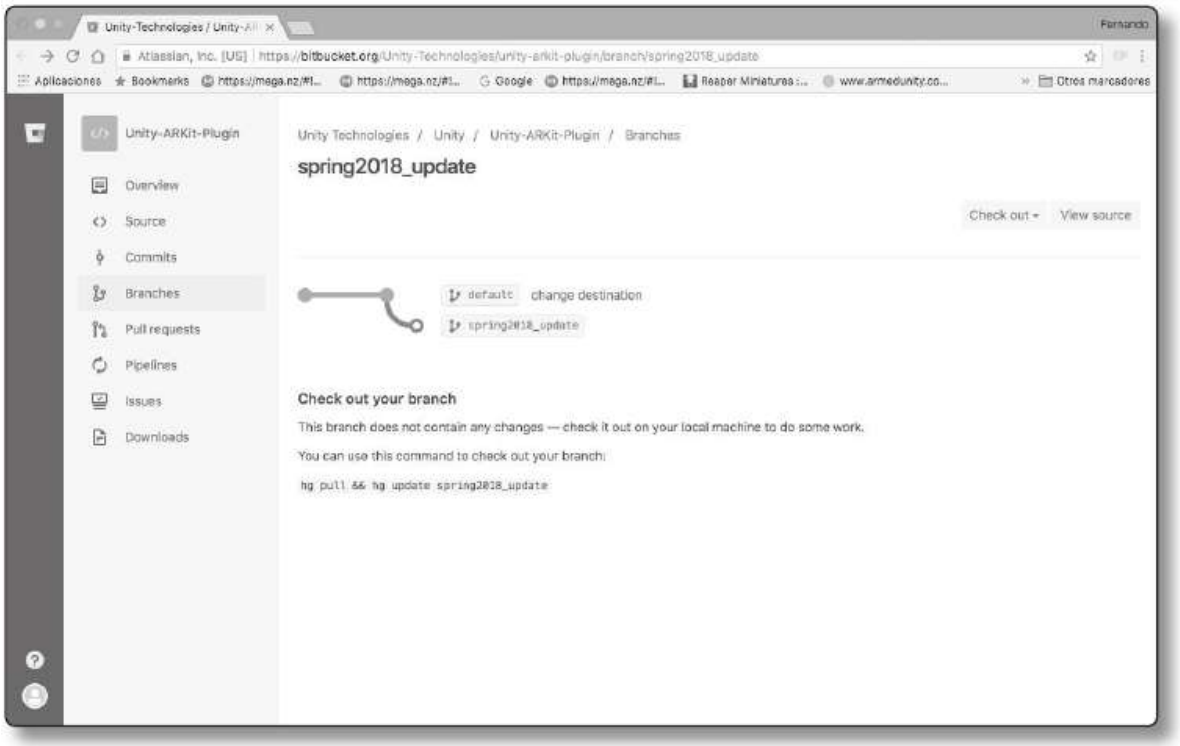
5.2.1.1 OBTENCIÓN UNITY ARKIT PLUGIN

Lo primero que hay que hacer para que una aplicación en Unity utilice el ARKit, es instalar el Unity ARKit *Plugin*.

El Unity ARKit *Plugin* permite manejar la librería incluida en iOS desde Unity y se puede obtener de dos formas: desde el repositorio de *Bitbuket* o desde la *Asset Store*.

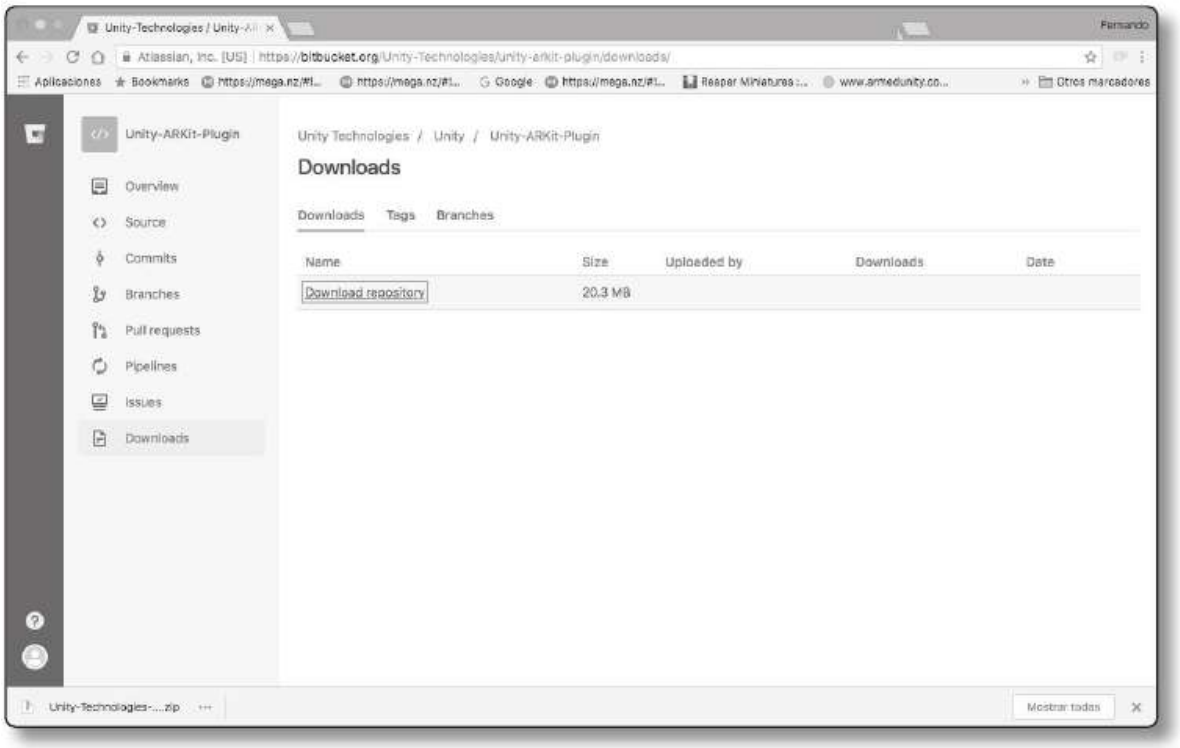
Para obtenerlo desde el repositorio de *Bitbucket* se debe acceder a la dirección *https://bitbucket.org/Unity-Technologies/unity-arkit-plugin/branch/spring2018_update*.

Si observamos la imagen, veremos que no estamos en la raíz del repositorio, sino en una rama completa, esto es porque esta es la rama correcta para que nuestra aplicación sea compatible con la versión 1.5 de ARKit, pero el plugin completo también incluye otras cosas, por ejemplos las librerías para trabajar con la beta de ARKit 2.0.

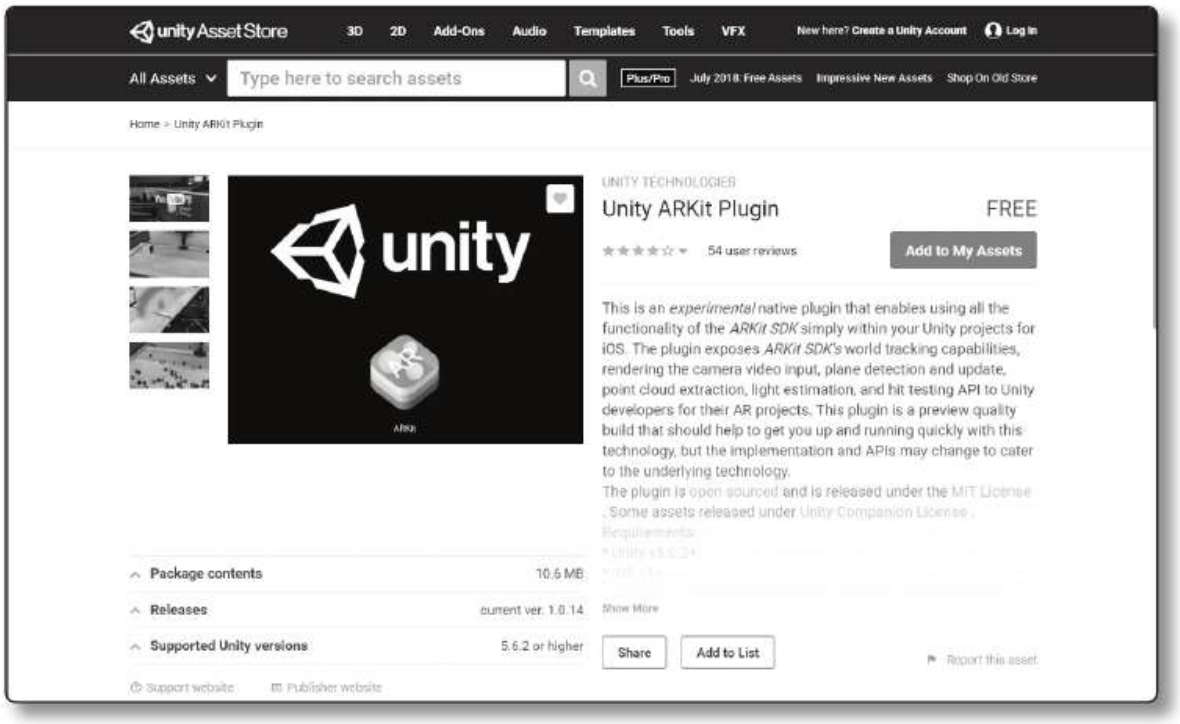


Para bajar el Plugin desde *Bitbucket*, solo tenemos que seleccionar la opción "*Download*" en el menú de la izquierda y en la página siguiente seleccionar "*Download repository*".

Esto nos descargará un archivo comprimido en ZIP, que cuando lo descomprimamos, contendrá la misma estructura de carpetas que un proyecto de Unity.



Hay una segunda forma de instalar el plugin para desarrollar aplicaciones con ARKit y es adquirirlo e instalarlo desde la *Asset Store*. El plugin se obtiene como cualquier otro *Asset* y es gratuito y desarrollado por Unity, solo hay que buscarlo por el nombre "*ARKit Plugin*" y nos aparecerá el primero.



Una vez que hayamos importado el *plugin*, veremos una determinada estructura en la jerarquía, que puede variar dependiendo del plugin importado, si hemos importado la rama de ARKit 1.5 o el Plugin completo. Para los ejemplos de este libro, hemos utilizado el *Plugin* de la *Asset Store*.

5.2.1.2 CONFIGURACIÓN DE UNITY ARKIT REMOTE

En primer lugar, vamos a ver una cosa que nos vendrá muy bien para desarrollar nuestras aplicaciones en ARKit, se trata del *ARKit Remote*.

ARKit Remote es una aplicación que ha desarrollado Unity y que forma parte del *ARKit Plugin* que nos evita en gran parte un trabajo bastante tedioso que solemos realizar cuando programamos con este tipo de tecnologías.

> **TRUCO**
>
> Los programadores que están acostumbrados a utilizar el sistema operativo Windows, a veces no se desenvuelven bien con MAC OS, y muchas veces hacen todo lo relativo a Unity en un ordenador con Windows y después de generar el proyecto en *XCode*, se copia este al ordenador MAC y se hace la compilación en él, de este modo el Mac solo se usa para construir el ejecutable
>
> En este caso, esto no serviría, ya que ARKit es una tecnología exclusiva de iOS, por lo que se necesitará un dispositivo iPhone o iPad para que funcione, con la consiguiente compilación desde un MAC.
>
> No obstante, Unity nos ha regalado una aplicación para poder utilizar remotamente las capacidades de ARKit suministradas por un iPhone o un iPad desde el escritorio.

Dado que ARKit es una tecnología propia del sistema operativo iOS, no funcionará por sí misma en otro sistema operativo, lo que quiere decir que si estamos programando con Unity en un MAC o en un PC no podremos probar la aplicación, sino que tendremos que compilar, generar el proyecto en *XCode*, pasarlo al MAC en el caso de que trabajemos con un PC, compilar la aplicación y ejecutarla en el iPad o en el iPhone y ver si funciona o no y vuelta a empezar cada vez que queramos realizar un testeo, lo que nos obligará a hacer multitud de compilaciones.

Esto, como podréis imaginar, el algo muy tedioso y que produce una gran pérdida de tiempo para pruebas muy simples y sobre todo cuando estamos realizando ajustes que debemos ejecutar la aplicación muchas veces muy seguidas en poco tiempo.

Para evitar esto existe el *ARKit Remote*, que consiste en un sistema dividido en dos partes, por una parte, una aplicación llamada *UnityARKitRemote* que se ejecuta en el dispositivo iOS y por otra, un *GameObject* llamado *ARKitRemoteConnection* que se incluye en la aplicación Unity.

De este modo, el dispositivo iOS envía la imagen de la cámara, información acerca de su movimiento, posición y rotación, de la nube de puntos que recoge y la información de los planos que va detectando o eliminando.

Esta información es recogida por un *GameObject*, que veremos cómo utilizar más adelante.

La aplicación *UnityARKitRemote* viene en el *Plugin*, por lo que tenemos que compilarla, para ello tenemos que crear su correspondiente *AppID* en el portal de desarrollo de Apple.

Una vez que tenemos creado el *AppID*, e importado el *ARKit Plugin* en Unity, abrimos la escena que tendremos en la carpeta **Assets\UnityARKitPlugin\ ARKitRemote** llamada **UnityARKitRemote.unity** (tendremos otra escena llamada **EditorTest**).

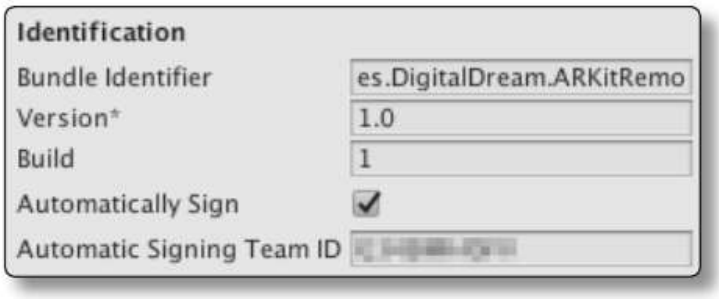
Una vez que tengamos la escena abierta, debemos ir al menú **File → Build Settings** y asegurarnos de lo siguiente:

- ⇨ En el apartado *Scenes to Build* solo debe encontrarse marcada la escena **UnityARKitPlugin/ARKitRemote/UnityARKitRemote**, puede haber otras, pero deben desmarcarse, ya que lo que pretendemos es hacer una aplicación solo de esta escena.

- ⇨ Marcar iOS como plataforma y hacer clic en el botón *Switch Platform*.

⇨ En la opción *Run XCode as* establecerla a *Release*

Una vez realizados estos dos pasos, hacer clic en el botón *Player Settings* y en el apartado *Identification* cumplimentar el *Bundle Identifier* con el mismo nombre que le hemos dado al *App ID* en el portal de desarrollo.

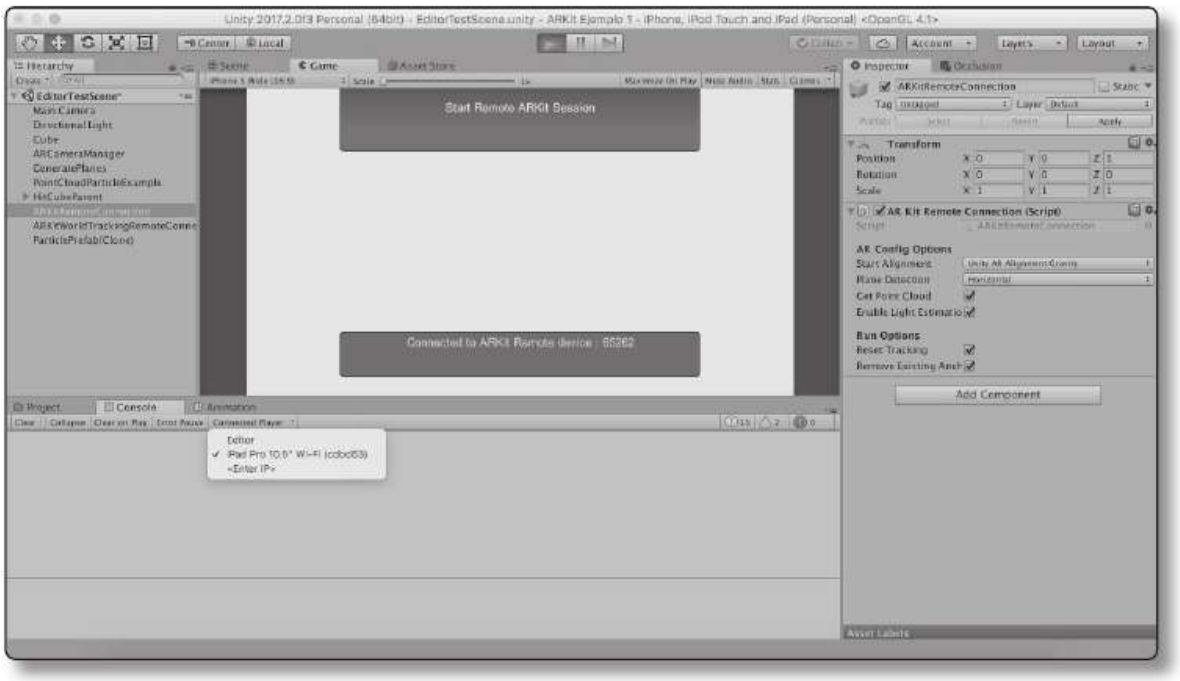


Una vez hecho esto junto con las cosas habituales a la hora de compilar una aplicación para iOS: icono, Nombre, versión, *build*, *Team ID*, etc. compilamos nuestra aplicación y la ejecutamos en el dispositivo.

Cuando ejecutemos la aplicación en el dispositivo, la pantalla se quedará en negro, mostrando el mensaje "*Waiting for editor connection..*"

Apartir de aquí, ya estaremos en condiciones de hacer funcionar una aplicación con tecnología ARKit desde el editor de Unity. Para ello, en la escena con la que estemos trabajando, debemos incluir un *Prefab* llamado **ARKitRemoteConnection** que se encuentra en la carpeta **Assets\ UnityARKitPlugin\ARKitRemote**.

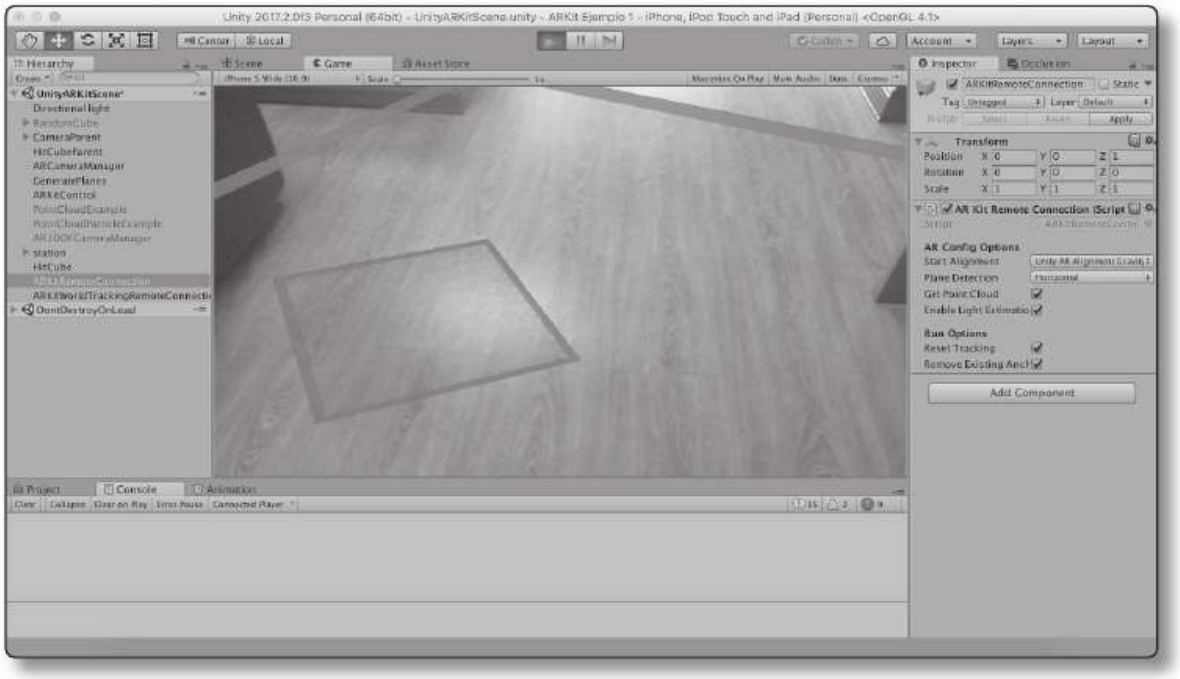
Cuando ejecutemos la aplicación nos aparecerá una pantalla como la que se muestra a continuación:



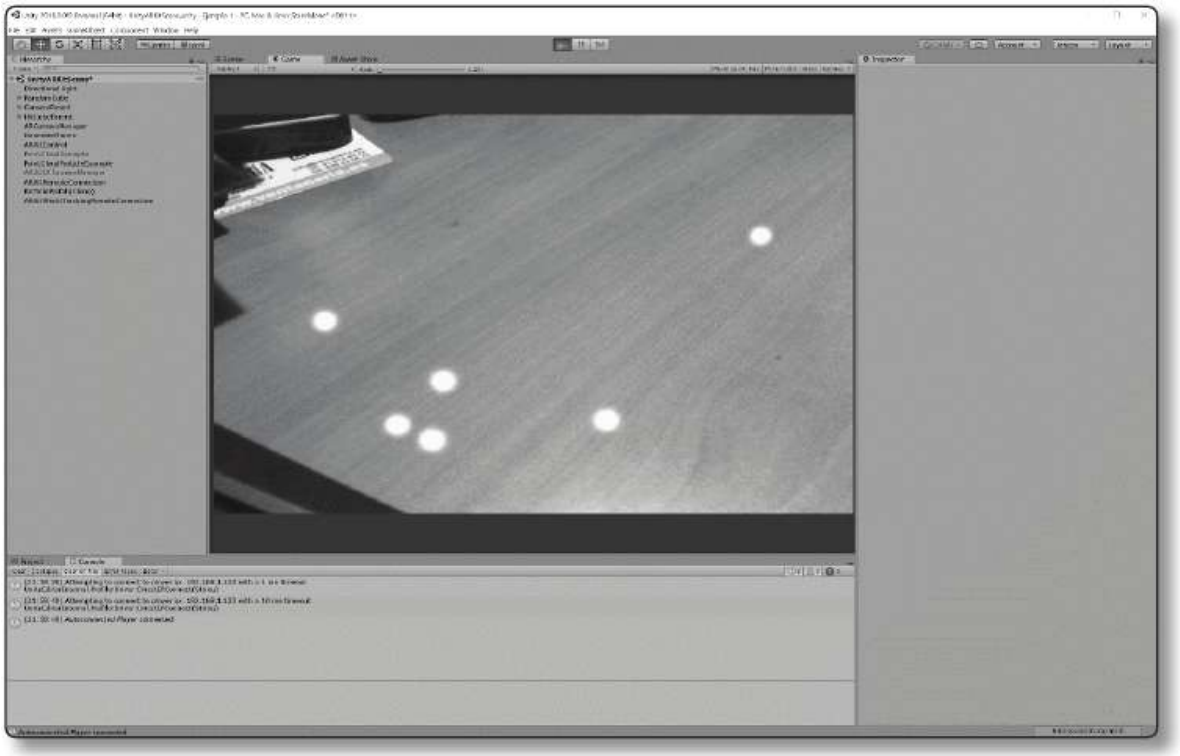
Desde esta pantalla debemos conectar con el dispositivo iPhone o iPad antes de proseguir para que el editor de Unity tenga disponible toda la información proveniente de ARKit.

Para ello debemos acceder al panel *Console* y veremos que nos ha aparecido una opción menú nueva llamada *Connected Player*, y si la desplegamos podemos ver el nombre del dispositivo que tengamos conectado por cable, como en el caso de la fotografía anterior y una opción llamada <*Enter IP*> que nos permite conectar mediante la red introduciendo la dirección IP del iPhone o iPad que queramos utilizar como suministrador de la información de ARKit y que deberá tener corriendo la aplicación Unity *ARKitRemote*.

Una vez realizada la conexión, la aplicación ejecutada dentro de nuestro editor de Unity, se comportará como si se ejecutase en un dispositivo iOS, como se puede ver en la siguiente imagen, donde aparece la imagen de la cámara capturada por un iPad y el resultado de una detección de planos realizado por ARKit a partir de una nube de puntos:



Esto es aún más interesante cuando el desarrollador está habituado a trabajar con Windows, ya que como se puede apreciar en la imagen de abajo también puede realizarse con este sistema operativo.



El único problema, es que no es, ni mucho menos, tan fluido como cuando se hace con un ordenador MAC. Comparando la ejecución del ejemplo anterior en un MAC con la ejecución en PC con características ligeramente superiores, el rendimiento es significativamente menor, aunque con un PC actual medio-alto es posible trabajar con comodidad con este sistema.

5.3 ESTRUCTURA DE UNA APLICACIÓN BÁSICA CON ARKIT

Con todo instalado y configurado, vamos a proceder a ver cómo está construida una aplicación con ARKit utilizando el Unity *ARKit Plugin*, el primer paso es crear un nuevo proyecto con Unity e indicarle que la plataforma de destino es iOS.

Para ello accederemos a la opción de menú **File → Build Settings** y en el apartado *Platform* seleccionaremos iOS y a continuación haremos clic en el botón *Switch Platform*, con esto, cuando queramos compilar la aplicación, Unity generará un proyecto *XCode* para construir la aplicación en iOS.

Como hemos visto, también tenemos que configurar todo lo relativo a ID de aplicaciones, nombre de paquetes, obtener los certificados, etc. tal y como establece Apple para realizar una compilación.

