



Universidad de Valladolid



**ESCUELA DE INGENIERÍAS
INDUSTRIALES**

Universidad de Valladolid

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Diseño Industrial
y Desarrollo del Producto

**REALIDAD AUMENTADA PARA LA VISUALIZACIÓN
E INTERACCIÓN CON OBJETOS 3D EN UNITY**

Autor: Pablo García Fuente

Tutor: David Escudero Mancebo

Departamento de Informática
Área de Ciencia de la Computación
e Inteligencia Artificial

Valladolid, Julio de 2021

La realidad aumentada es una tecnología emergente e innovadora, que genera un interés progresivo en el ser humano, al mismo tiempo que se abre paso dentro de nuestra vida cotidiana, consolidándose como una de las tecnologías del futuro. Cada vez son más los sectores en los que esta tecnología encuentra aplicación, siendo el diseño industrial uno de ellos. En base a esta premisa, se presenta este trabajo como una toma de contacto con la realidad aumentada, sirviendo como una introducción al estudio de la misma, y ofreciendo las herramientas para la creación de aplicaciones Android en Unity. Mediante un tutorial se mostrará cómo desarrollar y programar la aplicación Demostrador AR, que permite la visualización de modelos 3D propios utilizando los servicios de realidad aumentada de ARCore, con la posibilidad de interactuar con estos objetos, activar sus animaciones y modificar el material aplicado.

Augmented reality is an emerging and innovative technology that has generated a progressive interest in humans; at the same time, it is making its way into our daily lives, consolidating itself as one of the technologies of the future. The areas in which this technology finds application has been increasing, being industrial design one of them. Based on this premise, this work is presented as a first contact with augmented reality, serving as an introduction to the study of it, and offering the tools for the creation of Android applications in Unity. This tutorial will show how to develop and program the AR Demonstrator application, which allows the visualization of 3D models using ARCore's augmented reality services, with the possibility of interacting with these objects, activating their animations and modifying the applied material.

Realidad aumentada – Unity3D – ARCore – Android – Aplicación

Augmented Reality – Unity3D – ARCore – Android – Application

ÍNDICE DE CONTENIDO

1. INTRODUCCIÓN.....	9
1.1 Objetivos.....	10
1.2 Visión general.....	10
2. REALIDAD AUMENTADA.....	11
2.1 Definición y funcionamiento	12
2.2 Orígenes.....	14
2.3 Estado del arte	17
<i>Aplicaciones en educación</i>	18
<i>Aplicaciones en el arte</i>	19
<i>Aplicaciones en arquitectura</i>	20
<i>Aplicaciones en publicidad</i>	21
<i>Aplicaciones en industria 4.0</i>	22
<i>Aplicaciones médicas</i>	24
<i>Aplicaciones en entretenimiento</i>	25
2.4 Proyección de futuro	26
2.5 Conclusiones.....	28
3. TECNOLOGÍA EMPLEADA	29
3.1 Unity	29
<i>Interfaz</i>	31
3.2 Realidad aumentada en Unity	37
3.3 El motor de realidad aumentada ARCore.....	41
<i>Conceptos fundamentales</i>	42
4. DESARROLLO DE APLICACIONES EN UNITY	45
4.1 Preparación del Hardware.....	45
<i>Compatibilidad con ARCore</i>	45
<i>Opciones de desarrollador y depuración por USB</i>	46
4.2 Preparación del Software	47
<i>Descargar ARCore</i>	47
<i>Descargar Unity</i>	47
4.3 Creación un proyecto en Unity	50
4.4 Configuración inicial	50
4.5 GameObjects y Prefabs.....	57

4.6	Elementos requeridos en la escena.....	58
4.7	Configuración del SceneController.....	61
4.8	Detección y visualización de planos.....	62
4.9	Instanciación y manipulación de objetos.....	63
4.10	Instalación de la Aplicación Base.....	69
5.	APLICACIÓN DEMOSTRADOR.....	71
5.1	Modelos 3D propios.....	71
5.2	Modelos con animaciones.....	76
5.3	Cambios en el material del modelo.....	85
5.4	Mejoras en la interfaz.....	89
5.5	Instalación de la Aplicación Demostrador.....	99
5.6	Usabilidad.....	103
6.	CONCLUSIONES.....	109
7.	BIBLIOGRAFÍA.....	111
	APÉNDICE 1: Código de la Aplicación Base.....	115
	APÉNDICE 2: Código de la Aplicación Demostrador.....	117

ÍNDICE DE FIGURAS

Figura 1. Reality–virtuality continuum de Milgram y Kishino.	12
Figura 2. Esquema de tareas de un sistema RA.	13
Figura 3. Diseño del Sensorama.	14
Figura 4. "La espada de Damocles" de Ivan Sutherland.	14
Figura 5. Logotipo oficial de Realidad Aumentada.	15
Figura 6. Aplicación turística de RA creada con Layar.	16
Figura 7. Imagen publicitaria de las Google Glass de 2012.	17
Figura 8. Vista de la aplicación Construct3D.	18
Figura 9. Vista de la aplicación ARART.	19
Figura 10. Dibujo recreado en 3D con Quiver.	19
Figura 11. Vista de la aplicación AR Sketchwalk.	20
Figura 12. Realidad aumentada de Nike para la personalización de zapatillas. ...	21
Figura 13. Trabajador de Airbus utilizando las HoloLens 2.	22
Figura 14. Trabajador de ThyssenKrupp visualizando un ascensor en RA.	23
Figura 15. Vista de un conductor con la RA de Volvo.	23
Figura 16. Vista de la aplicación SNAP en funcionamiento.	24
Figura 17. Ejemplo de aplicación de RA en los informativos.	25
Figura 18. Gráfico de la evolución en los ingresos esperados por la RA y RA.	26
Figura 19. Diferentes partes de una lente de contacto de RA.	27
Figura 20. Plataformas compatibles con Unity.	29
Figura 21. Interfaz por defecto de Unity.	31
Figura 22. Ventana de proyectos de Unity.	32
Figura 23. Ventana de escena de Unity.	33
Figura 24. Funcionamiento de las herramientas de transformación.	34
Figura 25. Ventana de juego de Unity.	34
Figura 26. Ventana de jerarquía de Unity.	35
Figura 27. Dos vistas diferentes de la ventana inspector de Unity.	36
Figura 28. Funciones compatibles de AR Foundation.	38
Figura 29. Ejemplo de escena en Unity MARS.	38
Figura 30. Ejemplo de aplicación creada con Unity y Vuforia.	39
Figura 31. Logotipo de ARCore.	41

Figura 32. Activación de la Depuración USB.	46
Figura 33. Aplicación de Servicios de Google Play para RA.	47
Figura 34. Sección Installs en Unity Hub.	48
Figura 35. Ventana de elección de versiones de Unity Hub.	48
Figura 36. Ventana de selección de plataformas de Unity Hub.	49
Figura 37. Ventana de instalación de Visual Studio.	49
Figura 38. Ventana de creación de proyecto en Unity Hub.	50
Figura 39. Instalación de paquetes.	51
Figura 40. Configuración de reproductor necesaria.	52
Figura 41. Configuración para que la aplicación soporte ARCore.	53
Figura 42. Configuración de calidad.	53
Figura 43. Configuraciones de sombras.	53
Figura 44. Configuración del Gradle en Unity.	54
Figura 45. Configuración del JDK.	56
Figura 46. Cuatro tipos diferentes de GameObject.	57
Figura 47. Vista del inspector para ARCore Device.	59
Figura 48. Vista del Inspector para First Person Camera.	59
Figura 49. Vista del inspector para el Environmental Light.	60
Figura 50. Vista del inspector para el SceneController.	61
Figura 51. Vista del inspector para el PlaneGenerator.	62
Figura 52. Vista del Inspector para el PlaneDiscovery.	62
Figura 53. Vista del inspector para el ObjectGenerator.	67
Figura 54. Vista del inspector para el prefab Manipulator.	67
Figura 55. Objetos de juego secundarios del prefab Manipulator.	68
Figura 56. Vista del inspector para el ManipulationSystem.	68
Figura 57. Jerarquía de la aplicación base.	69
Figura 58. Configuración de construcción para la aplicación base.	69
Figura 59. Ventana de instalación vía USB.	70
Figura 60. Vistas de la aplicación base en ejecución.	70
Figura 61. Silla modelada en Blender.	72
Figura 62. Ventana de exportación FBX en Blender.	73
Figura 63. Vista del modelo importado en Unity.	73
Figura 64. Vista completa del prefabricado del modelo.	74

Figura 65. Vista de la silla en realidad aumentada.....	75
Figura 66. Silla de escritorio con animaciones en Blender.....	76
Figura 67. Configuración para exportar animaciones en Blender.....	77
Figura 68. Vista de la silla de escritorio importada en Unity.....	77
Figura 69. Clips de animación en Unity.....	78
Figura 70. Componente Animator para la silla de escritorio.....	78
Figura 71. Animator Controller de la silla de escritorio.....	78
Figura 72. Vista inicial de la ventana Animator.....	79
Figura 73. Vista final de la ventana Animator.....	79
Figura 74. Vista del Inspector para el Canvas.....	81
Figura 75. Modificación del texto de un botón.....	82
Figura 76. Vista de Escena del Canvas en 2D.....	83
Figura 77. Configuración On Click para la animación Rotar.....	83
Figura 78. Materiales y texturas usados en la silla de escritorio.....	84
Figura 79. Capturas antes y después de presionar el botón INCLINAR.....	84
Figura 80. Nuevos materiales para la silla de escritorio.....	85
Figura 81. Vista del Inspector para MaterialController.....	87
Figura 82. imágenes de las sillas tipo Sprite.....	87
Figura 83. Componente Image de uno de los botones.....	87
Figura 84. Configuración On Click para cambiar a Tela Gris.....	88
Figura 85. Capturas de la silla de escritorio con materiales diferentes.....	88
Figura 86. Nuevo diseño de los botones para los cambios de material.....	89
Figura 87. Configuración final On Click para cambiar a Tela Gris.....	91
Figura 88. Parte inferior del Canvas definitivo.....	91
Figura 89. Toggle Planes activado (izq.) y desactivado (dcha.).....	92
Figura 90. Configuración On Value Changed para el Toggle Planes.....	93
Figura 91. Variables públicas del DetectedPlaneGenerator.....	94
Figura 92. Toggle EII activado (izq.) y desactivado (dcha.).....	94
Figura 93. Configuración On Value Changed para el Toggle EII.....	95
Figura 94. Imagen del mensaje inicial.....	96
Figura 95. Animación para la imagen con el mensaje.....	97
Figura 96. Posición inicial (izq.) y final (dcha.) del mensaje.....	97
Figura 97. Variables públicas del CanvasController.....	98

Figura 98. Configuración para identificación de la aplicación.....	99
Figura 99. Iconos de las aplicaciones anteriores.	100
Figura 100. Legacy Icons para la aplicación Demostrador.....	100
Figura 101. Icono de la Aplicación Demostrador.....	100
Figura 102. Configuración de la Splash Image.	101
Figura 103. Vista de la pantalla de carga de la aplicación.....	101
Figura 104. Capturas de la Aplicación Demostrador en funcionamiento.	102
Figura 105. Gráfico de valoración SUS [49].....	107

1. INTRODUCCIÓN

En la época que vivimos podemos ver como muchas de las tecnologías que surgieron hace décadas, prometiendo revolucionar por completo el mundo en que vivimos, son ya una realidad. Algunos casos que nos son cercanos son los teléfonos inteligentes, los ordenadores personales o el internet, que no solo se han hecho un hueco en nuestras vidas, sino que muchos no podrían imaginarse la vida sin ellos. Estas tecnologías han llegado para quedarse, al menos hasta que puedan ser sustituidas por otras superiores.

Es una evidencia que nuestro mundo cambia constantemente, los avances tecnológicos se suceden a un ritmo sin precedentes, y es por ello por lo que debemos poner la vista en las nuevas tecnologías que están llamadas a marcar el futuro de los próximos años. Algunas de ellas están cada vez más presentes en lo que se conoce como Industria 4.0 como: el BigData, la realidad aumentada, la fabricación aditiva, el internet de las cosas (IoT), los robots autónomos, etc. Sin embargo, los avances no se reducen solo al sector industrial, sino que existen también otros sectores con especial interés en el desarrollo de estas y otras tecnologías. Además, aunque los campos de investigación pueden llegar a ser muy diversos, como la industria militar, los videojuegos, la medicina o la educación, estos se retroalimentan y entre todos impulsan el cambio.

La Realidad Aumentada es la tecnología que aquí nos ocupa. Se trata de una innovadora tecnología que genera un interés progresivo en el ser humano y al mismo tiempo se abre paso dentro de nuestra vida cotidiana mediante la combinación del virtualismo con el realismo en diferentes formas. Esta tecnología permite un nivel de inmersión que ningún equipo virtual puede proporcionar.

Un problema común entre profesionales que se dedican al mundo del modelado 3D, ya sean diseñadores, arquitectos o ingenieros, es la percepción que se puede llegar a tener de un modelo 3D. A menudo es difícil interpretar de manera precisa cómo es su forma, tamaño, aspecto o, incluso, la manera en que se relacionaría con su entorno. Sin lugar a duda, la realidad aumentada tiene mucho que aportar a este campo, ofreciendo una nueva forma de visualización.

Más en concreto, en el sector del diseño industrial, esta tecnología está destinada a tener un papel importante, y no solo para el modelado 3D, sino también para la comunicación de ideas. Su aplicación puede afectar a todo el proceso creativo que hay detrás de un producto. Desde la concepción de la idea, pasando por el prototipado y llegando hasta la presentación del producto final.

1.1 Objetivos

Teniendo en cuenta las necesidades antes mencionadas del diseño industrial, así como las oportunidades que la realidad aumentada puede ofrecer, podemos entender la motivación de este proyecto, que tiene como objetivos principales:

1. La realización de un pequeño estudio de la realidad aumentada, explicando su funcionamiento, revisando su historia, sus aplicaciones en la actualidad y su proyección de futuro. Todo ello para demostrar la gran potencialidad de esta tecnología.
2. La introducción al uso de Unity para la creación de aplicaciones Android que utilicen los servicios de realidad aumentada de ARCore.
3. El desarrollo de una aplicación de realidad aumentada, llamada Demostrador AR, para la visualización de productos, que sirva como herramienta a todo tipo de profesionales relacionados con el diseño industrial durante todo el proceso creativo.
4. La elaboración de un tutorial en el que se explique cómo crear esta aplicación, intentando servir de ayuda a otros estudiantes que quieran utilizarla para presentar sus productos.
5. La evaluación de la usabilidad de la aplicación creada a través de los resultados obtenidos en una prueba de usabilidad realizada a diferentes personas.

1.2 Visión general

Para comprender la manera en que se ha estructurado este documento es necesario indicar que existen principalmente tres partes.

En la primera (Capítulo 2), se presenta una investigación a cerca de la realidad aumentada. En ella empezaremos definiendo sus fundamentos, para después pasar a estudiar sus orígenes, sus aplicaciones en la actualidad y, finalmente daremos algunas ideas sobre el prometedor futuro de esta tecnología.

La segunda parte (Capítulo 3) servirá para introducirnos en las tecnologías que emplearemos para nuestra aplicación. Veremos qué es Unity, cómo es su interfaz y cuáles son las principales formas de implementar la realidad aumentada en este software. A continuación, se explicará cómo funciona ARCore a través de algunos conceptos fundamentales.

La tercera parte (Capítulos 4 y 5) tendrá formato de tutorial. En él se explicará, en primer lugar, cómo preparar el hardware y software necesario. Después, crearemos una aplicación base con los elementos necesarios para experiencia de RA. Finalmente, aprenderemos a desarrollar e instalar una aplicación en la que introducir modelos propios, activar animaciones y cambiar el material aplicado.

2. REALIDAD AUMENTADA

Desde hace décadas, la síntesis de imágenes por computadora ha sido una herramienta que científicos, ingenieros y humanistas han utilizado para representar sus datos e ideas visualmente. Tanto la tendencia a la representación de los datos computacionales con imágenes, muchas de ellas tridimensionales, como la comodidad que supone la interacción con las máquinas mediante metáforas gráficas, han impulsado el desarrollo de software y hardware realmente sofisticados con los que conseguir una síntesis de imágenes tridimensionales con gran detalle visual.

El realismo alcanzado por los sistemas visuales se basa en el detalle gráfico o foto-realismo, en la simulación de las reglas de movimiento del mundo real (principalmente la física de fuerzas, gravedad, fricción), y en la respuesta de los elementos en el mundo virtual ante los estímulos o comandos del usuario (la interactividad) [1].

Los ambientes gráficos pueden ser reconstrucciones o réplicas virtuales de espacios existentes en la realidad, pero también mundos virtuales que, dependiendo de su nivel de abstracción, puede llegar a estar muy lejos de nuestros referentes en el mundo real.

Sin embargo, en los últimos años, se han afianzado estrategias que combinan ambos mundos, como la Realidad Mixta, que puede ser definida como el continuo que va desde la realidad hasta la Realidad Virtual (simulaciones inmersivas de ambientes reales), pasando por la Realidad Aumentada (representaciones de entidades reales aumentadas con información digital) y la Virtualidad Aumentada (mundos virtuales como videojuegos o *Second Life*, aumentados con representaciones de entidades reales) [2].

En el presente trabajo nos vamos a centrar en el área de la Realidad Aumentada. Hace pocos años era una tecnología poco conocida, ya que para su uso era necesario integrar diferentes tecnologías a través de dispositivos especializados, como las lentes de Realidad Aumentada, que incluyen un sistema de captura de video y uno de proyección que necesitan ser integrados por un software específico. Sin embargo, hoy en día ya es posible aprovechar dispositivos más accesibles, como teléfonos inteligentes y tabletas, que ya integran los componentes necesarios y pueden ser adaptados para desarrollar experiencias de Realidad Aumentada.

2.1 Definición y funcionamiento

La combinación de objetos y mundos reales y virtuales se conoce como el área de los mundos integrados o la realidad mezclada. Este área se basa en una estrategia de visualización e interactividad que hace uso de múltiples tecnologías, y que ofrece un espectro de modalidades que se mueven entre dos extremos: si el entorno del espacio (el ambiente circundante) es predominantemente virtual y se le agregan objetos virtuales y reales, hablamos de realidad virtual; mientras que en el caso de un entorno dominante real al que se le agregan objetos virtuales, hablamos de realidad aumentada.

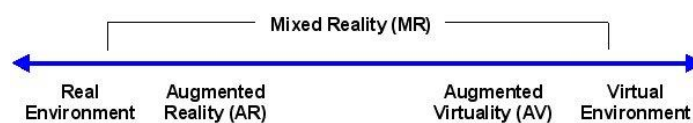


Figura 1. Reality–virtuality continuum de Milgram y Kishino.

En la figura 1 se muestra un gráfico que representa lo que Milgram y Kishino denominaron *Reality–virtuality continuum* en 1994. Consideran dos extremos, uno con entorno puramente real, y otro, con un entorno puramente virtual. Consideran que cualquier entorno que consiste en una mezcla de objetos reales y virtuales es Realidad Mixta (MR). Los entornos de realidad mixta donde el mundo real se aumenta con contenido virtual se denominan Realidad Aumentada (RA), mientras que aquellos en los que la mayor parte del contenido es virtual, pero existe cierta conciencia o inclusión de objetos del mundo real se denominan Virtualidad Aumentada (AV) [3].

Por lo tanto, podemos definir la realidad aumentada como una tecnología que integra señales captadas del mundo real (típicamente video y audio) con señales generadas por computadores (objetos gráficos tridimensionales); las hace corresponder para construir nuevos mundos coherentes, complementados y enriquecidos [1].

Sin embargo, estaríamos dando una definición incompleta si tratáramos de caracterizar la RA simplemente como la fusión de información virtual sobre el mundo físico. En este sentido, el investigador Ronald Azuma [4] [5] apunta que, además de la combinación de elementos reales y virtuales, la Realidad Aumentada también debe cumplir otras dos condiciones. Una de ellas es que tiene que ser interactiva en tiempo real, es decir, los objetos sintéticos agregados a una escena ayudarán al usuario a interactuar con el contexto. La otra es que debe ser registrada en 3D. Esto quiere decir que la información virtual tiene que estar vinculada espacialmente al mundo real de manera coherente. Se necesita saber en todo momento la posición del usuario respecto al mundo real y de esta manera puede lograrse el registro de la mezcla de información real y sintética.

TRABAJO DE FIN DE GRADO

APLICACIÓN DE REALIDAD AUMENTADA

En cuanto al funcionamiento de la Realidad Aumentada podemos decir, de forma sintetizada, que se trata de una combinación de dos tecnologías, los motores gráficos y la visión artificial [6]. El primero tiene como papel renderizar los contenidos, a menudo en 3D, que muestra la Realidad Aumentada. La visión artificial se encarga de que la ubicación de los contenidos aumentados sea la correcta en la escena, para que la composición sea coherente y entendible por el usuario.

El principal elemento que interviene en la parte gráfica es el renderizado. Este proceso consiste en la interpretación por parte del ordenador de una escena tridimensional para crear una imagen en dos dimensiones. Los datos que se procesan para llevar a cabo el render es la geometría del modelo 3D, su superficie (color y material), la iluminación de la escena y la localización de la cámara.

En cuanto a la visión artificial, existen técnicas muy variadas que se aplican a la Realidad Aumentada y se conocen generalmente como tecnologías de seguimiento o *tracking*. Existen muchos tipos, por ejemplo, el tracking facial, enfocado a detectar y seguir la posición de una cara, o el tracking de texturas, que posiciona una imagen de referencia en un sistema de coordenadas tridimensional. Las técnicas más novedosas que se utilizan en la actualidad incorporan sensores activos basados en luz estructurada, tracking SLAM (como ARCore) o tracking 3D.

Para que sea satisfactoria una experiencia de Realidad Aumentada, debe funcionar en tiempo real, como hemos indicado anteriormente. Esto significa que cada uno de los dos módulos debe hacerlo también. Por lo tanto, el módulo gráfico debe ser capaz de renderizar unas 60 imágenes por segundo, y el módulo de tracking de igual modo, ser capaz de analizar y extraer la información de 60 imágenes por segundo.

Se suele considerar que los sistemas de Realidad Aumentada funcionan de forma secuencial y ejecutan cuatro tareas, como se muestra en la Figura 2 [7].