A continuación, debemos importar el plugin de ARKit y si queremos trabajar desde el PC o el MAC, construir la herramienta *Unity ARKit Remote*. En este ejemplo no la utilizaremos para simplificar y para que no se vean más objetos de los necesarios.

Para realizar una aplicación con la tecnología de ARKit, necesitamos una serie de elementos básicos, el Plugin de ARKit, incluye dentro de sus ejemplos, una escena básica que es un excelente punto de partida para comenzar a construir una aplicación.

5.3.1 Objetos básicos que componen una escena

Vamos a abrir esta escena que se encuentra en **Assets\ UnityARKitPlugin\ Examples\UnityARKitScene** y que se llama, igual que la carpeta que la contiene, **UnityARKitScene**.

Esta escena es un ejemplo que escanea la superficie que enfoquemos con la cámara, marca los puntos detectado, busca planos y nos permite colocar un objeto en una superficie, en este caso un cubo, pero independientemente de lo que haga, vamos

a estudiar los componentes que lleva y vamos a ver para que sirven y como utilizar esta escena o una que nos construyamos a partir de ella para utilizarla como base de cualquier escena que necesitemos.

Solo se detallan los objetos importantes, ya que hay otros que son parte del ejemplo y no tienen relevancia para nuestro propósito que es comprender que objetos son necesarios y que función tiene cada uno para poder realizar una aplicación con la tecnología de ARKit.

Una vez abierta la escena, veremos los siguientes objetos en la jerarquía:

5.3.1.1 CAMARA PARENT CON EL OBJETO HIJO MAIN CAMERA

Esta es la cámara que hace la función de cámara de los objetos 3D del mundo virtual, pero que a la vez recoge la entrada de vídeo, superponiendo el mundo real y virtual y convirtiéndose por tanto en una cámara AR. Esta es la cámara que debemos utilizar en cualquier proyecto con ARKit y sustituye a la cámara tradicional de Unity. Contiene dos scripts:

- **UnityARVideo**:

  Este *script* se encarga de establecer los *Cliping Planes* y de limpiar el *background* de la cámara, también hace que utilice como fondo de renderizado un material con un *shader* especialmente diseñado para iOS que presenta el video capturado.

- **UnityARCameraNearFar**:

  Este *script* es un componente intermedio que comunica la cámara con *ARSession*, una clase que maneja los principales procesos de ARKit recopila y convierte datos recogidos del mundo real. Si los *Cliping Planes* se cambian por alguna razón, esta clase comunica estos cambios a *ARSession*.

5.3.1.2 AR CAMERA MANAGER

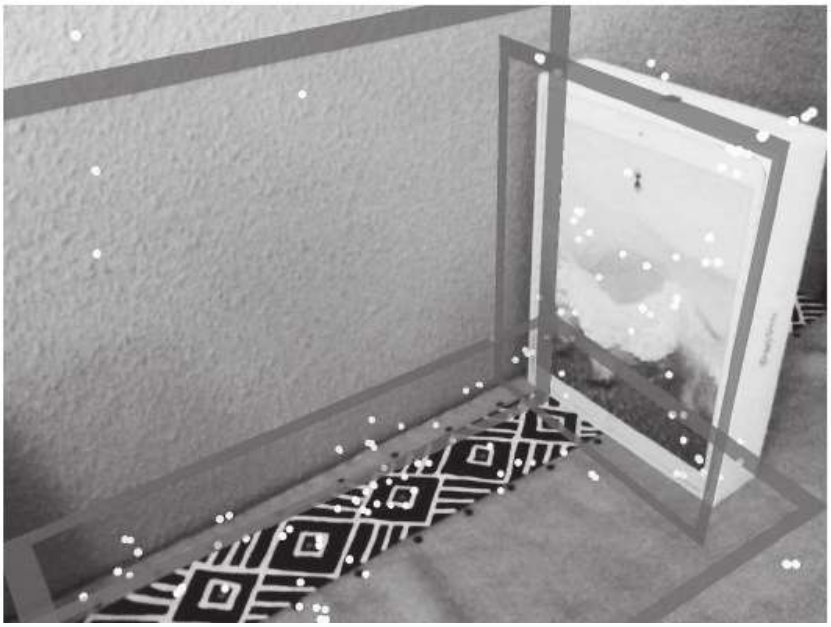
Contiene un script llamado **UnityARCameraManager** que referencia a la cámara anterior y gestiona distintos aspectos de esta como alineación, la orientación de los planos a detectar, la información de la nube de puntos, la estimación de la luz, etc.

Mediante su panel de configuración podemos establecer el comportamiento de ARKit en la aplicación:

⇨ *Start Alignment*: Establece como se orienta el mundo 3D con respecto al mundo real, existen tres opciones:

- *Unity AR Alignment Gravity*: El eje Y se alinea en horizontal con respecto al suelo y su origen es la posición inicial del dispositivo.
- *Unity AR Alignment Gravity and Heading:* El eje Y se alinea en horizontal con respecto al suelo y los ejes X y Z se alinean con respecto a la brújula y su origen es la posición inicial del dispositivo.
- *Unity AR Alignment Camera*: El sistema de coordenadas coincide con la orientación de la cámara según el sistema de la mano derecha, mencionado al inicio del capítulo.

⇨ *Plane detection*: Establece que tipo de planos serán detectados, en la imagen se pueden ver planos horizontales y verticales:



- Horizontal: ARKit únicamente detectará planos horizontales.
- Vertical: ARKit únicamente detectará planos Verticales.
- Horizontal and Vertical: ARKit detectará ambos tipos de planos.

⇨ *Detection Images*: Permite incluir un conjunto de imágenes que serán detectadas por ARKit como si fuesen marcadores, Mas adelante veremos un ejemplo de esto.

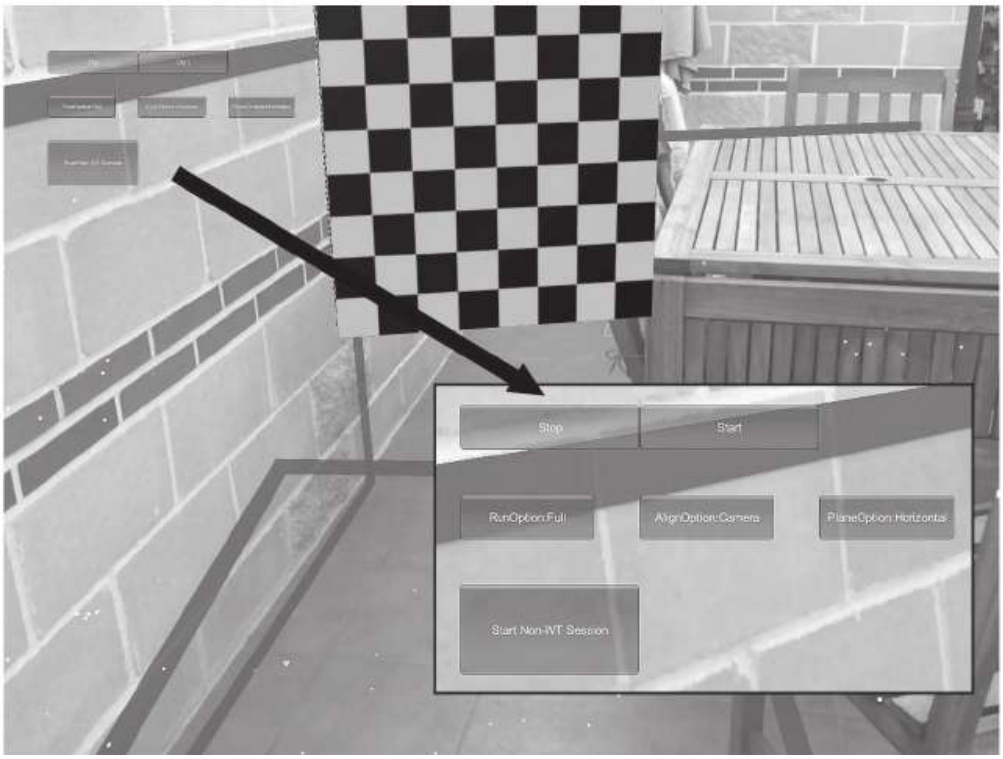
⇨ *Get Point Cloud*: Establece si se recoge información de la nube de puntos.

⇨ *Enable Light Estimation*: Activa la estimación de la luz de la cámara para aplicársela a los modelos 3D.

⇨ *Enable Auto Focus*: Activa el autofocus de la cámara.

5.3.1.3 ARKIT CONTROL

El objeto ARKit Control es un objeto vacío que tiene asociado un *script* llamado **UnityARKitControl** que inicialmente se encuentra desactivado. Este *script* se utiliza para depurar y permite activar y desactivas distintas características de la configuración de ARKit mediante la presentación de una interfaz de usuario en pantalla, nos permite cosas como ver y cambiar la alineación de la cámara o modificar la detección de planos. Si se activa se mostrarán unos controles como en la siguiente imagen:



5.3.1.4 GENERATE PLANES

El objeto **Generate Planes** es, como en el caso anterior, un objeto vacío que contiene un script llamado **UnityARKitGeneratePlanes**, a este script se le asigna un *Prefab* que se utiliza para superponer en la imagen de la cámara el plano que ARKit ha detectado.

En la imagen anterior se pueden observar los recuadros que son el resultado de la detección de los distintos planos, estos recuadros han sido colocados en la escena por este objeto.

5.3.1.5 POINT CLOUD EXAMPLE Y POINT CLOUD PARTICLES EXAMPLE

Estos dos objetos se utilizan con motivo de depuración para testear el reconocimiento de puntos, fundamentalmente para ver la calidad de patrones en imágenes o en distintas condiciones de luz y lo habitual es que en la aplicación final no se muestren,

Los dos objetos tienen un *script* que realiza una función similar, la diferencia es que el script **UnityPointCloudExample** se le asigna un *Prefab* con un objeto que colocará cuando detecte un punto de la nube y se le asigna un número máximo de puntos a representar con objetos, mientras que al *script* **UnityPointCloudParticlesExample** se le asigna un sistema de partículas para indicar los puntos y el tamaño de las mismas y al igual que en el caso anterior un tamaño máximo de puntos a representar. La diferencia es únicamente visual.

En la imagen anterior se pueden apreciar varios puntos sobre la imagen que son representados por el *script* **UnityPointCloudParticlesExample** que es el que estaba activo en la compilación de la aplicación a la que pertenece esa captura.

Sin embargo, en la imagen anterior a esta última, donde se ilustra las capturas de planos horizontales y verticales, se pueden apreciar unos puntos más gruesos que no son partículas, sino objetos y que son representados por el *script* **UnityPointCloudExample**.

Estos son los objetos básicos que intervienen en cualquier escena de una aplicación que use la tecnología de ARKit, el resto de objetos realizan las funciones específicas de la aplicación.

A continuación, como ya conocemos en qué consisten todos estos objetos, vamos a realizar una aplicación con una funcionalidad específica, creando y asignando todos estos objetos desde el principio, para posteriormente añadir otras funcionalidades que nos proporciona ARKit.

5.4 CREACIÓN DE UNA APLICACIÓN CON ARKIT

En primer lugar, ya tenemos claro lo que debemos hacer, construir un proyecto, importar el *ARKit Plugin*, configurar la cuenta de desarrollo de Apple, crear una nueva escena, añadirla a la lista de escenas en **Build Settings** de Unity, hacer el *Switch Platform* a iOs, establecer las opciones del *Player Settings*, el *ARKit Remote* para el desarrollo en el editor, etc. lo que vienen a ser los pasos habituales en todas las aplicaciones de este tipo.

5.4.1 Configuración de la cámara

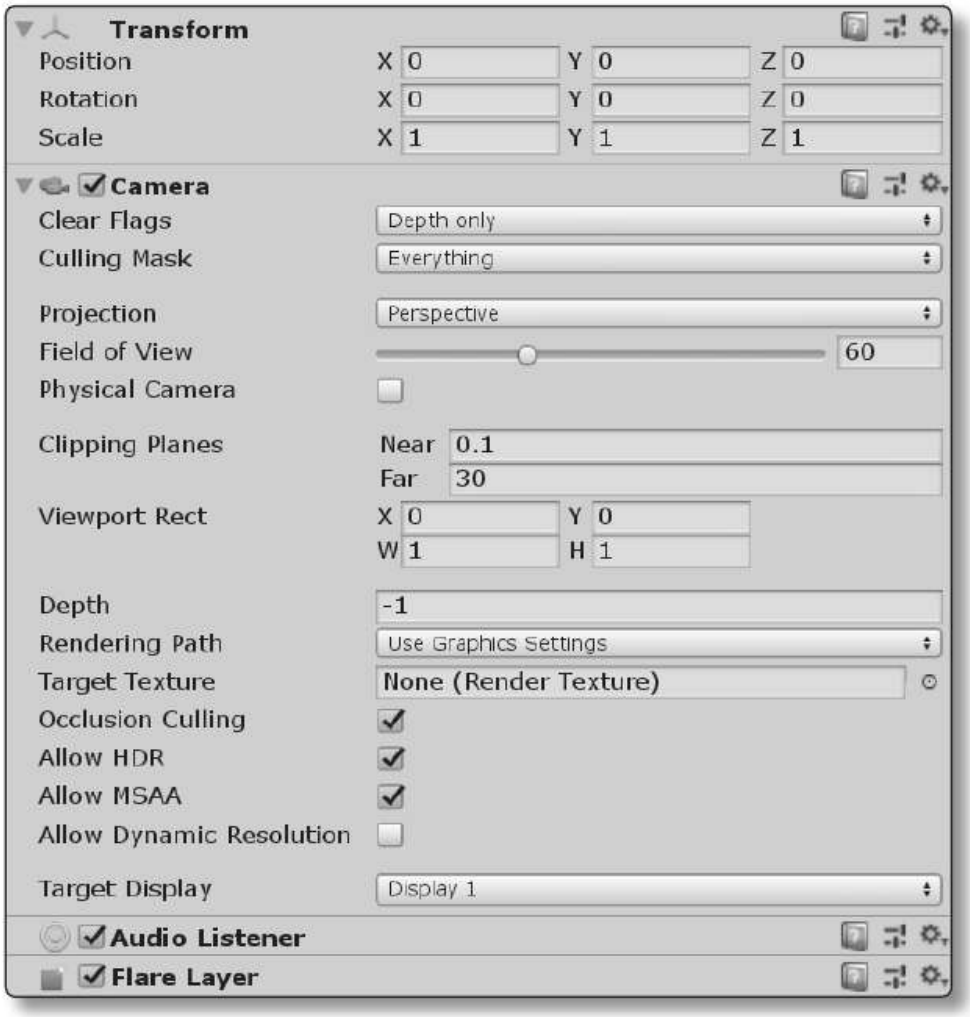
A continuación, creamos la cámara, a diferencia de otros SDKs, ArKit prácticamente no tiene *Prefabs*, así que lo que sería habitual, que es arrastrar un *Prefab*, no es o que se hace en este caso. El método de trabajo en ARKit, como ya hemos visto en el ejemplo, es crear un objeto vacío y asignarle scripts u otros objetos dependientes, ya que parte de su funcionamiento se basa en la jerarquía.

Por lo tanto, vamos a crear un objeto vacío, llamado *Camera Parent* y en su interior, otro objeto, de tipo *Camera*, llamado **Main Camera**.

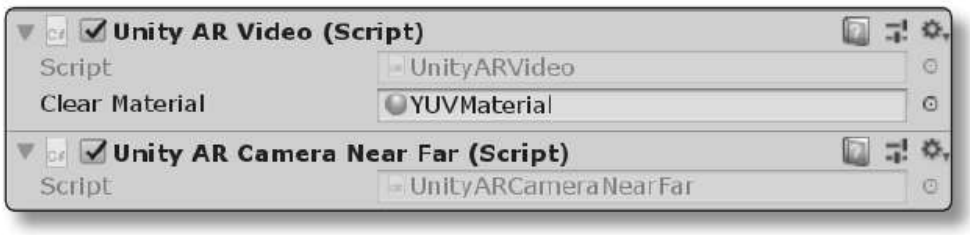
La opción *Clear Flags* de la cámara debe establecerse a *Depth Only* y el *Transform* debe posicionarse en las coordenadas 0,0,0. Esto es porque el mundo virtual siempre se va a crear a partir de la cámara y el punto de origen de la misma será el (0,0,0).

La siguiente opción a ajustar son los *Cliping Planes*, que definen, en el sistema de unidades de Unity, desde que distancia (*Near*) y hasta que distancia (*Far*) se muestran los objetos y fuera de ese rango son recortados, por defecto en todas las escenas de ejemplo de ARKit se utiliza 0,1 como valor del parámetro *Near* y 30 como valor del parámetro *Far* dado que son los valores más adecuados para una correcta optimización del hardware, pero pueden ser cambiados si necesita visualizarse más lejos, la cercanía es difícil que se necesite visualizar a menos de 0,1 m, en este tipo de aplicaciones, aunque también puede cambiarse.

Por tanto, la configuración de la cámara quedaría de la siguiente forma:



A continuación, añadimos los *scripts* de cámara, que como vimos con anterioridad son dos: el **UnityARVideo**, al que se asigna el material **YUVMaterial** y el **UnityARCameraNearFar**.



Lo siguiente es añadir el Objeto **Camera Manager** con el *script* **UnityARCameraManager** y asignarle la cámara **Main Camera** incluido dentro del objeto *Camera Parent* y establecer el comportamiento de la alineación, luz, planos, etc.

Con esto ya tenemos configurada la cámara y podemos proceder a los siguientes elementos:

5.4.2 Planos y nube de puntos

Si queremos que nuestra aplicación muestre los planos cuando los detecte, solo tenemos que añadir el objeto vacío *Generate Planes* y asignarle el *script* **UnityARGeneratePlanes** y asignarle el *Prefab* con el *GameObject* que se utilizará para dibujar los planos en la escena.

Si lo que pretendemos es que se muestren los puntos detectados según escaneamos el mundo real con la aplicación, crearemos un objeto vacío llamado *Point Cloud* o *Point Cloud Particle* y le asignaremos el *script* **UnityPointCloudExample** o bien **PointloudParticleExample** y configuraremos los detalles del número de puntos a representar. En el caso de las partículas el sistema y el tamaño de las mismas y en el caso de los puntos el objeto que los representa.

5.4.3 Depuración

Y como ya vimos, si necesitamos funcionalidad para la depuración en tiempo de ejecución de la aplicación, podemos añadir un objeto con el *script* **UnityARKitControl** para que nos permita cambiar algunos aspectos del funcionamiento de la cámara AR.

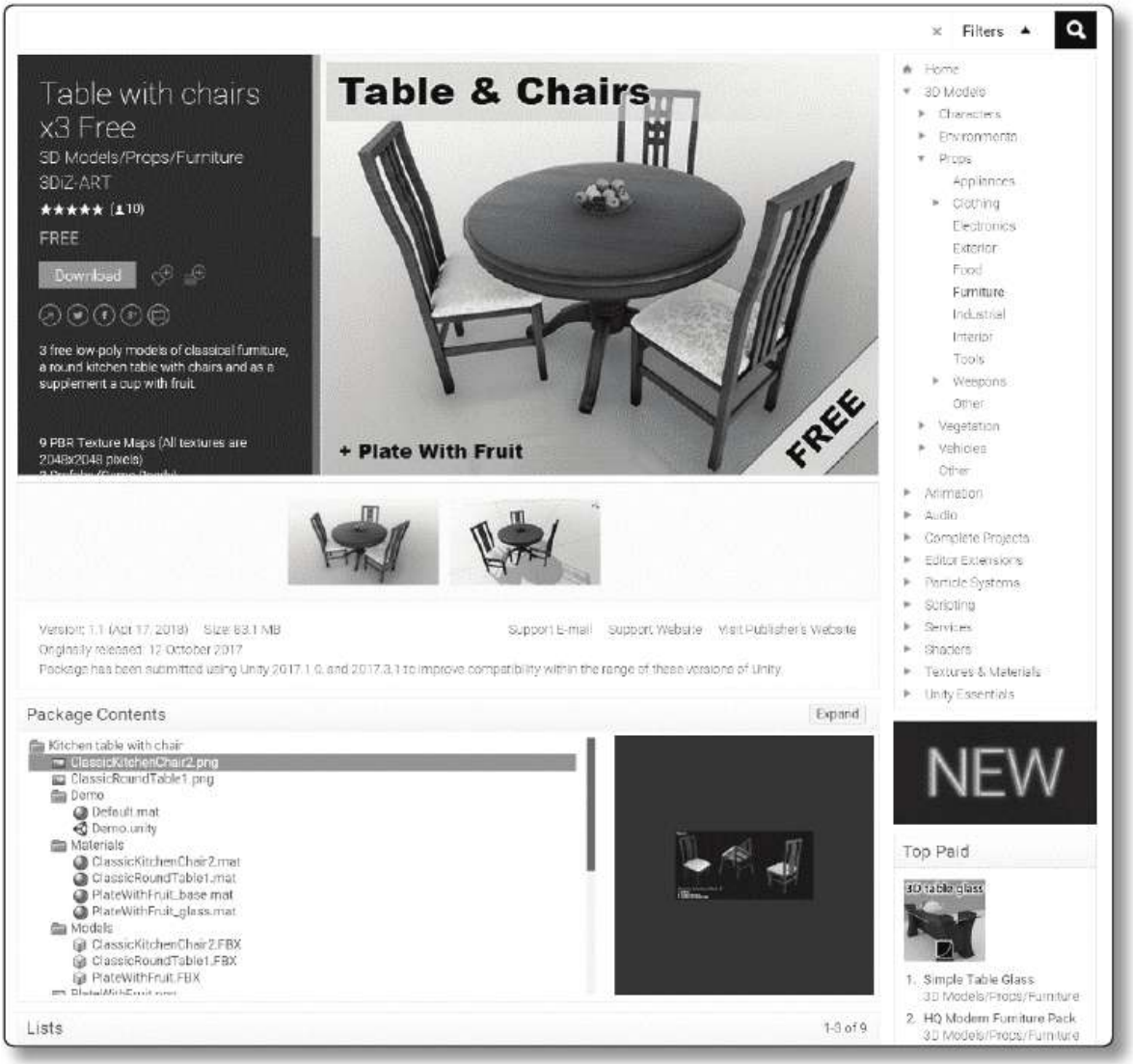
> **AVISO**
>
> Todos estos scripts que hemos visto, así como otros scripts y *Prefabs* utilizados por estos para el funcionamiento correcto del plugin y de la comunicación con ARKit, se encuentran en la carpeta **Assets\Plugins\iOS** y sus subcarpetas.
>
> Si en un momento dado, decides hacer limpieza y eliminar cosas como los ejemplos de utilización del Plugin, debes dejar intacta esa carpeta ya que, sin ellas, no puede funcionar la aplicación con ARKit.

5.4.4 Dando funcionalidad a la aplicación

Hasta aquí, lo que hemos hecho es replicar desde cero el ejemplo de una escena básica que viene incluido con ARKit, este ejemplo colocaba un cubo en un plano previamente detectado.

Lo que nosotros vamos a hacer es colocar otro objeto más realista para posteriormente aplicarle otras funcionalidades, como la generación de sombras y, además, permitiremos que el usuario realice ciertas acciones sobre el objeto como moverlo, rotarlo o escalarlo y que de este modo quede más integrado.

El objeto que hemos escogido para el ejemplo es un modelo en 3D gratuito de la *Asset Store* llamado "*Table with chairs X3 Free*" de 3DiZ-ART.



Una vez importado y descargado el modelo Vamos a crear un objeto vacío llamado Mesa y dentro de él, vamos a crear otro llamado Plano de sombra.

En este objeto, que va a contener el plano donde se deposite nuestra mesa y sillas y el plano con su correspondiente *Collider*, donde se deberán mantener, va a llevar asociado un *script* llamado **ColocacionMesa**, este script es prácticamente idéntico al del ejemplo de ARKit de la escena que nos permite colocar cubos en los planos detectados, pero lo que va a posicionar es la mesa que hemos importado.

Este *script*, que se encuentra en la carpeta **Assets** del Ejemplo 1 de ARKit, se explica con detalle en el documento PDF anexo al proyecto de ejemplo. Su objetivo es detectar cuando se toca la pantalla y si se toca en un plano previamente reconocido para colocar la mesa allí y tiene el siguiente código:

```
1  using System;
2  using System.Collections.Generic;
3  using UnityEngine.EventSystems;
4
5  namespace UnityEngine.XR.iOS
6  {
7      public class ColocacionMesa : MonoBehaviour
8      {
9          public Transform objetoTocado;
10         public float distanciaRayo = 30.0f;
11         public LayerMask capaColision = 1 << 10;  //ARKitPlane layer
12         private bool esDetectado;
13         bool HitTestWithResultType(ARPoint point, ARHitTestResultType resultTypes)
14         {
15             List<ARHitTestResult> resultadosToque =
16             UnityARSessionNativeInterface.GetARSessionNativeInterface().
17             HitTest(point, resultTypes);
18             if (resultadosToque.Count > 0)
19             {
20                 foreach (var resultadoToque in resultadosToque)
21                 {
22                     Debug.Log("Got hit!");
23                     objetoTocado.position = UnityARMatrixOps.GetPosition(resultadoToque.
   worldTransform);
24                     objetoTocado.rotation = UnityARMatrixOps.GetRotation(resultadoToque.
   worldTransform);
```

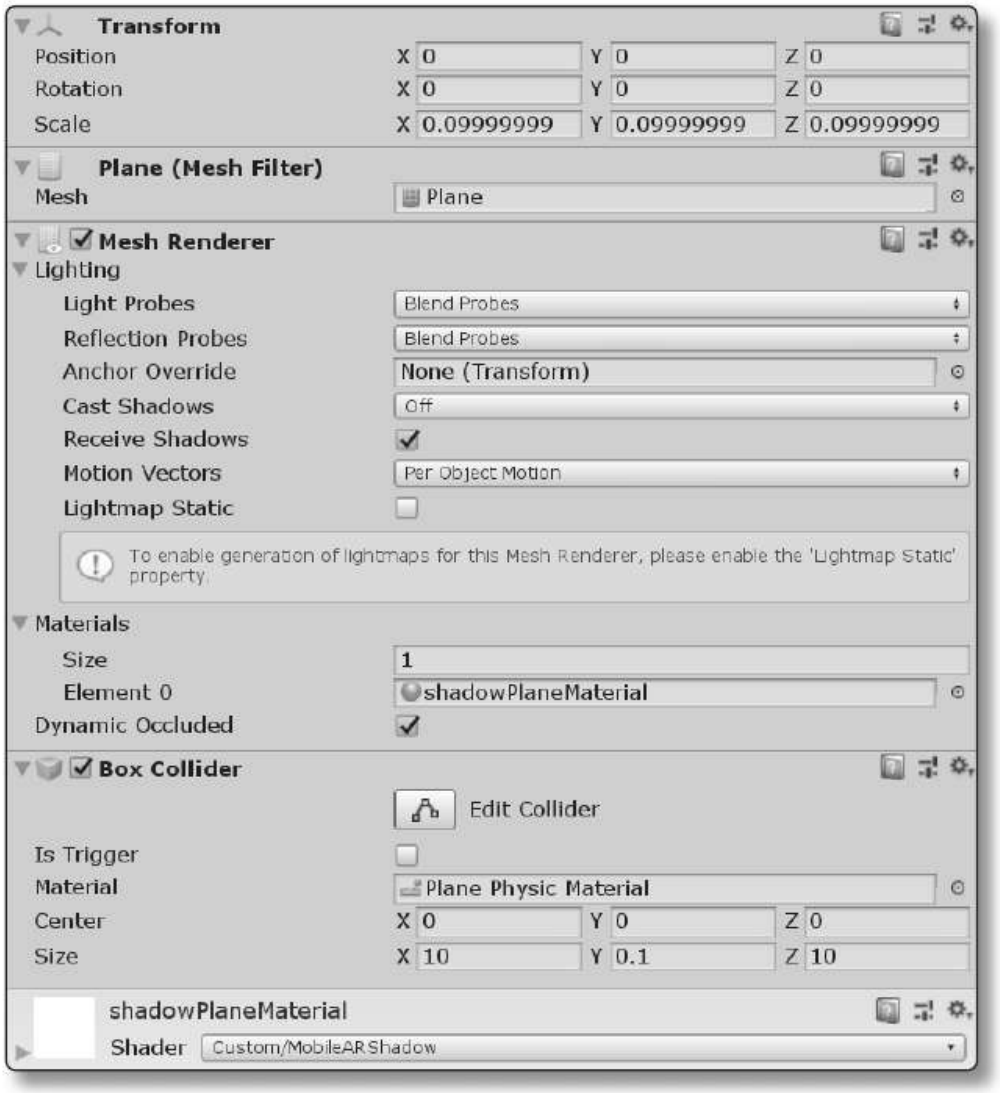
```
25                  Debug.Log(string.Format("x:{0:0.######} y:{1:0.######} z:{2:0.######}",
26                      objetoTocado.position.x, objetoTocado.position.y,
27                      objetoTocado.position.z));
28                  return true;
29              }
30          }
31          return false;
32      }
33
34      public void Start()
35      {
36          esDetectado = true;
37      }
38      public void DetectionOff()
39      {
40          esDetectado = false;
41      }
42      private bool IsPointerOverUIObject()
43      {
44      PointerEventData eventDataCurrentPosition = new PointerEventData(EventSystem.
    current);
45          eventDataCurrentPosition.position = new Vector2(Input.mousePosition.x,
46              Input.mousePosition.y);
47          List<RaycastResult> results = new List<RaycastResult>();
48          EventSystem.current.RaycastAll(eventDataCurrentPosition, results);
49          return results.Count > 0;
50      }
51      void Update()
52      {
53          if (Input.touchCount > 0 && objetoTocado != null)
54          {
55              var touch = Input.GetTouch(0);
56           if ((touch.phase == TouchPhase.Began || touch.phase == TouchPhase.Moved) &&
57                  esDetectado == true && !IsPointerOverUIObject())
58              {
59                  var screenPosition = Camera.main.ScreenToViewportPoint(touch.position);
60                  ARPoint point = new ARPoint
61                  {
```

```
62                  x = screenPosition.x,
63                  y = screenPosition.y
64              };
65              // prioritize reults types
66              ARHitTestResultType[] resultTypes = {
67                  ARHitTestResultType.ARHitTestResultTypeExistingPlaneUsingExtent,
68                  // if you want to use infinite planes use this:
69                  //ARHitTestResultType.ARHitTestResultTypeExistingPlane,
70                  ARHitTestResultType.ARHitTestResultTypeHorizontalPlane,
71                  ARHitTestResultType.ARHitTestResultTypeFeaturePoint
72              };
73              foreach (ARHitTestResultType resultType in resultTypes)
74              {
75                  if (HitTestWithResultType(point, resultType))
76                  {
77                      return;
78                  }
79              }
80          }
81      }
82  }
83 }
84
85
```

5.4.5 Aplicando sombras a los objetos virtuales

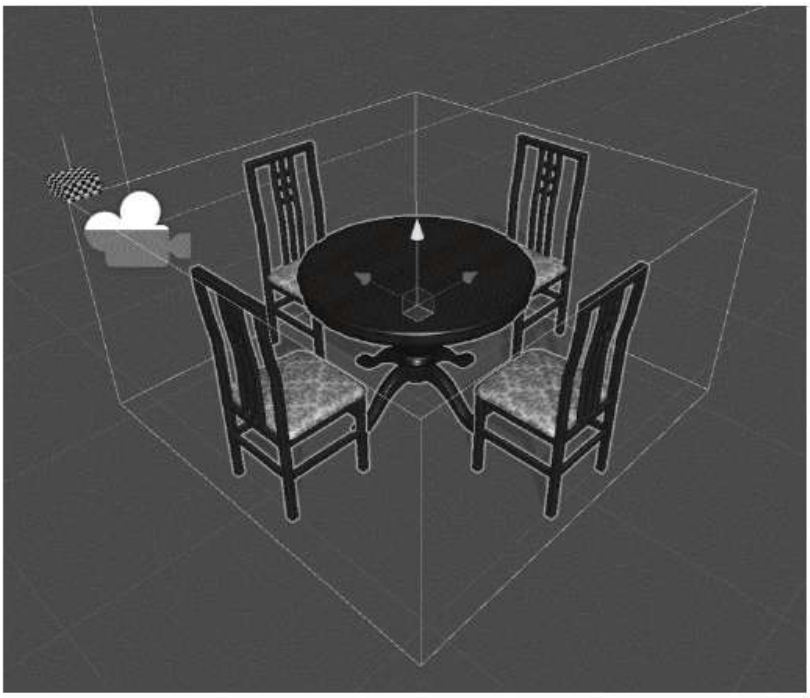
A continuación, vamos a crear un objeto vacío que se va a llamar "Plano" dentro del objeto "Plano de sombra" y que nos va a servir para dos cosas, para que las sombras proyectadas por la luz se reflejen en el suelo y para que exista un plano en el mundo virtual donde puedan reposar los objetos virtuales, esto nos será útil si queremos que los objetos virtuales tengan propiedades físicas, como gravedad.

Este objeto, necesita que añadamos un componente **MeshFilter** cuya malla es un plano y también necesita un componente **MeshRenderer** que tenga activada la opción *Receive Shadows* y además que tenga un material llamado *shadowPlaneMaterial* que se encuentra **Assets\UnityARKitExamples\Common\Materials**, también tiene que tener marcada la propiedad *Dynamic Occluded*.

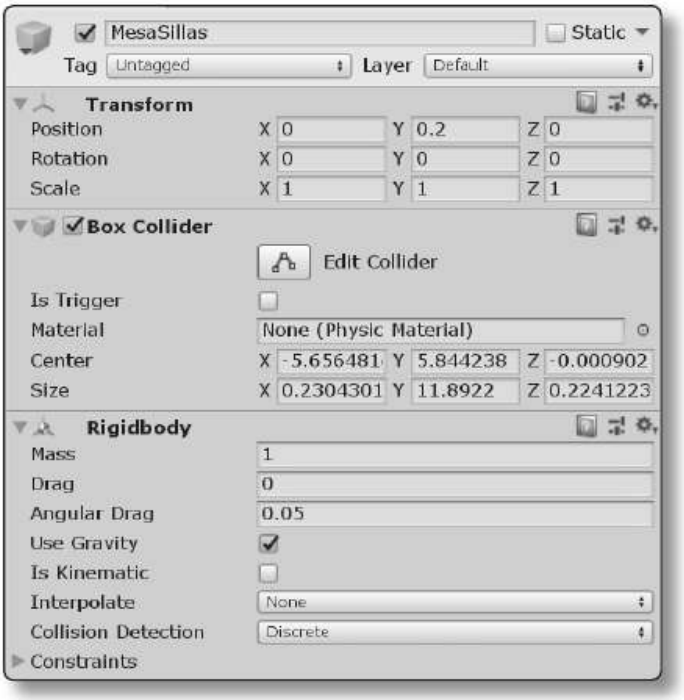


5.4.6 Objetos virtuales, movimiento y escalado

También crearemos un segundo objeto dentro de "Plano de sombras" llamado "Mesa Sillas", que contendrá los modelos 3D que vamos a utilizar en la escena, a este objeto que contendrá los modelos 3D le vamos a añadir un *BoxCollider* y un *RigidBody*, para que nos sea posible manipularlo y que tenga propiedades físicas y esté siempre tocando el plano que hace de suelo del mundo virtual por efecto de la gravedad y de este modo, la proyección de las sombras se realiza de forma adecuada. El objeto "Mesa" no debería quedar de forma similar al de la siguiente imagen:



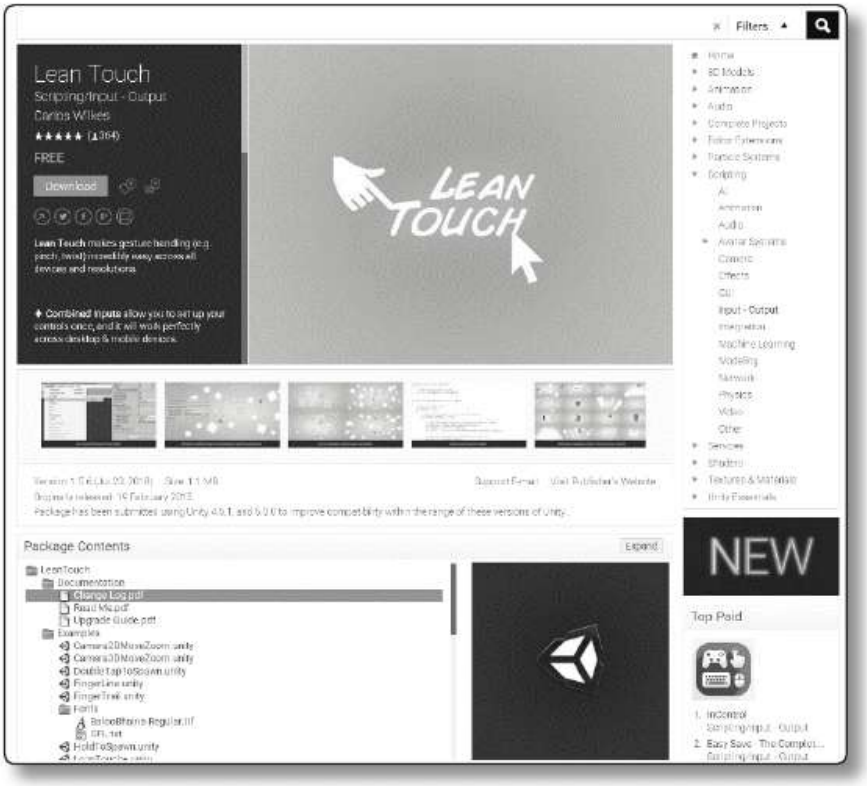
Y sus componentes y propiedades, debería quedar como se muestra a continuación:



Una vez que hemos realizado todo esto, la aplicación tiene la funcionalidad de rastrear una zona, detectar planos y colocar la mesa y las sillas en uno de los planos detectados, pero nos falta añadir la funcionalidad de manipular la mesa y las sillas para rotarla, moverla y cambiar su tamaño, para que el modelo 3D se integre lo mejor posible en el entorno real.

Como este tipo de funcionalidad, en realidad no tiene nada que ver con la programación de aplicaciones AR o MR, sino que se encuentra más bien en lo que sería la programación de interfaces de usuario de dispositivos táctiles, algo que se encuentra fuera del objeto de este libro, para utilizar esta funcionalidad vamos a utilizar otro *Asset* disponible en la *Asset Store*.

El *Asset* en cuestión es *Lean Touch*, del desarrollador *Carlos Willkes* y tiene dos versiones una gratuita y otra de pago, es un excelente *Asset* para manejar pantallas táctiles y la versión de pago es muy económica por lo que, si necesitamos realizar aplicaciones para tablets o teléfonos es muy conveniente considerar la compra de la versión de pago, ya que así, además, ayudamos al mantenimiento y la mejora del *Asset*, que de paso nos ahorra una gran cantidad de trabajo.



Una vez descargado e importado el *Asset*, para darle a la mesa y las sillas la funcionalidad de manipularlas mediante gestos, solo debemos asignar al objeto

"Mesa Sillas" dos *scripts* de este *Asset*, estos *scripts* son **LeanRotate** y **LeanScale**. Una vez añadidos, si compilamos y ejecutamos la aplicación, podremos colocar la mesa y las sillas y las podremos girar, colocarlas en el sitio que queramos y hacerlas más grandes o más pequeñas.

El resultado final de la aplicación puede verse en la siguiente imagen:

La mesa y las sillas se pueden manipular como un conjunto, tocando y arrastrando con el dedo, se mueven por el suelo. Si de toca con dos dedos y se abren o se cierran, se agrandará o se empequeñecerá y si se toca con dos dedos y se giran, la mesa y las sillas rotarán sobre sí mismos.

5.5 OTRAS CARACTERÍSTICAS

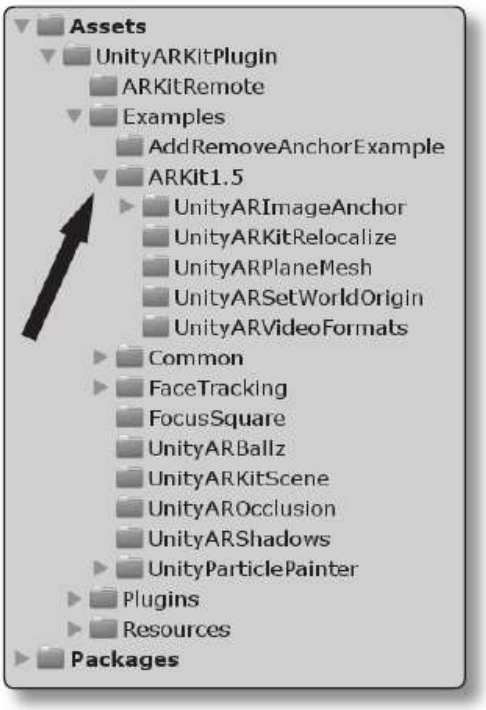
En la primavera de 2018, con la actualización de la versión 11.3 de iOS, se lanzó una versión de ARKit, la 1.5 que incluía algunas mejoras respecto a la versión 1.0 que hasta entonces era la versión que se encontraba en producción.

Alguna de ella ya la hemos visto, como por ejemplo la posibilidad de detectar planos verticales, que no estaba incluida en la versión original de ARKit, pero además hay algunas características nuevas bastante interesantes que es importante que veamos.

5.5.1 Detección de imágenes

Una de las nuevas posibilidades es la de reconocer imágenes y utilizarlas como marcadores. El objetivo de esto es que cuando la cámara de ARKit reconozca el patrón de una imagen que ya tiene asignada en la aplicación, instancie un *Prefab* de Unity, lo que no solo implica mostrar un modelo 3D o un panel 2D con información adicional, sino que este *Prefab* lleve código asociado y que lo pueda ejecutar, con la potencia que esto tiene, de esta forma podemos llevar ARKit a otro nivel en el desarrollo de aplicaciones de Realidad Aumentada ya que al detectar por ejemplo, marcadores asociados a un elemento, podemos hacer cosas como cambiar la interfaz de usuario en tiempo real y trabajar con un elemento del mundo real de forma distinta a otro que tenga asociada una imagen o marcador diferente.

Para ver cómo funciona la detección de imágenes, vamos a estudiar el ejemplo del *ARKit Plugin*, es importante que la versión instalada sea la que incluye la rama *Spring 2018 Update*, por eso, es mejor descargar el *plugin* del repositorio de *Bitbucked*, otras descargas de otros sitios pueden no ser compatibles con la versión 1.5. Esto lo sabremos porque al importar el *Plugin*, en la jerarquía tendremos una carpeta específica llamada **ARKit 1.5** dentro de la carpeta **Examples**.



En primer lugar, dentro de esta carpeta de ejemplos dedicada a la versión 1.5, vamos a encontrar otra llamada **UnityARImageAnchor** y dentro de ella una escena con el mismo nombre, que procederemos a abrir.

Una vez abierta la escena, veremos que tiene lo habitual en toda aplicación de ARKit, una **cámara AR** y el **CameraManager**, debidamente configurado.

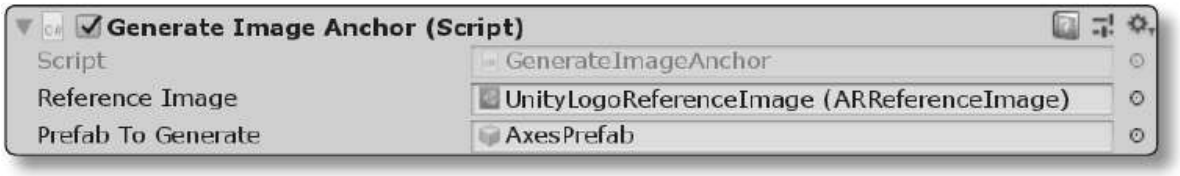
También tenemos el objeto *RandomCube* que Unity incorpora a todos los ejemplos para que veamos la orientación del sistema de coordenadas.

Y veremos dos objetos que no conocíamos hasta ahora, uno llamado **GenerateImageAnchor** y otro llamado **GenerateImageAnchorCube**, estos dos objetos son los que gestionan el reconocimiento de imágenes y lo que ocurre cuando una imagen es reconocida.

Vamos a ver en primer lugar el objeto llamado **GenerateImageAnchor**, si lo seleccionamos, veremos en el *inspector* que tiene un componente con el mismo nombre que tiene dos parámetros, uno llamado *Reference Image* y otro llamado *Prefab to Generate*.

El primero recibe la imagen de referencia que deberá ser reconocida y el segundo, recibe un *Prefab* que se instanciará cuando la imagen especificada anteriormente sea reconocida por la cámara de ARKit. Como hemos comentado, el *Prefab* no tiene por qué limitarse a un modelo 3D, puede ser cualquier cosa, incluido un objeto vacío con un script asociado que al instanciarse se ejecute y presente, por ejemplo, un panel 2D en un *Canvas* con información acerca de la imagen.

Pero si observamos con detenimiento el parámetro *Reference Image*, nos daremos cuenta que la imagen que se debe detectar no es una imagen normal y corriente, sino un objeto de la clase **ARReferenceImage** de ARKit.



Para crear un objeto de esta clase que contenga la imagen que queremos que se reconozca, debemos crearlo y para eso, al instalar el ARKit Plugin se nos han añadido al menú del editor de Unity, algunas herramientas. Para acceder a la herramienta de creación de imágenes de referencia, debemos acceder a la opción de menú **Asset → Create → Unity ARKit Plugin → ARReferenceImage**, una vez seleccionado, veremos que en el inspector nos aparecen tres campos para crear una imagen de referencia, también aparecen para que la editemos si hacemos clic en el botón de edición de la opción *Reference Image*.

En estos campos, incluiremos la siguiente información:

- ⇨ *Image Name*: Nombre de la imagen

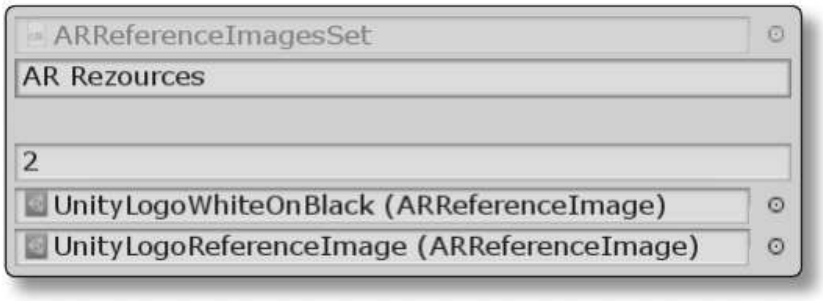
- ⇨ *Image Texture*: Textura 2D previamente importada. Para una correcta detección de las imágenes, conviene que cumplan unas determinadas características, estas suelen ser comunes a todos los sistemas de detección y se han explicado en el capítulo 4 en el que se ve como trabajar con marcadores con la herramienta Vuforia.

- ⇨ *Physical Size*: El tamaño real (ancho) que va a tener la imagen en la escena, en Unity, para estas medidas se utiliza el metro como unidad por lo que en la imagen de referencia del caso anterior la medida de ancho sería de 6 cm. Una medida inexacta no implica que la imagen no vaya a ser detectada, pero sí que la detección va a costar más trabajo y que puede ser más susceptible a fallar.

El otro objeto que hay en el ejemplo llamado **GenerateImageAnchorCube**, hace exactamente lo mismo, pero con otra imagen y otro *Prefab*, mientras uno nos mostrará una imagen 3D de un sistema de coordenadas, este nos mostrará un cubo. Por lo tanto, es necesario crear un objeto con un componente **GenerateImageAnchor** y su correspondiente imagen de referencia y asignarle un *Prefab*, por cada imagen que queramos detectar.

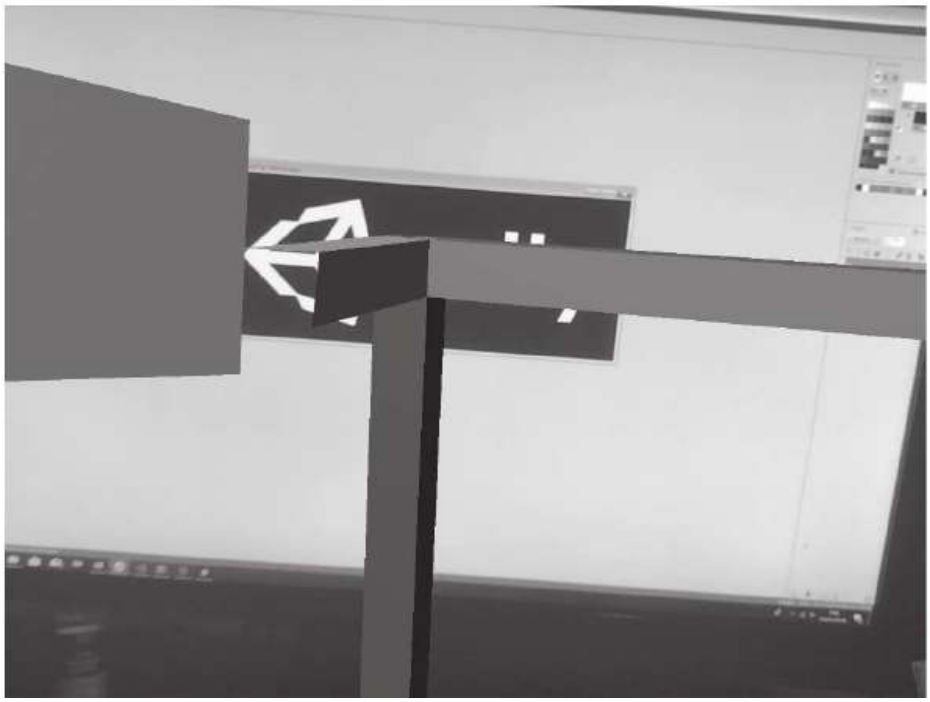
Una vez que tenemos todas las imágenes que queremos detectar, debemos hacer una colección y asignársela a la cámara, esto se hace con otra herramienta a la que podemos acceder desde la opción del menú **Asset → Create → Unity ARKit Plugin → ARReferenceImagesSet**.

Esta opción de menú abrirá la herramienta en el inspector y allí habrá que indicarle el número de imágenes que va a tener la colección y asignarles la imagen de referencia. Estas imágenes de referencia se habrán guardado con la extensión *.asset* en la misma carpeta de la escena.



Una vez creada la colección, se nos generará un *asset* en la carpeta de la escena y si recordamos cuando al principio del capítulo estudiábamos las propiedades del *script* **Unity AR Camera Manager**, del objeto **camera Manager**, había una, que dijimos que comentaríamos más adelante, llamada *Detection Images* a la que se le debe asignar un objeto de la clase **ARReferenceImagesSet**, es aquí donde precisamente debemos asignar el *asset* que nos ha generado la herramienta anterior.

Si compilamos la aplicación y escaneamos con el dispositivo iOS cualquiera de las imágenes, veremos cómo se instancian los *Prefabs* asignados a cada una. En la siguiente imagen se pueden ver las dos imágenes a capturar presentadas en la pantalla de un PC desde una aplicación de diseño y al enfocar con un iPad a la pantalla, son reconocidas e instanciados los dos *Prefabs*, una de las imágenes es ocultada por el *Prefab* del cubo.