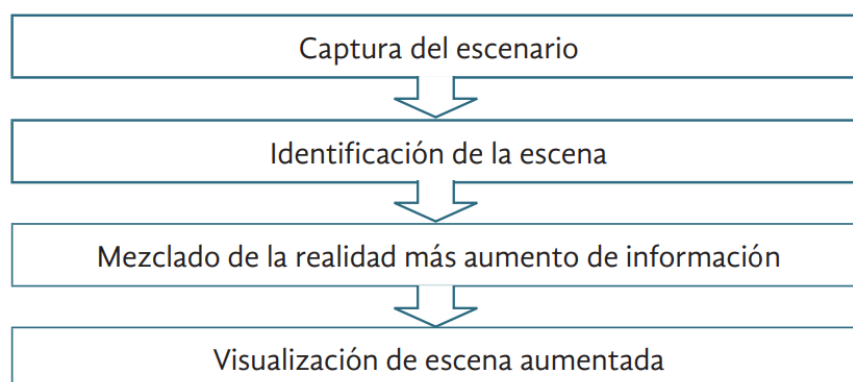


Figura 2. Esquema de tareas de un sistema RA.

2.2 Orígenes

Sería un error pensar que la Realidad Aumentada es una tecnología de naturaleza reciente, pues ya en los años 90 el término era conocido y se utilizaba en distintos campos que iban desde la Medicina, a la Aeronáutica, la Robótica y el Turismo. Incluso podemos remontarnos más atrás en el tiempo. La primera concepción de la Realidad Aumentada ocurrió en una novela de Frank L Baum escrita en 1901. En ella incluyó la idea de unas gafas electrónicas que superponían datos sobre personas que visualizaban, este invento era llamado “character maker” [8].

El segundo antecedente fue creado por el cinematógrafo Morton Heilig en 1962. Su invento llamado “Sensorama” incluía imágenes, sonido, vibración y olfato, en un intento de añadir información adicional a una experiencia.

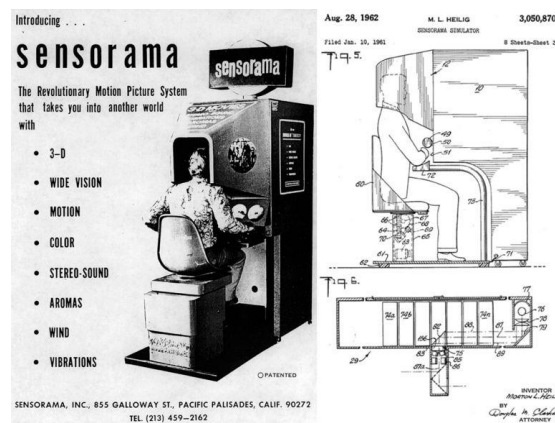


Figura 3. Diseño del Sensorama.

Años más tarde podemos encontrar otros casos como la conocida “Espada de Damocles”, inventada en 1968 por el profesor de Harvard Ivan Sutherland. Consistía en un artilugio que se colocaba sobre la cabeza (Head Mounted Display, HMD) y mediante el cual el usuario podía experimentar gráficos generados por ordenador [9]. En 1973, Ivan Sutherland inventará el casco de realidad virtual. Sin embargo, estos inventos se acercan más a la realidad virtual que a la realidad aumentada.



Figura 4. "La espada de Damocles" de Ivan Sutherland.

El primer gran desarrollo en realidad aumentada fue "Videoplace", creado en 1974 por Myron Krueger. Este invento combinaba un sistema de proyección y videocámaras que producían sombras, generando un entorno interactivo en una realidad artificial que rodeaba a los usuarios y respondía a sus movimientos y acciones [10].

Sin embargo, es en los años 90 cuando la realidad aumentada toma un verdadero impulso. Es en estos años cuando la RA incursiona en el mundo científico, en un contexto en el que la tecnología se centraba en los ordenadores de procesamiento rápido, el renderizado de gráficos en tiempo real y sistemas de seguimiento de posición portables [7]. Esto permite lograr la implementación y combinación de imágenes generadas por ordenador sobre la visión del mundo real. Es en este momento cuando los sistemas comienzan a usar la concepción actual de RA.

Fue en 1992 cuando el investigador Tom Caudell, de la empresa Boeing, acuñó el término "Realidad Aumentada" para referirse a los dispositivos utilizados por los electricistas de las fábricas aeronáuticas cuando tenían que realizar cableados complicados. Estos dispositivos eran diagramas digitales de RA sobrepuestos en un tablero donde se organizaba el cableado. A lo largo de esa década surgieron aplicaciones de desarrollo de RA, pero sus requerimientos técnicos y de costo no permitían un fácil acceso a esta tecnología.

En 1999 Kato y Billinghurst presentan por primera vez ARToolKit, un software de aplicación de código libre muy utilizado en la RA con más de 650.000 descargas en 2014, que captura acciones en el mundo real y las combina con objetos virtuales, permitiendo crear aplicaciones de Realidad Aumentada y teniendo una gran influencia sobre las aplicaciones actuales. Años más tarde, en 2009, ARToolKit es incorporado a Adobe Flash, con lo que la RA llega al navegador Web.

En el mismo año se crea el logotipo oficial de Realidad Aumentada con el fin de estandarizar la identificación de la aplicación de esta tecnología en cualquier soporte o medio por parte del público general.

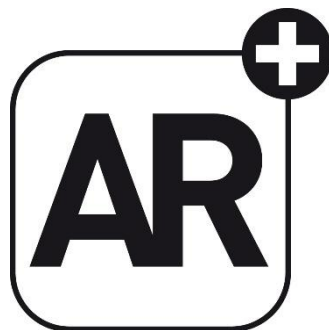


Figura 5. Logotipo oficial de Realidad Aumentada.

Ya en el siglo XXI, según la empresa Innovae [6], que tiene amplia experiencia creando experiencias de RA y RV, podemos clasificar el desarrollo de esta tecnología en tres etapas.

La primera de ellas es la RA en ordenadores personales. Aproximadamente en el año 2006, gracias al impulso del mundo de los videojuegos y las mejoras técnicas, con procesadores más potentes y tarjetas gráficas dedicadas, se hizo posible por primera vez la creación de experiencias de Realidad Aumentada de calidad a un precio razonable. Entre sus aplicaciones destacaron las relacionadas con el mundo del marketing.

La segunda etapa sería la RA en teléfonos inteligentes. La incursión en el mercado de smartphones supuso una verdadera revolución social y tecnológica que también actuó como catalizador en el crecimiento de la Realidad Aumentada. Estos dispositivos, junto con las tabletas, permitieron a los usuarios hacerse partícipes de experiencias de RA mientras se incrementaba el desarrollo de estas. En esta etapa aparecieron las primeras apps vinculadas a revistas, catálogos o carteles publicitarios. Pero también otras en el sector turístico, con herramientas como Layar o Wikitude, que agregaban información de la ciudad a una capa superpuesta a la cámara del móvil, en función de la orientación y la localización de un usuario, gracias al sistema de geolocalización y los sensores del móvil. También surgieron empresas que enfocaron sus aplicaciones a la formación y el entretenimiento.

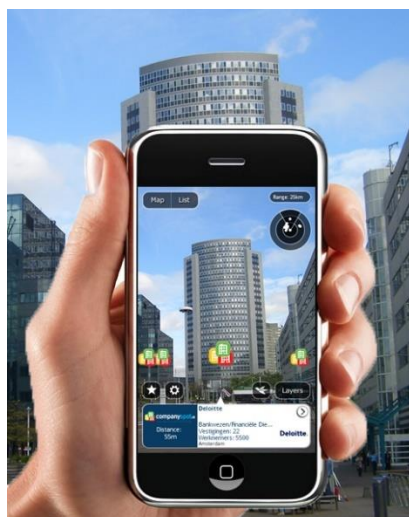


Figura 6. Aplicación turística de RA creada con Layar.

Por último, consideramos que la tercera etapa comienza cuando entran en juego las primeras gafas y visores RA. El primero que se promocionó a lo grande como un dispositivo de Realidad Aumentada, fueron las Google Glass en 2012, que a pesar del esfuerzo tecnológico no tuvo mucho éxito. Sin embargo, este fracaso provocó que se comenzara a conocer de forma masiva esta tecnología.

Posteriormente se han lanzado al mercado otros modelos de este tipo de dispositivos como HoloLens, Magic Leap o Nreal, que van que van confirmando la solidez que hay en la apuesta por esta nueva industria.



Figura 7. Imagen publicitaria de las Google Glass de 2012.

La RA ha demostrado ser una tecnología multidisciplinar que con los años ha irrumpido en diferentes ámbitos de aplicación. A pesar de ser todavía una tecnología emergente, ya podemos afirmar que la RA se encuentra omnipresente en nuestra vida cotidiana, siendo acogida con altos grados de aceptación entre los usuarios y compatible con diversos dispositivos tecnológicos.

2.3 Estado del arte

La Realidad Aumentada actualmente es considerada una de las más importantes tendencias tecnológicas, colocándose en un lugar prominente y siendo empleada en una gran variedad aplicaciones y contextos de uso.

Las aplicaciones pertinentes a la RA son aquellas que requieren la reformulación del mundo con información multidimensional, para presentar elementos virtuales que revelen conocimiento. Las principales aplicaciones se han dado en campos muy diversos. Los casos mejor documentados son en educación, el arte, entretenimiento, medicina, difusión de la ciencia y la tecnología, entrenamiento industrial, museos, narraciones interactivas, presentación de productos e industria militar.

A continuación, vamos a ver algunos de los ejemplos más desatacados en los principales campos de aplicación de la Realidad Aumentada.

Aplicaciones en educación

Uno de los aspectos más relevantes en la aplicación de realidad aumentada al mundo educativo es que aporta a los estudiantes un entorno interactivo con el conocimiento. La posibilidad de vincular a los materiales escolares información en tres dimensiones facilita en gran medida la comprensión de las materias y, además, permite aumentar la motivación de los estudiantes.

Incluso se han realizado numerosos estudios comprobando la eficacia de la realidad aumentada en personas con Trastornos del Espectro Autista (TEA). En la realidad aumentada tanto el factor auditivo como el visual están presentes en cada dinámica por lo que se produce un periodo más amplio de atención en la tarea. Se ha comprobado que la RA en personas con TEA trabaja habilidades como: la atención, el razonamiento, la reflexión, la resolución, la memoria y la concentración [11].

Quizá una de las aplicaciones más conocidas de la Realidad Aumentada en la educación sea el proyecto "Magic Book" del grupo activo HIT de Nueva Zelanda. El alumno lee un libro real a través de un visualizador de mano y ve sobre las páginas reales contenidos virtuales [12].

Por otra parte, existen proyectos relacionados con el mundo de la química, como el "Augmented Chemistry", que enseña a los estudiantes un átomo o molécula RA hasta que, al mover los marcadores, expone la reacción de seis ejemplos de metales al mezclarlos con cuatro de soluciones [13].

En cuanto a las aplicaciones de realidad aumentada enfocadas en la educación superior podemos encontrar proyectos de enseñanza de conceptos de ingeniería mecánica en combinación con Web3D o para enseñar geometría como "Construct3D" [14]. En la Figura 8 podemos ver esta última aplicación en funcionamiento.

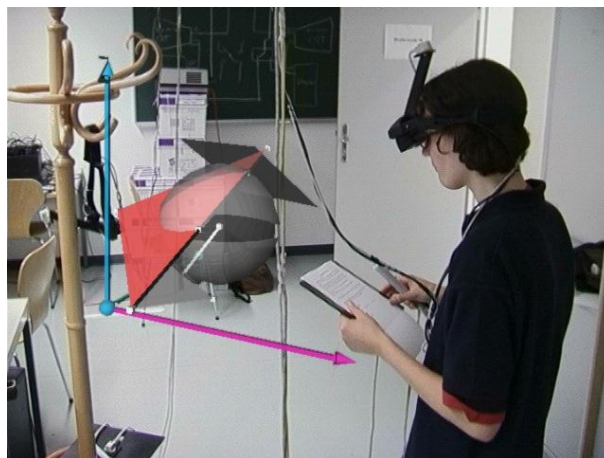


Figura 8. Vista de la aplicación Construct3D.

Aplicaciones en el arte

Otra de las disciplinas donde la realidad aumentada tiene una incidencia cada vez mayor es el arte. Aquí podemos destacar “ARART”, que en palabras de sus creadores “presenta una nueva plataforma de expresión que conecta la creación artística con la realidad”. Una de sus funciones más populares es la animación de pinturas. ARART permite que se desarrolle una nueva historia, como si el tiempo atrapado dentro de la pintura hubiera cobrado vida. Ofrece un vistazo a la historia oculta que se esconde detrás del cuadro [15].



Figura 9. Vista de la aplicación ARART.

Otro ejemplo de aplicación, aunque para niños, es “Quiver”. Esta aplicación reproduce en tres dimensiones los dibujos creados sobre el papel. Se parte de unos dibujos impresos, que los niños colorean y que, escaneándolos con su aplicación móvil, toman forma 3D y reproducen animaciones.



Figura 10. Dibujo recreado en 3D con Quiver.

Aplicaciones en arquitectura

La arquitectura es otro de los campos en los que se ha encontrado aplicación para la realidad aumentada. Son cada vez más las empresas de este sector que apuestan por esta tecnología. De hecho, alrededor del 12% de las empresas españolas que apuestan por la RA se dedican al diseño o construcción de elementos arquitectónicos [16].

La realidad aumentada en arquitectura permite ver cómo quedará una obra arquitectónica añadiendo de forma virtual los elementos que luego se incorporarán de forma real. Esto implica un enorme ahorro de tiempo y dinero, a la vez que ofrece mayores garantías tanto a arquitectos como clientes.

En este campo vamos a destacar tres aplicaciones. La primera es “AR Sketchwalk”, que permite realizar diseños y bosquejos para que el cliente puede tener una visión mucho más aproximada de la obra, implementando los diseños realizados sobre el plano real. De esta manera el cliente pueda interactuar en tiempo real.



Figura 11. Vista de la aplicación AR Sketchwalk.

En segundo lugar, tenemos la aplicación “ARKI”, un software cuya finalidad es mostrar modelos en 3D sobre planos de planta en 2D, permitiendo la interacción con los diseños.

Por último, “SmartReality”, que es una app similar a la anterior, pero que también permite ver modelos en 3D creados a partir de elementos virtuales sobre planos impresos. Además, se puede ver las distintas etapas de un proyecto, visualizar las diferentes capas del diseño o grabar en vídeo su experiencia.

Aplicaciones en publicidad

La Realidad Aumentada ha supuesto un gran avance para el sector publicitario, usando como herramienta los teléfonos inteligentes y fusionando publicidad con entretenimiento. El juego publicitario crea un ambiente que conecta al cliente con el producto. De esta manera se produce un mayor impacto y aumentan las posibilidades de éxito.

Una tienda de Nike en París ha utilizado la realidad aumentada de una forma muy innovadora. A través dispositivos basados en la proyección, el cliente puede personalizar las zapatillas a través de un ordenador, smatphone o tablet. Todos los colores y diferentes patrones elegidos se proyectan sobre una zapatilla real expuesta en el mostrador [17].



Figura 12. Realidad aumentada de Nike para la personalización de zapatillas.

Otra posibilidad consiste en aumentar directamente los productos, por ejemplo, en un concesionario, se podría ver de forma superpuesta a cada coche, información adicional que muestre equipamientos opcionales o partes del motor que no se encuentran a la vista. En el stand de Toyota Polonia en el Salón del Automóvil de Poznań se utilizó una pantalla de rayos X que escaneaba el coche Prius Plug-in Hybrid y mostraba cómo funcionaba su interior. La instalación disponía con un panel táctil para que los usuarios pudieran cambiar el color de la carrocería, mirar el interior de los coches y comparar todas las versiones.

Los catálogos aumentados son también muy habituales dentro de las aplicaciones para el marketing. Permiten al usuario entender mejor como funciona un producto, como son los pasos para montarlo o como se integra en un conjunto mayor.

Aplicaciones en industria 4.0

Sin duda, la industria está siendo uno de los sectores en los que antes se está adoptando la tecnología de realidad aumentada. Aplicaciones en ámbitos como el mantenimiento, la fabricación, el marketing, la logística o la formación son ya habituales en el mundo de la industria 4.0.

Los principales usos que se le da a la RA están relacionados con la presentación de procedimientos guiados paso a paso, que orientan a un operario a la hora de realizar un montaje o una reparación, o con sistemas de teleasistencia que facilitan la resolución de incidencias de forma remota, gracias a que un experto puede ayudar de forma más efectiva gracias a las instrucciones en pantalla basadas en esta tecnología en el objeto sobre el que se trabaja.

Una empresa pionera en el uso de la realidad aumentada es Airbus, que aplica esta tecnología de múltiples formas. En 2019 comenzó una colaboración con Microsoft para implementar su tecnología holográfica de RA y, de hecho, se llegaron a identificar más de 300 casos de uso donde esta tecnología podría ser muy útil [18].

La realidad aumentada permite a los aprendices aeroespaciales aprender en un entorno virtual inmersivo sin necesidad de encontrarse en una aeronave o utilizar piezas físicas reales. Por otra lado, HoloLens ayuda a los diseñadores de Airbus a evaluar de forma virtual sus diseños y comprobar si se encuentran listos para la fase de fabricación. La RA acelera sustancialmente este proceso y disminuye el tiempo que se invierte para ello, normalmente en un 80 por ciento.



Figura 13. Trabajador de Airbus utilizando las HoloLens 2.

Podemos encontrar otro ejemplo en la inspección de aviones militares que Airbus realiza junto con el Ejército del Aire español. La tecnología se basa en drones equipados con sensores y cámaras de alta definición que escanean, en tan solo unas horas en lugar de días, el exterior de una aeronave que sea sometida a una inspección de mantenimiento. Por medio de una conexión segura se pueden

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

mostrar en tablets y en gafas de realidad aumentada los datos y la información que se generan [19].

Otra empresa que apuesta decididamente por el uso de la realidad aumentada es ThyssenKrupp, que en 2016 anunció también la incorporación de Microsoft HoloLens a la actividad de sus 24.000 técnicos de mantenimiento. De esta forma se permite la visualización e identificación de los problemas de los ascensores, sin necesidad de desplazamientos, y tener acceso remoto a la información técnica lo que implica una capacidad de maniobra y operatividad cuatro veces más rápida que sin la utilización del dispositivo [20].



Figura 14. Trabajador de ThyssenKrupp visualizando un ascensor en RA.

En el año 2019, la empresa automovilística Volvo mostró un adelanto de una innovadora tecnología desarrollada junto con Varjo, otra empresa que se dedica a la fabricación de visores de RA de alta calidad. De esta forma, se podría evaluar con mayor precisión y exactitud prototipos, diseños y tecnologías de seguridad activa, pudiendo conducir un automóvil real mientras se usa uno de estos dispositivos de realidad mixta que agrega elementos virtuales que parecen reales tanto para el conductor como para los sensores del coche [21].



Figura 15. Vista de un conductor con la RA de Volvo.

Aplicaciones médicas

Uno de los aspectos más potentes de la realidad aumentada es la capacidad de mostrar a través de diferentes dispositivos cosas que no son normalmente visibles. En el ámbito de la medicina, esta posibilidad tiene multitud de aplicaciones.

Por ejemplo, un cirujano podría tener información ampliada del paciente mientras está realizando una operación. Incluso podría superponer su cuerpo unas guías que le muestren por donde debe intervenir gracias al uso de imágenes médicas con realidad aumentada (TAC, radiografía o resonancia magnética).

Una de las ventajas de mayor interés de las gafas de realidad aumentada para los entornos quirúrgicos es la posibilidad de manejarlas con la voz y los gestos, por lo que no sería necesario tocar nada.

La plataforma de navegación quirúrgica avanzada "SNAP" permite a los médicos mostrar sus "planes" para una cirugía. Está conectado a los sistemas de navegación de sala de operaciones estándar y proporciona capacidades de imagen avanzada, incluyendo múltiples puntos de vista 3D, que permiten a los cirujanos ver su caso tanto desde la perspectiva de un microscopio como con otra vista desde detrás del tumor [22].



Figura 16. Vista de la aplicación SNAP en funcionamiento.

También se están probando en la medicina aplicaciones de realidad aumentada para la visualización de datos clínicos. Aplicaciones cuyo objetivo es el visionado ágil de datos clínicos a través de ultrasonidos, imágenes de tomografía, obteniendo una visión más precisa de esos datos, pudiendo mejorar diagnósticos y facilitar la toma de decisiones para posibles intervenciones quirúrgicas. Por ejemplo "AccuVein AV400" ayuda a los profesionales sanitarios a localizar las venas de los pacientes con mayor precisión.

Aplicaciones en entretenimiento

Uno de los campos más estudiados y explotados donde la tecnología de la realidad aumentada se encuentra bastante extendida es en sistemas y aplicaciones de entretenimiento. Esto es debido a que es un mercado que arroja un alto porcentaje de rentabilidad.

Hoy en día ya son muchos juegos los que cuentan con la aplicación de la RA, algunos pioneros basados en la ARToolKit (una biblioteca que permite la creación de RA) como "AquaGauntlet", "ARHockey" o "AR-Bowling", entre otros muchos. Pero sin duda, el ejemplo más destacado es "Pokémon GO", que pulverizó todos los récords de descargas en unos pocos días después de su lanzamiento.

Sin embargo, el futuro del entretenimiento pasa por el uso de elementos hardware más inmersivos, como los HMD (Head Mounted Display), que por el momento no son rentables debido a su alto coste. No obstante, siguiendo el camino en innovación de las principales compañías desarrolladoras de videojuegos, es posible que en una próxima generación de videoconsolas se incluyan ya estos dispositivos, de forma que lleguen masivamente al público general.

Por otro lado, las retransmisiones deportivas son otro de los ámbitos en los que podemos ver muchos ejemplos de la aplicación de la RA. Piscinas, campos de fútbol, pistas de carreras o cualquier otro entorno deportivo puede ser preparado fácilmente para el uso de esta tecnología.

Además de esto, la RA cada vez se utiliza más en los espacios informativos de muchas cadenas de televisión, como es el caso de Atresmedia, que ha apostado por la aplicación de esta tecnología con el propósito de lograr captar la atención del espectador y emplear así un nuevo lenguaje [23].



Figura 17. Ejemplo de aplicación de RA en los informativos.

2.4 Proyección de futuro

Basta con observar algunos datos de mercado para darnos cuenta de que el sector de la realidad aumentada se encuentra en pleno auge, como podemos observar en el gráfico de la Figura 18. En el mismo puede apreciarse la notable diferencia de crecimiento entre esta y la realidad virtual. Los datos son suficientemente ilustrativos sobre la tendencia de mercado [24]. Sin embargo, hay que tener claro que ambas industrias no son competidoras sino totalmente complementarias.