5.5.2 Relocalización

ARKit también ha incorporado una característica importante que es la relocalización. Cuando al inicio del capítulo estuvimos explicando brevemente cómo funcionaba ARKit y otros sistemas basados en *Odometría Inercial Visual*, nos pudimos dar cuenta de que estos sistemas saben dónde nos encontramos y como calcular la posición de los objetos 3D porque constantemente comparaban la información de la nube de puntos con la información de los sensores de posición del dispositivo, tal y como se apreciaba perfectamente en el video que pusimos de ejemplo.

El problema de esto, es que, si se interrumpe al análisis de la nube de puntos y la comparación con los sensores de posición, se deja de conocer la posición del dispositivo en el mundo real y, por tanto, el sistema no sabe dónde nos encontramos y como consecuencia de eso no puede colocar los objetos virtuales en el sitio correcto.

ARKit tenía este problema en su versión inicial, si mientras estábamos utilizando una aplicación con ARKit, se utilizaba el teléfono en un iPhone o nos saltaba una llamada de Skype o por cualquier razón la aplicación pasaba a segundo plano, el tracking se interrumpía y se perdía la posición.

Desde la versión 1.5, ARKit incorpora la posibilidad de mantener el tracking, aunque la aplicación pase a segundo plano y cuando se vuelva a utilizar continuar con el mismo sin perder información, a condición de que el dispositivo no se mueva demasiado.

Esto se hace cambiando el valor de *ARTrackingStateReason*, y estableciéndolo a *ARTrackingStateReasonRelocalizing*, esto no tiene una API con un objeto creado para hacerlo directamente desde Unity, por lo que hay que hacerlo a través de la clase **UnityARSessionNativeInterface** de la siguiente forma.

```
private UnityARSessionNativeInterface augmentedRealitySession;
augmentedRealitySession.ARTrackingStateReason =
    augmentedRealitySession.ARTrackingStateReason.
ARTrackingStateReasonRelocalizing;
```

Para revertir el sistema de *tracking* para que no se efectúe la relocalización, solo hay que hacer la siguiente llamada:

```
UnityARSessionNativeInterface .ARSessionShouldAttemptRelocalization = false;
```

6

ARCORE

6.1 INTRODUCCIÓN

Después de ver Vuforia y ARKit paramos en la última estación de Realidad aumentada que nos falta, ARCore.

ARCore es la nueva tecnología de Realidad Aumentada de Google y nace como la contrapartida a ARKit en smartphones sobre su sistema operativo Android. Fue presentado en agosto del 2017 y un poco después se anunció los dispositivos compatibles entre los que estaban el Samsung Galaxy S8 o el Google Pixel. La lista de dispositivos compatibles ha ido creciendo desde entonces y podemos consultarla aquí *(https://developers.google.com/ar/discover/supported-devices)*

Al igual que ARKit esta tecnología está basada en VIO (*Visual Inertial Odometry*) y también nos permite enlazar los datos de la cámara con los datos del giroscopio (Posición y rotación) y los proporcionados por el acelerómetro. Con todo esto somos capaces de posicionar objetos virtuales en el mundo real sin necesidad de marcadores, así como establecer un seguimiento de estos con una precisión bastante aceptable. También incluimos en sus características el ajuste de la iluminación del objeto virtual a la luz real que capta la cámara.

A diferencia de *Tango*, el otro proyecto de Realidad Aumentada de Google, no necesitamos de un hardware adicional y usamos la propia cámara del Smartphone para hacer que todo funcione. Los dispositivos *Tango* disponían de 3 cámaras que hacían en conjunto las veces de un scanner 3D y daban una gran precisión, aunque solamente estaba disponible para un reducido número de dispositivos, por lo que Google arrancó el proyecto ARCore que pretende estar disponible para un número mucho mayor de dispositivos, al no depender de hardware especial.

Vamos a explicar las tres tecnologías en las que se basa ARCore para conseguir esto con algo más de detalle:

⇨ **Motion Tracking:**

Esta tecnología permite al smartphone conocer su posición en el mundo real utilizando la cámara, el giroscopio y acelerómetro. ARCore utiliza un proceso llamado Odometría concurrente o COM para saber en todo momento la posición del smartphone con respecto al mundo que le rodea.

Básicamente lo que hace es capturar puntos del entorno (*feature points*) fácilmente reconocibles con su algoritmo de visión artificial y usarlos, junto a la información inercial proporcionada por acelerómetro y giroscopio, para su propio autoposicionamiento a lo largo del tiempo. Esto último es importante ya que es capaz de guardar información sobre el recorrido que hagamos y "Recordar" la posición del punto de partida y la posición relativa de los distintos objetos virtuales que pudiéramos colocar a lo largo de este recorrido.

⇨ **Enviromental understanding:**

Nos permite "medir" el tamaño y localización de todo tipo de superficies, horizontales y verticales, que tenemos alrededor para usarlas como soporte para objetos virtuales en nuestra aplicación.

La limitación, al igual que con ARKit, son las superficies blancas o sin textura que son muy difíciles de reconocer para este tipo de algoritmos. En la práctica, las paredes blancas o muy parecidas al suelo no son capaz de detectarlas.

⇨ **Light Estimation:**

Esta característica, también disponible en ARKit, nos permite calcular la iluminación de los objetos virtuales basándonos en la luz del mundo real donde los vamos a colocar. Esto incrementa el realismo de la escena que componemos.

6.2 CONFIGURACIÓN DEL MOTOR

Ahora vamos a configurar nuestro motor favorito para afrontar proyectos con ARCore. Los requerimientos básicos para poder preparar nuestro proyecto son:

⇨ Disponer de una versión de Unity 3d 2017.4 o superior.

⇨ Tener descargado la última versión de ARCore (En el momento de la publicación de este libro es la 1.3.0)

⇨ Disponer de una versión del SDK de Android 7.0 (*API level 24*) o superior, ya que las anteriores no son compatibles.

Para la descarga del SDK de ARCore disponemos de este enlace *https://github.com/google-ar/arcore-unity-sdk/releases*

Nosotros elegimos la versión en formato *.unitypackage* y descargamos el fichero. Para la integración del paquete en nuestro sistema, como hemos hecho hasta ahora, nos vamos a **Assets→Import Package→Custom package**, elegimos el paquete descargado y finalmente pulsamos en *Import* para cargar los *Assets*.

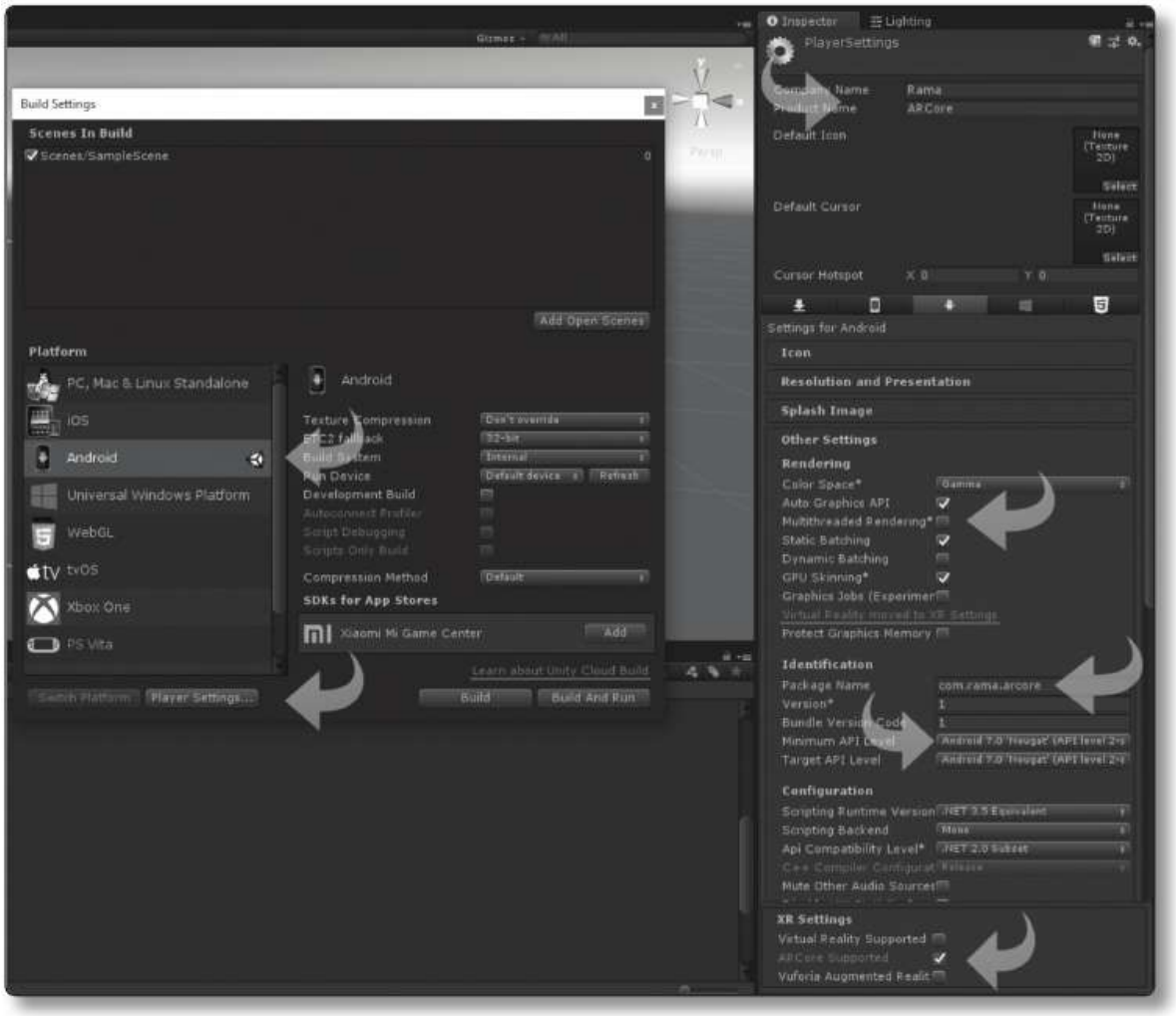
Una vez importado el SDK vamos a preparar nuestro proyecto para Android. Para esto nos vamos a **File→Build Settings** y seleccionamos Android en el selector de la izquierda y pulsamos el botón *Switch Platform*. Después pulsamos en *Player Settings* para configurar nuestro proyecto. Aquí debemos de modificar parámetros en varios lugares como mínimo para que la cosa funcione correctamente:

⇨ Lo primero es configurar el nombre de nuestra compañía y del producto en la cabecera de esta sección.

⇨ Después desplegamos la pestaña *Other Settings* y configuramos varias cosas:

- Desmarcamos la opción *Multithreaded Rendering*.
- Configuramos un nombre de paquete acorde al nombre de la aplicación. En nuestro caso hemos puesto "com.rama.arcore".
- Definimos *Minimun API Level* a *Android 7.0 "Nougat"* ya que es la primera API que incluye compatibilidad con ARCore
- En *Target API Level* también marcamos *Android 7.0 "Nougat"*

⇨ Finalmente desplegamos la pestaña *XR Settings* y marcamos la opción *ARCore Supported*.



Con esto ya tenemos nuestro Unity 3D preparado para desarrollo de proyectos con ARCore.

6.3 DETECCIÓN DE IMÁGENES

Para entender los distintos objetos que vamos a manejar en un proyecto con este SDK vamos a cargar uno de los ejemplos que tenemos por defecto y explicamos cada uno de los *Gameobject* que incluye para que todo funcione.

En esta última versión del SDK han incluido una funcionalidad nueva que nos va a resultar familiar, la detección de imágenes 2D. Usaremos el ejemplo que es capaz de detectar imágenes 2D y seguirlas por su sencillez para empezar. ARCore puede detectar Imágenes 2D y seguirlas con su cámara mostrando elementos virtuales referenciados a estas. No es capaz (como sí que lo es Vuforia) de usar

marcadores cilíndricos o cúbicos por lo que las imágenes deben de estar en planos para reconocerlas correctamente.

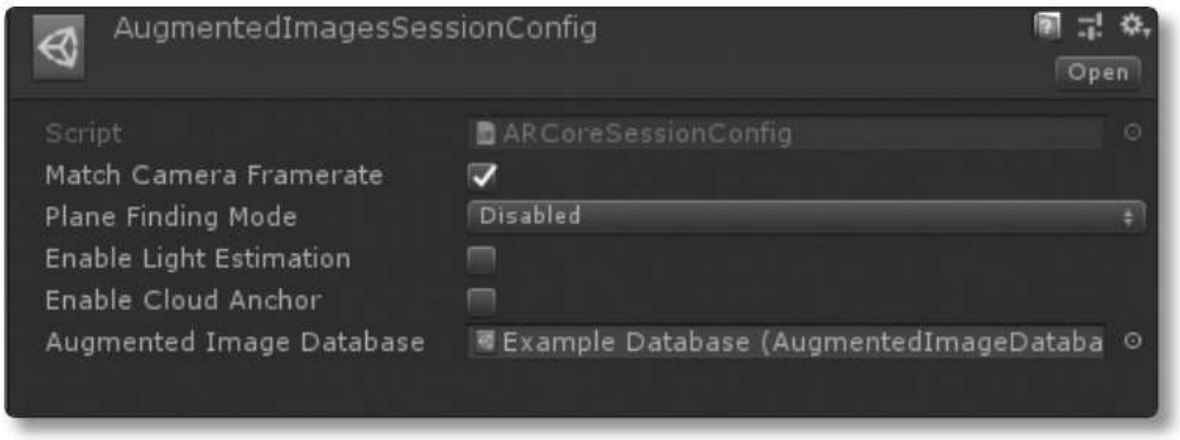
El ejemplo lo tenemos en:

Assets\GoogleARCore\Examples\AugmentedImage\Scenes\AugmentedImage.

Nosotros hemos creado una copia en nuestro proyecto (Contenidos adicionales del capítulo) en **Assets\00–Scenes** para que esté más accesible y junto al resto de escenas del capítulo.

Una vez cargada la escena vemos que dispone de diferentes objetos que vamos a comentar:

- **ArCore Device**: Este objeto es el que configura la sesión de ARCore y carga una configuración definida para nuestra escena. Disponemos de un *Prefab* con el objeto en **Assets/GoogleARCore/Prefabs/ArCore Device** (por si necesitamos componernos la escena nosotros mismos). Del *gameobject* **ArCore Device** cuelga otro llamado **Fisrt Person Camera** con la cámara de ARCore y el *script* que renderiza los objetos. Este subobjeto no debemos de tocarlo. Si pulsamos en **ARCore Device** y nos vamos al *Inspector* podemos ver que el objeto dispone de un *script* llamado **ARCoreSession**. En este *script* se gestiona la sesión de ARCore y se carga (Lo más importante a reseñar) la configuración de escena en nuestro proyecto. Si damos un clic en el objeto *Session Config* vemos que nos lleva a la configuración por defecto, que no es otra cosa que un fichero de configuración situado en **Assets\GoogleARCore\Examples\ AugmentedImage \Configuration.**



Vamos a ver que es cada cosa:

- *Match Camera Framerate*: Aquí permitimos a ARCore introducir un retardo sobre la velocidad a la que la cámara del dispositivo entrega fotogramas (Sobre unos 30 fotogramas en la mayoría de los casos). En nuestro caso lo dejaremos marcado.
- *Plane Finding Mode*: Definimos que tipos de planos vamos a buscar en el mundo real. Por defecto se buscan planos horizontales y verticales, aunque podemos restringir o desactivar esta búsqueda. En este caso concreto lo dejaremos en *Disabled* porque no es necesario para nuestro ejemplo de detección de imágenes.
- *Enable Light Estimation*: Activamos o no la estimación de luz en escena para que se ajuste la iluminación de los objetos virtuales a la luz detectada en el mundo real. En este caso también la desactivamos al no ser necesario.
- *Enable Cloud Anchor*: Un anchor o anclaje es una descripción de una ubicación y orientación fijas en el mundo real. Existe la posibilidad de que estos anclajes se compartan en la nube para múltiples usuarios y dispositivos. También desmarcado para este caso
- *Augmented Image Database*: En ARCore podemos detectar imágenes 2D también en el mundo real. ARCore es capaz de detectarlas y seguirlas (*tracking*) con la cámara. Aquí es donde cargamos nuestra base de datos de imágenes detectables. Al igual que pasaba con Vuforia no todas las imágenes son "fácilmente reconocibles". Depende de la cantidad de patrones detectables y no repetitivos que tenga serán más sencillas o complicadas de detectar. ArCore es capaz de seguir más de 20 imágenes simultáneamente, aunque no puede seguir múltiples instancias de la misma imagen. Para este ejemplo disponemos de una base de datos previamente creada en **Assets\GoogleARCore\Examples\AugmentedImage\Scenes\Images\ExampleDatabase**. Podemos usar esta o simplemente crear las nuestra con nuestras propias imágenes. Hacer esto es tan sencillo como seleccionar todas las imágenes de la carpeta donde las hemos importado previamente y hacer clic derecho y **Create→GoogleArCore→Augmented Image Database**. Crea un fichero con la base de datos en la misma carpeta y ya podemos arrastrarlo a nuestro fichero de configuración.

⇨ *ExampleController*: Este objeto contiene el controlador de la escena y disponemos de varios Scripts ejemplo para entender cómo manejar los distintos elementos de la API de ARCore. En este caso usamos el *script* **AugmentedImageExampleController** que básicamente lo que hace es crear un "Marco de cuadro" alrededor de las imágenes de la base de datos. Podemos observar que le pasamos un par de variables:

- **AugmentedImageVisualizerPrefab**: Es un *prefab* que consiste en un objeto con un script llamado **AugmentedImageVisualizer**. El *script* de este *prefab* lo que hace es colocar un marco de cuadro alrededor de la imagen detectada. Para implementaciones propias **es aquí donde nosotros debemos de preparar el elemento virtual que queremos mostrar tomando como referencia la Imagen**.
- **FitToScanOverlay**: Esto es simplemente un **UI/Image** que activamos cuando no estamos siguiendo ninguna imagen.

Si le echamos un vistazo al código vemos como en el *Update()* va registrando todas las imágenes reconocibles e instancia el *Prefab* pasándole la imagen reconocida en una variable pública (Este *Prefab* se encarga posteriormente de enmarcar la imagen dependiendo de sus dimensiones). En caso de estar en modo de seguimiento (*tracking*) oculta el objeto **FitToScanOverlay** y si no lo activa.

Si compilamos y generamos él *.apk* podemos ver que al detectar la imagen efectivamente la enmarca como si fuese un cuadro.

6.4 DETECCIÓN DE PLANOS

Tras detectar imágenes, pasamos a lo realmente característico de este SDK, que es la posibilidad de detectar planos e iluminar los objetos virtuales dependiendo de la luz del mundo real. Al igual que el anterior usaremos una escena de los ejemplos e iremos explicando los objetos que la integran. Esta escena será la base de los siguientes ejemplos que vamos a ver.

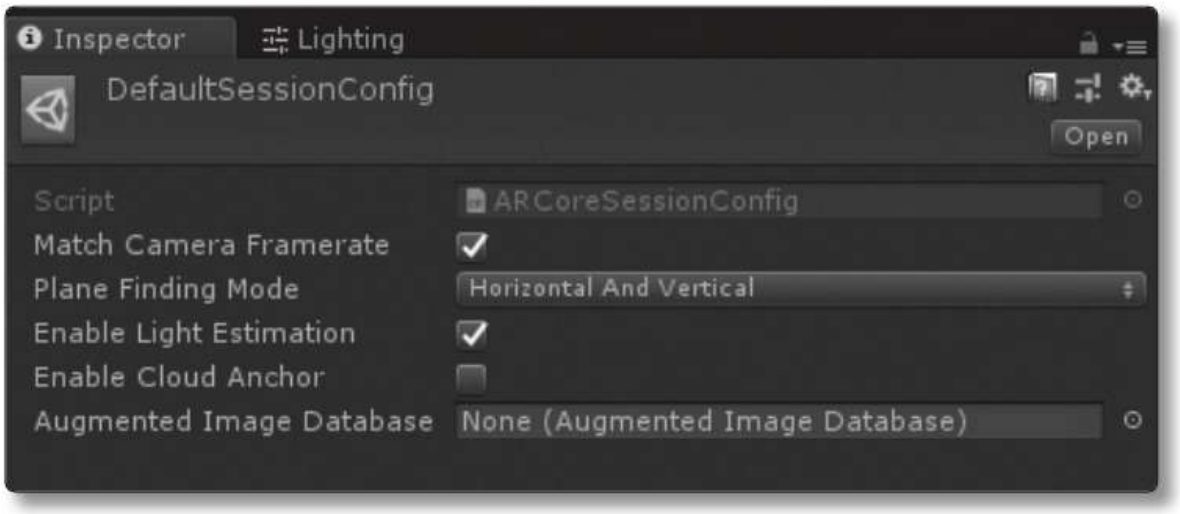
La escena podemos cargarla desde **Assets\GoogleARCore\Examples\HelloAR\Scenes\HelloAR** o en nuestro proyecto en contenidos descargables en **Assets\00–Scenes\HelloAR**.

Una vez cargada podemos ver algunos objetos iguales u otros nuevos. Pasamos a detallar la funcionalidad de cada uno y los cambios con respecto a la escena anterior:

- **ARCore Device**: Aquí, como explicamos antes, se carga la configuración de la escena. En este caso cambia con respecto al ejemplo anterior. Si nos vamos a **ArCoreSession** vemos que el fichero de configuración ha cambiado. Ya no le pasamos ninguna base de datos de imágenes como parámetro y hemos activado la estimación de luz y la detección de planos.

  Este fichero de configuración lo podemos encontrar en

  Assets\GoogleARCore\Configurations\DefaultSessionConfig



⇨ **Enviromental Light**: Este objeto es el encargado de regular la iluminación de los objetos virtuales de la escena. Como vemos es simplemente un *script* que mediante un *shader* cambia la iluminación de los objetos.

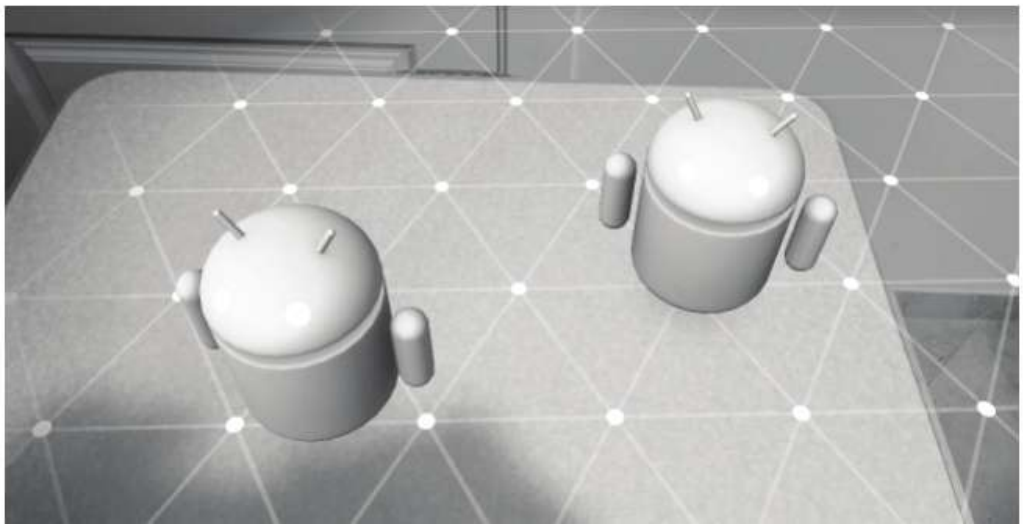
Aquí nosotros no tenemos que cambiar nada. Si quisiéramos añadir este objeto manualmente a una escena lo tendríamos disponible como *Prefab* en **Assets\GoogleARCore\Prefabs**. Si hemos definido la opción *Enable Light Estimation* en configuración debemos de tenerlo incluido en la escena.

⇨ **Plane Generator**: Es el objeto encargado de generar los planos. Al igual que el anterior no es conveniente modificarlo y ya veremos cómo manipularlo más delante desde el *Script* del *controller*. Básicamente está compuesto por un *script* llamado **DetectedPlaneGenerator** que lo que hace es instanciar un *Prefab* que vinculamos en la variable **DetectedPlanePrefab** sobre el plano detectado. El *Prefab* que instancia por defecto (Unos puntos interconectados) lo tenemos en **Assets\Examples\Common\Prefabs\DetectedPlaneVisualizer**. Al igual que el anterior, si hemos activado en configuración la detección de planos, debe de estar incluido en la escena.

⇨ **ExampleController**: Al igual que en el anterior ejemplo aquí tenemos el controlador que se encarga de gestionar las instancias de los elementos 3D que queremos introducir en escena. En este caso concreto lo que hacemos es introducir a *Andy*, el pequeño robot verde que conforma el logo de Android. Como podemos ver dentro del *Gameobject* disponemos de un *script* llamado **HelloARController** al cual le pasamos la cámara de la escena **FirstPersonCamera**, el *prefab* que muestra el plano (Recordamos que lo tenemos en **Assets\Examples\Common\Prefabs\DetectedPlaneVisualizer**), el *prefab* con *Andy* (Lo tenemos en **Assets\Examples\Common\Prefabs\Andy**) y un *gameobject* **UI.Image** que muestra un mensaje "*Searching for surfaces*"). Si miramos que hace el *script* (Al final del capítulo mostraremos un ejemplo totalmente comentado donde lo explicamos con detenimiento) en la función *Update()* intenta detectar planos en el mundo real y si detecta alguno oculta el **UI.Image** (El que muestra buscando). En caso de haber encontrado un plano, y realizar un *Touch* en la pantalla, lo que hacemos es instanciar a *Andy* sobre ese plano basándonos en un *raycasting* desde el sitio donde hemos realizado el *Touch*. Como vemos es algo muy sencillo y con esta base de ejemplo, y cambiando a *Andy* por cualquier otro modelo 3D, podemos

incluir cualquier objeto sobre el plano del mundo real y que se ilumine con la luz detectada.

Si probamos a compilar el resultado debe de ser algo parecido a esto al ejecutar la APP:



6.4.1 El Pozo

Hasta ahora hemos usado el código propuesto en los ejemplos de ARCore. Una vez conocida la estructura y los distintos *gameobjects* y *Prefab* que debemos de incluir en la escena vamos a construir un proyecto propio con algunos cambios. Para este nuevo ejemplo tomamos como base el Ejemplo anterior (*HelloAR*), pero vamos a hacer algunos cambios en la escena.

En el proyecto de contenidos adicionales del libro la escena que vamos a crear está disponible en **Assets\00 –Scenes\Pozo**.

Vamos a detallar las distintas acciones a seguir:

⇨ Incluiremos dos **UI.Buttons** (que llamarán a una función del *script* de controlador) en el *Canvas*. Como hemos podido observar el *grid* de puntos está bien para ver que detecta nuestro sistema y que no, pero a la hora de hacer una demostración, captura de imagen o video llega a molestar. En estos botones que acabamos de incluir llamaremos a una función para activar/desactivar este *grid*. La función concretamente es *GridActiv()* y le pasamos un parámetro booleano a true o false dependiendo de si lo

queremos activar o desactivar. (Comentaremos esta función con el resto del código al final de esta sección).

⇨ También incluimos un modelo 3D, pero con algunas características especiales. Concretamente incluiremos un Pozo trampa con pinchos en su interior. El *prefab* de este modelo lo tenemos disponible en **Assets\50–Prefabs\Pozo** y si nos acercamos a él podemos ver que no es visible desde todos los lados. Como podemos ver en la captura está preparado para solo ser visible desde arriba y muestra su interior a través de agujero de arriba.

Vista superior y lateral del modelo 3D donde se ve lo que queda y no queda visible.

Con esto conseguimos un efecto de profundidad que al usar la aplicación en el mundo real parece realmente que existe un pozo allí al no mostrar el interior nada más que desde el agujero superior. Al asomarnos al interior es cuando muestra los pinchos de abajo.

Este efecto se consigue con varios *shaders* y es aplicable a cualquier modelo. En nuestro caso los *shaders* están disponibles en **Assets\10–Shaders**.

Si vemos la estructura del Pozo está formado por el elemento portal que es el que nos deja ver el interior del pozo y que usa el *shader* "*PortalWindows*", los elementos "Botton" y "Spikes" que usan el *shader* "*StandarARC*" y el elemento "Top" que usa un *shader* standard y siempre está visible.

⇨ Por último, cambiamos el *Script* del controlador. Nuestro *gameobject* ahora se llamará **ControladorAR** y el *script* encargado de gestionar todo **ControladorARCore**. El script está disponible en **Assets\40–Scripts\ ControladorARCore.cs**. Ya llegó el momento de pararnos y explicar

las partes más interesantes de este script. Esto nos ayudará a entender las posibilidades de los distintos objetos que nos proporciona ArCore y cómo usarlos. Hemos castellanizado el código para entenderlo mejor.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
/// Añadimos esta librería para poder manipular el UI de la escena
using UnityEngine.UI;

/// Muy Importante, añadimos esta librería para poder operar con ARCore
using GoogleARCore;
using GoogleARCore.Examples.Common;
public class ControladorArCore : MonoBehaviour {
//Creamos una variable pública de la cámara
public Camera camara;
 //Creamos una variable pública para poner el prefab de los planos de traking creados por ARCore
public GameObject planosDeTracking;
//Creamos una Variable pública del objeto que vamos a instanciar
public GameObject objetoIns;
//Creamos una variable pública del objeto text del canvas
public GameObject buscandoPlano;
//Variable booleana necesaria para las funciones extraidas del código de comprobación de Google

 private bool m_IsQuitting = false;
    //Variable booleana para detectar los planos, lo damos el valor true como inicial

private bool mostrarUI = true;

private List<DetectedPlane> m_NuevosPlanos = new List<DetectedPlane>();

private List<DetectedPlane> m_TodosLosPlanos = new
List<DetectedPlane>();


void Update () {

//Condicional para salir de la aplicación cuando se pulsa el botón back (1)
```

```
35          if (Input.GetKey(KeyCode.Escape)) {
36              Application.Quit();         }
37
38  //Inicializamos la funcion para que compruebe que existen los permisos y que no existan
    errores (2)
39          _QuitOnConnectionErrors();
40          //Comprobamos si hay tracking (3)
41          if (Session.Status != SessionStatus.Tracking)
42          {
43              const int lostTrackingSleepTimeout = 15;
44              Screen.sleepTimeout = lostTrackingSleepTimeout;
45              if (!m_IsQuitting && Session.Status.IsValid())
46              {
47                  //Activamos el text gameobject del UI
48                  buscandoPlano.SetActive(true);
49              }
50              return;
51          }
52          //Función nativa de Unity para controlar la atenuación de pantalla
53          //En este caso, siempre estará activa (4)
54          Screen.sleepTimeout = SleepTimeout.NeverSleep;
55          //Iteramos sobre los planos de este frame e instaciamos los objetos que
56            //correspondan (5)
57          Session.GetTrackables<DetectedPlane>(m_NuevosPlanos,
58  TrackableQueryFilter.New);
59          for (int i = 0; i < m_NuevosPlanos.Count; i++)
60          {
61              //Instanciamos el prefab para la visualización del plano y que los siga.
62              //La posición se ajusta al origen con una identidad de rotación para que
63              //se actualice con las coordenadas globales de Unity
64              GameObject objetoPlano = Instantiate(planosDeTracking, Vector3.zero,
65              Quaternion.identity, transform);
66              objetoPlano.GetComponent<DetectedPlaneVisualizer>().Initialize
67               (m_NuevosPlanos[i]);
68
69          }
70  //Ocultamos el texto de Buscando planos ... ya que hay tracking
71          Session.GetTrackables<DetectedPlane>(m_TodosLosPlanos);
72
73          for (int i = 0; i < m_TodosLosPlanos.Count; i++)
```

```
74          {
75              mostrarUI = true;
76              //Condicional sobre si hay tracking
77              if (m_TodosLosPlanos[i].TrackingState == TrackingState.Tracking)
78              {
79                  mostrarUI = false;
80                  break;
81              }
82          }
83          //Activamos o no el gameobject de texto
84          buscandoPlano.SetActive(mostrarUI);
85          //Empezamos a configurar la acción de instanciar el objeto a mostrar en
86            //los planos de tracking mediante el táctil de la pantalla (6)
87          Touch touch;
88          if (Input.touchCount < 1 || (touch = Input.GetTouch(0)).phase !=
89  TouchPhase.Began)
90          {
91              return;
92          }
93
94          TrackableHit hit;
95          TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon |
96              TrackableHitFlags.FeaturePointWithSurfaceNormal;
97          if (Frame.Raycast(touch.position.x, touch.position.y, raycastFilter, out hit))
98          {
99  //Creamos una variable nueva e instanciamos el objeto a mostrar en la
100 //posición obtenida del touch
101             var objetoEnPlano = Instantiate(objetoIns, hit.Pose.position, hit.Pose
102 rotation);
103             //Creamos un anclaje para permitir a ARCore poder seguir el punto
104 //generado por el táctil de la pantalla y que se mueve con respecto
105 //al mundo real (7)
106             var anchor = hit.Trackable.CreateAnchor(hit.Pose);
107             //El objeto instanciado debería de mirar a la cámara y debe de estar
108             //situado a ras del plano
109             if ((hit.Flags & TrackableHitFlags.PlaneWithinPolygon) !=
110 TrackableHitFlags.None)
111             {
112             //Obtiene la posición de cámara y ajusta la posición Y(altura) con el
113             //punto del táctil
```