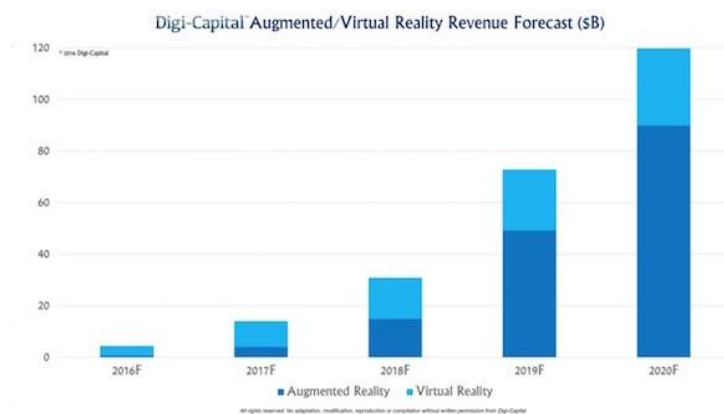


Figura 18. Gráfico de la evolución en los ingresos esperados por la RA y RV.

Es difícil llegar a comprender hoy en día en qué medida nos va a cambiar la vida, de aquí a unos años, la llegada de nuevas aplicaciones de realidad aumentada. Se puede establecer un paralelismo con los profundos cambios que los teléfonos inteligentes han traído a nuestra vida cotidiana: era imposible imaginarlos para la generación de los 90.

El hardware que sirve de soporte a la RA se halla en pleno desarrollo y aún le queda mucho margen de mejora. No obstante, empiezan a aparecer proyectos que permiten hacernos una idea sobre futuro de esta tecnología. Es primordial enfatizar que el futuro de la realidad aumentada no reside en las pantallas de los teléfonos o en un par de gafas, sino en sistemas completamente discretos que ostentan una portabilidad extrema. En este sentido las lentes de contacto de RA son el principal ejemplo. Estas disponen de filtros duales capaces de permitir que el ojo enfoque las cosas que están cerca y las que están lejos al mismo tiempo. El filtro central dirige la luz desde el HUD en el centro de la pupila del usuario. Al mismo tiempo, el filtro exterior dirige la luz a la llanta de la pupila permitiendo que ambas imágenes lleguen al ojo al mismo tiempo.

Diversas empresas se encuentran investigando distintos tipos de recursos que hagan posible la concretización de tecnologías aún más innovadoras dentro del campo de la inmersión. Innovega está desarrollando una lente de contacto

TRABAJO DE FIN DE GRADO
APLICACIÓN DE REALIDAD AUMENTADA

llamada iOptik que proporcionará el paso crucial necesario para percibir vista aumentada. Con estos lentes de contacto, el usuario puede ver el mundo natural, sus ojos pueden moverse normalmente y no existen gafas pesadas para dificultar el movimiento o la percepción [25].

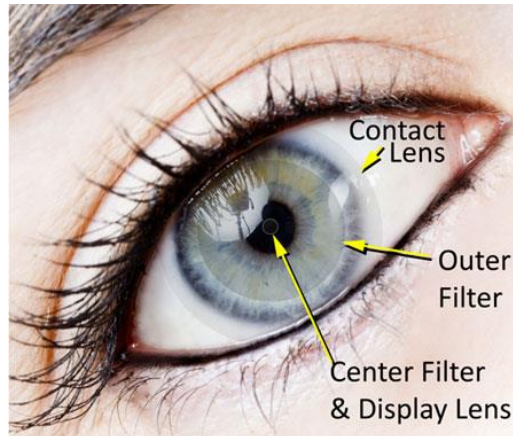


Figura 19. Diferentes partes de una lente de contacto de RA.

Por otro lado, los sistemas de RA más actuales requieren de usuarios expertos para operarlos, si las aplicaciones de realidad aumentada se convirtieran en sistemas de usos comunes y frecuentes en las personas en su vida cotidiana, entonces, los sistemas deberían ser cómodos, discretos, amigables y operables por usuarios no expertos. Por lo tanto, se requiere de sistemas más robustos, que eviten la calibración y los requisitos de configuración.

Cabe destacar también que los investigadores y desarrolladores de realidad aumentada se han centrado principalmente en aumentar el sentido de la vista. En una proyección al futuro, los entornos de RA pueden requerir la participación de otros sentidos como el tacto, audición, olfato o hasta el gusto. Con esto podemos inferir una amplia variedad de aplicaciones en el campo de la medicina, donde personas con discapacidades podrían interactuar normalmente con el mundo con la ayuda de estos nuevos sistemas. Definitivamente el modo de vivir de las personas cambiaría por completo, pudiendo muchas de ellas vivir, sentir o experimentar nuevas experiencias que antes por alguna incapacidad se tornaba casi imposible.

La nueva era de la información está llegando a su punto álgido. Las posibilidades empresariales en este ámbito son enormes y asistiremos al desarrollo de muchas de ellas durante los próximos años.

2.5 Conclusiones

Con todo lo visto en este capítulo hemos aprendido, en primer lugar, qué es lo que se conoce como Realidad Aumentada y cuál es, a grandes rasgos, el funcionamiento de esta tecnología. También hemos realizado un pequeño repaso de sus orígenes, mostrando que, aunque se trata de un campo relativamente nuevo, su investigación data ya de varias décadas atrás. Los avances tecnológicos han provocado que en solo unos pocos años la RA se abra paso y se asiente como una tecnología a tener en cuenta en cada vez más sectores. Por esta razón, se han enumerado también varios ejemplos de su uso correspondientes a los principales campos en los que la realidad aumentada tiene aplicación. Por último, se han expuesto algunas de las líneas de investigación que con probabilidad se seguirán en un futuro, orientadas a conseguir una mejor experiencia inmersiva y al alcance de cada vez más sectores.

A pesar de las limitaciones con las que todavía cuenta, esta tecnología crece y se innova constantemente, razón por la cual podemos suponer que la realidad aumentada tiene un futuro prometedor y seguirá siendo por lo pronto un campo recurrente de investigación en los próximos años.

Sin duda, en el ámbito del diseño industrial esta tecnología puede tener un gran potencial, ya sea para facilitar tareas de diseño, para comunicar ideas o para visualizar productos terminados de cara a la venta. Es esta la razón de ser del presenta proyecto, que pretende poner a disposición de los diseñadores del producto (principalmente) una herramienta con la que visualizar y exponer al mundo sus propios productos.

3. TECNOLOGÍA EMPLEADA

En este capítulo se realizará una presentación de la tecnología que posteriormente se empleará para el desarrollo de nuestra aplicación. Veremos qué es Unity, cómo es su interfaz, las diferentes posibilidades para crear RA en él y, por último, nos centraremos en el motor de realidad aumentada ARCore.

3.1 Unity

Unity es un software de desarrollo de videojuegos en tiempo real. Esta herramienta, creada por Unity Technologies, engloba motores para renderizar imágenes, motores de audio y motores de animación. En la actualidad es la plataforma líder en la creación de contenido interactivo en tiempo real [26] [27].

Unity está disponible como plataforma de desarrollo para Microsoft Windows, Mac OS y Linux. Cuenta con soporte de compilación con diferentes tipos de plataformas:

- Microsoft (Windows, Xbox 360, Phone)
- OS X
- Linux
- Play Station (PS3, PS4, PS Vita)
- Nintendo (Wii, Wii U)
- Iphone
- Android



Figura 20. Plataformas compatibles con Unity.

La primera versión de Unity se lanzó en la Conferencia Mundial de Desarrolladores de Apple en 2005. En un principio fue creado exclusivamente para funcionar y generar proyectos en los dispositivos de la plataforma Mac. Dado el gran éxito que obtuvo, se continuó con el desarrollo del motor y sus herramientas.

En septiembre de 2010 fue lanzada la versión 3 de Unity y se centró en empezar a introducir más herramientas que los estudios de alta gama por lo general tienen a su disposición, con el fin de captar el interés de los desarrolladores más grandes.

La versión, Unity 5, fue lanzada a principios de 2015 e incluye añadidos como Mecanim animation, soporte para DirectX 11 y soporte para juegos en Linux y arreglo de bugs y texturas [28].

A continuación, se enunciarán algunas de las principales características de Unity:

- Puede usarse junto con Blender, 3ds Max, Maya, Softimage, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks y Allegorithmic Substance. Si se realizan cambios en los modelos generados en cualquiera de estos programas, todas las instancias de ese objeto se actualizan automáticamente sin tener que volver a importarlo manualmente.
- El motor gráfico utiliza OpenGL (para los sistemas operativos Windows, Mac y Linux), Direct3D (solo en Windows), OpenGL ES (para dispositivos con Android y iOS), e interfaces propietarias (como la consola Wii). Tiene soporte para mapeado de relieve, mapeado de reflejos, mapeado por paralaje, oclusión ambiental en espacio de pantalla, sombras dinámicas utilizando mapas de sombras, render a textura y efectos de post-procesamiento de pantalla completa.
- Se usa el lenguaje ShaderLab para la creación de sombreadores. Pueden escribirse shaders en tres formas distintas: como Surface shaders, como Vertex and Fragment shaders, o como shaders de función fija.
- Soporte integrado para Nvidia, el motor de física PhysX, (a partir de Unity 3.0) con soporte en tiempo real para mallas arbitrarias y sin piel, ray casts gruesos, y las capas de colisión.
- El script se basa en Mono, la implementación de código abierto de .NET Framework. Los programadores pueden emplear UnityScript (un lenguaje personalizado inspirado en la sintaxis ECMAScript), C# o Boo (que tiene una sintaxis inspirada en Python).
- Incluye Unity Asset Server, una solución de control de versiones para todos los assets de juego y scripts.
- Cuenta con un amplio manual de usuario, que se va actualizando a medida que se actualiza el software.

Interfaz

A continuación, vamos a identificar los elementos principales de la interfaz de Unity [29] que encontraremos por defecto al abrir el programa. En la siguiente imagen aparecen enumeradas distintas zonas de la interfaz:

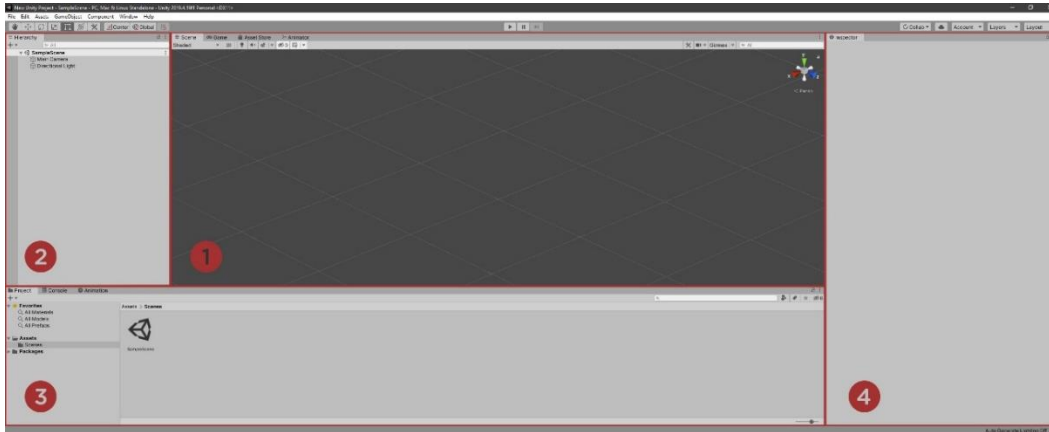


Figura 21. Interfaz por defecto de Unity.

1. En la vista de **Escena** es donde se construye mediante objetos las escenas del juego y permite una navegación visual del mismo. Además, puede mostrar una perspectiva 2D o 3D dependiendo del tipo de proyecto.
2. Esta es la ventana de **Jerarquía**, que es una representación de texto jerárquico de cada objeto en la escena. Cada uno tiene una entrada en la dentro de esta ventana, por lo que está inherentemente vinculada a la vista de escena. La jerarquía revela la estructura de cómo los objetos están agrupados el uno al otro.
3. La venta la de **Proyecto**, muestra nuestros *assets* de librería que están disponibles para ser usados. Cuando importamos nuevos *assets* al proyecto, estos aparecen aquí. La parte izquierda contiene una estructura jerárquica de carpetas.
4. Para visualizar y editar todas las propiedades del objeto seleccionado tenemos la ventana del **Inspector**. También nos permite añadir nuevos parámetros y configurar algunos mediante *scripts*.

El *layout* es la disposición concreta de la interfaz. Podemos elegir uno predefinido en el menú del mismo nombre que se encuentra en la parte superior derecha, o bien personalizarlo nosotros añadiendo las ventanas necesarias.

Ahora que tenemos una visión general de la interfaz vamos a describir con mayor detalle cada una de estas ventanas principales y algunas otras que también utilizaremos.

Proyecto (Project)

Esta ventana muestra todos los archivos relacionados con nuestro proyecto. Es un reflejo de las carpetas que lo constituyen y que se encuentran organizadas en el ordenador. A través de ella podemos navegar y encontrar todos los activos (assets) presentes en él, como escenas, scripts, modelos 3D, texturas, archivos de audio y prefabricados.

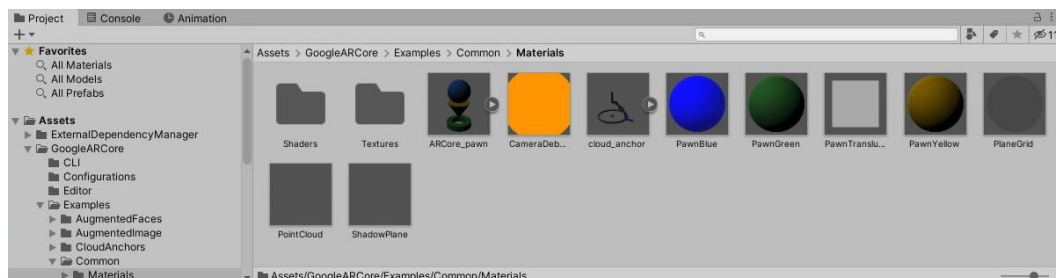


Figura 22. Ventana de proyectos de Unity.

El panel izquierdo muestra la estructura de carpetas del proyecto como una lista de jerarquía. Cuando una carpeta es seleccionada de una lista, su contenido se muestra en el panel a la derecha.

Los assets individuales aparecen en el panel de la mano derecha como iconos que indican su tipo (script, material, subcarpeta, etc.). Los iconos pueden ser redimensionados usando el deslizador que está en la parte inferior del panel. El espacio a la izquierda del deslizador muestra el elemento actualmente seleccionado, incluyendo una ruta completa al elemento si se está realizando una búsqueda.

En la parte superior se encuentra la barra de herramientas. En ella podemos realizar búsquedas de activos, ya sea por su nombre, por el tipo de elemento o la etiqueta (*label*) que tengan asignada. En el extremo izquierdo tenemos un botón que permite crear una gran variedad de elementos para el proyecto.

Escena (Scene View)

La vista de escena es la ventana en la cual podemos interactuar con el mundo virtual que estamos creando. Esta ventana es utilizada para crear las escenas del proyecto, seleccionar y posicionar personajes, luces, cámaras y todos los demás tipos de objetos de la escena. Manipular y modificar objetos dentro de la vista de escena es una de las funciones más importantes en Unity.

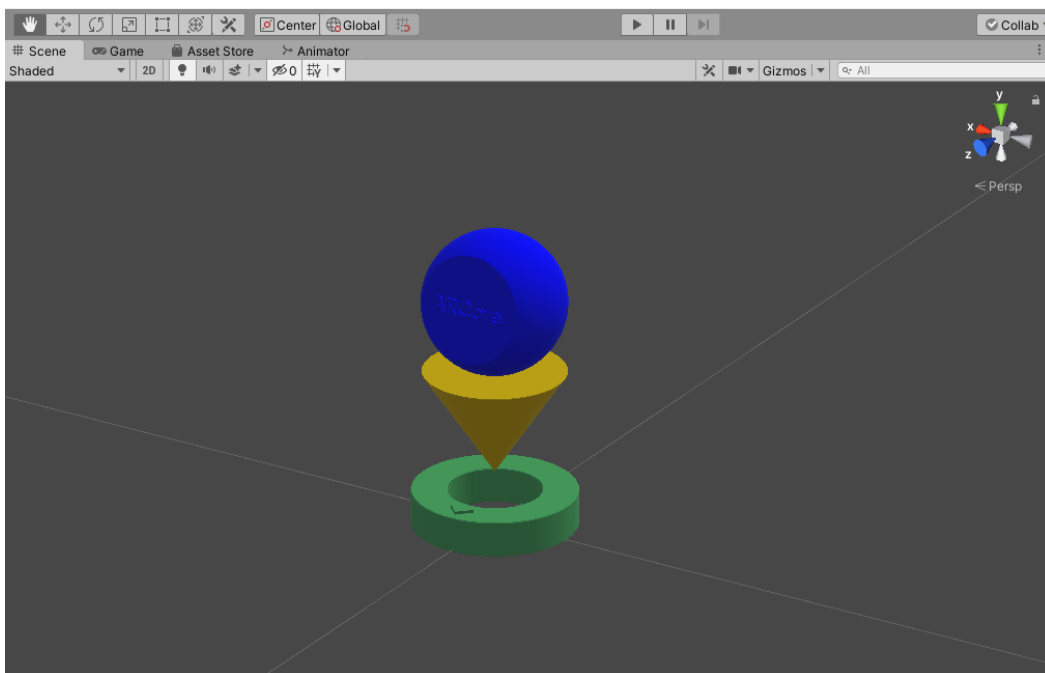


Figura 23. Ventana de escena de Unity.

Para poder trabajar de forma fluida dentro de esta vista es necesario poder navegar por el espacio tridimensional de manera cómoda. En Unity existe un botón de navegación en la esquina superior izquierda, representado por una mano. Seleccionando esta herramienta tenemos tres estados diferentes:

- Mover: arrastrando con el botón izquierdo del ratón.
- Orbitar: pulsando *Alt + botón izquierdo del ratón*.
- Zoom: pulsando *Alt + botón derecho del ratón*.

También podemos posicionar nuestra cámara del visor de escena a través del Gizmo que se encuentra en la parte superior derecha. Interactuando con él podemos visualizar en todos los ejes de coordenadas e incluso cambiar entre dos modos de visualización; perspectiva y ortográfica.

A la derecha del botón de navegación existen otros botones, denominados herramientas de transformación, con diferentes funciones para modificar los objetos de la escena, como trasladarlos, escalarlos o rotarlos. Cualquier transformación realizada con estas herramientas se verá reflejada en el componente de transformación asociado al objeto modificado, en la ventana del inspector.

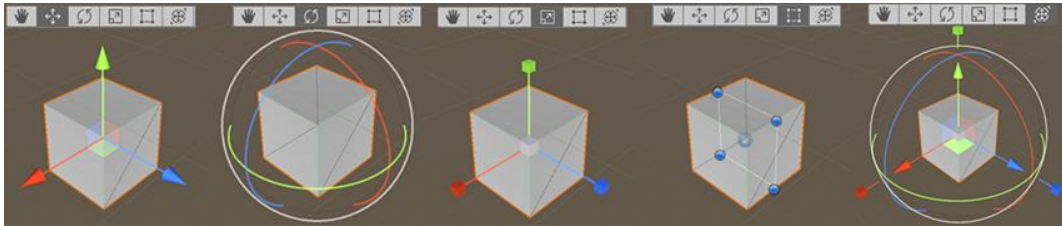


Figura 24. Funcionamiento de las herramientas de transformación.

Dentro de esta ventana también existe una barra de control con varias opciones. Entre otras cosas, permite modificar el aspecto con que se muestran los objetos (*Shaded, Wireframe, etc.*), o cambiar entre una vista 3D y 2D.

Juego (Game View)

A través de esta vista podemos visualizar una escena renderizada, es decir, el resultado de la compilación de todos los objetos, animaciones, luces, etc. Todo ello es generado desde la(s) cámara(s) de la escena.

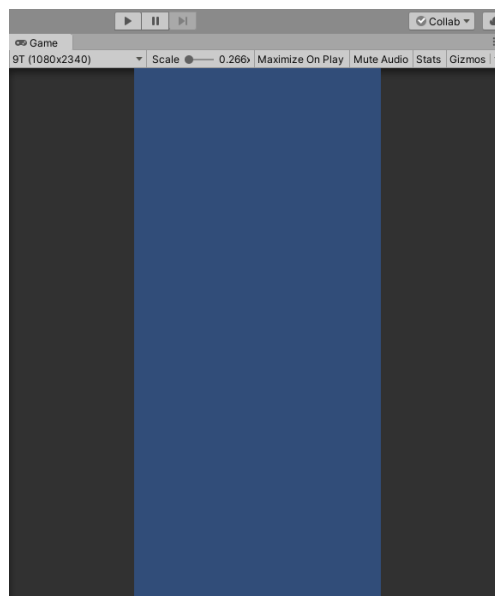


Figura 25. Ventana de juego de Unity.

Los tres botones superiores sirven para controlar el modo de reproducción. Cuando se está ejecutando, todos los cambios que se realicen serán temporales, y será inicializados de nuevo al finalizar.

En la barra de control podemos modificar algunos parámetros como la resolución o la escala (que permite hacer zoom) y activar o desactivar la visualización a pantalla completa, el audio, las estadísticas y los gizmos.

En nuestro caso esta vista no nos será especialmente útil ya que, al tratarse de una aplicación de realidad aumentada, la ventana no mostrará los resultados de la cámara de nuestro dispositivo móvil y por ello aparecerá con un fondo azul. Sin embargo, sí podemos visualizar elementos UI como botones.

Jerarquía (Hierarchy window)

La ventana jerarquía contiene cada GameObject (objetos, luces, cámaras) que se encuentra dentro de la escena existente. Algunos de estos son instancias directas de archivos de asset como modelos 3D, y otras son instancias de Prefabs. Cuando se añade un objeto a la escena éste aparecerá en la lista y desaparecerá al eliminarlo.

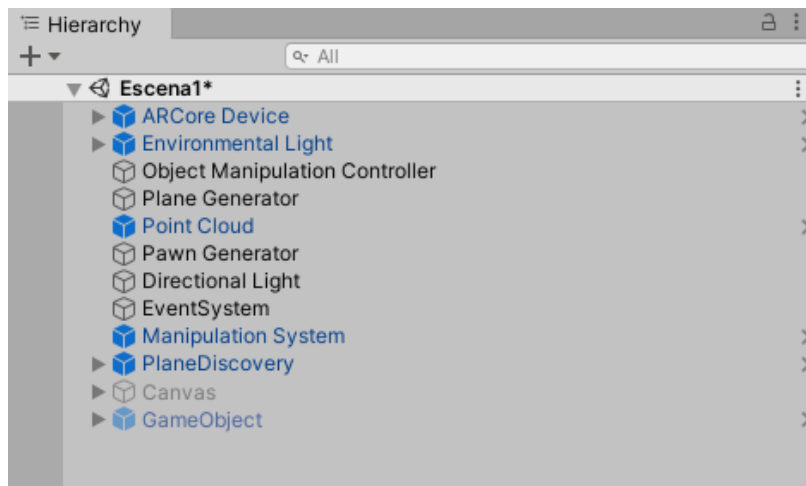


Figura 26. Ventana de jerarquía de Unity.