```
114                 Vector3 cameraPositionSameY = camara.transform.position;
115                 cameraPositionSameY.y = hit.Pose.position.y;
116             //Hacemos que el objeto instanciado mire hacia la cámara,
117             //respectando su perspectiva en el eje Y.
118                 objetoEnPlano.transform.LookAt(cameraPositionSameY,
119 objetoEnPlano.transform.up);
120             }
121             //Hacemos que el objeto instanciado sea hijo del anclaje
122             objetoEnPlano.transform.parent = anchor.transform;
123         }
124
125     }
126     //segundo ejemplo, función para visualizar el grid o no (8)
127     //Función para botones para mostrar los planos de suelo de ARCore o no
128     public void GridActiv(bool ch)
129     {
130         foreach (GameObject plane in GameObject.FindGameObjectsWithTag
131 ("Player"))
132         {
133             Renderer r = plane.GetComponent<Renderer>();
134             DetectedPlaneVisualizer t = plane.GetComponent<DetectedPlane
135 Visualizer>();
136             r.enabled = ch;
137             t.enabled = ch;
138         }
139     }
140 //Son funciones para que se cierre la aplicación en el caso de error además
141 //de garantizar los permisos necesarios para ejecutarla (2)
142     private void _QuitOnConnectionErrors()
143     {
144         if (m_IsQuitting)
145         {
146             return;
147         }
148
149         if (Session.Status == SessionStatus.ErrorPermissionNotGranted)
150         {
151             _ShowAndroidToastMessage("Camera permission is needed to run this
152 application.");
153             m_IsQuitting = true;
```

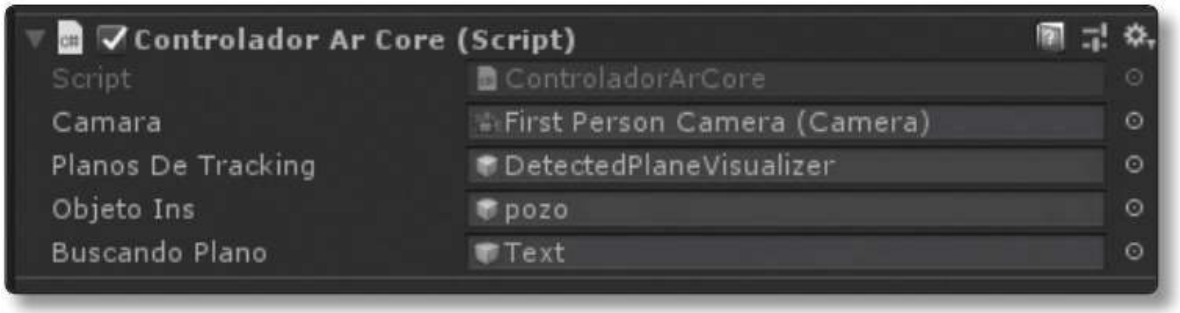
```
154             Invoke("_DoQuit", 0.5f);
155         }
156         else if (Session.Status.IsError())
157         {
158             _ShowAndroidToastMessage("ARCore encountered a problem
159 connecting.  Please start the app again.");
160             m_IsQuitting = true;
161             Invoke("_DoQuit", 0.5f);
162         }
163     }
164     //Función para salir de la APP
165     private void _DoQuit()
166     {
167         Application.Quit();
168     }
169     //Función para mostrar mensajes en Android
170     private void _ShowAndroidToastMessage(string message)
171     {
172         AndroidJavaClass unityPlayer = new AndroidJavaClass("com.unity3d
173 player.UnityPlayer");
174         AndroidJavaObject unityActivity = unityPlayer.GetStatic<AndroidJava
175 Object>("currentActivity");
176         if (unityActivity != null)
177         {
178             AndroidJavaClass toastClass = new AndroidJavaClass("android.widget
179 Toast");
180             unityActivity.Call("runOnUiThread", new AndroidJavaRunnable(() =>
181             {
182                 AndroidJavaObject toastObject = toastClass.CallStatic<AndroidJavaO
183 bject>("makeText", unityActivity,
184                     message, 0);
185                 toastObject.Call("show");
186             }));
187         }
188     }
189 }
190
```

Dentro de la función *Update()* vamos a controlar todo *frame* a *frame*. Comentamos los puntos más representativos del código:

1. Si pulsamos ESC salimos de la aplicación.
2. *QuitOnConnectionErrors():* Esta función se encara e comprobar que disponemos de todos los permisos necesarios para ejecutar ARCore. En caso de no sea así mostramos un mensaje a nivel de sistema operativo con *ShowAndroidToastMessage()*
3. Buscamos si estamos haciendo tracking. En caso de no conseguirlo, pasamos al siguiente *frame* y activamos el **UI.Text** que indica que estamos todavía buscando planos.
4. Evitamos que la pantalla se atenúe.
5. Recorremos los planos detectados y activamos el objeto visualizador para cada uno de ellos.
6. Si hemos realizado un *touch* en la pantalla, mediante la técnica *raycasting* y con las coordenadas del *touch* localizamos donde colisionaría en el plano y allí instanciamos nuestro elemento 3D.
7. Creamos un anclaje en el punto en el que hemos situado nuestro objeto virtual para permitir a ARCore hacer el seguimiento en el resto de la sesión.
8. Función que muestra u oculta el *grid* de los planos. Esta es la función que vincularemos a los 2 **UI.Buttons** que hemos incluido en pantalla.

Como podemos ver este *script* es básicamente el que realiza todo el trabajo de control. Debemos de vincularle la Cámara (**First Person Camera**, la encontramos en **ArCoreDevice**), el *prefab* encargado de visualizar planos (**Assets\GoogleARCore**

Examples\common\Prefabs\DetectedPlaneVisualizer), el objeto a insertar (Nuestro Pozo) y el **UI.Text** que muestra el texto "Buscando plano .." cuando no ha encontrado ninguno.



El resultado, si compilamos todo debe de mostrar un resultado parecido a este en el mundo real:

6.4.2 El Portal

Seguramente la mayoría de los lectores habrán visto varios ejemplos de portal dimensional en Internet usando Realidad Aumentada. Este tipo de efecto es muy impresionante y en este libro queríamos también crear uno usando ARCore. La escena que estamos describiendo la tenemos en **\Assets\00–Scenes\Portal** en el ejemplo de materiales adicionales.

Para los que no sepan a qué nos estamos refiriendo dejamos algunos enlaces de YouTube para que puedan visualizar algunos ejemplos:

Ejemplo de portal 1: *https://www.youtube.com/watch?v=rIPfpGCxONQ*
Ejemplo de portal 2: *https://www.youtube.com/watch?v=NesmV1p-qZE*

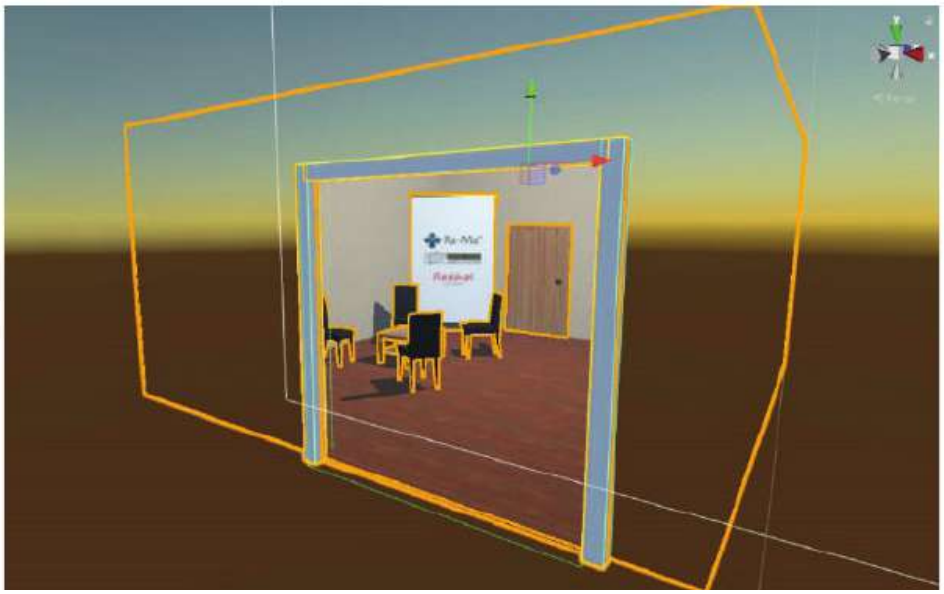
O la recreación en Realidad Aumentada del video de A-ha que muchos conocemos *https://www.youtube.com/watch?v=ZBdRAdSosv4*

Bien, pues para hacer esto con lo que ya sabemos, es algo tremendamente sencillo.

Dentro de los *assets* disponibles en materiales adicionales tenemos mobiliario y una habitación virtual en 3D, justo los que hemos usado para el capítulo 3. En el proyecto de este capítulo lo tenemos todo preparado en un *prefab* en **Assets\50–Prefabs\PortalHabitacion**.

Si arrastramos este *Prefab* a escena y arrastramos el *gameobject* dentro del *script* del controlador tendríamos ya nuestro portal preparado.

Al igual que hemos hecho con el pozo, el secreto está en ocultar a la vista mediante un *shader* (En nuestro caso **Assets/10–Shaders/StandardARC**) los objetos del interior de la habitación y solamente los dejamos ver mediante otro *shader* (**Assets \10 –Shaders\PortalWindow**) que tenemos aplicado a un plano en el ventanal de la habitación. Los objetos siempre visibles como el marco de la ventana tienen aplicado un *shader standard*.



El resultado en la calle queda espectacular una vez compilada la aplicación.

7

MICROSOFT WINDOWS MIXED REALITY

7.1 INTRODUCCIÓN

Windows Mixed Reality (**WMR**) es una plataforma de realidad mixta incluida como parte del sistema operativo Windows 10.

WMR proviene de una plataforma anterior llamada **Windows Holographic**, que ofrece la posibilidad de presentar hologramas virtuales combinados con el mundo real a través de un casco autónomo (un dispositivo llamado *Microsoft HoloLens*) y poder interactuar con ellos mediante gestos teniendo en cuenta las características del entorno real en el que nos encontramos, tales como distancias y volúmenes, recogidas mediante distintos sensores existentes en el casco.

La plataforma **WMR**, utiliza dos tipos de dispositivos, por una parte, las ya citadas *Microsoft Hololens*, que tienen las características que hemos mencionado y un nuevo tipo de gafas bastante más sencillas cuyas características difieren bastante de las *Hololens*.

En este momento, todos estos nuevos dispositivos compatibles con la plataforma **WMR** solo ofrecen la posibilidad de presentar experiencias de Realidad Virtual y no Realidad Mixta, como hace el dispositivo *HoloLens* de Microsoft, es decir, con estos dispositivos no podremos ver el mundo real, tampoco detectar distancias y volúmenes a excepción de un área de seguridad en la que movernos sin riesgo a tropezarnos con elementos como muebles o paredes, ni podremos manejar las aplicaciones mediante gestos, debiendo utilizar dos mandos de forma similar a como se hace con otras plataformas de Realidad Virtual como *Oculus* o HTC.

Tampoco existe ningún dispositivo que sea autónomo como el dispositivo *HoloLens*, sino que tienen que estar conectados a un PC mediante dos cables, uno USB y otro HDMI y una conexión *Bluetooth* para los mandos, con las limitaciones que el cable nos impone para realizar movimientos o desplazarnos.

Por tanto, aunque Microsoft llame a su plataforma **Windows Mixed Reality** (Realidad Mixta de Windows), nosotros consideramos que estos dispositivos están más cerca de la Realidad Virtual que de la realidad Mixta, aunque al contar con sensores para poder detectar unos límites en el espacio que usaremos como zona de trabajo y que estos estén incorporados en las gafas, en lugar de ser independientes, como ocurre con otros dispositivos como *Oculus* o *HTC*, Microsoft los considera Realidad Mixta por la combinación sensores + Realidad Virtual.

Este es otro claro ejemplo de lo variable que puede ser la consideración de lo que es la Realidad Mixta, como comentábamos al inicio del libro cuando nos referíamos al continuo de lo virtual.

Actualmente hay disponibles cinco dispositivos de este tipo compatibles con **WMR** de distintos fabricantes, todos ellos tienen características y diseños similares y los mismos requerimientos para su funcionamiento. Las diferencias entre ellos se centran en las calidades de construcción y precio, desde nuestro punto de vista como desarrolladores de aplicaciones para este tipo de dispositivos, no existe ninguna diferencia entre ellos.

7.2 INSTALACIÓN Y PRIMEROS PASOS

Para poder crear aplicaciones que funcionen con estas gafas previamente debemos establecer una configuración determinada en el PC en el que vamos a desarrollar, lo primero es que el PC esté preparado para funcionar con la Realidad Mixta de Microsoft y esto, aunque con alguna salvedad, lo podemos comprobar con la aplicación **Windows Mixed Reality PC Check** que está disponible de forma gratuita en la tienda de Microsoft.

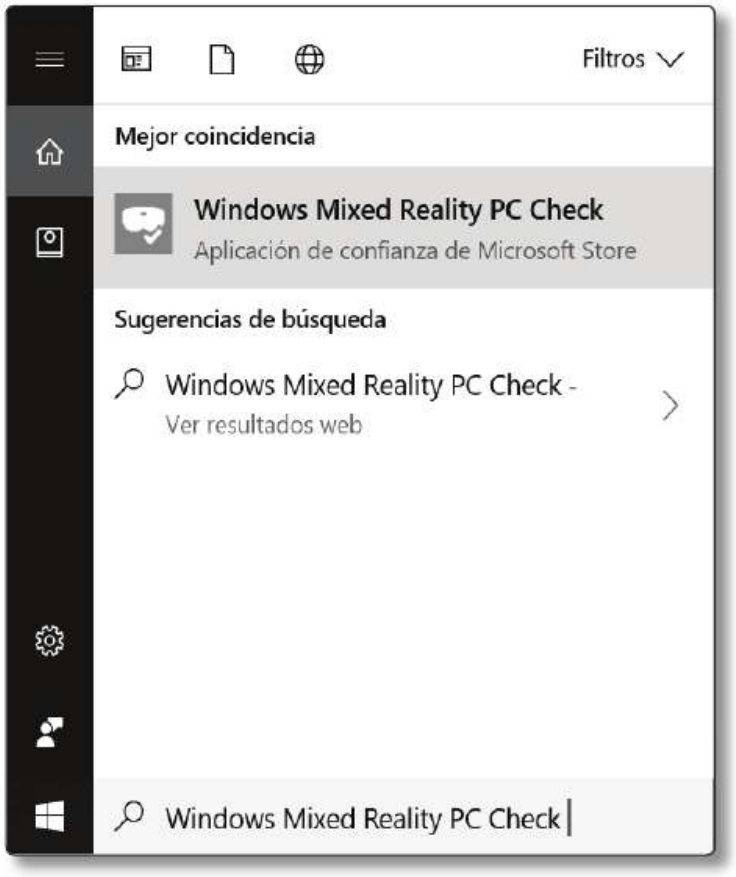


Figura 7.1. Aplicación de chequeo

La instalación del software necesario para unas gafas compatibles con el sistema **WMR** es automática, una vez que se conectan las gafas, estas son detectadas y todo el portal de WMR se descarga y se autoinstala, incluyendo posibles actualizaciones del firmware de las gafas, por lo que, en principio, no es necesario que tengas que hacer ninguna operación especial, siempre que tu PC sea compatible.

Si ya dispones de unas gafas y las has conectado, es posible que tengas instalada la aplicación de chequeo en el PC, lo puedes comprobar escribiendo "*Windows Mixed Reality PC Check*" en el cuadro de búsqueda y aparecerá como aplicación o bien lo puedes buscar en la tienda de Microsoft.

Una vez lanzada la aplicación, esta comprobará los componentes del PC y presentará una ventana con información de los mismos y un icono para cada uno de ellos:

Figura 7.2. Resultados de la aplicación Windows Mixed Reality PC Check

Los iconos pueden ser uno de los tres siguientes:

✓	Las características del PC son suficientes para el funcionamiento de **WMR**
⚠	El sistema **WMR** funcionará, pero puede tener problemas debido a que el hardware no rinde lo esperado y es recomendable actualizarlo
✗	El sistema **WMR** no funcionará con esta configuración.

ⓘ ATENCIÓN

La aplicación *Windows Mixed Reality PC Check* no es, ni mucho menos, infalible. Puede informarte que el sistema es 100% compatible y no serlo, hay varios casos en los que no informa de la compatibilidad correctamente.

La aplicación *PC Check* ha tenido fama de dar bastantes fallos, uno que ha producido muchos quebraderos de cabeza era si el sistema operativo era una versión N de Windows, una versión especial comercializada en la Unión Europea que no incluye todos los elementos de una versión normal de Windows y que te da la opción de seleccionar el navegador en la instalación y que se comercializó así por cuestiones de competencia con otros fabricantes de software y por lo tanto muy extendida en España. En este caso, WMR no es compatible con Windows N, sin embargo, la aplicación **Windows Mixed Reality PC Check** informaba que el sistema operativo era compatible, aunque las gafas no funcionaban.

Se puede comprobar este punto en la configuración de Windows, tal y como se muestra en la siguiente imagen:

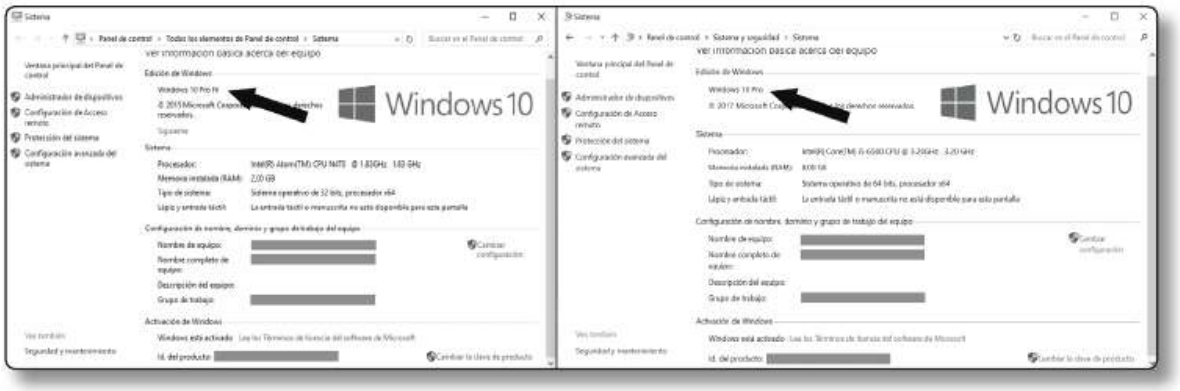


Figura 7.3. Comprobación de la versión de Windows

Si *PC Check* nos avisa de este u otra incompatibilidad deberemos solucionarla ya que, si no, de ninguna manera podremos utilizar la Realidad Mixta, en este caso concreto, la solución pasa obligatoriamente por obtener una nueva licencia de Windows y reinstalar el sistema operativo con la nueva versión.

Algunos usuarios han reportado otro tipo de problemas, tales como puertos USB que no suministraban la potencia necesaria para una correcta alimentación de las gafas, aunque al ser USB 3.0 se daban por compatibles y que se han solventado instalando placas de puertos USB 3.0 o 3.1 adicionales con su propia alimentación independiente o la utilización de adaptadores de *DisplayPort* a HDMI que funcionaban con monitores de hasta 4K pero que no suministraban imagen y sonido a las gafas. En cualquier caso, el usuario debe ser consciente de que pueden aparecer incompatibilidades que no sean detectadas por la aplicación **Windows Mixed Reality PC Check** aunque normalmente son debido a configuraciones muy especiales de hardware producidas por adaptadores o elementos intermedios, difícilmente detectables por *PC Check*.

Afortunadamente, según hemos podido comprobar a lo largo del proceso de redacción de este libro, Microsoft ha actualizado la aplicación y ha mejorado considerablemente el chequeo del sistema, este problema que se producía con las versiones de Windows N, ya no ocurre con las últimas versiones de *PC Check*, se detectan adecuadamente como no compatibles y además han mejorado muchas más cosas, lo que indica que Microsoft está realizando una labor continuada de mejora en su plataforma de Realidad Mixta y en la actualidad, se encuentra bastante madura pese a ser de las que menos tiempo lleva en el mercado.

Una vez que nos hemos asegurado que nuestro equipo es totalmente compatible, debemos configurarlo en función del uso que le vayamos a dar. Dentro de los pasos de configuración aparecerá una pantalla donde debemos elegir si vamos a configurar las Gafas *para todas las experiencias (*que además es la opción recomendada) o si solamente las vamos a usar *sentado y de pie*. Este proceso de configuración aparece la primera vez que accedemos al Portal de Realidad Mixta, pero podemos reconfigurar el sistema cuantas veces queramos mediante la opción de configuración que figura en el mismo.

Figura 7.4. Configuración del tipo de experiencia

Adicionalmente, aparte de la aprobación de los elementos de hardware por la aplicación de chequeo, deberemos realizar las siguientes operaciones:

⇨ Activar el "Modo de desarrollador en Windows".

Para activar el modo de desarrollador hay que ir a Configuración → Actualizaciones y seguridad → Para programadores y marcar la opción *Modo de programador*

⇨ Instalar Visual Studio 2017 con el SDK de *Windows 10 Fall Creators Update*.

Cualquier versión de **Visual Studio 2017** nos será válida, si no utilizamos una versión especifica de Visual Studio para otro tipo de desarrollo y optamos por usar la que viene incluida en la distribución de Unity, debemos de asegurarnos de haberla instalado o de instalarla con posterioridad si no lo habíamos hecho.

Para ello, bien al instalar Unity o con posterioridad, ejecutando de nuevo el instalador, seleccionaremos la opción *Visual Studio Community* tal y como se ve en la imagen:

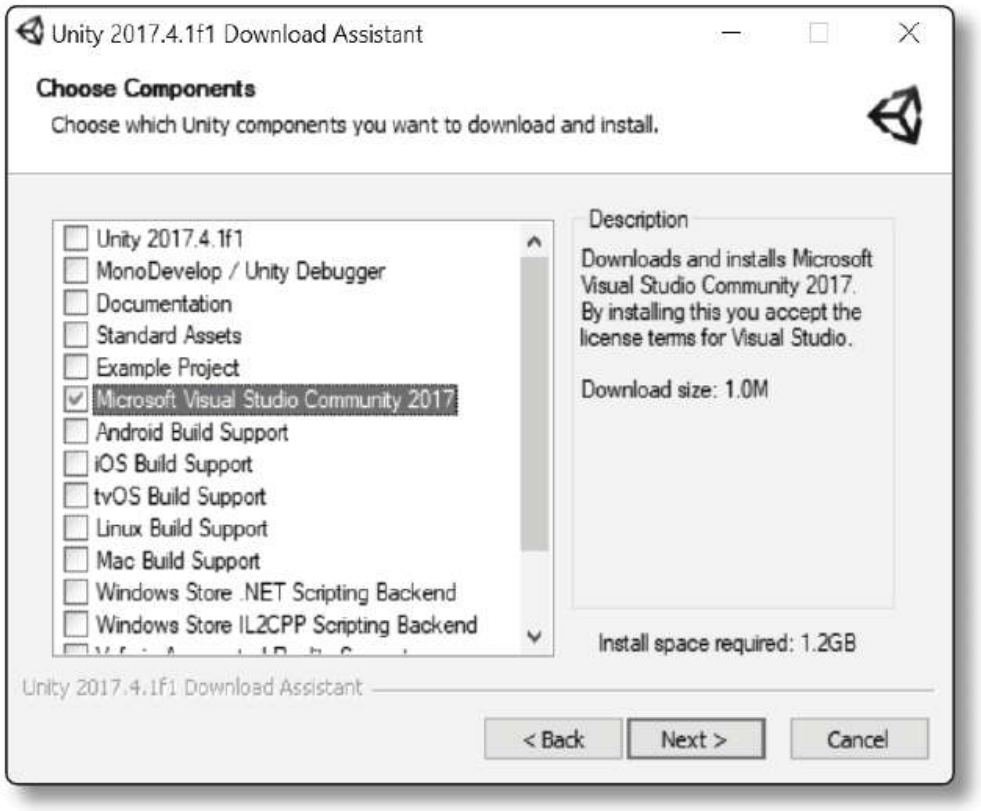


Figura 7.5. Instalación de Visual Studio Community incluida en Unity

⇨ Instalar el SDK de *Windows 10 Fall Creators Update* o *UWP (Universal Windows Platform)*.

Las últimas versiones de Visual Studio 2017 ya lo llevan incluido, así que si lo acabas de instalar no será necesario. Si la versión de Visual Studio es anterior a la actualización *Fall Creators Update* de Windows 10 que se realizó en Noviembre de 2017, entonces si necesitarás actualizar el SDK, que puedes descargar desde la siguiente página de Microsoft: *https://developer.microsoft.com/en-US/windows/downloads/windows-10-sdk*

Si la versión de Visual Studio que tienes instalada es la que viene con Unity, es necesario instalar el SDK, ya que esta no lo incorpora, lo sabrás porque más adelante, cuando configures el proyecto en Unity, verás el error *Could not find any supported UWP SDK installations*:

Figura 7.6. Error indicando la falta del SDK de UWP

Esto ocurre tanto con la versión que se suministra con la instalación de Unity 2017 como con la que se suministra con Unity 2018.

⇨ Configurar Visual Studio para trabajar con UWP y Unity.

Una vez que tenemos instalado Visual Studio 2017, debemos abrirlo, iniciar sesión en nuestra cuenta de desarrollo e irnos a la opción **Herramientas → Obtener herramientas y características** y una vez allí, en la pestaña *cargas de trabajo* marcar las opciones *Desarrollo de la plataforma universal de Windows* bajo el apartado *Windows* y *Desarrollo de juegos con Unity* bajo el apartado *Dispositivos móviles y juego* y posteriormente pulsar modificar para instalar los componentes.

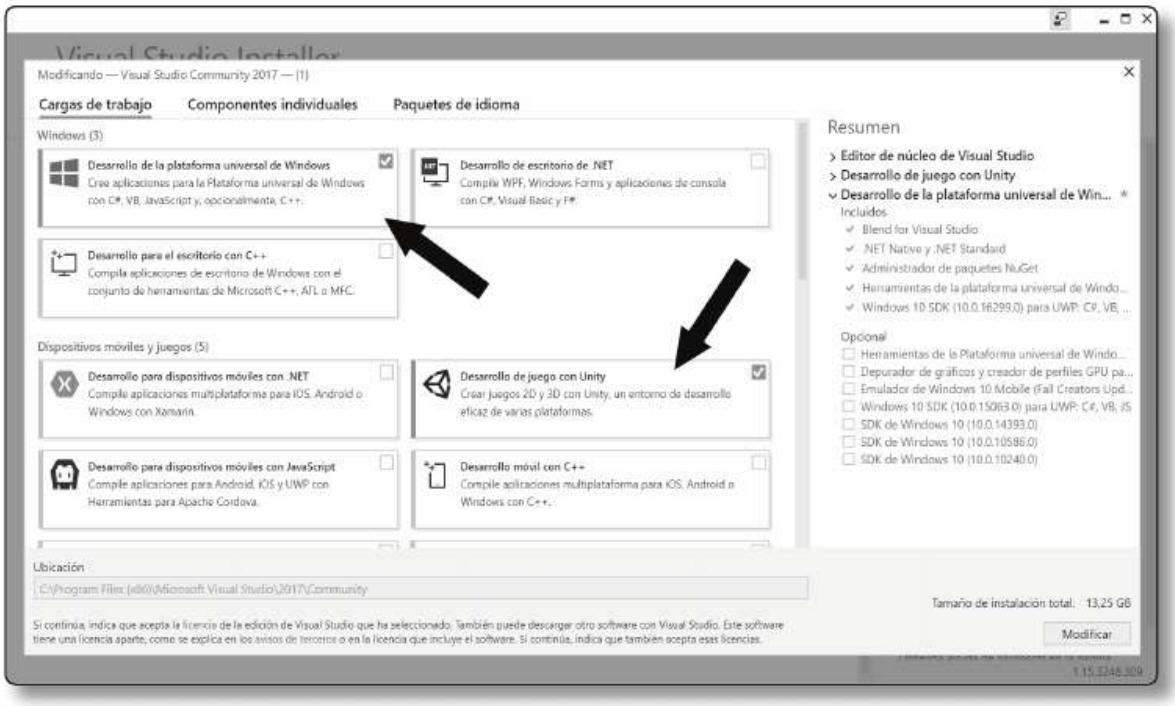


Figura 7.7. Configuración de los componentes adicionales de Visual Studio

Es posible que salga un aviso que indique que hay componentes en ejecución que deben de ser parados y que se volverán a iniciar tras la instalación de los nuevos paquetes.

ⓘ ATENCIÓN

Cabe la posibilidad de que, si se instalan las cargas de trabajo en Visual Studio con posterioridad al SDK de UWP, Unity nos siga mostrando el error de que no disponemos del SDK para compilar aplicaciones para UWP, en este caso hay que realizar los dos últimos pasos en orden inverso, es decir, primero instalar Visual Studio y las cargas de trabajo necesarios y posteriormente el SDK de UWP.

⇨ Unity 2017.4 o superior.

La versión de Unity necesaria para poder realizar aplicaciones para WMR será la 2017.4 o superior.

⇨ Actualización de los *drivers* de la tarjeta gráfica

Es muy recomendable actualizar los drivers de la tarjeta gráfica a su última versión. Siempre es preferible utilizar los drivers nativos del fabricante en lugar de un driver genérico para aprovechar al máximo las características de la tarjeta gráfica ya que las aplicaciones de realidad virtual suelen ser bastante exigentes en el apartado gráfico.

ⓘ NOTAS

Aunque hayas seguido estos pasos, se puede dar el caso de que Visual Studio requiera algún componente adicional. No debes sorprenderte si cuando realices la primera compilación, aparece algún mensaje informativo que indica que necesita un determinado componente o que debe actualizar alguno. Simplemente realiza la instalación y compila de nuevo. Esto suele ser habitual sobre todo si dispones de varias configuraciones instaladas simultáneamente en el mismo PC, por ejemplo, varias versiones distintas de Unity y de Visual Studio.

7.3 CONFIGURAR UNITY PARA CREAR UN PROYECTO WMR

Una vez que ya tenemos todo configurado, podemos empezar a crear nuestro primer proyecto para la plataforma WMR con Unity. Para que nuestro proyecto funcione correctamente con unas gafas WMR necesitamos especificar unas opciones de construcción determinadas que veremos a continuación. Para ellos, vamos a crear un nuevo proyecto o a abrir el proyecto de ejemplo llamado **Ejemplo 1 WMR**, que se encuentra en la carpeta **Capitulo 7 → Ejemplo 1**

En primer lugar, debemos establecer las opciones del proyecto, para ello lo primero es indicar cuál es la plataforma de desarrollo, esto se hace en la opción **File → Build Settings**, una vez allí, veremos la ventana de opciones de proyecto, seleccionaremos *Universal Windows Platform* y haremos clic sobre el botón *Switch platform* como se puede ver en la siguiente imagen:

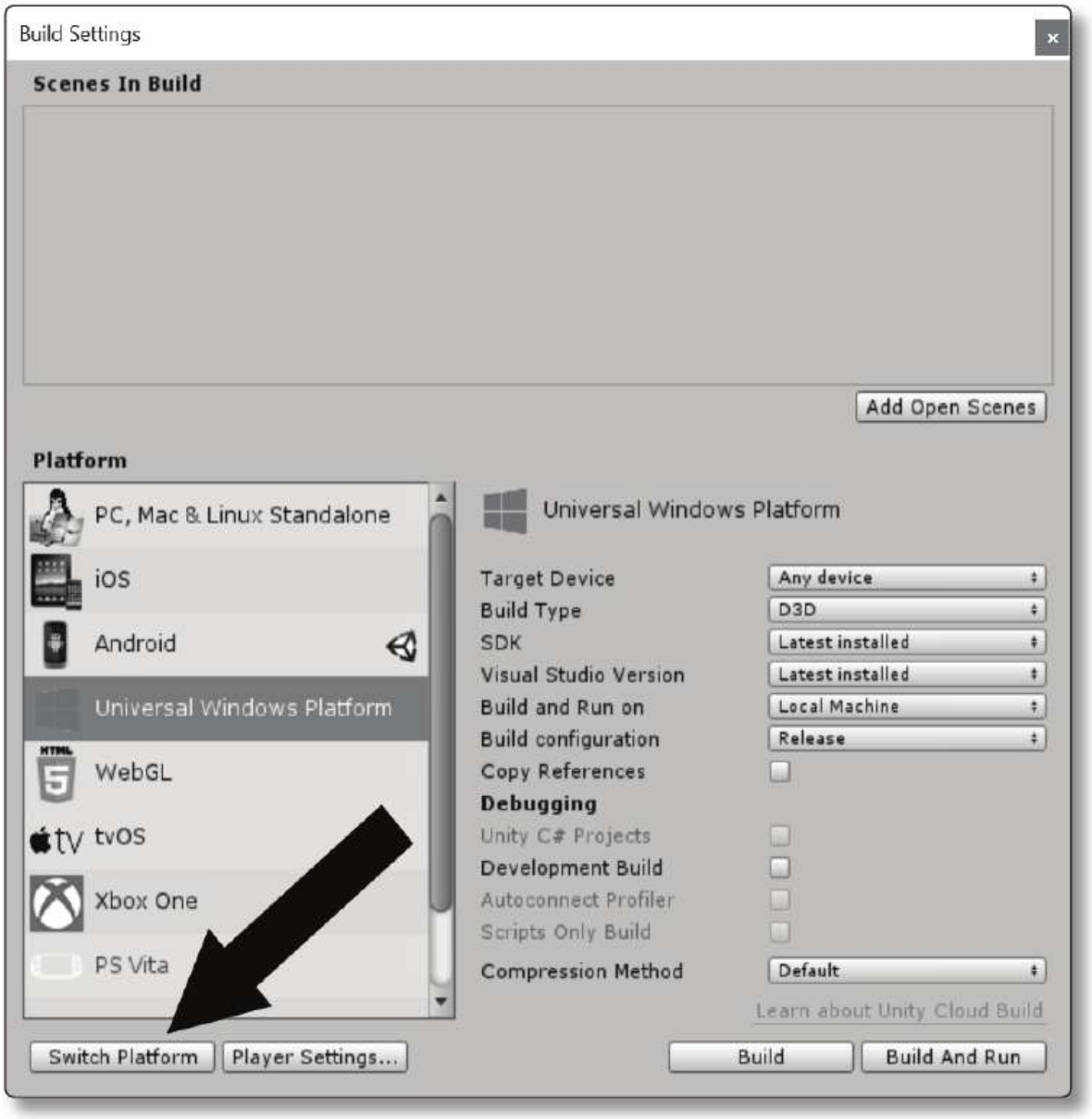


Figura 7.8. Establecimiento de la plataforma UWP

Una vez que hagamos clic en el botón, dependiendo de si es un proyecto aún vacío o de si ya tenemos escena y programas, se realizará una comprobación y recompilación, en cuyo caso podrá tardar algo de tiempo en realizarse.

Una vez establecida **UWP** como la plataforma de desarrollo, debemos indicar los siguientes valores en las opciones de la parte derecha:

- ⇨ En la opción *Target Device* seleccionaremos *Any device*, esta opción se utiliza para gafas WMR, hay otras opciones para crear aplicaciones UWP en dispositivos móviles, en PC o en gafas *Hololens*.

- ⇨ En la opción *Build Type* seleccionaremos D3D. La opción D3D se diferencia de la opción XAML en cómo se manejan los contenedores gráficos, optimizándolos para presentar objetos 3D (D3D) o combinado con elementos 2D (XAML). Esto es, dicho muy a groso modo, una forma de gestionar dichos contenedores mediante distintas clases y dependerá del tipo de aplicación que se quiera hacer si se utiliza un sistema u otro. Por ejemplo, si se usan elementos 2D mezclados en el entorno holográfico 3D como, por ejemplo, el teclado en pantalla que es un elemento 2D, es necesario utilizar el modo XAML, pero esto es una explicación extremadamente básica, para ampliar la información, te recomendamos que consultes la documentación de Microsoft acerca de la *Universal Windows Platform*.

- ⇨ En la opción SDK tendremos todas las versiones del SDK de UWP que tengamos instaladas en el sistema, lo habitual será que seleccionemos la opción "*Latest Installed*" para crear la aplicación con la versión más reciente a no ser que por circunstancias determinadas requiramos una versión específica.

- ⇨ Lo mismo haremos con la siguiente opción, en la que indicaremos que versión de Visual Studio queremos utilizar, por defecto tendremos la última instalada, que podremos cambiar si queremos utilizar otra distinta.

- ⇨ Y en la opción *Build and Run on* seleccionaremos *Local Machine*, ya que es ahí donde tenemos instaladas las gafas y el portal de WMR. Esto nos permitirá que, al pulsar el botón de ejecución de Unity, la aplicación se visualice directamente en las gafas a la vez que en la pestaña *Game* de Unity.

Mas adelante veremos el resto de opciones que afectan a la ejecución y construcción del proyecto. De momento hay otra parte que debemos configurar para que el proyecto funciona correctamente en un entorno WMR.

En la misma ventana donde indicamos las opciones de construcción, junto al botón para establecer la plataforma, veremos también un botón que nos permite establecer las opciones específicas del proyecto de Unity y que viene indicado como

Player Settings. Al hacer clic sobre él, veremos que en el *Inspector* nos aparece el siguiente panel:

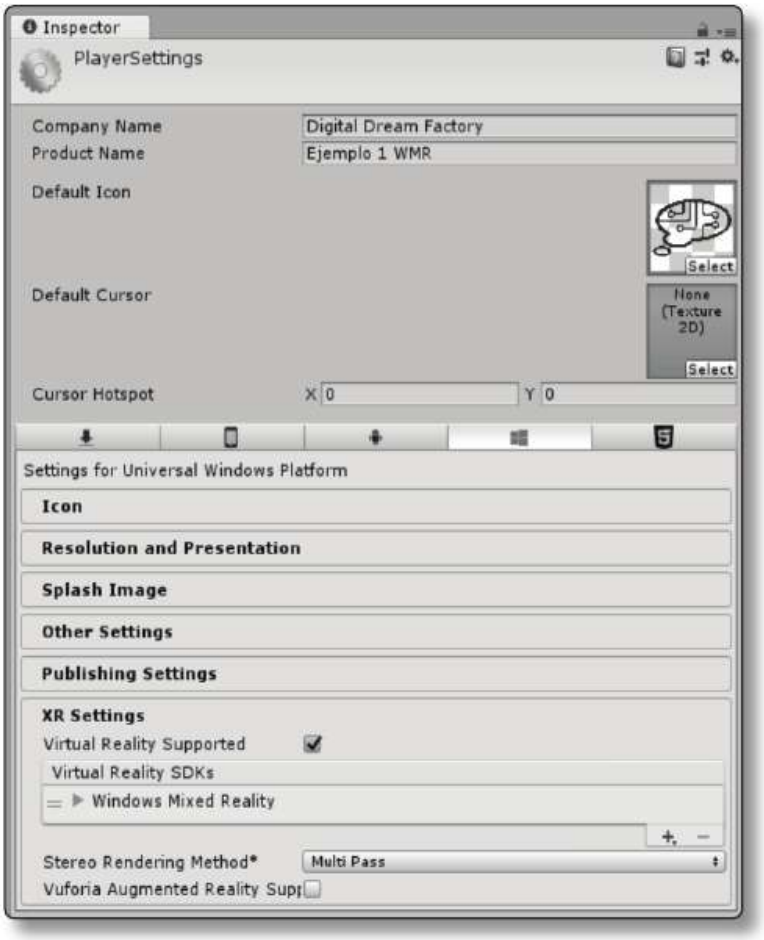


Figura 7.9. Configuración de las opciones del proyecto para WMR

En la pestaña correspondiente a *Universal Windows Platform*, podremos ver un apartado llamado *XR Settings*, en él se encuentra una opción para indicar a Unity que necesitamos utilizar el soporte de Realidad Virtual, así que marcaremos la casilla indicada con el texto *Virtual Reality Suported*, al hacerlo, se nos desplegará una lista con los distintos SDK para Realidad Virtual o Aumentada soportados por UWP que tengamos instalados y por defecto nos aparecerá añadido el de WMR.

También podemos incluir otros, como Vuforia para trabajar con marcadores o *Stereo Display* para que aparezca la pantalla doble y posteriormente visualizar la aplicación con un visor, pero en este caso, sin tener el soporte de las gafas, como por ejemplo el tracking de los mandos o la detección de los límites de la zona de seguridad. Esto último serviría para realizar aplicaciones para dispositivos que ejecuten UWP, por ejemplo, móviles con sistema operativo Windows.

ⓘ ATENCIÓN

Si estás usando una versión anterior a Unity 2018 o Unity 2017.4, quizá no encuentres estas opciones, ya que en versiones anteriores se encontraban dentro del apartado Other Settings , como puedes ver en la siguiente figura, correspondiente a Unity 2017.4 , se indica que se han movido al apartado XR Settings.

Figura 7.10. Aviso de cambio de ubicación de opciones.

ⓘ ATENCIÓN

Es importante destacar que, en versiones anteriores de Unity, en lugar de encontrarnos *Windows Mixed Reality* como el SDK utilizado para las gafas, podemos ver un SDK llamado *Windows Holographics*.

Para un mejor seguimiento de este libro, te recomendamos que utilices la versión 2018 de Unity, aunque es posible que todos los ejemplos funcionen en Unity 2017.X, aparte de cosas como esta, Unity ha hecho una importante labor de integración, incluyendo como parte del motor, productos de terceros que anteriormente había que instalar y configurar aparte, como por ejemplo Vuforia o los SDK de WMR o ARCore, etc.

Hay una excepción a esto y es cuando utilicemos el *Mixed Reality Tookit*, un conjunto de *Assets* que implementan cosas para el desarrollo con WMR, como el teletransporte, giro de cámaras, selección de objetos y muchas cosas más, estas librerías están desarrolladas para Unity 2017.2.

Con esta configuración ya podemos implementar un sistema de Realidad Virtual básico, aunque aún le debemos indicar que vamos a utilizar funcionalidades adicionales, siempre, claro está, que queramos utilizarlas.

Figura 7.11. Capabilities

Esto es lo que se llaman *Capabilities* y se indican en el apartado del mismo nombre que se encuentra dentro de la opción *Publising Settings*.

Hay una buena cantidad de ellas, alguna de las cuales, como la llamada *Spatial Perception*, no son utilizables por WMR y si por las gafas *Hololens*.

El que nos encontremos distintas *Capabilities* que sean exclusivamente para el uso de las gafas *Hololens* o para el de las gafas WMR y que no funcionen con los otros dispositivos es debido a que Microsoft ha integrado en el anterior sistema, llamado *Windows Holographics*, las características de los nuevos dispositivos, creando el sistema WMR que soporta los dos dispositivos.

Esto hace que existan capacidades como la percepción espacial o el uso de gestos para manipular la interfaz de usuario que no funcionen con estas gafas y otros, como el uso de mandos o la detección del área de trabajo que no funcionen con las *Hololens*.

Esta capacidad que hemos mencionado en concreto, la *SpatialPerception*, sirve para que se reconozca el volumen de nuestro entorno como si fuese una malla 3D y se pueda, por ejemplo, detectar colisiones entre los objetos reales y los virtuales, capacidad que no tienen las gafas WMR y si las *Hololens*.

Por ello, no vamos a ver todas las que hay en la lista, solo las más importantes. A continuación, vamos a listarlas indicando cuales están soportadas por las gafas inmersivas y cual por las *Hololens* y para qué es cada una de ellas:

Capability	Gafas WMR	Hololens	Descrpción
InternetClient	S	S	Proporciona acceso de salida a Internet, WMR lo necesita para el reconocimiento de voz.
InternetClientServer	S	S	Proporciona acceso de salida y entrada a Internet, igual que el anterior pero bidireccional, no requiere *InternetClient* si se utiliza este.
MusicLibrary	S	N	Proporciona acceso a la biblioteca de sonidos y a la grabación de los mismo incluyendo la grabación de sonido incluida en el vídeo.
PicturesLibrary	S	S	Proporciona acceso a la biblioteca de imágenes
VideosLibrary	S	N	Proporciona acceso a la biblioteca de vídeos.
Webcam	N	S	Permite capturar imagen y video desde las aplicaciones
Microphone	S	S	Permite el uso del reconocimiento de voz en las aplicaciones.
Bluetooth	S	S	Permite la comunicación con dispositivos Bluetooth en la aplicación. WMR lo utiliza para la comunicación con los mandos.
SpatialPerception	N	S	Permite el uso del mapeado espacial y el reconocimiento de los volúmenes del entorno.

7.4 EL PRIMER PROYECTO EN WMR

Una vez que hayamos configurado Unity tal y como hemos explicado en las páginas anteriores, ya nos encontramos en condiciones de crear nuestro primer proyecto y hacerlo funcionar en unas gafas compatibles con WMR.

Esto es, en realidad, más fácil de lo que podemos pensar, solo tenemos que crear una escena, colocar un plano a modo de suelo o bien un terreno, colocar algún

elemento, bien sean algunas formas geométricas o elementos 3D, aplicar materiales a nuestro gusto o si tenemos un terreno colocar árboles, agua, etc. En fin, lo que nos parezca bien.

A continuación, colocamos la cámara en la escena, para un proyecto WMR no es necesario que tengamos una cámara especial, con lo que la cámara que se crea automáticamente al crear la escena es perfectamente válida.

Si una vez creada nuestra escena, con las gafas conectadas, hacemos clic en el botón *Play* de Unity, la aplicación se ejecutará. En una instalación normal de Unity, sin gafas veríamos la escena desde la vista de cámara, pero en este caso veremos la vista desde las gafas y si nos movemos o movemos la cabeza nuestro punto de vista cambiará y por lo tanto también cambiará lo que se ve en la ventana *Game* de Unity.

Por supuesto, si nos giramos, girará la cámara y podremos ver todos los elementos que se encuentren detrás de nosotros.

Si nuestras gafas están configuradas en modo límite y tenemos definida un área de trabajo, nos podremos mover libremente por esta a la vez que Unity moverá la cámara y por tanto tendremos la sensación de acercarnos o alejarnos a los distintos elementos existentes en la escena y apreciaremos todo lo que ello conlleva: aumento o disminución del volumen de las fuentes de sonido, cambios de iluminación, etc., es decir tendremos una sensación total de inmersión.

Si observamos la siguiente imagen, veremos que hay una diferencia entre la cámara previsualizada y la que se ve cuando la aplicación está en ejecución y podremos observar que esta es cuadrada, esto es porque lo que se envía a las gafas son dos imágenes simultaneas de 1440 x 1440 *pixels* y además si cerramos uno u otro ojo y comparamos la imagen que vemos con la que hay en la pantalla tampoco coincide, ya que a las gafas se les envían dos imágenes ligeramente desplazadas para que cuando nuestro cerebro las procese, las combine creando una imagen tridimensional.

Si abrimos el proyecto de ejemplo llamado **Ejemplo 1 WMR**, que se encuentra en la carpeta **Capitulo 7 → Ejemplo 1** veremos un ejemplo similar a este, de momento solo podemos desplazarnos por la escena si tenemos configuradas las gafas en modo limite y tanto como la distancia que hayamos definido y que el cable nos permita, aunque podemos hacer distintas pruebas cambiando la ubicación de la cámara principal.

Para hacer este ejemplo solo hemos descargado el *Asset* gratuito **Mega Fantasy Props Pack**, del desarrollador *karboosx* y hemos abierto una de sus escenas de ejemplo, la que se ve en esta imagen es la llamada *House of night*. Si el proyecto se ha configurado correctamente, visualizaremos la aplicación en RV.

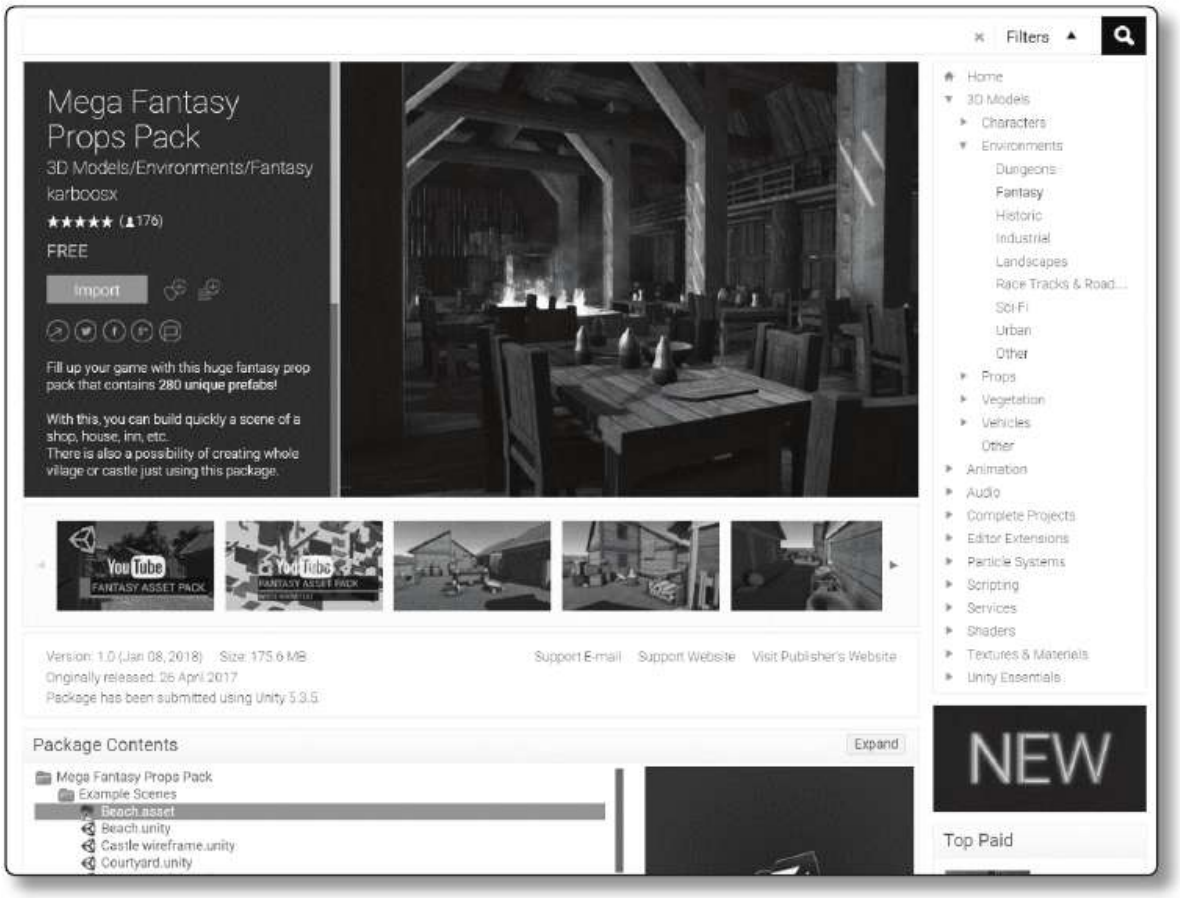


Figura 7.12. Pack de Assets de la Store a integrar

7.5 LOS MOTION CONTROLLERS DE WMR

Todas las gafas compatibles con el sistema WMR disponen de dos mandos que se conectan al PC vía Bluetooth, es necesaria una conexión Bluetooth 4.0 para que funcionen correctamente, por lo que si nuestra versión de Bluetooth es inferior tendremos que actualizar el hardware.

Desde el punto de vista de Unity, un *Motion Controller* de WMR es como cualquier otro mando y se configura en el *Input Manager*, accediendo a él posteriormente desde los métodos **GetInput** y **GetAxis** de la clase **Input**.

No obstante, hay una API específica para WMR, desde la que es más cómodo gestionar los mandos, está API es el *Mixed Reality Toolkit*, que veremos más adelante.

A lo largo del capítulo veremos cómo movernos utilizando un mando para apuntar a un punto del terreno y transportarnos al punto seleccionado utilizando un botón.

7.6 EL WINDOWS MIXED REALITY TOOLKIT

El SDK de *Windows Mixed Reality* es muy amplio y, por lo tanto, con él se pueden realizar absolutamente todas las tareas necesarias para cualquier aplicación. No obstante Microsoft ha desarrollado un conjunto de utilidades, funciones y componentes, llamado *WMR Toolkit* que ha puesto a disposición de la comunidad de desarrolladores en un repositorio de *GitHub*.

GitHub es una plataforma de desarrollo colaborativo que se utiliza para alojar el código fuente de proyectos y que utiliza un sistema de control de versiones llamado *Git*.

Lo más habitual es que los proyectos de *GitHub* se almacenen y se pongan a disposición de la comunidad de forma pública mediante una cuenta gratuita que dispone de distintas herramientas para colaborar en el proyecto, aunque también pueden utilizarse cuentas de pago para gestión de repositorios privados.

El *WMR Toolkit* puede obtenerse desde el sitio web de *GitHub*, accediendo a la dirección *https://github.com/Microsoft/MixedRealityToolkit-Unity/*, una vez que cargue la página del proyecto, que se muestra en la imagen siguiente, veremos un botón de color verde con el texto *Clone or Download*, que en la imagen se encuentra marcado con el número ①, si lo seleccionamos, se desplegará una cortina en la que se nos ofrecerán dos opciones de descarga: una para descargarlo a nuestra aplicación *GitHub* local y otra, marcada con el número ②, que generará un archivo .ZIP con el proyecto que deberemos descomprimir para utilizarlo posteriormente.

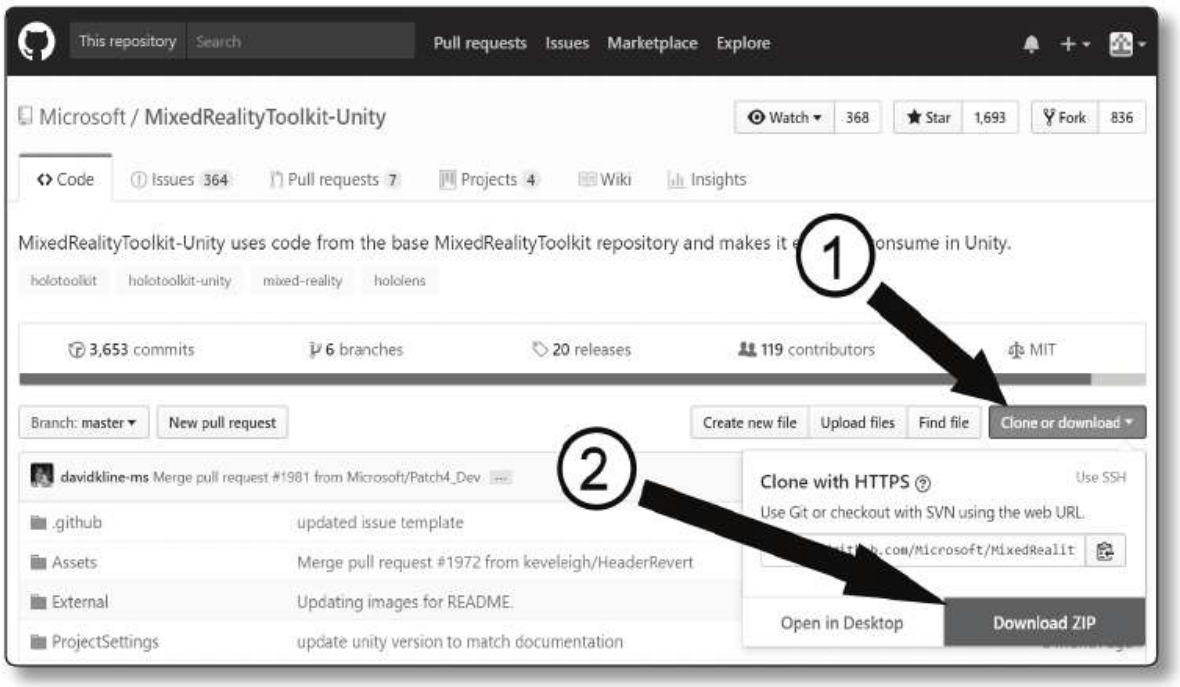


Figura 7.13. Repositorio GitHub del Mixed Reality Toolkit para Unity

GitHub es un sistema de completo de gestión del código fuente que nos permite hacer una gran variedad de tareas, si quieres profundizar en él, hay muchos tutoriales y documentación en Internet, es especialmente recomendable una guía en español, cuyo proyecto, por supuesto, se encuentra y mantiene en *GitHub* y al que se puede acceder en la dirección: *https://github.com/Hispano/Guia-sobre-Git-Githuby-Metodologia-de-Desarrollo-de-Software-usando-Git-y-Github*. Es altamente recomendable.

Una vez descargado el archivo .ZIP, debemos descomprimirlo y abrirlo con Unity como cualquier otro proyecto y a continuación, crear el paquete que importaremos cuando queramos utilizar algún componente que se encuentre contenido en el *WMR Toolkit* igual que si de cualquier otro *asset* se tratase.

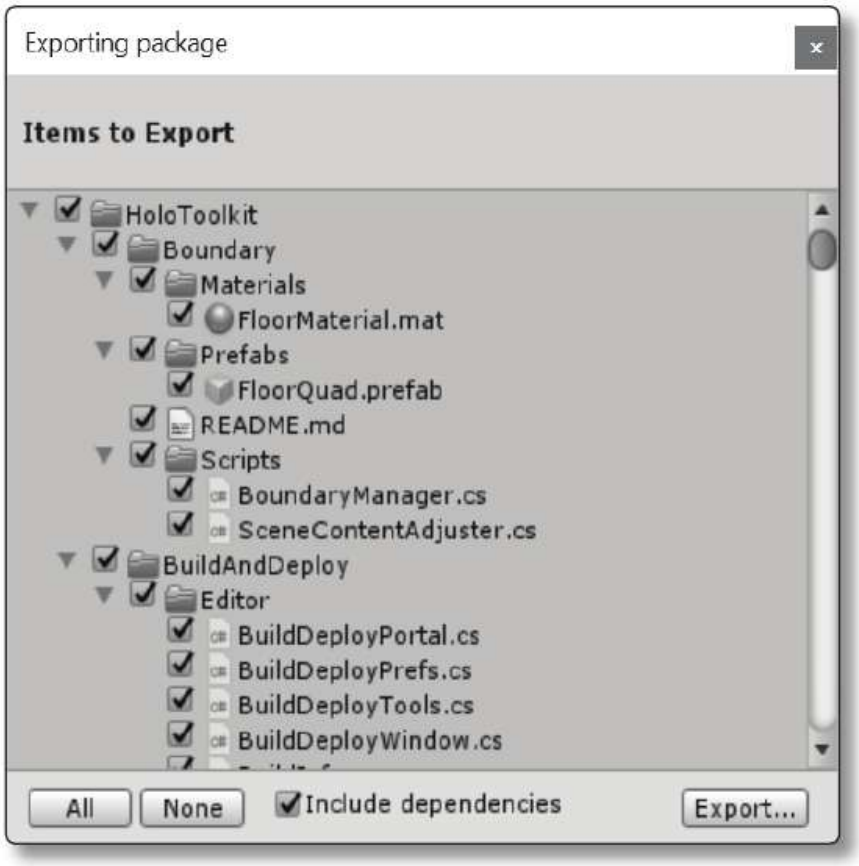


Figura 7.14. Exportación del paquete WMR Tookit

Cuando hayamos abierto e importado el proyecto en Unity, tendremos que generar el paquete mediante la opción de menú **Package → Export** y se creará un archivo con el nombre que le hayamos dado y la extensión *.unityPackage***,** que importaremos en cada proyecto en el que queramos utilizar el *WMR Toolkit*. Nosotros ya hemos realizado este paso con la última versión disponible en el momento de

la publicación de este libro, podrás encontrar el paquete en la carpeta **Capitulo 7\Ejemplo 3. WMR Toolkit\WMR Toolkit.unitypackage**. Por supuesto, también se puede abrir el proyecto resultante de la descompresión del archivo *.zip*, pero con seguridad cuando ya conozcamos el *WMR Toolkit* con más detalle, nos será más cómodo quitar lo que no utilicemos como los ejemplos y quedarnos con un *package* que importemos directamente en nuestro proyecto.

> **NOTAS**
>
> En el momento de escribir este libro, la versión de Unity con la que se ha desarrollado y probado los ejemplos es la 2018.1, mientras que la última versión del *WMR Toolkit*, es compatible con la versión 2017.2.2, por lo que al abrir el paquete en una versión que no sea esa, te dará un aviso de que hay APIS obsoletas y de posibles incompatibilidades. Debes asegurarte de que lo que vas a usar funciona correctamente en otra versión o utilizar la versión de Unity adecuada si usas el *WMR Toolkit*.

Vamos a crear un nuevo proyecto y a configurarlo y prepararlo para utilizar los componentes y funcionalidades que nos ofrece *WMR Toolkit*. Para hacer esto, hemos utilizado la versión de Unity 2018.1.6, para garantizar la compatibilidad con la versión actual del *WMR Toolkit*, aunque es posible que muchas de las cosas funcionen en versiones superiores.

Para ello, creamos un proyecto de la forma en la que habitualmente lo hacemos en Unity, y una vez que nos aparezca el entorno de Unity importamos el paquete que hemos generado en el paso anterior mediante la opción de menú **Assets → Import Package → Custom Package…** y seleccionando el archivo del paquete que contiene el **WMR Toolkit**.

Antes de seguir debemos de configurar el proyecto para WMR en **File → Proyect Settings** eligiendo plataforma *Universal Windows Platform (Switch Platform)* y luego revisando que todo esté correcto *Player Settings***,** aunque esto veremos que se puede hacer de manera automática un poco más adelante**.**