La lista se crea en el orden en que se van agregando o creando objetos, pero podemos cambiarlo con tan solo arrastrar los objetos encima o debajo. Además de esto, también podemos emparentar objetos. Unity usa un concepto llamado *Parenting*. Para hacer que cualquier GameObject sea hijo de otro, basta con arrastrar el hijo sobre el padre deseado. El hijo heredará el movimiento y la rotación de su padre.

Inspector (Inspector window)

Todo lo que contiene una escena en Unity son GameObjects. Cuando se selecciona uno de ellos en la jerarquía o en la escena, el inspector va a mostrar todos los componentes (*components*) y materiales que tiene asociado el objeto y permite modificarlos y reordenarlos.

Cualquier propiedad puede ser modificada directamente. Incluso las variables de los scripts pueden ser modificadas sin cambiar el script. Se pueden cambiar las variables mientras el juego se está ejecutando para realizar pruebas. Si en un script hay definida una variable pública de un tipo objeto, se puede arrastrar y soltar el GameObject o Prefab dentro del Inspector para asignarlo como variable del script.

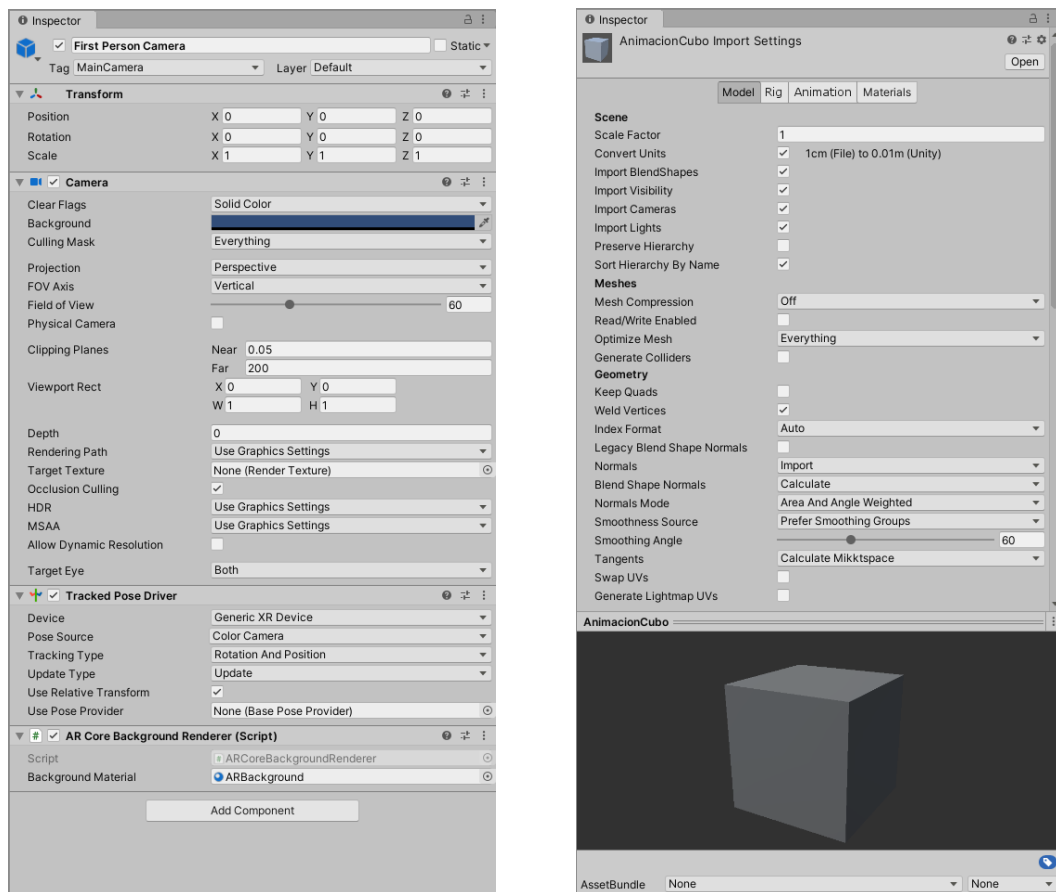


Figura 27. Dos vistas diferentes de la ventana inspector de Unity.

Cuando seleccionamos un asset en la ventana de proyecto, lo que mostrará el inspector serán las configuraciones que controlan la forma en que Unity importa y usa el asset en tiempo de ejecución. Cada tipo de activo tiene su propia configuración.

Además, en la parte superior también tenemos algunas otras configuraciones como activar/desactivar el objeto, cambiar su nombre o asignarle una determinada etiqueta o capa.

Debido a la gran cantidad de components que podemos encontrar en Unity, tan solo explicaremos los necesarios para nuestra aplicación a medida que se vayan implementado.

Otras ventanas

Las ventanas explicadas anteriormente nos sirven para comprender la interfaz básica de Unity, sin embargo, existen algunas otras vistas que podemos utilizar para tareas más específicas:

- Console: muestra mensajes, advertencias y errores.
- Animation View: para animar objetos en la escena.
- Animator View: para crear, organizar y conectar estados en un controlador de animación.
- Profiler: Usado para investigar y encontrar el performance de los cuellos de botella en el juego.
- Asset Server View: Usado para administrar versión del proyecto usando el Asset Server de Unity.
- Ligthmapping View: Usado para administrar iluminación de mapas.
- Occlusion Culling View: Usado para administrar el Occlusion Culling y mejorar su performance.

3.2 Realidad aumentada en Unity

Unity comenzó siendo desarrollada como una plataforma de creación de videojuegos multiplataforma, sin embargo, ha evolucionado hasta llegar a convertirse en una herramienta mucho más polivalente que ofrece muchas otras posibilidades, entre las que se encontramos la creación de aplicaciones de realidad aumentada [30].

Para desarrollar este tipo de aplicaciones se complementa con otros kits de desarrollo de software. A continuación, se muestran algunas de los principales complementos para crear realidad aumentada en Unity.

AR Foundation

AR Foundation incluye funciones centrales de ARKit, ARCore, Magic Leap y HoloLens, así como funciones únicas de Unity, para crear aplicaciones robustas. Este marco de trabajo permite aprovechar todas estas funciones en un flujo de trabajo unificado [31]. Además, AR Foundation se puede integrar con Unity MARS.

Functionality	ARCore	ARKit	Magic Leap	HoloLens
Device tracking	✓	✓	✓	✓
Plane tracking	✓	✓	✓	
Point clouds	✓	✓		
Anchors	✓	✓	✓	✓
Light estimation	✓	✓		
Environment probes	✓	✓		
Face tracking	✓	✓		
Meshing			✓	✓
2D Image tracking	✓	✓		
Raycast	✓	✓	✓	
Pass-through video	✓	✓		
Session management	✓	✓	✓	✓

Figura 28. Funciones compatibles de AR Foundation.

Unity MARS

Unity Mars es una herramienta para creación de realidad aumentada en Unity que permite la creación de apps inteligentes basadas en esta tecnología. Gracias a esta solución se pueden llevar los datos del entorno y de los sensores al flujo creativo, lo que permite la creación de apps inteligentes de AR que interactúen con el mundo real mediante información virtual de cualquier tipo.

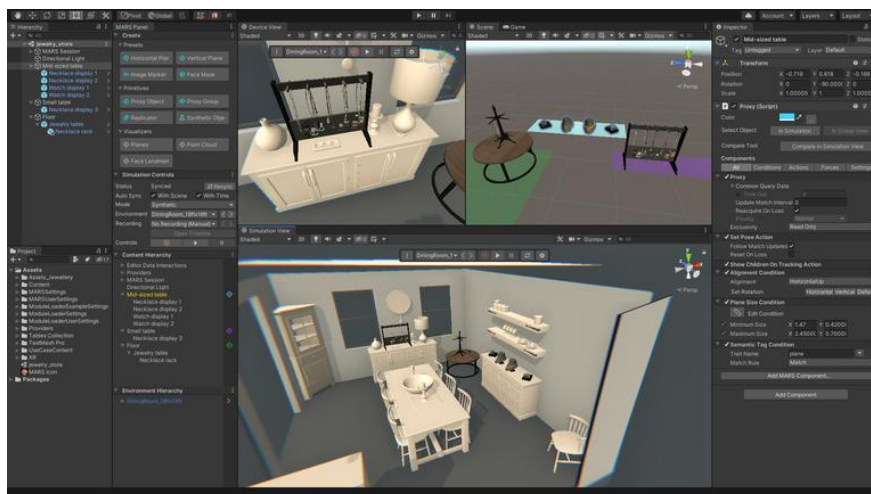


Figura 29. Ejemplo de escena en Unity MARS.

Entre sus principales ventajas se encuentra la posibilidad de crear apps de realidad aumentada complejas y orientadas a objetos, probar las funcionalidades de las apps sin tener que salir del editor y desarrollar aplicaciones AR que funcionen en diferentes plataformas [32].

Biblioteca Unity

Gracias a esta biblioteca se pueden integrar muchas funcionalidades y elementos de Unity en las aplicaciones de realidad aumentada ya creadas, sin estar obligado a reconstruir desde cero la aplicación. No solo nos permite añadir elementos de realidad aumentada, sino también imágenes en 2D y 3D renderizadas en tiempo real, minijuegos y mucho más [33].

XR Interaction Toolkit

Permite agregar funciones de interactividad a las apps de realidad aumentada en Unity. De este modo, ya no es necesario tener que programar estas interacciones desde cero [34].

Vuforia Engine

Se trata de un kit de desarrollo de software (SDK) para crear aplicaciones de realidad aumentada. Los desarrolladores pueden agregar fácilmente funcionalidad avanzada de visión por computadora a cualquier aplicación, lo que permite reconocer imágenes y objetos e interactuar con espacios en el mundo real. Admite el desarrollo de aplicaciones AR para dispositivos Android, iOS, Lumin y UWP [35].



Figura 30. Ejemplo de aplicación creada con Unity y Vuforia.

Unity Ads

Permite añadir anuncios o promociones a las apps de realidad aumentada con Unity. Así se pueden crear campañas publicitarias originales que consigan un mayor impacto de los clientes y contribuyan a mejorar la imagen de marca.

RA instantánea

Es un servicio especial de Unity que permite a los usuarios crear videojuegos o contenidos de realidad aumentada que sean pequeños, rápidos y ligeros. Esta plataforma todavía se encuentra en fase de desarrollo.

ARCore

Para el desarrollo de la aplicación del presente proyecto utilizaremos la plataforma creada por Google para crear experiencias de realidad aumentada.

ARCore proporciona dos SDK diferentes para implementar su tecnología en Unity a fin de desarrollar aplicaciones AR para Android:

- AR Foundation:
Como se explicó anteriormente, es un marco multiplataforma que le permite crear experiencias de realidad aumentada para dispositivos Android o iOS. En este caso es necesario la instalación de las extensiones ARCore XR Plugin y ARCore Extensions.
- SDK de ARCore para Unity:
Se trata de un SDK independiente que permite usar las funciones de ARCore orientado a Android. Se proporciona como un paquete independiente (*.unitypackage*). Este SDK no es compatible con AR Foundation [36].

En este caso se ha elegido la segunda opción, es decir que trabajaremos con las herramientas proporcionadas por el SDK para crear nuestra aplicación de realidad aumentada. Más adelante se explicarán los pasos a seguir para descargarlo e instalarlo en Unity.

3.3 El motor de realidad aumentada ARCore

ARCore es la plataforma desarrollada por Google para crear experiencias de realidad aumentada [37]. Proporciona diferentes APIs (Interfaces de Programación de Aplicaciones) para que, dentro de sistemas de desarrollo, como Unity o Unreal, se puedan crear estas experiencias. ARCore hace posible que nuestro teléfono pueda detectar su entorno, entender el mundo que lo rodea e interactuar con la información. Algunas de las APIs están disponibles en Android e iOS para permitir experiencias de RA compartidas [38].



Figura 31. Logotipo de ARCore.

El funcionamiento de ARCore se basa en tres capacidades fundamentales para integrar contenido virtual en el mundo real tal y como es visto desde la cámara de nuestro teléfono. Estas capacidades son el seguimiento de movimiento, la comprensión ambiental y la estimación de luz, que se describirán más adelante.

La tecnología de seguimiento de movimiento de ARCore utiliza la cámara del teléfono para identificar puntos interesantes, que se denominan características, y realiza un rastreo para determinar cómo se mueven esos puntos a lo largo del tiempo. Combinando el movimiento de estos puntos y las lecturas de los sensores inerciales del teléfono, ARCore es capaz de determinar tanto la posición como la orientación del teléfono a medida que se mueve por el espacio.

Además de identificar puntos clave, ARCore puede detectar superficies planas, como una mesa o el suelo, y también puede estimar la iluminación promedio en el área a su alrededor. Estas capacidades se combinan para permitir que ARCore construya su propia comprensión del mundo que lo rodea.

Esta interpretación del mundo real le permite colocar objetos, anotaciones u otra información de una manera que se integra perfectamente en el entorno físico.

ARCore necesita que el móvil tenga instalado Servicios de Google Play para RA, disponible en Google Play. No obstante, las aplicaciones que necesitan ARCore pueden invocar su instalación. La ventaja de ser una API universal es que los desarrolladores pueden usar sus funciones sin tener que crear su propia implementación desde cero.

Conceptos fundamentales

A continuación, vamos a realizar un desglose más detallado del funcionamiento de ARCore a partir de sus conceptos fundamentales [39].

Rastreo de movimiento

Para poder comprender cual es la posición del teléfono en relación con el mundo que lo rodea, ARCore utiliza un proceso denominado localización y mapeo simultáneos (SLAM) a medida que nuestro dispositivo móvil se mueve por el mundo. Consiste en detectar características visualmente distintas en la imagen capturada por la cámara llamadas puntos de características y usar estos puntos para calcular el cambio en la ubicación. La información visual se combina con mediciones inerciales de la IMU (Unidad de Medición Inercial) del dispositivo para estimar la pose (posición y orientación) de la cámara en relación con el mundo a lo largo del tiempo.

Al hacer coincidir la pose de la cámara virtual que renderiza el contenido 3D con la posición y orientación de la cámara del dispositivo proporcionada por ARCore, se consigue representar el contenido virtual desde la perspectiva correcta. La imagen virtual renderizada se superpone sobre la imagen obtenida de la cámara del dispositivo, haciendo que parezca que el contenido virtual es parte del mundo real.

Comprensión ambiental

ARCore mejora constantemente su comprensión del entorno del mundo real mediante la detección de planos y puntos característicos. ARCore busca grupos de puntos de características que parecen estar en superficies horizontales o verticales comunes, como mesas o paredes, y además puede determinar los límites de cada plano. Estos planos quedan disponibles para su aplicación, así como la información acerca de sus límites. Toda esta información puede ser utilizada para colocar objetos virtuales apoyados en superficies planas.

En el caso de intentar reconocer superficies planas sin texturas, como una pared blanca, es posible que no ARCore no consiguiera detectarla ya que su tecnología de detección de planos se basa en el reconocimiento de puntos característicos de las superficies.

Comprensión de profundidad

Otra de las posibilidades que ofrece ARCore es la creación de mapas de profundidad, es decir, imágenes que contienen información sobre la distancia entre superficies desde un punto dado, utilizando la cámara RGB principal de un dispositivo que ha de ser compatible. Estos datos del mapa de profundidad pueden ser empleados para crear experiencias de usuario inmersivas y realistas, como hacer que los objetos virtuales aparezcan delante o detrás de los objetos del mundo real o que choquen con precisión con las superficies detectadas.

Estimación de luz

La estimación de luz de ARCore permite un aumento en la sensación de realismo ya que la información obtenida le permite iluminar los objetos virtuales en unas condiciones similares a las del entorno del mundo que los rodea. A partir de una imagen de cámara determinada se obtiene la información sobre la intensidad media y la corrección de color.

Interacción del usuario

ARCore utiliza la prueba de impacto para tomar una coordenada (x, y) correspondiente a la pantalla del dispositivo móvil (proporcionada por un toque o cualquier otra interacción que desee que admita su aplicación) y proyecta un rayo en esa coordenada en la vista del mundo de la cámara, devolviendo cualquier punto característico que el rayo cruza, junto con la posición y orientación de esa intersección en el espacio real. Esto permite a los usuarios seleccionar o interactuar con objetos en el entorno.

Imágenes aumentadas

Otra de las funciones de ARCore son las imágenes aumentadas, que permiten la creación de aplicaciones de realidad aumentada que se activan a través de imágenes 2D específicas. Los usuarios pueden generar experiencias de realidad aumentada cuando apunten la cámara de su teléfono a determinadas imágenes.

ARCore también rastrea imágenes en movimiento como, por ejemplo, una valla publicitaria en el costado de un autobús en movimiento. Las imágenes se pueden compilar sin conexión para crear una base de datos de imágenes, o se pueden agregar imágenes individuales en tiempo real desde el dispositivo. Una vez registrado, ARCore detectará estas imágenes, los límites de las imágenes y devolverá la pose correspondiente.

Puntos orientados

Cuando se realiza una prueba de impacto que devuelve un punto característico, ARCore observará también los puntos característicos cercanos y los utilizará para intentar estimar el ángulo de la superficie en el punto característico dado. Luego devolverá una pose (posición y rotación) que tenga en cuenta ese ángulo. Por lo tanto, los puntos orientados permiten colocar objetos virtuales en superficies en ángulo.

Anclas y rastreables

Las poses pueden cambiar a medida que ARCore mejora la comprensión de su propia posición dentro de su entorno. Cuando se desea colocar un objeto virtual, se debe definir un ancla para asegurar que ARCore rastrea la posición del objeto a lo largo del tiempo. A menudo, se crea un ancla basada en la pose devuelta por una prueba de impacto, como se ha descrito anteriormente en la interacción del usuario.

El hecho de que las poses puedan cambiar significa que ARCore puede actualizar la posición de objetos ambientales como planos y puntos característicos a lo largo del tiempo. Los planos y los puntos son un tipo especial de objeto llamado rastreable. Como sugiere el nombre, estos son objetos que ARCore rastreará a lo largo del tiempo. Se pueden anclar objetos virtuales a rastreables determinados para asegurar que la relación entre el objeto virtual y el rastreable permanezca estable incluso cuando el dispositivo se mueve. Esto quiere decir que, si se coloca un modelo virtual en una mesa, si ARCore luego ajusta la posición del plano asociado con la mesa, el modelo permanecerá encima de la mesa.

Intercambio

ARCore Cloud Anchor API permite crear aplicaciones colaborativas o multijugador para dispositivos Android e iOS. Con Cloud Anchors, un dispositivo envía un ancla y puntos de características cercanos a la nube para su alojamiento. Estos anclajes se pueden compartir con otros usuarios con dispositivos en el mismo entorno. Esto permite que las aplicaciones representen los mismos objetos 3D adjuntos a estas anclas, y que los usuarios puedan tener la misma experiencia de RA simultáneamente.

Explicados estos conceptos fundamentales, ya podemos tener una idea amplia de lo que es ARCore, cuál es su funcionamiento y las posibilidades que ofrece para la creación experiencias de realidad aumentada.

4. DESARROLLO DE APLICACIONES EN UNITY

En este capítulo se explicará, en forma de tutorial, cómo desarrollar aplicaciones en Unity que utilicen los servicios de realidad aumentada de ARCore. En primer lugar, veremos cómo preparar el hardware y cómo descargar e instalar el software necesario. Después, crearemos un nuevo proyecto, con algunas configuraciones iniciales y con el SDK de ARCore para Unity. De este modo iremos añadiendo nuevos elementos a la escena para llegar a obtener una primera aplicación base que permite la detección de planos y la colocación y manipulación de objetos.

4.1 Preparación del Hardware

Antes de comenzar con el desarrollo de nuestra aplicación debemos de asegurarnos de que nuestro dispositivo cumple con los requisitos necesarios para habilitar las funciones de ARCore de la aplicación. Además, será necesario modificar algunas de las configuraciones de nuestro teléfono para poder instalarla en él. Lo haremos utilizando un cable USB, desde nuestra máquina de desarrollo. En este apartado se explicarán todos los pasos a seguir para llevar a cabo esta preparación del hardware.

Compatibilidad con ARCore

Que un dispositivo sea compatible con ARCore significa, básicamente, que ha superado su proceso de certificación. La importancia de esta certificación se encuentra en su deseo de ofrecer siempre una buena experiencia con las aplicaciones de realidad aumentada. Para conseguirlo es fundamental un buen funcionamiento del seguimiento de movimiento sensible, que se lleva a cabo combinando la imagen de la cámara con la entrada del sensor de movimiento para determinar cómo se mueve el dispositivo del usuario en el entorno real.

Para certificar cada dispositivo, se realiza una verificación de la cámara, los sensores de movimiento y la arquitectura de diseño para asegurarse de que la tecnología de RA funciona como se espera. Además, para garantizar un buen rendimiento y cálculos efectivos en tiempo real, el dispositivo debe tener una CPU lo suficientemente potente para que se integre adecuadamente con el diseño del hardware.

En el siguiente enlace podemos encontrar el listado completo de los modelos de dispositivos específicos que son compatibles en la actualidad, ya sean emuladores, Android (Google Play o China) o IOS [40].

➤ <https://developers.google.com/ar/devices>

Opciones de desarrollador y depuración por USB

Una vez que nos hemos asegurado de que nuestro dispositivo está certificado y es compatible con ARCore, vamos a realizar algunos cambios en los ajustes de éste.

En primer lugar, vamos a activar las opciones de desarrolladores, ya que en versiones posteriores de Android 4.1 estas opciones no están disponibles de forma predeterminada. Para habilitar estas opciones debemos presionar 7 veces la opción "número de compilación". Podemos encontrar esta opción en una de las siguientes ubicaciones, según la versión de Android:

- Android 9 (nivel de API 28) y versiones posteriores: **Configuración > Acerca del dispositivo > Número de compilación**
- Android 8.0.0 (API nivel 26) y Android 8.1.0 (API nivel 26): **Configuración > Sistema > Acerca del dispositivo > Número de compilación**
- Android 7.1 (nivel de API 25) y versiones anteriores: **Configuración > Acerca del dispositivo > Número de compilación**

Para poder usar el depurador y otras herramientas, debemos habilitar la depuración por USB, que permitirá que Unity reconozca nuestro dispositivo cuando se conecta mediante USB y así poder instalar nuestra aplicación de RA [41].

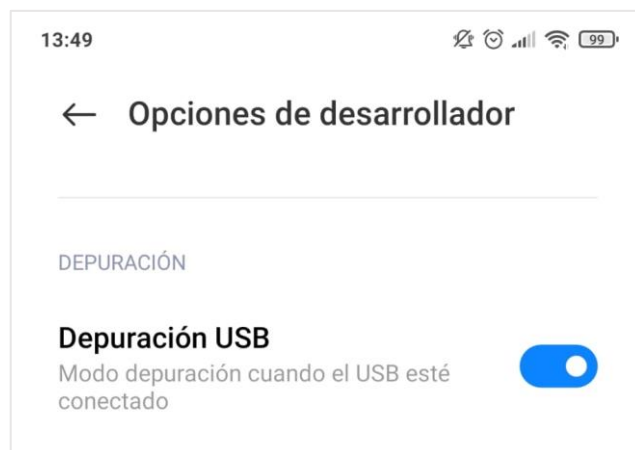


Figura 32. Activación de la Depuración USB.

4.2 Preparación del Software

Descargar ARCore

Para poder disfrutar de la realidad aumentada de ARCore en nuestro teléfono tan solo debemos tener descargada y actualizada la aplicación Servicios de Google Play para RA.

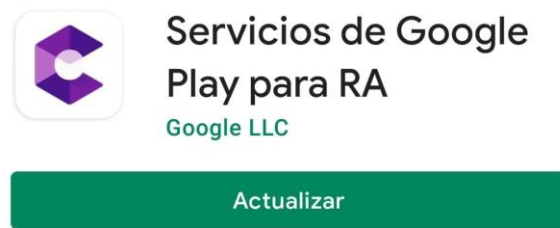


Figura 33. Aplicación de Servicios de Google Play para RA.

Podemos encontrarla en Play Store como cualquier otra aplicación, siempre y cuando nuestro dispositivo móvil cumpla los requisitos impuestos por ARCore.

Descargar Unity

Para el desarrollo de la aplicación será necesario instalar Unity en su versión 2017.4.40 o superior. Está disponible para Windows, Mac y Linux. Podemos acceder a la página de descarga del programa a través del siguiente enlace:

➤ <https://unity3d.com/es/get-unity/download>

En ella veremos que tenemos la opción de descargarlo de dos formas:

- Descargando únicamente la versión que nos interese.
- Descargando una herramienta denominada Unity Hub.

Escogeremos la segunda opción y posteriormente elegiremos la versión que queramos. De este modo también tenemos la posibilidad de descargar recursos, crear proyectos, añadir o quitar componentes a una versión ya instalada y desinstalar versiones previamente instaladas.

DESARROLLO DE APLICACIONES EN UNITY

Dentro de Unity Hub, en la sección **Installs** podemos ver las versiones instaladas, así como instalar otras a través del botón **ADD**. La versión con la cual se ha realizado este proyecto es la 2019.4.19f1, que era la recomendada en el momento de la instalación (LTS significa Long Term Support y son versiones de Unity a las que se darán soporte durante al menos dos años).

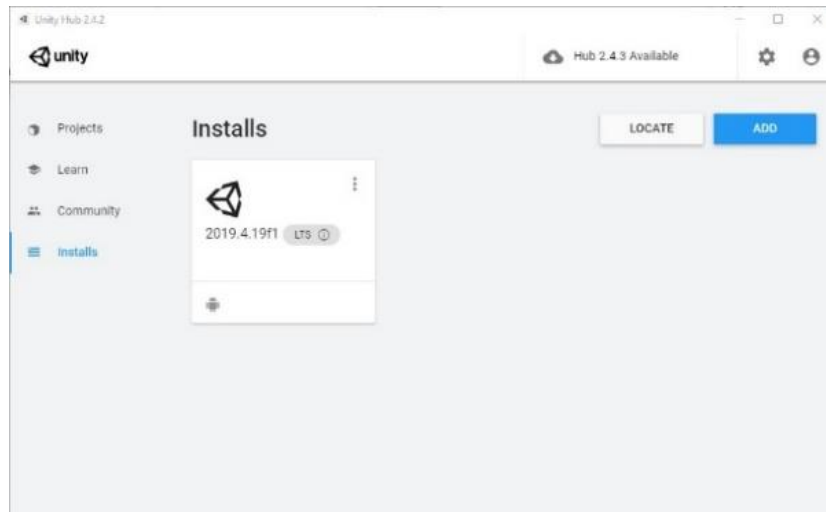


Figura 34. Sección Installs en Unity Hub.

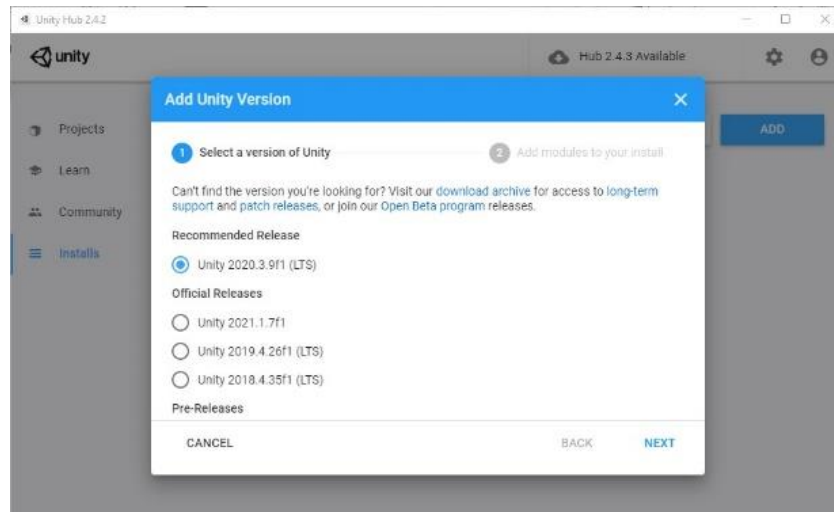


Figura 35. Ventana de elección de versiones de Unity Hub.

A continuación, escogeremos las plataformas sobre las que vamos a generar el juego (PC, Android, IOS, Mac, ...). Si cambiamos de idea y vamos a querer generar el juego para una plataforma que al inicio no contemplábamos, podremos volver a esta pantalla y añadirla.

TRABAJO DE FIN DE GRADO

APLICACIÓN DE REALIDAD AUMENTADA

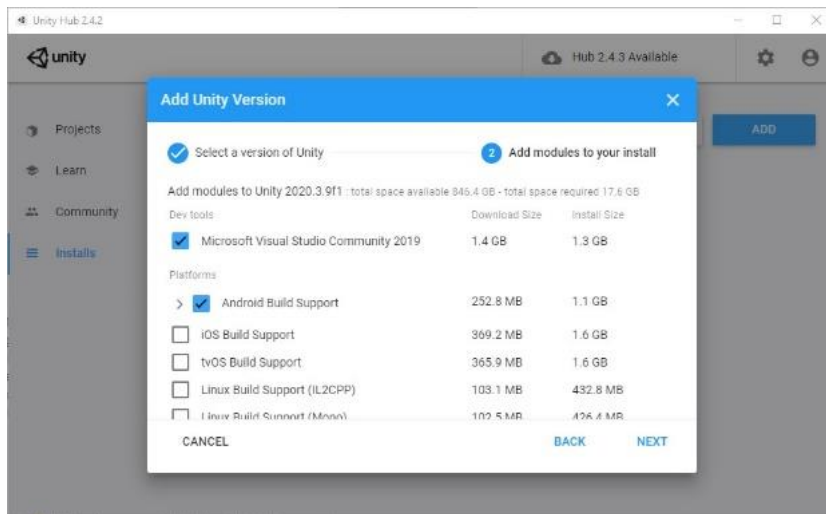


Figura 36. Ventana de selección de plataformas de Unity Hub.

En la siguiente ventana aceptaremos los términos de licencia para que también se instale Microsoft Visual Studio, que será la plataforma que usemos para la programación del código de Unity.

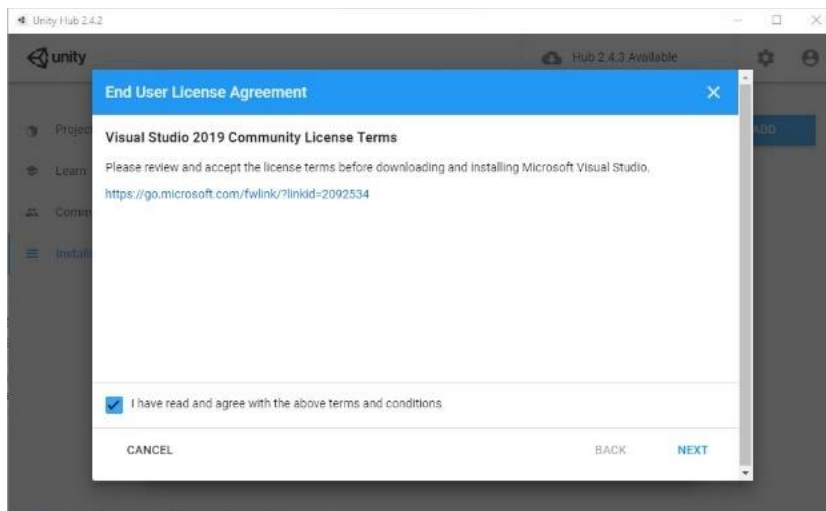


Figura 37. Ventana de instalación de Visual Studio.

Por último, aceptaremos también los términos de licencia para la descarga e instalación del SDK y NDK de Android. Hecho esto comenzará la instalación de la versión escogida de Unity.

4.3 Creación un proyecto en Unity

Para crear un nuevo proyecto lo haremos a través de Unity Hub, en la sección **Projects**, haciendo clic en el botón **NEW**.

En la ventana emergente elegiremos la opción de **3D** en el menú **Templates**, introduciremos un nombre para el proyecto y estableceremos la localización deseada para guardar el proyecto en nuestro ordenador. Ahora, pulsando el botón **CREATE** se abrirá inmediatamente Unity con nuestro nuevo proyecto.

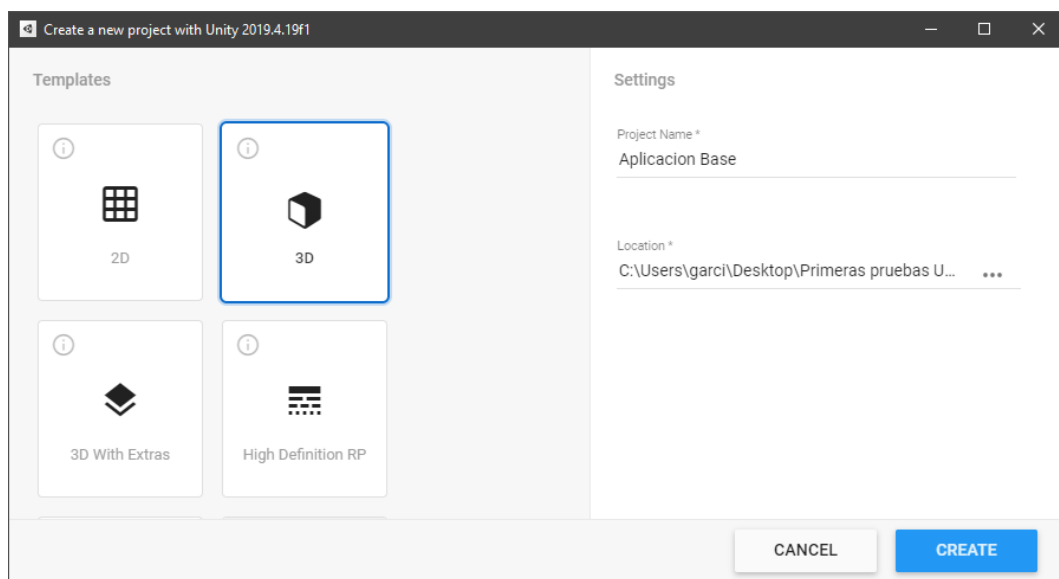


Figura 38. Ventana de creación de proyecto en Unity Hub.

4.4 Configuración inicial

En esta parte del tutorial explicaremos cuáles son las configuraciones que deberemos establecer en nuestro proyecto para desarrollar una aplicación de realidad aumentada con ARCore. Es decir, cuáles son los paquetes que debemos instalar, cómo descargar e importar el SDK, los cambios a realizar en la configuración del reproducción y en Gardle.

Instalación de paquetes

Una vez en Unity procederemos a instalar algunos paquetes necesarios para nuestra aplicación. Para ello iremos a **Window > Package Manager**, allí buscaremos e instalaremos:

- **Multiplayer HLAPI**, requerido por el ejemplo de CloudAnchors [42] (para crear experiencias de RA multijugador o colaborativas).
- **XR Legacy Input Helpers**, requerido para Instant Preview [43] (para obtener una vista previa de la aplicación utilizando la entrada y salida real del dispositivo).

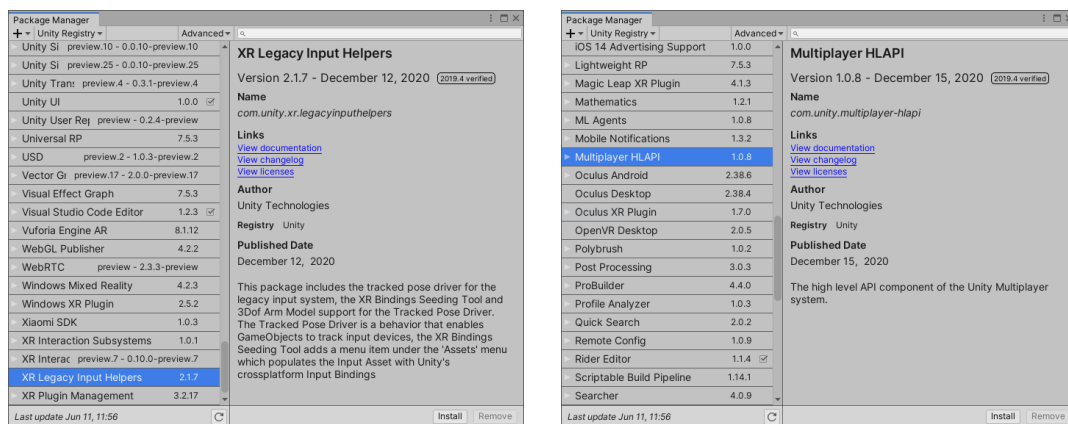


Figura 39. Instalación de paquetes.

Descarga e importación del SDK

Descargaremos la versión 1.23.0 del SDK de ARCore para Unity. Lo haremos a través el siguiente enlace, en la parte inferior de la página:

➤ <https://github.com/google-ar/arcore-unity-sdk/releases/tag/v1.23.0>

Para importarlo vamos a **Assets > Import Package > Custom Package** y seleccionamos el archivo **arcore-unity-sdk-1.23.0.unitypackage**.

En el cuadro de diálogo emergente debemos asegurarnos de que todas las opciones del paquete están seleccionadas y, entonces, hacemos clic en **Import**.

Cambios en la configuración del reproductor

Ahora vamos a cambiar la plataforma para la cual vamos a crear nuestra aplicación. Para ello vamos a **File > Build Settings** seleccionamos **Android** y hacemos clic en **Switch Platform**.

Después ingresamos en **Player Settings**, en la zona inferior de la ventana. En la tabla de configuraciones para Android, en la sección **Other Settings** modificamos lo siguiente:

- Seleccionamos **Auto Graphics API**.
- Deseleccionamos **Multithreaded Rendering** (es posible que los activos 3D no siempre se rendericen correctamente cuando una aplicación coloca una gran carga en el hilo de renderizado).
- Podemos introducir un nombre para el paquete creando un ID de aplicación único con un formato de nombre de paquete de Java.
- En **Minimum API Level** seleccionamos **Android 7.0 'Nougat' (API Level 24)** u otro superior.
- Establecemos el **Scripting Backend** en **IL2CPP** y seleccionamos **ARM64** en **Target Architectures** para que la aplicación sea compatible con dispositivos de 64 bits.

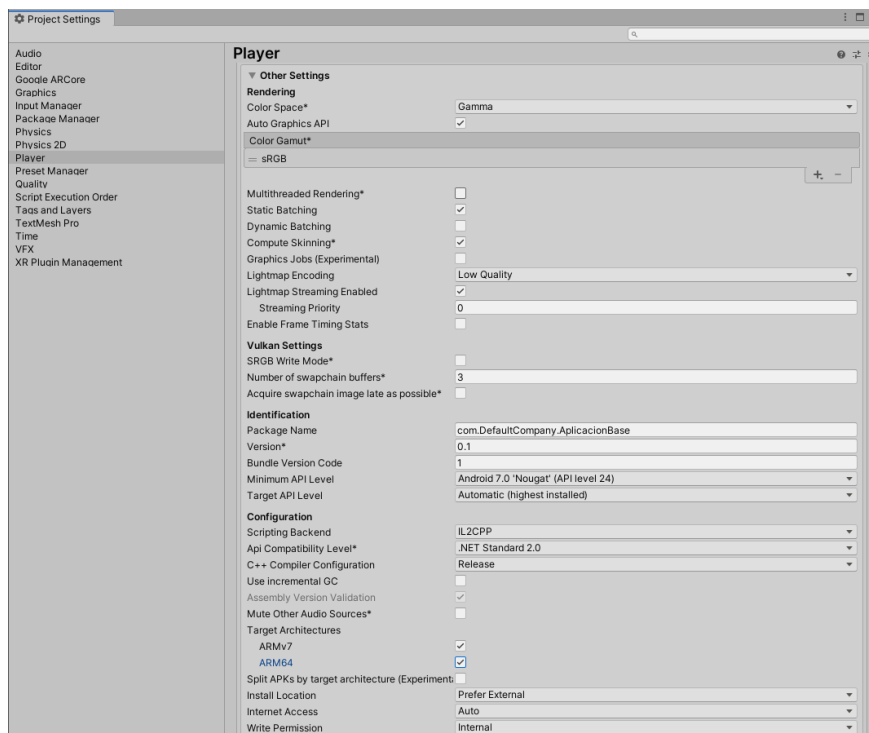


Figura 40. Configuración de reproductor necesaria.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

También en la tabla de configuraciones para Android, pero ahora en la sección **Resolution and Presentation** vamos a establecer la orientación por defecto en **Portrait**.

En la sección **XR Settings** marcaremos la opción de **ARCore Supported**.

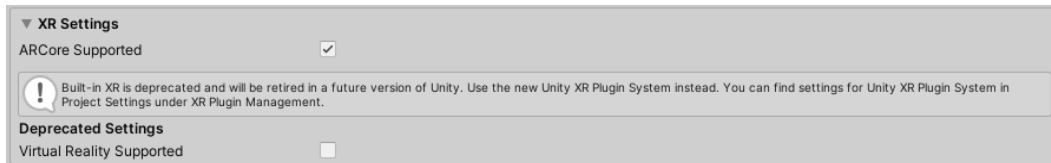


Figura 41. Configuración para que la aplicación soporte ARCore.

Cambios en la configuración de calidad

Dentro de la sección **Quality** vamos a establecer en la parte superior el nivel de calidad por defecto en **Ultra** para Android.

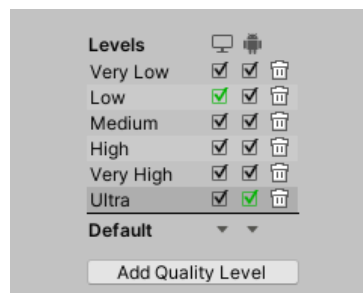


Figura 42. Configuración de calidad.

En el apartado **Shadows** realizaremos las siguientes modificaciones:

- Seleccionamos la opción **Very High Resolution** en **Shadow Resolution**.
- Establecemos la **Shadow Distance** en 10.
- Establecemos la configuración **Shadow Near Plane Offset** en 0.

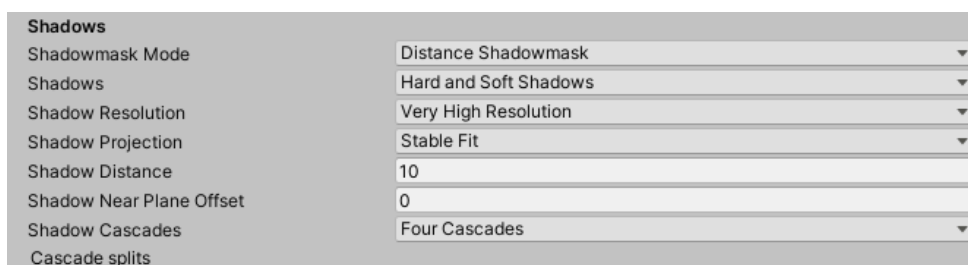


Figura 43. Configuraciones de sombras.

Configuración del Gradle

Gradle es un sistema de compilación de Android que automatiza varios procesos de compilación y evita muchos errores de compilación comunes. Unity usa Gradle para todas las compilaciones de Android.

Si en este momento intentáramos compilar la aplicación e instalarla en nuestro dispositivo móvil, es probable que la consola nos advirtiera de la existencia de un problema. Esto es porque al utilizar el SDK de ARCore (1.19 o posterior) con una versión de Unity 2018.4 o posterior, se requiere la versión 5.6.4 de Gradle o superior.

Para evitar este problema vamos a tener que personalizar el Gradle que Unity trae por defecto. El proceso a seguir para llevar a cabo esta configuración depende de la versión de Unity, como se explica en el siguiente enlace:

➤ <https://developers.google.com/ar/develop/unity/android-11-build7>

Aquí solo se recogen los pasos a seguir para las versiones 2019.3 y 2019.4.

En primer lugar, descargamos la versión actualizada del Gradle desde la página de descargas de su web (<https://gradle.org/releases/>). Hacemos clic en **complete** y comenzará la descarga de un archivo .zip que debemos descomprimir.

De nuevo en Unity, vamos a **Edit > Preferences > External Tools > Android**, desmarcamos la versión recomendada y seleccionamos la que hemos descargado.

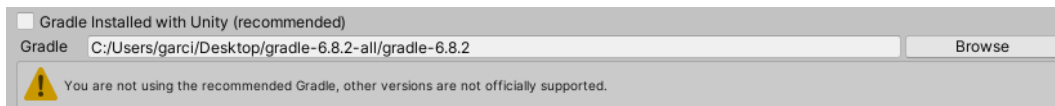


Figura 44. Configuración del Gradle en Unity.