Lo primero que veremos una vez importado, será que bajo nuestra carpeta *Assets* tenemos cuatro carpetas nuevas: una llamada **HoloToolkit**, otra llamada **HoloToolkit-Examples,** otra llamada **HoloToolkit-Preview** y finalmente otra llamada **HoloTollkit-UnitTest.**

Otra cosa que veremos, es que nos ha aparecido un nuevo menú en la barra llamado **Mixed Reality Toolkit** que tiene varios submenús. El primero de ellos es **Configure** y tiene cuatro apartados:

⇨ **Apply Mixed Reality Project Settings** nos permite establecer las distintas opciones del proyecto para que funcione en unas gafas WMR, estas opciones, que se pueden ver en la siguiente figura, son las mismas que ya vimos con anterioridad, solo que después de haber instalado el *WMR Toolkit* podemos establecerlas desde aquí en lugar de hacerlo desde las opciones *Build Settings* y *Player Settings*. Marcando los distintos tags y pulsando posteriormente en "*Apply"* configuramos el proyecto de manera automática.

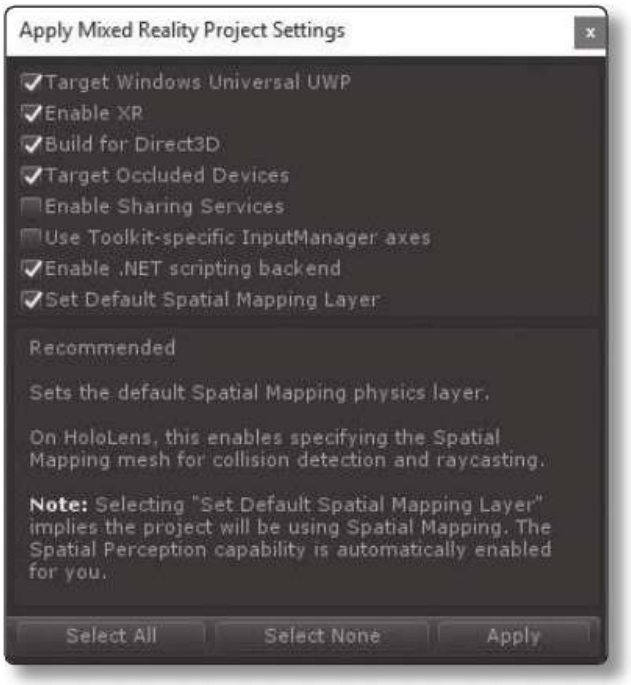


Figura 7.15. Configuración de proyecto desde WMR Toolkit

- *Target Windows Universal UWP:* Se encarga de elegir UWP como plataforma del aplicativo
- *Enable XR:* Activa la compatibilidad del proyecto con WMR en **Player Settings→ Other Settings**
- *Build for Direct3D:* Recomendable para casi todos los proyectos por ser muchísimo más rápido que si no lo incluimos. A menos que necesitemos superponer contenido XAML sobre el de Unity, o alternar entre vistas XAML y de Unity, debe permanecer marcada.
- *Target Occluded Devices:* Aquí elegimos el tipo de casco que vamos a usar. Se marca esta casilla si vamos a usar dispositivos oclusivos como las gafas Acer HMD o las Lenovo Explorer (Las usadas para desarrollar los ejemplos de este capítulo). Básicamente es para diferenciar estos tipos de cascos de dispositivos que permiten

transparencia y muestran los gráficos mediante realidad aumentada como pueden ser las *Hololens*. Esto sirve para configurar el nivel de gráficos a usar (También en ***Edit→Proyect→Quality***) de manera automática. Si el dispositivo no es oclusivo entonces se recomienda un nivel de gráficos más bajo.

- *Enable Sharing Services:* Activa los servicios compartidos y la comunicación entre dispositivos (No es algo que vamos a ver en este capítulo). En nuestro caso lo dejamos desmarcado.
- *Use Toolkit-specific InputManager axes:* Activa el uso de los controladores de Xbox. Esto es necesario si usásemos unas *Hololens*, pero en nuestro caso lo dejaremos desmarcado.
- *Enable .NET scripting backend:* Si está activado usa el módulo .NET de Unity y IL2CPP si no está activado. Esto debemos de activarlo.
- *Set Default Spacial Mapping Layer*: Establece el mapa espacial por defecto. En nuestro caso lo dejamos activo.

⇨ **Apply Mixed Reality Scene Settings** nos permite definir los parámetros de la escena que tenemos cargada. Cuando pulsamos *Apply,* como en el anterior menú, se cargan las opciones automáticamente.

Figura 7.16. Configuración de la escena desde WMR Toolkit

- *Add the Mixed Reality Camera Prefab:* Añade manera automática el *prefab* de cámara a la escena con todos los componentes necesarios. En nuestro caso lo vamos a marcar. Si necesitamos acceder a este *prefab* de manera manual lo encontraremos en:

  Assets/HoloToolkit/Input/Prefabs/MixedRealityCameraParent.

- *Move camera to origin:* Centra la cámara en el punto (0,0,0) de la escena. Es importante que la cámara este centrada en el mundo virtual en el que estamos para que las referencias a los distintos objetos sean las correctas. Este punto también debe de estar marcado.

  Add the input manager Prefab: Añade el *prefab* encargado de gestionar los mandos (Si existen) del dispositivo que estés usando. En nuestro caso lo marcamos también. El *prefab* se puede encontrar para añadir manualmente en:

  Assets/HoloToolkit/Input/Prefabs/InputManager
- *Add the default cursor Prefab:* Nos permite gestionar animaciones del cursor en aplicaciones WMR. En nuestro caso lo marcaremos también. El *prefab* está disponible también en:
- *Assets/HoloToolkit/Input/Prefabs/Cursor/DefaultCursor*
- *Update World Space Canvases:* Activa el uso del **UIRaycastCamera** y su sistema de eventos. Nosotros lo marcaremos también en nuestro ejemplo.
- *Add Spacial Mapping Prefab:* Para nuestro ejemplo, por el dispositivo que usamos (Lenovo Explorer), no lo vamos a activar.

⇨ **Apply UWP Capability Settings** nos permite incluir distintos periféricos como la Webcam o el micrófono, así como los *prefabs* necesarios para la comunicación.

Figura 7.17. Configuración de las capacidades de nuestro proyecto desde WMR Toolkit

- *Microphone:* Imprescindible para el reconocimiento de voz (*DictationRecognizer*, *GrammarRecognizer* y *KeywordReconizer*) y uso de Cortana en nuestros proyectos. Nosotros lo marcaremos para nuestro ejemplo.
- *WebCam*: Necesitamos activarlo si vamos a usar la cámara RGB del dispositivo. A menos que necesitemos hacer uso de ella de manera programada es recomendable que no lo marquemos ya que es un recurso bastante sensible.
- *Spacial Perception*: Es necesario marcarlo si vamos a hacer uso de las capacidades de mapeado espacial de las *Hololens*. En nuestro caso no será necesario.
- *Internet Client:* Nos permite crear un cliente TCP/IP y recibir información desde internet.
- *Internet Client Server:* Además de recibir información nuestra APP podría actuar como servidor.
- *Private Network Client Server:* Actúa como la anterior, pero enfocado a que la comunicación sea exclusivamente dentro de una red local.

Una vez configurado el proyecto y la escena pasamos a crear la escena propiamente dicha. Para este ejemplo vamos a crear un entorno controlado y cerrado del que no nos podemos salir. Para esto hemos usado los *assets* de la habitación que hemos visto en otros ejemplos anteriores de este libro. El ejemplo que vamos a generar lo hemos llamado **Ejemplo 2 WMR** y se encuentra en la carpeta **Capitulo 7 → Ejemplo 2.**

Para facilitar la creación y el seguimiento del ejemplo, hemos creado un paquete unity3D llamado **GeometríaWMR** que contiene todos los *assets* necesarios para la habitación. Para importarlo hay que seleccionar la opción de menú **Assets → Import Package → Custom Package…** y seleccionando el archivo del paquete. El *prefab* a integrar en la escena lo podemos localizar en **Assets/Assets3d/Geometría de Escena.**

En este momento ya tendríamos una escena completa con todos los *prefabs* necesarios para desplazarnos por ella y usar los mandos (Aunque no tengamos nada para interactuar con ellos). La apariencia de la escena sería parecida esta imagen:

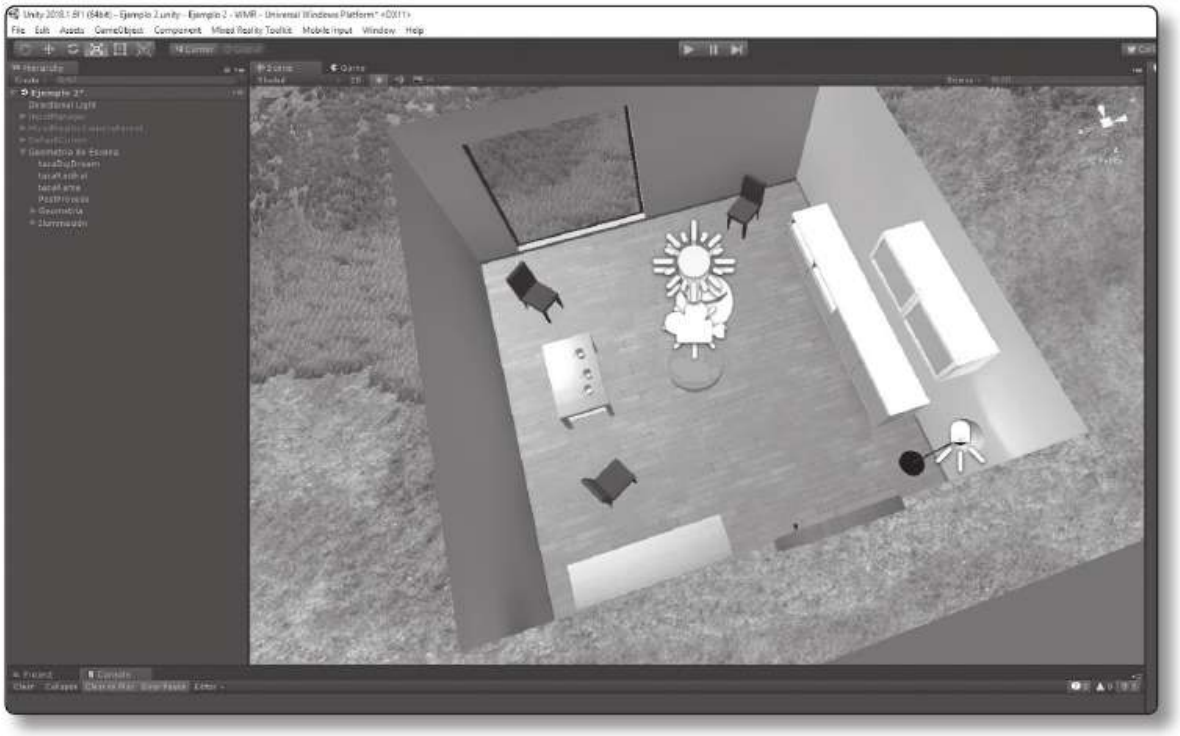


Figura 7.18. Vista de la escena desde el editor en nuestro ejemplo

Como podemos ver ya se han incluido (Si hemos "Aplicado" los cambios en *Apply Mixed Reality Scene Settings*) los *prefabs* correspondientes a la cámara (*MixedRealityCameraParent*), a los mandos (*InputManager*) y al cursor por defecto (*DefaultCursor*). Además, la geometría aparece centrada en nuestra escena y correctamente iluminada (Se ha incluido la iluminación también en el mismo *prefab* que la geometría).

Bastaría ejecutar el proyecto y ponernos las gafas para introducirnos dentro de la escena. Podemos girar la cabeza y mirar alrededor y, si tenemos los tenemos activos, aparecerán los mandos en nuestras manos.

Podemos comprobar también que el movimiento y los botones que se pulsan en los mandos en el mundo real se ven reflejados fidedignamente en el virtual.

Como movimientos básicos tenemos los siguientes:

1. Podemos rotar a la derecha o izquierda moviendo el *Touch* analógico del mando en el eje horizontal ← →

2. Nos desplazamos ligeramente hacia atrás moviendo el mismo *Touch* analógico hacia abajo ↓

3. Nos desplazamos a un lugar concreto (Marcado por una flecha 3D) moviendo el *Touch* analógico hacia arriba ↑ y luego soltando en el lugar que queremos.

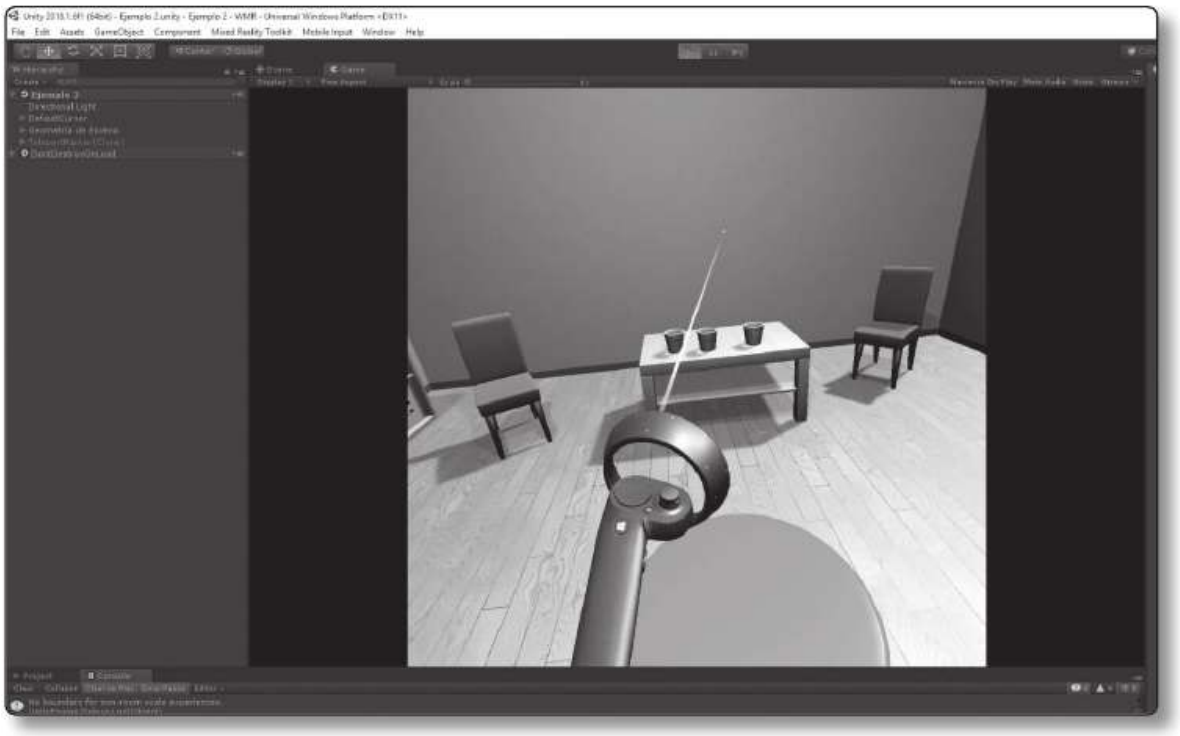


Figura 7.19. Escena en ejecución con el mando activado

Como vemos montar una escena básica es relativamente sencillo y existen *prefabs* para casi todo lo que pudiéramos necesitar.

Antes de continuar debemos de tener en cuenta un par de detalles que nos hemos encontrado y que conviene saber:

- ⇨ El primero es sobre la posición del *prefab* de la cámara. Al comienzo del capítulo comentamos que debíamos elegir la configuración *para todas las experiencias o recomendada* (Ilustración 4). Si no elegimos esta opción el punto de pivote de la cámara se nos queda a nivel del suelo. Para solucionar esto tenemos dos opciones, o configuramos la opción por defecto o elevamos el *gameobject MixedRealityCameraParent* en el eje Y.

- ⇨ La segunda tiene que ver con el funcionamiento de los mandos. En nuestro caso debemos de encenderlos antes de arrancar el editor de Unity 3D, aunque posteriormente cuando pasa cierto tiempo sin usarlos se apaguen. Si no hacemos esto nos encontramos que a posteriori el editor no es capaz de detectarlos.

Esto tiene que ver con nuestra experiencia con el dispositivo concreto que hemos usado para el desarrollo de este capítulo (Lenovo Explorer) y puede ser que con otros dispositivos no pase o que en la nueva versión del *Toolkit* quede solucionado.

Una vez desplegada la escena podemos comprobar que no podemos salir de la habitación en ningún momento. Esto sucede porque no podemos desplazarnos sobre nada que no disponga de un *Collider*. Tampoco podemos atravesar nada que disponga de un *Collider*. En nuestro caso la geometría de la propia habitación nos sirve como límite. Si anulamos el *gameobject* que muestra toda la geometría (En este caso con desactivar el objeto *Box001* dentro de *Geometría de escena* tenemos suficiente) vemos que ya no podemos desplazarnos con el mando (Situar la fecha 3D de la nueva ubicación). Como vemos el terreno no sirve de base para desplazarnos.

Esto es importante para poder limitar nuestro movimiento dentro de nuestro mundo virtual si fuese necesario y no atravesar las paredes, por ejemplo. En nuestro caso concreto hemos definido un objeto que son las paredes y la ventana que nos encapsulan, pero se puede definir mediante algo más simple. Un plano, por ejemplo.

Como ejercicio proponemos crear un plano 3D dentro de la escena que disponga de un *Collider* (*Mesh Collider*) y vemos que efectivamente podemos desplazarnos encima de el sin ningún tipo de problema. Si hacemos el plano lo suficientemente grande (En nuestro caso lo hemos centrado en el punto 0,0,0 y le hemos dado una escala de 10) vemos que tampoco podemos desplazarnos a los límites de este. Pero, ¿Por qué sucede esto si los límites los marcamos mediante c*olliders* como hemos dicho?

Como regla general podemos desplazarnos sobre cualquier objeto con un *collider*, pero además WMR dispone de unos límites que podemos marcar a nuestro antojo y que restringen aún más el área por la que podemos desplazarnos. Para delimitar estos límites se usa un objeto *Quad* (*Gameobject->3d Object->Quad*).

Para ver esto vamos a generar un nuevo objeto llamado **"Límites"** de tipo *Quad*. Centraremos el objeto en la escena, lo giraremos sobre el eje x en 90 grados y lo escalaremos a 20 para ver cómo queda.

Una vez hecho esto debemos de decirle al sistema que este objeto "marcará los límites*"*. Para esto último desplegamos el *Gameobject MixedRealityCameraParent* y seleccionamos *Boundary*. Es este *gameobject* disponemos de un script llamado *Boundary Manager*. Debemos arrastrar nuestro *gameobject* "**límites**" en la variable *Floor Quad* y vemos que ya podemos desplazarnos hasta los límites del objeto sin problema.

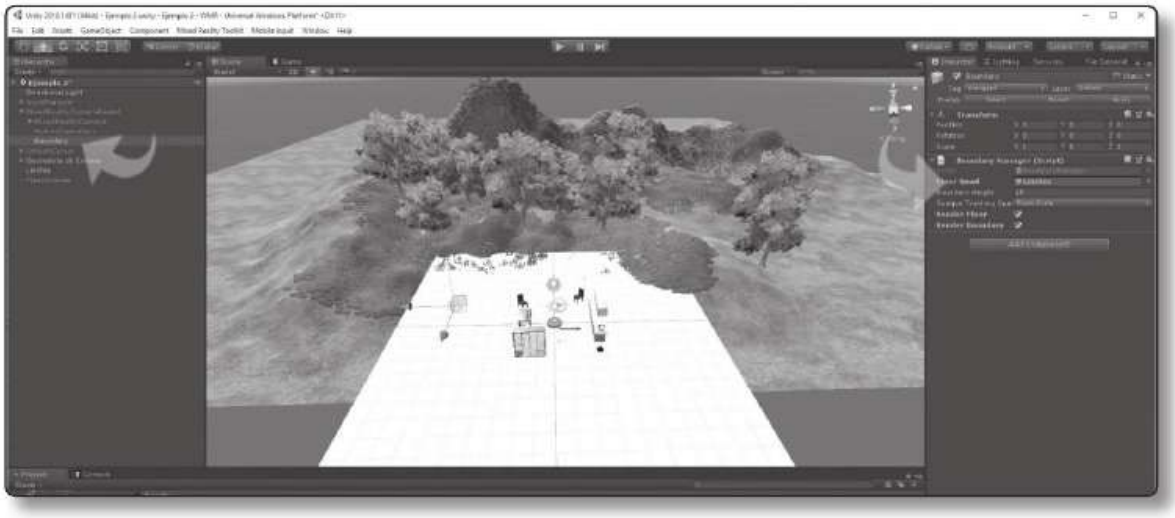


Figura 7.20. Ubicación del gameobject Boundary y de la variable FloorQuad

Una vez que sabemos desplazarnos por la escena y establecer los límites de esta vamos a aprender como interactuar con los *gameobjects* que la conforman. Para esto construiremos un selector de colores que cambiaran el color de la estantería de la escena.

Lo primero que necesitamos es el *Script cambioColor*. Lo tenemos disponible en el paquete **ScriptsYLibreriasWMR.unitypackage** de los contenidos adicionales. Para importarlo nos vamos a ***Assets→Import Package→Custom package***, lo seleccionamos y le damos al botón ***import***.

Este script se lo podemos asignar a cualquier mueble, pero en nuestro caso concreto lo haremos a la estantería. Para ello desplegamos el *gameobject* de la geometría, buscamos la estantería (***Geometría de escena→Geometría→prefab_estanteriaCajon→estantería***) y arrastramos el *script* en el inspector (El *script* está en la carpeta *Assets/Scripts*). Como vemos en el Inspector podemos configurar 3 colores. Básicamente lo que hacemos es tintar la estantería con un color concreto. El color base es el blanco para que la textura permanezca del color de la madera, como primer color pondremos el rojo y como segundo color el negro.

ⓘ NOTAS

Debemos de asegurarnos de que el Alpha de los colores elegidos este a 0 para que todo funcione correctamente más adelante.

Ya tenemos todo preparado para llamar a la función que cambia el color desde un *canvas*. En WMR podemos generar *canvas* integrados dentro de la escena y con los cuales se interactúa con los mandos. El *canvas* en si no tiene nada de particular para WMR y se genera como siempre, solamente que ahora queda integrado dentro del mundo virtual. Esto es especialmente útil para generar menús interactivos dentro de la escena. Veamos cómo hacerlo.

Creamos un *gameobject empty* llamado *AjustadorCanvas*, lo posicionamos en el (0,0,0) y le asignamos el *script* **SceneContentAdjuster** ubicado **en Assets\HoloToolkit\Scripts\SceneContentAdjuster**. Dentro de este *gameobject* creamos un objeto *canvas* (**UI→Canvas**) y le asignamos las siguientes propiedades:

⇨ *Render Camera = WorldSpace*

⇨ En *Rect Transform*

- *Width*: 70
- *Heigh*: 180
- *Scale*: 0.01,0.01,0.01

⇨ Comprobamos que dispone del *script* **Canvas Helper** y si no se lo asignamos (En este último caso arrastramos el mismo objeto sobre la variable *canvas* del script)

⇨ Añadimos el modificador **Grid Layout Group** (Pulsando en *Add Component* y buscándolo con el buscador) y le asignamos los siguientes parámetros:

- *Cell Size*: 60, 60
- *Start Axis*: Vertical
- *Child Aligment*: *Middle Center*
- *Constraint*: *Fixed Row Count*
  - *Constraint Count*: 3

Lo siguiente que tenemos que hacer es posicionar el *canvas* al lado de la estantería.

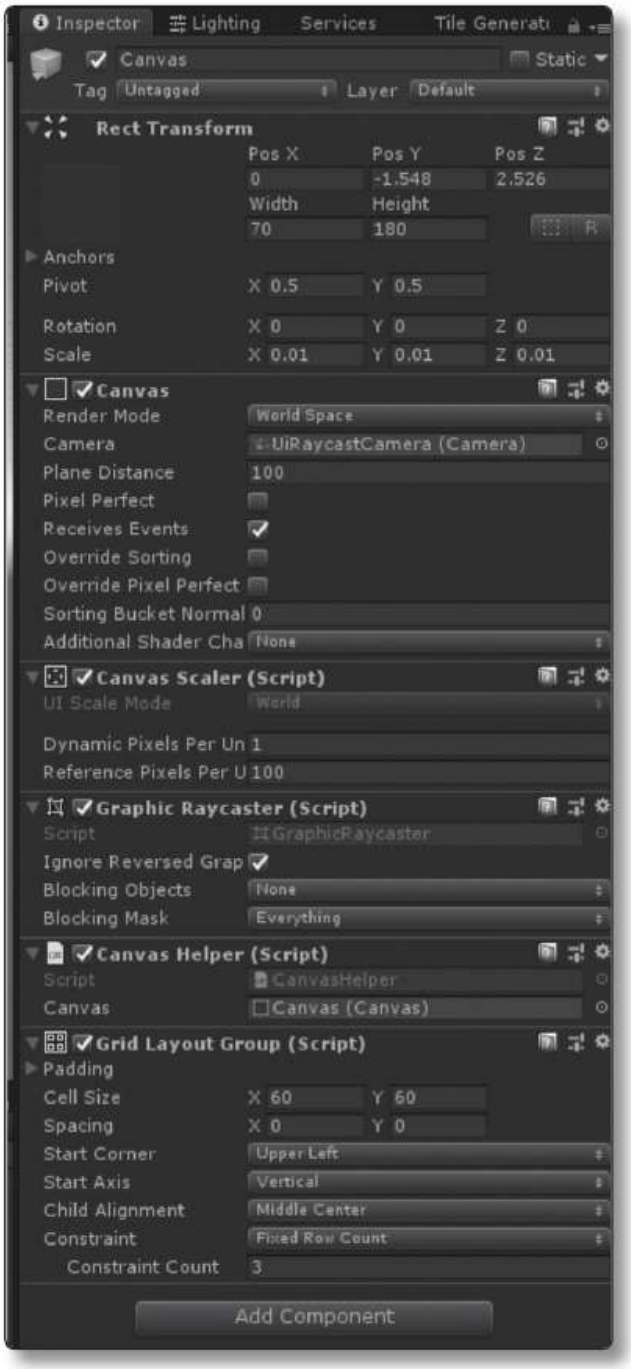


Figura 7.21. Configuración canvas

Ahora asignamos un **UI.Button** a nuestro *canvas* donde le ponemos el color Blanco. Cambiamos *Highlighted color* por un gris y le asignamos una acción a realizar en el *OnClick*. Para esto último basta con arrastrar el *gameobject* de nuestra estantería y elegir la función *ChangeColor ()* del script *cambioColor*. El valor que le asignamos al color para el blanco o "color madera" es el 0.

Con esto ya tendríamos definido el botón blanco. Para el rojo y el negro bastaría con duplicar el botón, cambiarle el nombre y el valor de la variable de *ChangeColor* a 1 (Para el rojo) o 2 (Para el negro).

Ya tenemos todo listo. Ahora arrancamos la escena y veremos los 3 colores al lado de la estantería. Si apuntamos con nuestro mando a cada uno de ellos se tornarán grises y al pulsar el gatillo veremos como la estantería cambia de color.

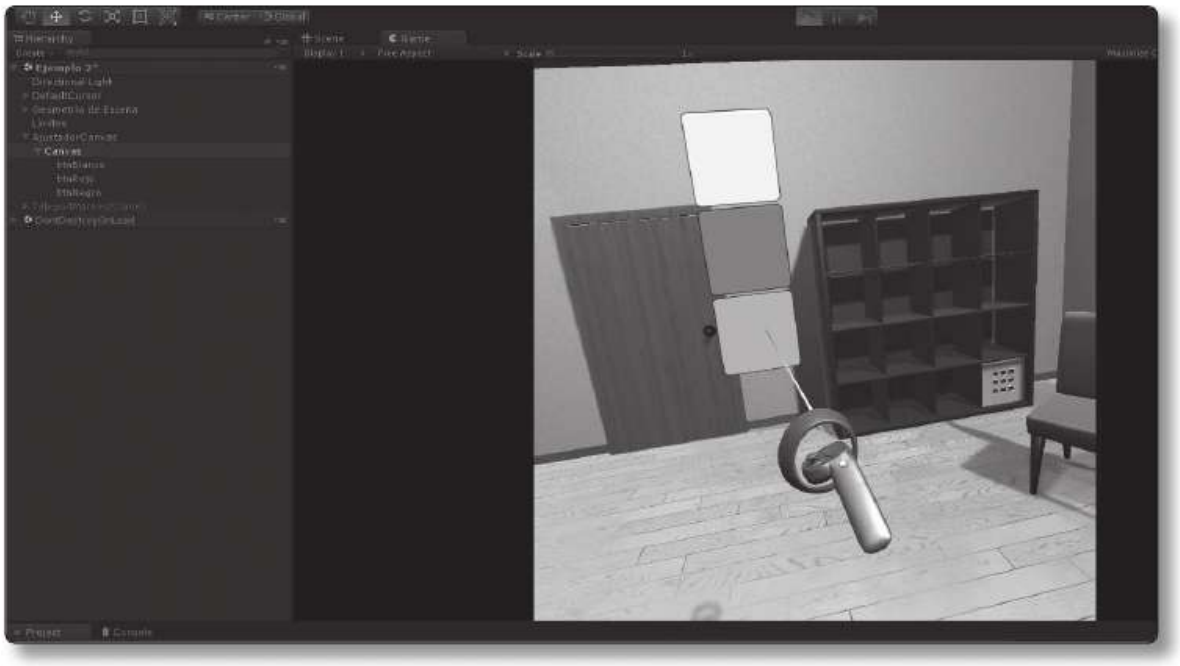


Figura 7.22. Selector de color en acción

Finalmente, para acabar con los mandos, vamos a ver como manipular espacialmente objetos.

Como podemos ver disponemos de unas tazas encima de la mesa. Para configurar un objeto para que sea manipulable por los mandos del dispositivo debemos de añadir 2 cosas:

1. Añadimos un *Box Collider* al objeto. Es importante que sea un *Box Collider* para que funcione todo correctamente. Una vez añadido en el inspector debemos de ajustarlo si es necesario en el botón *Edit Collider* y que las líneas verdes abarquen todo el objeto.
2. Añadimos el *script* **TwoHandManipulate** al *inspector* (**Assets\HoloToolkit\Input\Scripts\Utilities\Interactions\TwoHandManipulate**) y lo configuramos de la siguiente manera:
   - *Bounding Box Prefab*: Elegimos *BoundingBoxBasic* (Lo podemos encontrar en **Assets\HoloToolkit\UX\Prefabs\BoundingBoxes**)

- *Manipulation Mode*: *Move And Rotate*
- *Rotation Constraint: None*
- *Enable One Hand Move*: Activado

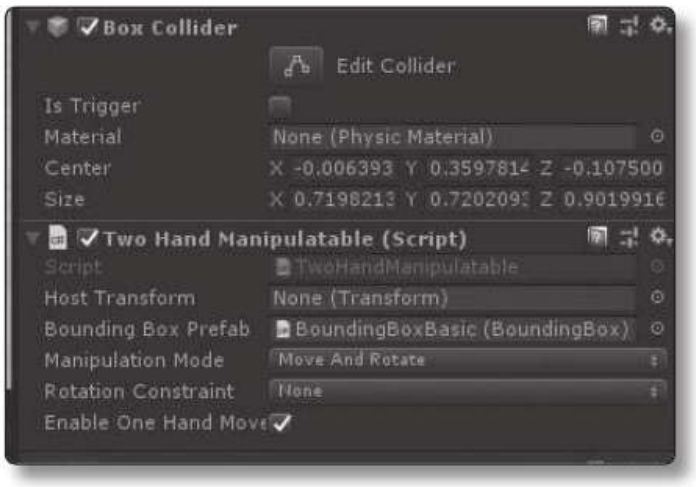


Figura 7.23. Configuración de la taza para ser manipulable

Ahora ya podríamos arrancar la escena y manipular las tazas. Para coger y trasladar debemos de apuntar al objeto, en este caso nuestra taza, y pulsar el gatillo 1. Para girar el objeto debemos de apuntar con los dos mandos al objeto y pulsar ambos gatillos (Gatillo 1 y 2) y mover los mandos para la rotación. Si cruzamos los mandos vemos como el objeto gira.

Figura 7.24. Ubicación de los gatillos o botones

7.7 EL CONTROL POR VOZ

En las aplicaciones de Realidad Virtual, la interfaz de usuario es bastante distinta a la de una aplicación convencional y en determinadas ocasiones realizar tareas como introducir comandos mediante un teclado virtual puede resultar tedioso, en otras ocasiones una interfaz basada en botones o menús puede no resultar adecuada para una buena experiencia de usuario.

Habitualmente, los usuarios pueden tener problemas con la interfaz de usuario debido a que la presentación de opciones menú, paneles o elementos de **UI**, pueden tapar elementos del mundo virtual, también pueden desaparecer de nuestro campo de visión si están asociados a un objeto cuando el usuario se mueve e incluso verse "del revés" cuando el usuario se sitúa detrás de ellos o molestar en el campo de visión debido a que los dispositivos tienen un campo de visión o FOV limitado y diferente según cada dispositivo.

En ocasiones toda esta casuística llamada, más que interfaz de usuarios, experiencia de usuario, puede llegar a resultar muy complicada, por eso es importante reducir en lo posible el número de elementos y la posible confusión o error con los mismos y por tanto en alguno de estos casos podemos utilizar como parte de esta interfaz de usuario el reconocimiento de voz para controlar la aplicación.