A continuación, vamos a **Project Settings > Player > Android > Publishing Settings > Build** y seleccionamos:

- **Custom Main Gradle Template**
- **Custom Launcher Gradle Template**

Para finalizar vamos a modificar el código de estos dos archivos que se han generado y que se encuentran en **Assets/Plugins/Android** dentro de la carpeta donde está guardado el proyecto.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Abrimos ambos archivos con Visual Studio y eliminamos el siguiente comentario que aparece en la parte superior:

```
// GENERATED BY UNITY. REMOVE THIS COMMENT TO PREVENT OVERWRITING WHEN  
EXPORTING AGAIN
```

En su lugar copiamos el código que aparece a continuación:

```
buildscript {  
    repositories {  
        google()  
        jcenter()  
    }  
    dependencies {  
        // Must be Android Gradle Plugin 3.6.0 or later. For a list  
of  
        // compatible Gradle versions refer to:  
        // https://developer.android.com/studio/releases/gradle-  
plugin  
        classpath 'com.android.tools.build:gradle:3.6.0'  
    }  
}  
  
allprojects {  
    repositories {  
        google()  
        jcenter()  
        flatDir {  
            dirs 'libs'  
        }  
    }  
}
```


Configuración del JDK

Podría ocurrir que al intentar instalar la aplicación en un dispositivo la consola nos advirtiera también de otro error, esta vez relacionado con el JDK (Java Development Kit), que es un software que provee herramientas de desarrollo para la creación de programas en Java.

Este error puede deberse a que Unity no reconoce correctamente la ruta en la que se encuentra ubicado el JDK que se ha instalado por defecto desde Unity Hub. La solución es tan sencilla como ir a **Edit > Preferences > External Tools > Android** y entonces desmarcar y volver a marcar la casilla de la opción que dice "JDK installed With Unity". De esta forma reconocerá la ruta correctamente y quedará solucionado.



Figura 45. Configuración del JDK.

Una vez que hemos configurado nuestro proyecto, como se explica anteriormente, y hemos importado el SDK podemos empezar a crear la aplicación.

Para comenzar con el desarrollo de nuestra aplicación partiremos de una primera aplicación base que contendrá los elementos necesarios para conseguir una aplicación de realidad aumentada totalmente funcional. De este modo iremos explicando la función de los diferentes elementos que la componen a medida que los vamos incorporando, así como la forma en que se coordinan entre ellos.

Con esta escena básica podremos localizar planos en el entorno, posicionar objetos virtuales en esos planos y manipularlos a través de diferentes entradas táctiles.

En primer lugar, estudiaremos dos conceptos esenciales para comprender el funcionamiento de Unity. Después añadiremos los tres primeros elementos a la escena, fundamentales para una aplicación de ARCore. Seguiremos configurando un controlador de escena y un generador de planos. Veremos cómo instanciar y manipular objetos y, finalmente, instalaremos la aplicación en nuestro dispositivo.

4.5 GameObjects y Prefabs

Para comenzar, añadiremos a la escena algunos elementos requeridos para usar ARCore en Unity. Estos elementos cumplen diversas funciones de control sin tener ninguna apariencia concreta dentro de la escena.

Para comprender mejor esto y, en general, la forma en que funciona Unity, vamos a explicar dos conceptos que son fundamentales: los GameObjects y los Prefabs.

Los **GameObjects** son objetos fundamentales en Unity, que pueden representar personajes, cámaras, escenarios, etc. Al fin y al cabo, todo lo que contiene una escena son GameObjects. Estos no representan nada por sí mismos, pero funcionan como “contenedores” para *Components*, que son los que caracterizan a estos objetos e implementan una verdadera funcionalidad en ellos. Por ejemplo, un objeto de luz funciona como tal porque tiene asociado un componente del tipo *Light*.



Figura 46. Cuatro tipos diferentes de GameObject.

Un GameObject siempre tendrá un componente *Transform* asociado (para representar la posición y orientación) y no es posible eliminarlo. Los otros

componentes que le dan al objeto su funcionalidad pueden ser agregados desde el menú *Component* del Inspector o desde un script. También hay muchos objetos útiles pre-construidos (figuras primitivas, cámaras, etc.) disponibles en el menú *GameObject > 3D Object*.

En el caso de los **Prefabs** se trata básicamente de un *GameObject* predefinido. Unity dispone de un tipo de asset denominado *Prefab*, que permite almacenar un objeto de juego con unos determinados componentes y propiedades. El *prefab* actúa como una plantilla a partir de la cual se pueden crear nuevas instancias del objeto en la escena.

Podemos crear uno arrastrando un *GameObject* desde la Jerarquía hasta la ventana de Proyecto. Los objetos creados como instancias de *prefab* serán mostrados en la vista de jerarquía en texto azul (los objetos normales son mostrados en texto negro).

4.6 Elementos requeridos en la escena

A continuación, vamos a explicar cómo añadir los tres primeros elementos de nuestra aplicación: *ARCore Device*, *Environmental Light* y *EventSystem*. Se trata de objetos de juego (los dos primeros serán prefabricados) indispensables para la creación de una aplicación de realidad aumentada con ARCore en Unity.

ARCore Device

En primer lugar, vamos a eliminar el objeto de juego *Main Camera*, ya que en su lugar utilizaremos la cámara en primera persona del prefabricado *ARCore Device*.

Para agregar este elemento podemos buscarlo en la ventana proyecto en **Assets/GoogleARCore/Prefabs** y después arrastrarlo en la escena o en la jerarquía. Debemos asegurarnos de que la posición establecida en su componente *Transform* es la (0,0,0).

El *ARCore Device* es un prefabricado que administra la sesión de ARCore. Cuando está activo en una escena, este *prefab* creará e inicializará una sesión de ARCore y renderizará la imagen de fondo de una cámara Unity que sigue la posición y orientación del dispositivo según lo estimado por ARCore.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Todos los procesos de AR, como el seguimiento de movimiento, la comprensión del entorno y la estimación de la iluminación, ocurren dentro de una sesión de ARCore. La *Session* es el principal punto de entrada a la API de ARCore. Administra el estado del sistema AR y maneja el ciclo de vida de la sesión, lo que permite que la aplicación cree, configure, inicie o detenga una sesión. Lo más importante es que permite que la aplicación reciba el acceso a la imagen vista desde la cámara y la pose del dispositivo [44].

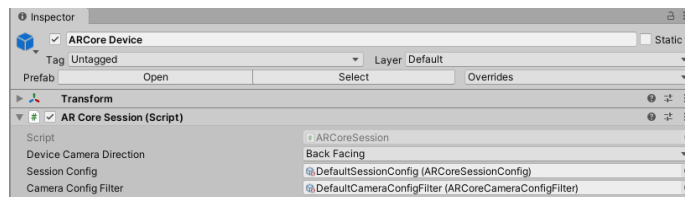


Figura 47. Vista del inspector para ARCore Device.

Contiene un componente de tipo script denominado *ARCoreSession*, responsable de administrar la sesión de ARCore. En él podemos establecer la dirección de la cámara. También contiene una configuración de sesión (*DefaultSessionConfig*) y un filtro de configuración de cámara (*DefaultCameraConfigFilter*).

CameraConfig proporciona detalles de la configuración de la cámara que ARCore usa para acceder al sensor de la cámara para una sesión determinada. Estos detalles incluyen, por ejemplo, la velocidad de fotogramas de captura objetivo y si un sensor de profundidad está presente y se utiliza [45].

Al crear una nueva sesión, ARCore usa *ARCoreCameraConfigFilter* para filtrar la configuración de la cámara que mejor coincide con la lista de configuraciones disponibles.

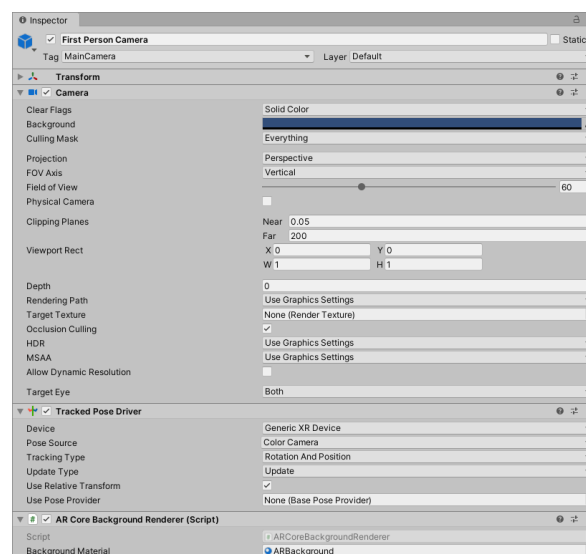


Figura 48. Vista del Inspector para First Person Camera.

ARCore Device contiene también otro objeto de juego, la *First Person Camera*, que es su hijo dentro de la jerarquía. Este *GameObject* es la cámara principal dentro de nuestra escena 3D.

Tiene un componente asociado denominado *TrackedPoseDriver* que actualiza la posición y la rotación locales de la transformación del objeto del juego, para que coincida con la *pose* estimada de la cámara.

El script adjunto *ARCoreBackgroundRenderer* muestra la imagen del sensor de la cámara del dispositivo físico como una textura de fondo de la cámara de Unity.

Environmental Light

El segundo elemento que vamos a añadir será el *Environmental Light*, otro prefabricado que además se encuentra en el mismo directorio que el anterior (*Assets/GoogleARCore/Prefabs*). Debemos eliminar también el objeto de juego *Directional Light* predeterminado.

La API de estimación de iluminación analiza una imagen determinada en busca de señales visuales discretas y proporciona información detallada sobre la iluminación en una escena determinada. Luego, puede usar esta información al renderizar objetos virtuales para iluminarlos en las mismas condiciones que la escena en la que están colocados, haciendo que estos objetos se puedan apreciar más realistas [46].

Este prefabricado ajusta la iluminación en la escena AR usando una estimación del brillo promedio de píxeles en la imagen que captura nuestra cámara. Para hacer esto, en cada *frame*, el script del controlador *EnvironmentalLight* obtiene la intensidad de píxeles de *LightEstimate* y los valores de corrección de color para la imagen capturada por la cámara.

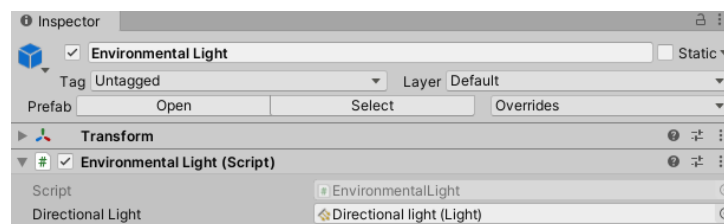


Figura 49. Vista del inspector para el *Environmental Light*.

Después, el script usa estos valores para generar un valor de intensidad de píxel normalizado y establecer un valor de sombreado *_GlobalColorCorrection*. Además, este script utiliza como variable una luz direccional que es un objeto hijo del mismo prefabricado.

EventSystem

En tercer lugar, añadiremos el elemento *EventSystem*. En este caso no se trata de un objeto prefabricado, sino que vamos a crearlo nosotros mismos desde el menú: **GameObject > UI > EventSystem**.

El *EventSystem* es una manera de enviar eventos a objetos en la aplicación basado en inputs, ya sean de teclado, mouse o tacto. Está diseñado como un administrador y facilitador de comunicaciones entre módulos del *EventSystem*. Algunas de sus funciones son: controlar qué *GameObject* es considerado como seleccionado, qué *InputModule* está en uso, maneja el *Raycasting* (más adelante lo utilizaremos) y actualiza todos los *InputModule* como es requerido [47].

4.7 Configuración del SceneController

A continuación, vamos a crear y configurar un elemento controlador de escena, que nos servirá para coordinar la actividad de ARCore y Unity. Añadiremos un objeto de juego vacío a través del menú **GameObject > Create Empty** y cambiaremos su nombre por *SceneController* en la ventana inspector.

Este tipo de objeto no tiene ninguna funcionalidad por sí mismo. Para que cumpla su papel como controlador de escena vamos a agregarle un componente de tipo C# script. Este script será el *ObjectManipulationController* que pertenece a la escena *ObjectManipulation* del SDK, que es un ejemplo de aplicación muy similar a lo que queremos conseguir con esta aplicación base. Podremos encontrarlo en **Assets/GoogleARCore/Examples/ObjectManipulation/Scripts**.

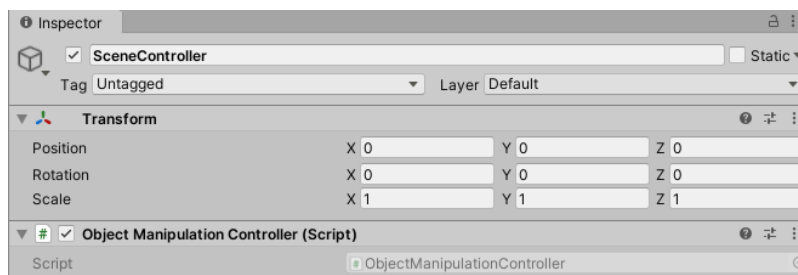


Figura 50. Vista del inspector para el *SceneController*.

Gracias a este script nuestra aplicación realizará diversas funciones de control sobre la sesión AR, como habilitar la captura de cámara a 60 FPS para dispositivos compatibles en la función **Awake()**, administrar su ciclo de vida a través del método **UpdateApplicationLifecycle()** o mostrar un mensaje de aviso (*toast*) de Android con **ShowAndroidToastMessage(string message)**.

4.8 Detección y visualización de planos

En este apartado vamos a ver cómo podemos hacer para que nuestra aplicación detecte los planos del entorno, ya sean verticales u horizontales, y, además, podamos visualizarlos.

ARCore utiliza una clase llamada *DetectedPlane* para representar los planos detectados. Esta clase no es un objeto de juego por sí mismo, sino que necesitamos de un prefabricado para renderizar estos planos. Afortunadamente en el SDK de ARCore para Unity contamos con un prefab de este tipo, llamado *DetectedPlaneVisualizer*.

Para cada uno de los planos detectados, se creará un objeto de juego que renderizará el plano usando el prefabricado *DetectedPlaneVisualizer*. Entre los assets del SDK encontramos también un C# script denominado *DetectedPlaneGenerator.cs*, que cumple con esta función.

Vamos a agregar este script como un componente en un nuevo objeto de juego vacío que llamaremos *PlaneGenerator*. Para ello seleccionamos este objeto de juego, hacemos clic en el botón **Add Component** en la ventana inspector y escribimos *DetectedPlaneGenerator* en el buscador. Luego establecemos el valor *Detected Plane Prefab* del script con el prefabricado *DetectedPlaneVisualizer*, que podemos encontrar en la ventana proyecto en **Assets / GoogleARCore / Examples / Common / Prefabs**.

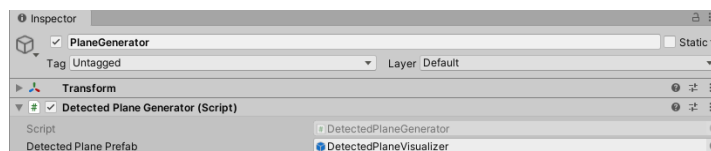


Figura 51. Vista del inspector para el *PlaneGenerator*.

Para mejorar la experiencia de usuario con nuestra aplicación y que la detección de planos sea más intuitiva, vamos a añadir también a la escena el prefabricado *PlaneDiscovery*. Lo podemos encontrar entre los assets del SDK, en la misma carpeta que el prefab anterior.

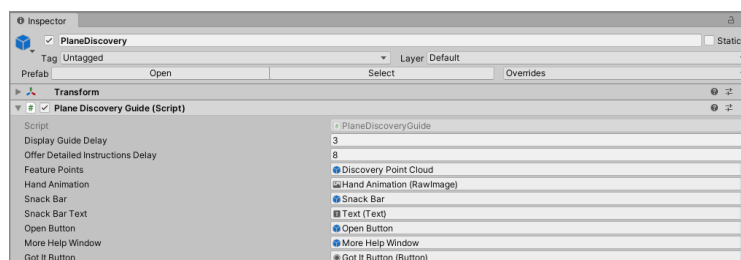


Figura 52. Vista del Inspector para el *PlaneDiscovery*.

Este objeto de juego guía a los usuarios para escanear los alrededores y descubrir planos. Consiste en una animación de una mano que sostiene un dispositivo móvil e instrucciones breves que indican cómo moverlo para que se detecten puntos característicos. Si no se encuentra ningún plano después de un cierto período de tiempo, la barra de refrigerios muestra un botón que ofrece abrir una ventana de ayuda con instrucciones más detalladas.

Si en este momento ejecutáramos la aplicación veríamos como al mover nuestro dispositivo se van detectando los planos del mundo real, representados con una malla triangular de color blanco.

Dependiendo de las características físicas del entorno, podrían pasar un par de segundos hasta que se detecte el primer plano. La detección de planos se puede mejorar con buena iluminación y algún tipo de patrón o textura.

4.9 Instanciación y manipulación de objetos

En esta parte del tutorial vamos a explicar por fin qué es lo que debemos hacer para que nuestra aplicación sea capaz de generar instancias de un objeto, colocándolo sobre un plano detectado y con la posibilidad de manipularlo con entradas táctiles. Para ello necesitaremos añadir dos nuevos objetos de juego a la escena: un generador de objetos que llamaremos *ObjectGenerator*, y un sistema para controlar la manipulación del objeto, que será el prefab *ManipulationSystem* del SDK.

ObjectGenerator

Este objeto de juego se encargará de gestionar la colocación del objeto en la escena AR. Lo hará gracias a un script de manipulación que tendrá como complemento. Este script tendrá referencias a la cámara en primera persona, al prefabricado *Manipulator* del SDK y a al prefabricado del objeto que queramos colocar.

En primer lugar, vamos a crear un objeto de juego vacío como se ha explicado anteriormente (*GameObject > Create Empty*) y cambiaremos su nombre por *ObjectGenerator*. A este *GameObject* le vamos a añadir un componente C# script que nombraremos como *ObjectManipulator*.

Realmente este script que vamos a programar es similar al *PawnManipulator.cs* de la una de las escenas de ejemplo del SDK, sin embargo, nos interesa ir creando el código poco a poco para detallar cuál es su función. Además, vamos a explicar cómo hacer para limitar el número de instancias al número que queramos.

Vamos a abrir con Visual Studio el script *ObjectManipulator* que acabamos de crear para empezar a programarlo desde cero, eliminando el código inicial.

En primer lugar, añadiremos el *namespace* que contiene las clases referidas al ejemplo *ObjectManipulation* e importamos los espacios de nombres de GoogleARCore y UnityEngine con *using*, como se ve en el siguiente código:

```
namespace GoogleARCore.Examples.ObjectManipulation
{
    using GoogleARCore;
    using UnityEngine;
```

Ahora añadiremos una clase pública con el nombre del script seguido de la palabra **Manipulator**, en vez de *MonoBehaviour*. De este modo estableceremos que la clase hereda de la clase base *Manipulator*, que recibirá devoluciones de llamada de *ManipulationSystem* cuando se detecten los diferentes gestos del usuario. En estas devoluciones de llamada, *Manipulator* puede decidir si se inicia el gesto, qué hacer mientras avanza y cuándo termina, anulando diferentes métodos de la clase base.

```
public class ObjectManipulator : Manipulator
```

Dentro de esta clase vamos a declarar tres variables públicas: la cámara en primera persona, el objeto que queramos instanciar y el prefabricado del manipulador.

Si además queremos poner un límite al número de instancias añadiremos dos variables más, una pública con la que podremos establecer el número desde el inspector y otra privada que nos servirá de contador.

```
public Camera FirstPersonCamera;
public GameObject ObjectPrefab;
public GameObject ManipulatorPrefab;

public int Instances;
private int count = 0;
```

Ahora crearemos el método **CanStartManipulationForGesture (TapGesture gesture)**. Para evitar instanciar un objeto cuando tocamos uno ya existente, el gesto tan solo puede iniciarse cuando el objetivo del gesto no es ningún objeto, es decir, que es *null*.

```
protected override bool CanStartManipulationForGesture(TapGesture gesture)
{
    if (gesture.TargetObject == null)
    {
        return true;
    }

    return false;
}
```

Además de la devolución de llamada para decidir si se puede iniciar la manipulación para un tipo de gesto determinado, la clase base *Manipulator* proporciona devoluciones de llamada para implementar diferentes comportamientos durante cada paso del ciclo de vida del gesto: `OnStartManipulation()`, `OnContinueManipulation()` y `OnEndManipulation()`.

En este caso, con *TapGesture* (es decir, un toque en la pantalla) crearemos una instancia de un objeto desde el prefabricado en el momento en que el *TapGesture* finaliza. Para ello introduciremos a continuación el método **OnEndManipulation (TapGesture gesture)**.

Dentro añadiremos, en primer lugar, las dos siguientes declaraciones lógicas *if*, que forzarán la terminación de la ejecución del método en el caso de que el gesto sea cancelado o que apunte a un objeto existente, respectivamente.

```
protected override void OnEndManipulation(TapGesture gesture)
{
    if (gesture.WasCancelled)
    {
        return;
    }

    if (gesture.TargetObject != null)
    {
        return;
    }

    ...
}
```

A continuación, también dentro del método, vamos a rastrear la ubicación en la que el usuario toca la pantalla en busca de planos. Cuando tocamos la pantalla, *Frame.Raycast()* utiliza un *Raycast* desde la posición de toque para detectar si tocamos un plano o puntos orientados. En caso afirmativo se crea una instancia del objeto en la posición del *TrackableHit* donde el usuario realiza el toque. En el mismo lugar se crea también un instancia del prefabricado manipulador.

Seguidamente, se establece el objeto como un hijo del manipulador, se crea un ancla para mantener la posición en relación con el mundo real, se hace que el manipulador sea hijo del ancla y, por último, queda seleccionado.

Para limitar el número de instancias lo que haremos será imponer como condición que el contador no sea mayor o igual al número de instancias deseadas. El contador aumentará en una unidad cada vez que se genere un objeto nuevo.

```
TrackableHit hit;
TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon;

if (Frame.Raycast(
    gesture.StartPosition.x, gesture.StartPosition.y, raycastFilter, out hit))
{
    if (((hit.Trackable is DetectedPlane) &&
        Vector3.Dot(FirstPersonCamera.transform.position - hit.Pose.position,
            hit.Pose.rotation * Vector3.up) < 0) || (count >= Instances))
    {
        Debug.Log("Hit at back of the current DetectedPlane");
    }
    else
    {
        var gameObject = Instantiate(ObjectPrefab, hit.Pose.position,
            hit.Pose.rotation);
        count++;

        var manipulator =Instantiate(ManipulatorPrefab, hit.Pose.position,
            hit.Pose.rotation);

        gameObject.transform.parent = manipulator.transform;

        var anchor = hit.Trackable.CreateAnchor(hit.Pose);

        manipulator.transform.parent = anchor.transform;

        manipulator.GetComponent<Manipulator>().Select();
    }
}
}
```

Con esto ya hemos terminado con la programación del script *ObjectManipulator* (en el Apéndice 1 se puede ver el código completo).

Ahora debemos establecer un valor para las variables que hemos declarado como públicas. Para añadir la cámara en primera persona simplemente podemos arrastarla desde la jerarquía hasta el espacio de la variable en el inspector. Por el momento, como objeto prefabricado utilizaremos el modelo *PawnPrefab* del SDK (**Assets/GoogleARCore/Examples/Common/Prefabs**), que nos servirá para probar el funcionamiento de la aplicación base. El prefabricado *Manipulator* lo podemos encontrar en la ventana de proyecto en **Assets / GoogleARCore / Examples / ObjectManipulation / Prefabs**. Por último, vamos a fijar el número de instancias en uno, para evitar crear objetos en el entorno por error al tocar la pantalla.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

El objeto de juego *ObjectGenerator* quedaría configurado como se ve en la siguiente imagen.

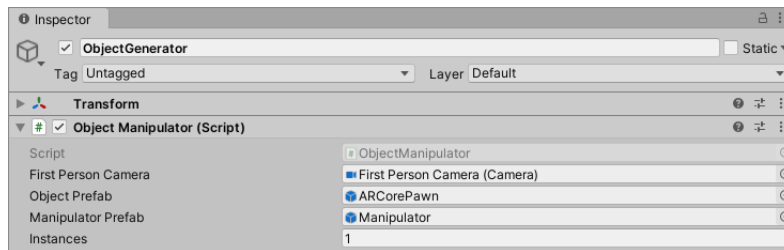


Figura 53. Vista del inspector para el *ObjectGenerator*.

Como hemos explicado antes, la instancia del objeto se crea como hija del *ManipulatorPrefab*. Este prefabricado contiene varios scripts manipuladores que amplían la clase base *Manipulator*, y que anulan las funciones de llamada anteriores para determinados tipos de gestos que en su lugar implementan diferentes comportamientos a los que reacciona el objeto: selección, traslación, escala, rotación y elevación. Algunos parámetros de estos scripts se pueden personalizar abriendo el prefabricado en la ventana inspector.

En el caso de nuestra aplicación base, vamos a mantener desactivados (desmarcando los componentes como se ve en la Figura 54) los scripts *ScaleManipulator* y *ElevationManipulator*. Lo haremos así porque queremos mantener cierto realismo a la hora de presentar el objeto en realidad aumentada y no nos interesa modificar su escala ni dejarlo levitando por encima del suelo.

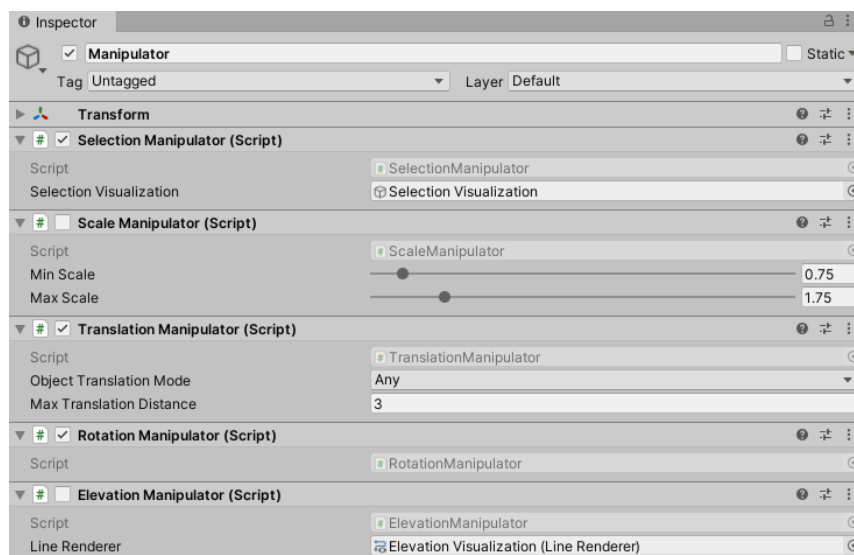


Figura 54. Vista del inspector para el prefab *Manipulator*.

Por otra parte, al abrir el prefabricado veremos que, además de los scripts mencionados, contiene dos objetos de juego secundarios: *Selection Visualization* y *Elevation Visualization*. Los componentes tipo script *SelectionManipulator* y *ElevationManipulator* hacen referencia a estos dos objetos de juego respectivamente en el prefab *Manipulator*. La visualización de selección se habilitará mientras el objeto está seleccionado, y la visualización de elevación se habilitará mientras la manipulación de elevación esté activa.

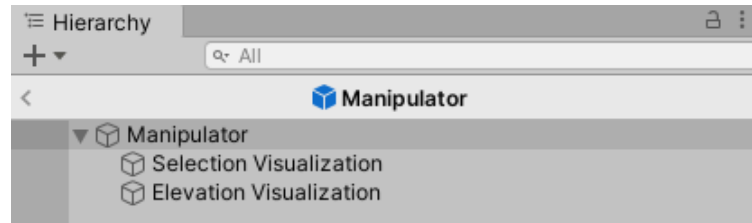


Figura 55. Objetos de juego secundarios del prefab Manipulator.

ManipulationSystem

Por último, vamos a introducir en la escena el prefabricado *ManipulationSystem* que encontraremos también entre los assets del SDK en **Assets / GoogleARCore / Examples / ObjectManipulation / Prefabs**.

Este prefab tan solo está formado por un componente de tipo C# script. Gracias a la programación de este script se puede llevar a cabo la detección de las entradas táctiles para después notificar a todos los manipuladores de la escena acerca de estos gestos.

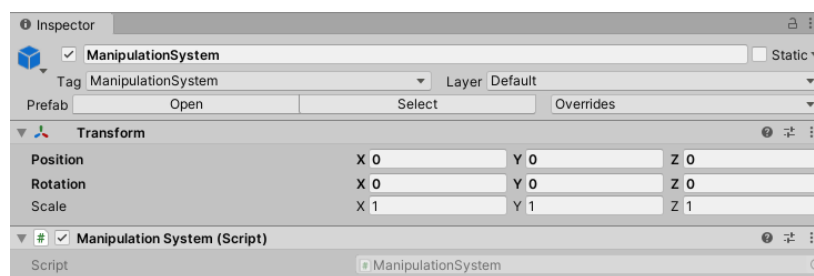


Figura 56. Vista del inspector para el ManipulationSystem.

Ahora ya tenemos todos los elementos necesarios para poder instanciar y manipular objetos en nuestra aplicación base de realidad aumentada.

4.10 Instalación de la Aplicación Base

Con todos estos elementos creados y configurados ya tenemos todo lo necesario para que nuestra aplicación funcione correctamente. Ahora vamos a ver qué tenemos que hacer para instalar esta aplicación en nuestro dispositivo móvil.

En primer lugar, vamos a guardar la escena a través del menú **File > Save** y la nombraremos Aplicación Base. La jerarquía de nuestra escena debería quedar como se ve en la siguiente imagen: los elementos *ARCore Device*, *EnvironmentalLight* y *EventSystem* del apartado 5.2, el *SceneController* del 5.3, el *PlaneGenerator* y *PlaneDiscovery* del 5.4, y, por último, el *ObjectManipulator* y *ManipulationSystem* del apartado 5.5.

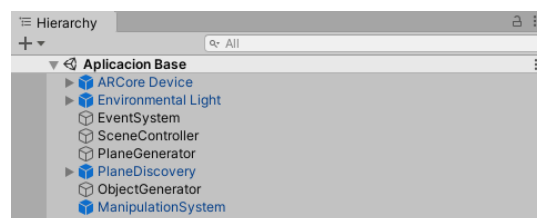


Figura 57. Jerarquía de la aplicación base.

A continuación, vamos a abrir la ventana **Build Settings** en el menú **File**. Aquí debemos añadir nuestra escena a las escenas en construcción (Scenes In Build), para ello pulsamos en botón **Add Open Scenes**. También tenemos que asegurarnos de que la plataforma de construcción seleccionada es Android.

Es el momento de conectar nuestro teléfono móvil al ordenador a través de un cable USB y activar la depuración por USB como se explicó en el apartado 4.1. Hecho esto podemos dar al botón **Refresh** y nuestro dispositivo debería aparecer entre las opciones del desplegable de la izquierda.

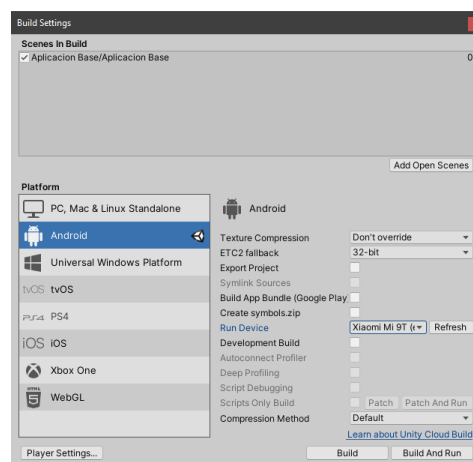


Figura 58. Configuración de construcción para la aplicación base.

DESARROLLO DE APLICACIONES EN UNITY

Para comenzar a compilar la aplicación debemos pulsar el botón **Build and Run**, que instalará la aplicación en nuestro teléfono y la ejecutará directamente. Antes, Unity nos pedirá que guardemos el archivo .apk así que lo llamaremos también “Aplicación Base” y le daremos a **Guardar**. En este momento comenzará la compilación de la aplicación y cuando finalice, si todo funciona correctamente, nos aparecerá una ventana en el móvil que nos preguntará si deseamos instalar la aplicación vía USB, a lo que responderemos afirmativamente.



Figura 59. Ventana de instalación vía USB.

Tan solo tenemos que dar permiso para que utilice la cámara de nuestro dispositivo (Figura 60 izq.) y ya podremos probar la aplicación. Si movemos el teléfono veremos cómo empiezan a visualizarse los planos a los que apuntamos con la cámara. Si realizamos un toque en la pantalla sobre un plano aparecerá el objeto prefabricado, que también podremos trasladar y rotar (Figura 60 dcha.).

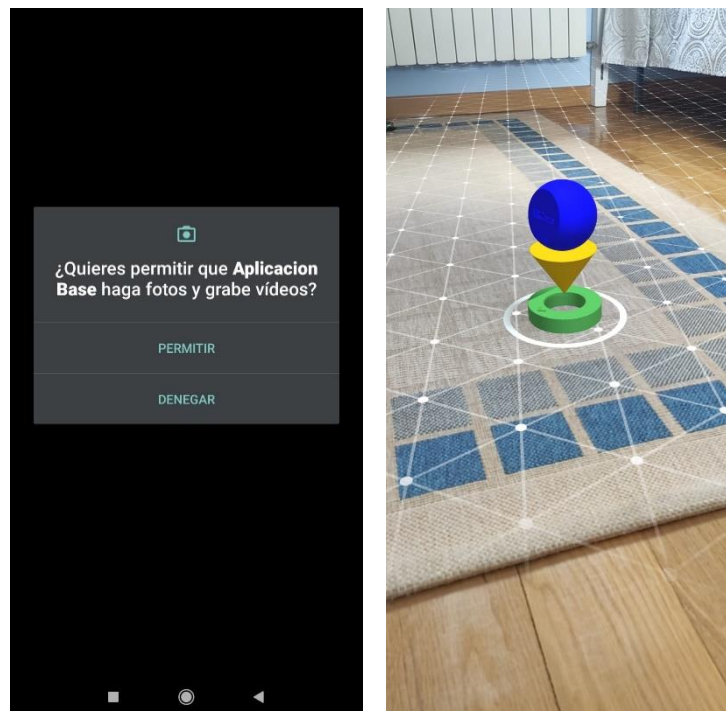


Figura 60. Vistas de la aplicación base en ejecución.

5. APLICACIÓN DEMOSTRADOR

En esta segunda parte del tutorial vamos a mejorar la aplicación base implementando elementos que aportarán nuevas funcionalidades. En primer lugar, aprenderemos a importar nuestros propios modelos 3D con sus propias animaciones y materiales. Después veremos cómo activar las animaciones y modificar los materiales mientras se ejecuta la aplicación. También realizaremos una mejora de su interfaz y la instalaremos. Finalizaremos el capítulo con una prueba para medir su usabilidad.

5.1 Modelos 3D propios

Antes de empezar a añadir elementos vamos a hacer una copia de la Aplicación Base para no modificar la original. Para ello iremos a **File > Save As** y guardaremos la nueva escena con el nombre "Modelos Propios". Para mantener un poco de orden en los assets vamos a crear dos carpetas nuevas (Botón derecho > Create > Folder), una para la Aplicación Base, en la que meteremos esa escena y el script *ObjectManipulator*, y otra en la que meteremos la nueva escena y los nuevos assets que incorporemos a esta escena.

Importación de modelos con materiales

Importar modelos 3D en Unity resulta realmente sencillo, basta con arrastrar el archivo correspondiente y soltarlo en la ventana proyecto. Unity soporta la importación de modelos desde las aplicaciones de modelado 3D más populares: Maya, Cinema 4D, 3ds Max, Cheetah3D, Modo, Lightwave, Blender y SketchUp. A través del siguiente enlace podemos acceder a información más específica sobre la importación para cada uno de estos programas:

➤ <https://docs.unity3d.com/es/530/Manual/HOWTO-importObject.html>

La importación de mallas en Unity se puede lograr mediante dos tipos de archivos principales:

- **Formatos de archivo 3D exportados.** Unity puede leer archivos .FBX, .dae (Collada), .3DS, .dxf y .obj.
- **Archivos propios de aplicación 3D.** Unity también puede importar, mediante conversión archivos : Max, Maya, Blender, Cinema4D, Modo, Lightwave & Cheetah3D, por ejemplo .Max, .MB, .MA, etc.

En el caso de importación de formatos del primer tipo existen algunas ventajas como que se exportan solo los datos necesarios, que son archivos generalmente

más pequeños y que soporta otros paquetes 3D para cuyo formato propietario no hay un soporte directo. Sin embargo, el segundo tipo de archivos también tiene puntos positivos, como un proceso rápido de iteración (al guardar el archivo fuente, Unity re-importa automáticamente) y la simplicidad en su uso.

Un archivo modelo puede contener un modelo 3D, como lo es un personaje, un edificio, o un mueble. El modelo es importado en múltiples assets. En la vista de proyecto, el objeto principal importado es un modelo prefabricado que puede contener, además de la malla 3D, otros elementos como un material o animaciones asociados a él. En la ventana inspector podemos acceder a las configuraciones de importación para el propio modelo 3D, su *rig* (para avatares animados), sus materiales y animaciones.

En este apartado del tutorial vamos a trabajar con un modelo sencillo de una silla, que ha sido creada en Blender. Por supuesto, el lector podrá utilizar el modelo 3D y el programa de modelado que desee, siempre y cuando el modelo exportado se encuentre en un formato compatible con Unity.

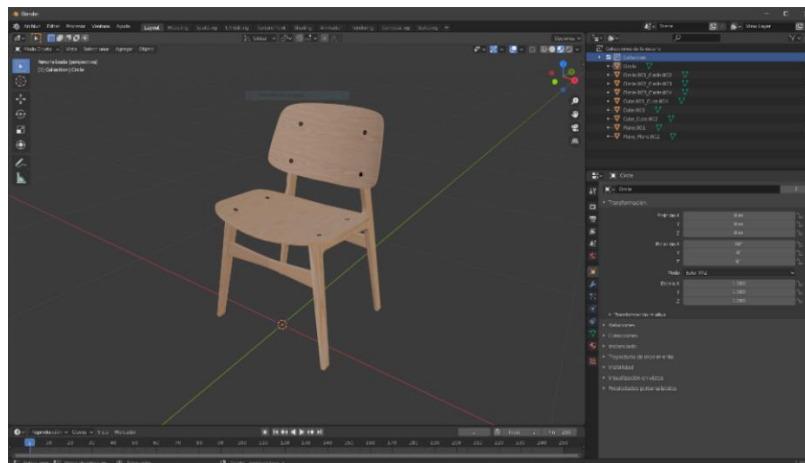


Figura 61. Silla modelada en Blender.

En este caso, como los archivos de Blender son compatibles con Unity, podríamos arrastrarlo directamente a la ventana de proyecto. Sin embargo, vamos a importar los modelos en Unity en formato **FBX**, que es un método más genérico ya que estos archivos pueden ser generados por muchas de las aplicaciones de modelado 3D. En este formato además del modelo podemos exportar cámaras, luces, animaciones, rigs, etc. Las aplicaciones a menudo permiten exportar objetos seleccionados o una escena entera.

Antes de proceder a exportar el modelo debemos asegurarnos de que se encuentra en una escala que se ajuste a la realidad, que el objeto esté bien centrado en el espacio y que su orientación es la correcta.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Ahora vamos a Archivo > Exportar > FBX (en Blender) y se abrirá una ventana con opciones de exportación en la cual debemos asignar un nombre al archivo y elegir dónde queremos guardarlo. Además, como la silla tiene asignados materiales con texturas vamos a establecer la configuración de modo de rutas (*Path Mode*) en **Copiar** y activamos la casilla de la derecha para incorporar las texturas, como se ve en la siguiente figura.

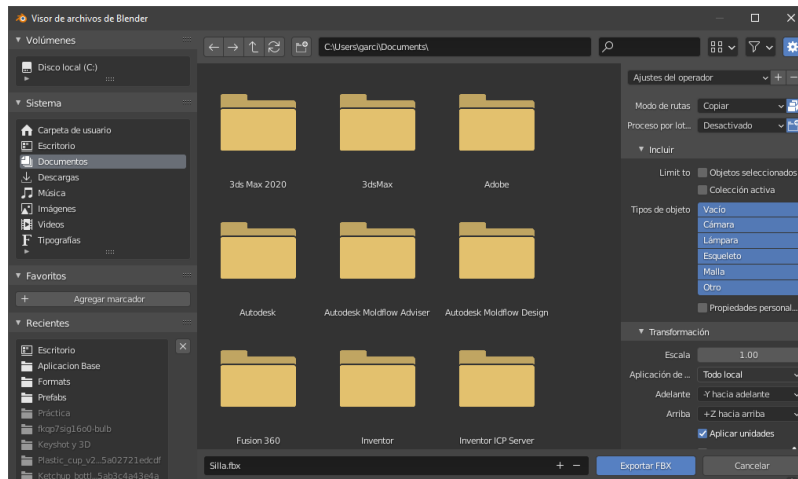


Figura 62. Ventana de exportación FBX en Blender.

Una vez exportado solo tenemos que ir al archivo y arrastrarlo hasta la ventana proyecto de Unity. Cuando se complete la importación veremos que entre los assets aparece nuestro modelo con una pestaña a la derecha, si la pulsamos se desplegará y podremos ver todo lo que contiene el archivo FBX, en este caso todas las piezas de la silla y los materiales.



Figura 63. Vista del modelo importado en Unity.

También podemos ver que los materiales aplicados al modelo no tienen las texturas que sí tenían en Blender. Para que se importen también estas texturas tenemos que ir a las **Import Settings** (en el Inspector) y en **Materials** debemos pulsar **Extract Textures**. En la ventana emergente seleccionamos la carpeta que contiene esta escena, aunque también podemos crear otra nueva dentro para guardar texturas y materiales. Veremos que las texturas se importan y se aplican a nuestro modelo.

Ahora vamos a activar la opción de **Generate Colliders**, también dentro de las configuraciones de importación, pero esta vez en el menú **Model > Meshes**. Así se generará un componente **Mesh Collider** para cada parte del modelo, es decir, se creará un colisionador a partir de la geometría de la malla correspondiente. De otra forma no sería posible manipular el modelo con entradas táctiles, porque los rayos generados por el *Raycast* no impactarían contra él y no sería reconocido.

También debemos asegurarnos de que el **Scale Factor** es 1 y que la conversión de unidades entre el archivo y Unity es la correcta (1 cm (File) to 0.01 m (Unity)).

Creación de un Prefab

Podríamos utilizar el modelo tal y como está ahora mismo, sin embargo, vamos a añadir otro elemento que nos servirá para proyectar las sombras del modelo sobre el plano en que se encuentra. Lo haremos a través de un prefab que contenga tanto el modelo de la silla como un plano transparente.

Para ello vamos a crear en la Jerarquía un nuevo objeto de juego vacío con el nombre "SillaPrefab", que debe estar en la posición (0,0,0). Después arrastraremos el modelo importado hasta el nuevo objeto para que sea su hijo.

Para crear el plano vamos a **GameObject > 3D Object > Quad**, lo llamamos "ShadowQuad" y lo hacemos hijo igual que antes. Debemos orientarlo horizontalmente, posicionarlo en el centro y escalarlo si hiciera falta. Para que el plano sea transparente y se proyecten sombras sobre él vamos a asignarle un material del SDK que se llama *ShadowPlane* y podemos encontrar en **Assets / GoogleARCore / Examples / Common / Materials**.

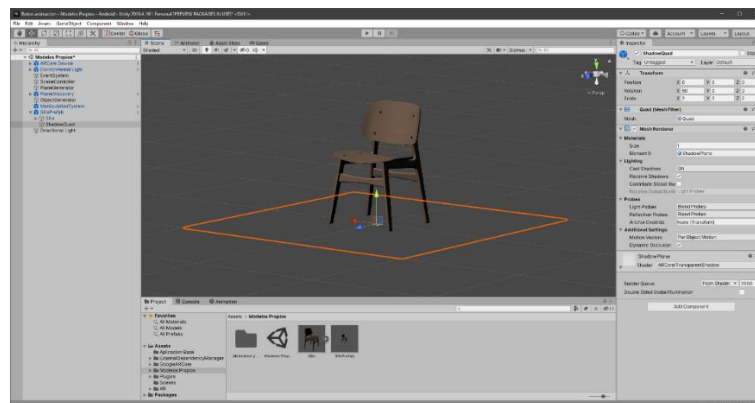


Figura 64. Vista completa del prefabricado del modelo.

Para que este objeto de juego que hemos creado se convierta en un prefabricado simplemente debemos arrastrarlo hasta nuestros assets en la ventana Proyecto. Ahora ya podemos eliminarlo de la Jerarquía y asignar el asset *SillaPrefab* como la variable *Object Prefab* del script *ObjectManipulator*.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Cabe aclarar que si vemos el modelo totalmente negro al añadirlo a la escena es porque no tenemos ninguna fuente de luz por defecto. Podemos añadir un objeto de juego de luz direccional para visualizar mejor el objeto, pero debemos recordar desactivarlo antes de instalar la aplicación.

Para este tutorial, como solo vamos a generar una instancia en la aplicación no nos interesa visualizar constantemente el anillo de selección que teníamos en la Aplicación Base. Por lo tanto, vamos a desactivar el script *SelectionManipulator* del prefabricado *Manipulator*.

Antes de llevar a cabo la instalación (igual que en el apartado 5.6) vamos a guardar la escena. También la agregamos a las escenas en construcción en **Build Settings** y desmarcamos la escena de la Aplicación Base. Por último, en **Player Settings > Player** y cambiamos el **Product Name** por "Modelos Propios".

Ahora ya podemos visualizar nuestro propio modelo 3D en realidad aumentada.



Figura 65. Vista de la silla en realidad aumentada.

5.2 Modelos con animaciones

Ahora que ya sabemos emplear nuestros propios modelos 3D, vamos a empezar a trabajar con animaciones, lo cual aumentará un poco el grado de complejidad de la aplicación. El objetivo es aprender a importar modelos con sus propias animaciones y activarlas mientras se ejecuta la aplicación a través de botones.

Para esta parte del tutorial vamos a utilizar otro modelo de ejemplo diferente. En este caso se trata de una silla de oficina, que al igual que antes, ha sido modelada en Blender. Esta silla es muy apropiada para animar ya que tiene varias partes móviles que se pueden regular.

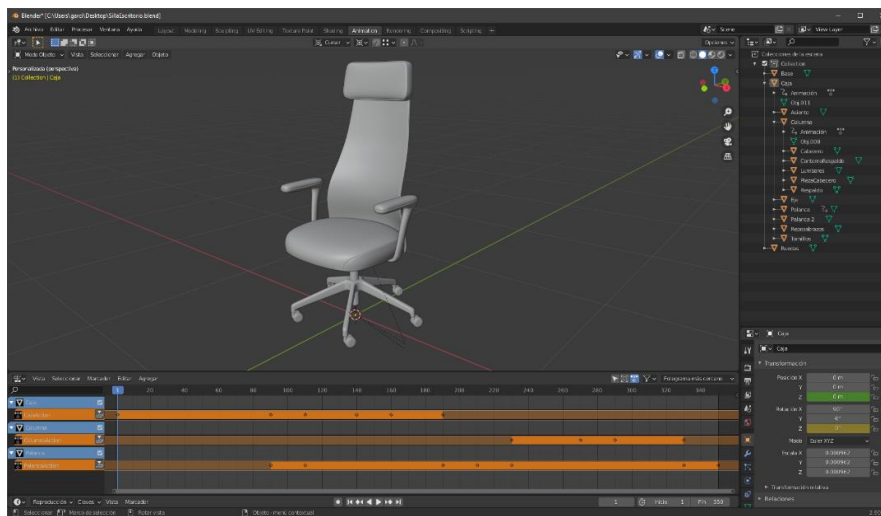


Figura 66. Silla de escritorio con animaciones en Blender.

Para este modelo se han creado tres animaciones diferentes: en la primera la parte superior realiza un giro de 360 grados sobre el eje de la silla, en la segunda se acciona la palanca hacia arriba, el asiento se eleva y después vuelve a su posición, y, por último, la palanca se acciona hacia afuera, el respaldo se reclina y retoma también su posición original. Estas acciones se desencadenan una detrás de otra con una duración total de 350 frames (fotogramas). Debemos recordar establecer la frecuencia de fotogramas de Blender en 30 fps, que es como funciona Unity.

Como se puede ver en la Figura 66, no hemos aplicado materiales en Blender. En este caso los crearemos en Unity, lo cual nos permitirá manipularlos más fácilmente en el siguiente apartado.

Una vez listo el modelo 3D podemos exportarlo en formato FBX, como hicimos con la silla anterior. Ahora no es necesario incorporar las texturas, pero sí debemos asegurarnos de que esté marcada la opción Capturar Animación y, dentro de ella, estén desactivadas las tres primeras configuraciones. De este modo las animaciones se guardarán en el archivo como una única animación

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

global y no se asignará cada animación a cada una de las partes del modelo. Después nos encargaremos de dividir la animación como corresponde dentro de Unity.

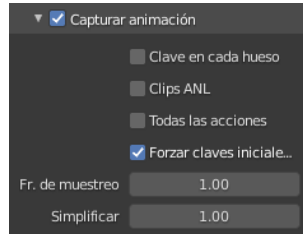


Figura 67. Configuración para exportar animaciones en Blender.

Antes de importar el nuevo archivo en nuestro proyecto, vamos a hacer una copia de la escena como hicimos al comienzo del apartado anterior. En este caso la llamaremos "Modelos Animados" y crearemos también una carpeta del mismo nombre para almacenarla junto con los nuevos assets.

Importación de modelos con animaciones

A continuación, vamos a importar el archivo FBX en la misma carpeta. Si lo desplegamos en la ventana Proyecto, veremos, además de todas sus partes, un material sin nombre y una animación llamada Scene.

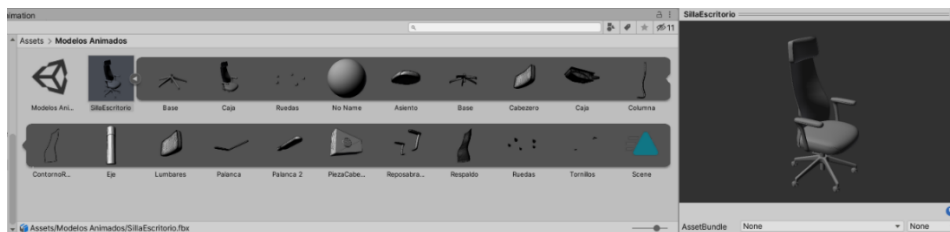


Figura 68. Vista de la silla de escritorio importada en Unity.

En las **configuraciones de importación** del modelos vamos a activar la generación de colisionadores y después, en el menú **Animation** vamos a dividir la animación global en tres clips, uno para cada una de las animaciones que se habían creado en Blender.

Para ello debemos fijarnos en las duraciones de las animaciones en Blender, por ejemplo, nuestra primera animación va desde el fotograma 1 hasta el 90. Ahora añadimos un clip a la lista, le asignamos un nombre y los fotogramas inicial y final como corresponda. Debemos hacer lo mismo con el resto de las animaciones y obtendremos una lista de clips como en la Figura 69. Finalmente, le daremos a **Apply** y veremos que aparecen tres animaciones más entre los assets asociados a nuestro modelo.

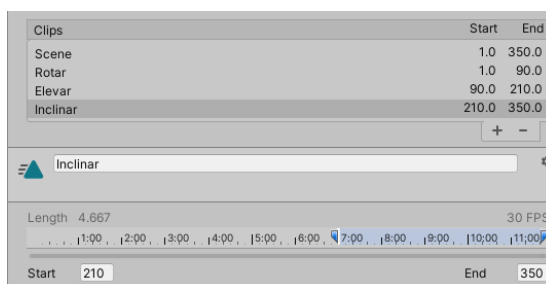


Figura 69. Clips de animación en Unity.

Ahora que ya tenemos el modelo y las animaciones listas vamos a crear un objeto de juego prefabricado como hicimos con la silla del ejemplo anterior, añadiendo igualmente un plano que reciba las sombras del objeto. El prefabricado se llamará “SillaEscritorioPrefab”.

El componente Animator

Después abrimos el prefab desde la ventana Proyecto para agregarle un componente **Animator** al modelo 3D. Este tipo de componentes son utilizados para asignar una animación (o varias) a un GameObject dentro de la escena. Como veremos, requiere una referencia a un *Animator Controller* que define qué clips de animación usar, y controla cuándo y cómo mezclar y hacer una transición entre estos.

Para crear este componente en la silla es tan sencillo como arrastrar los tres clips de animación que hemos creado hasta el modelo dentro del prefabricado.

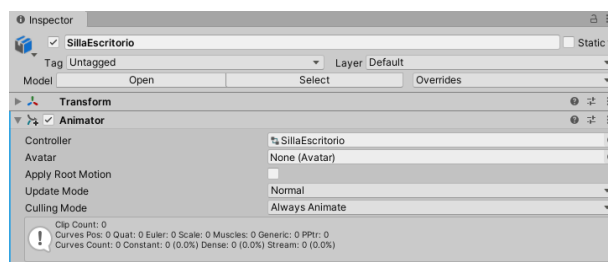


Figura 70. Componente Animator para la silla de escritorio.

Veremos que con el nuevo componente también ha aparecido un *Animator Controller* ya asignado, con el mismo nombre que el objeto de juego y que aparece como un nuevo elemento entre nuestros assets.



Figura 71. Animator Controller de la silla de escritorio.

Si hacemos doble clic en el controlador de animación veremos cómo se abre una nueva ventana en Unity llamada *Animator*. Desde esta ventana podremos organizar nuestras animaciones, crear transiciones, añadir nuevos estados, etc.

En nuestro caso tendremos por defecto algo similar a lo que se muestra en la Figura 72, es decir, los tres clips de animación, una entrada (en verde), una salida (en rojo) y "Any State" (en azul), que sirve para pasar a un estado específico sin importar el estado actual. La animación que aparece en naranja es la que se considera el estado por defecto, por lo que en este caso el modelo ejecutaría esta acción inmediatamente después de aparecer, y no es lo que queremos.

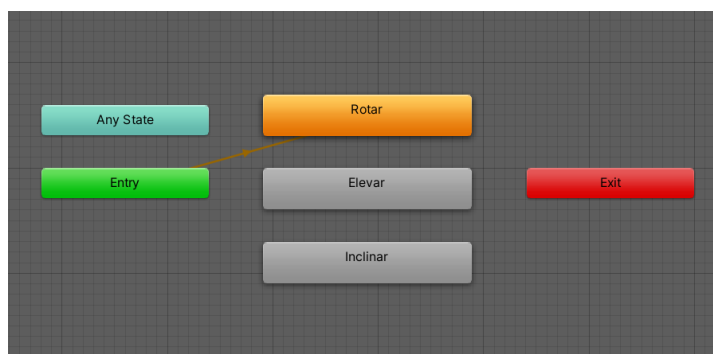


Figura 72. Vista inicial de la ventana Animator.

Nuestro objetivo es activar las animaciones mediante botones, por lo que necesitamos establecer un estado por defecto que no lleve a cabo ninguna acción. Para ello hacemos **Botón Derecho > Create > Empty** y aparecerá un nuevo estado vacío. Para asignarlo como estado por defecto hacemos clic con el **Botón Derecho** sobre él y le damos a **Set As Layer Default State**. Además, queremos que cuando finalice cualquiera de las animaciones, el modelo vuelva a su estado por defecto. Para ello vamos a crear transiciones haciendo de clic con el **Botón Derecho** sobre la animación, dando a **Make Transition** y arrastrando hasta el nuevo estado. En la Figura 73 podemos ver el resultado final.

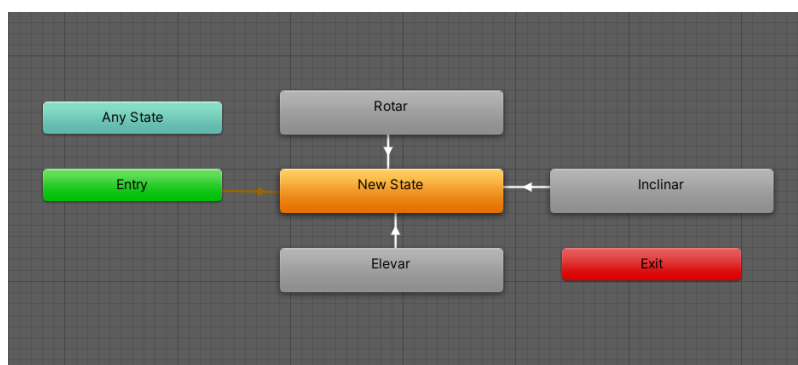


Figura 73. Vista final de la ventana Animator.

Ahora que ya tenemos las animaciones bien organizadas en el *Animator*, vamos a ver cómo podemos hacer para activarlas mediante botones. Podríamos intentar crear un botón y asignarle directamente el objeto prefabricado para activar alguna de sus animaciones. Sin embargo, al ejecutar la aplicación veríamos que el objeto no hace nada porque se trata de una instancia y no del prefab original. Por lo tanto, cuando queramos activar la animación debemos asegurarnos de hacer referencia al objeto instanciado. Con este fin vamos a crear un controlador de animaciones para las instancias.

Interacción para activar animaciones

Lo que pretendemos conseguir con este nuevo elemento es encontrar, mediante un script, el objeto instanciado, acceder a su componente *Animator* y crear funciones que activen las animaciones.

Añadiremos a la escena un nuevo objeto de juego vacío que llamaremos *InstanceAnimator*, al cual agregaremos un componente C# script del con el mismo nombre.

En primer lugar, declararemos dos variables privadas. La primera, de tipo `GameObject`, se llamará "Instance", y, como su propio nombre indica, nos servirá para almacenar la instancia del prefabricado. La segunda será de tipo *Animator* y la usaremos para el componente que vamos a tomar de la instancia.

```
private GameObject instance;  
private Animator anim;
```

A continuación, crearemos el método **FindObject()**, donde vamos a buscar la instancia de nuestro modelo de silla (no el prefabricado que contiene el modelo) a través de su nombre para asignar ese `GameObject` a la variable *instance*. Después, le extraeremos su componente *Animator* y se lo asignaremos a la variable *anim*.

```
void FindObject()  
{  
    instance = GameObject.Find("SillaEscritorio");  
    anim = instance.GetComponent<Animator>();  
}
```

Después vamos a crear una función pública para cada animación. En este caso serán tres, que denominaremos **PlayRotar()**, **PlayElegar()** y **PlayInclinar()**. Dentro de cada una ellas, llamaremos a la función anterior **FindObject()** y después, se activará la animación correspondiente.

```
public void PlayRotar()
{
    FindObject();
    anim.Play("Rotar", -1, 0f);
}

public void PlayElevar()
{
    FindObject();
    anim.Play("Elevar", -1, 0f);
}

public void PlayInclinar()
{
    FindObject();
    anim.Play("Inclinar", -1, 0f);
}
```

Una vez tenemos el script programado podemos pasar a estudiar cómo crear los botones y hacer que activen las funciones que acabamos de crear.

Canvas

En primer lugar, vamos a crear un nuevo objeto de juego **Canvas**, que representa un espacio abstracto en el cual los elementos UI (Interfaz de Usuario) son colocados y renderizados. Todos los elementos de este tipo deben ser hijos del Canvas. Para crearlo vamos al menú **GameObject > UI > Canvas**.

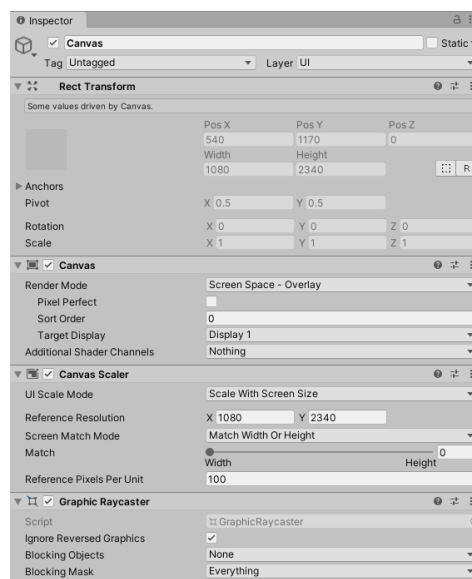


Figura 74. Vista del Inspector para el Canvas.

Este objeto de juego contiene cuatro componentes diferentes. En primer lugar, el *Rect Transform*, que es la contraparte del diseño 2D del componente Transform. Después tenemos el propio componente Canvas. Dentro de este, vamos a establecer el **Render Mode** en **Screen Space – Overlay**, que coloca los elementos UI en la pantalla mostrada en la parte superior de la escena. Si el tamaño de la pantalla es modificado o cambia la resolución, el Canvas automáticamente cambiará el tamaño para que coincida. En tercer lugar, se encuentra el *Canvas Scaler*, en el cual elegiremos **Scale With Screen Size** como modo de escalado para conseguir que el Canvas se escale en función de una resolución de referencia (1080x2340 en el caso de mi móvil). Por último, el *Graphic Raycaster* que es utilizado para hacer raycast (emitir rayos) contra un Canvas. El Raycaster mira todos los gráficos en el Canvas y determina si uno de ellos ha sido golpeado.

Ahora vamos a crear tres botones sencillos para activar nuestras animaciones. Para ello vamos al menú **GameObject > UI > Button** y veremos que se crea directamente como hijo del Canvas. Hacemos lo mismo para crear otros dos botones y a cada uno de estos objetos de juego le asignamos el nombre de la animación que van a activar.

Para cambiar el texto que aparece en el botón debemos acceder al objeto de juego *Text* que es hijo suyo. En él podemos editar también otros aspectos como la tipografía, el tamaño de letra, la alineación, el color, etc.

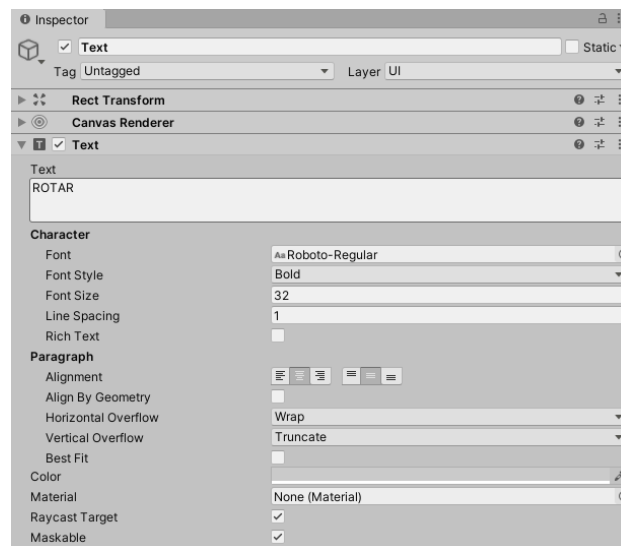


Figura 75. Modificación del texto de un botón.

En el caso de que quisiéramos crear un botón a partir de una imagen en vez de con texto deberíamos eliminar el objeto de juego *Text* y después, arrastrar nuestra imagen hasta la opción *Source Image* dentro del componente *Image* del botón.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

También podemos modificar otros parámetros del botón, sin embargo, no nos detendremos en ellos puesto que tan solo necesitamos tres botones sencillos. Lo que si haremos será posicionarlos en la parte inferior de la pantalla de forma ordenada. Para visualizar mejor los elementos del Canvas es recomendable activar la opción 2D de la ventana Escena.

Vamos a utilizar el componente *Rect Transform* para modificar la posición de los botones. Gracias a la opción **Anchor Presets** (en la parte superior izquierda) podemos alinearlos al centro y establecer su ancla en la parte inferior para que se mantengan abajo, aunque cambie la resolución de la pantalla.

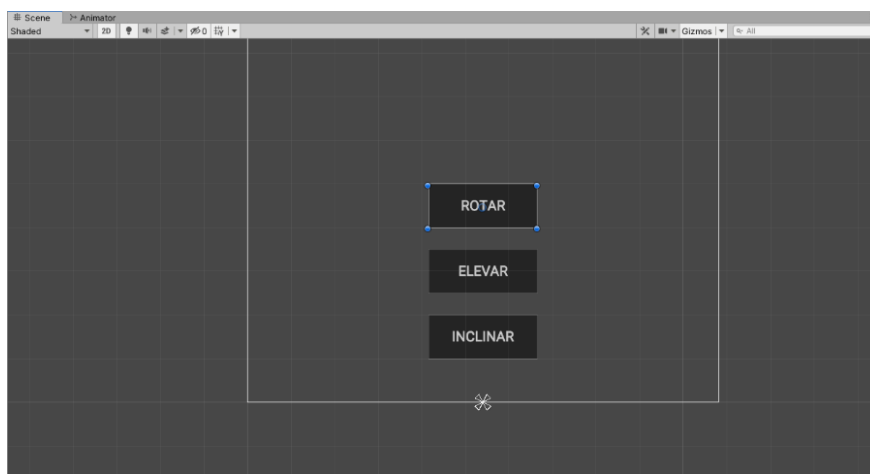


Figura 76. Vista de Escena del Canvas en 2D.

Ahora ya solo nos falta asignarle una acción a los botones. En el componente *Button* de los botones encontraremos una lista vacía con el título **On Click ()**, que lo que hace es invocar un evento cuando el usuario hace clic en el botón y lo suelta. Añadiremos un elemento a la lista y arrastraremos el objeto *InstanceAnimator* hasta el hueco disponible. Después, en la opción desplegable de la derecha seleccionamos el script del objeto (*InstanceAnimator*) y la función que activa la animación que corresponda.

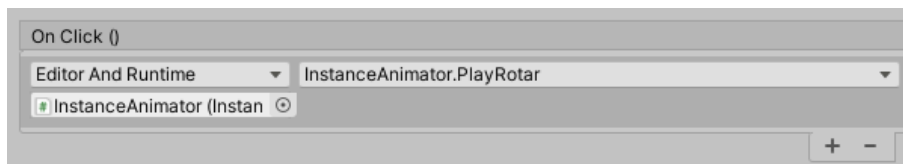


Figura 77. Configuración On Click para la animación Rotar.

Ahora ya tenemos programados los botones con las funciones que activan las animaciones, pero antes de proceder a instalar la aplicación vamos a aplicarle materiales a nuestro modelo.

Añadiremos una carpeta nueva en la que almacenar los materiales e importar las texturas (en el caso de que se usen). Para crear los materiales hacemos clic con el **Botón Derecho** en la ventana proyecto y le damos a **Create > Material**. Para aplicarlo solo hay que arrastrarlo hasta la parte del modelo deseada, ya sea en la Escena o en la Jerarquía. La configuración de los materiales es muy similar a otros programas de 3D, por lo que no se va a entrar en detalle.



Figura 78. Materiales y texturas usados en la silla de escritorio.

Para finalizar, vamos a instalar la aplicación como hemos hecho anteriormente. Debemos recordar añadir esta escena a las **Scenes in Build** y deseleccionar las anteriores. También podemos cambiar el **Product Name** por “Modelos Animados”.

En la Figura 79 podemos ver algunas capturas de esta aplicación en ejecución con la silla en diferentes posiciones.



Figura 79. Capturas antes y después de presionar el botón INCLINAR.

5.3 Cambios en el material del modelo

Para una experiencia de usuario más amplia con nuestra aplicación de visualización de productos en realidad aumentada, vamos a implementar nuevos elementos que permitirán una nueva funcionalidad, la personalización de los materiales del modelo en tiempo de ejecución de la aplicación.

Al igual que en otras ocasiones, vamos a partir de la escena anterior, por lo que haremos una copia de esta que llamaremos "Cambiar Material". También crearemos una carpeta con el mismo nombre para almacenar los nuevos assets.

Para esta parte del tutorial seguiremos utilizando el mismo objeto prefabricado con la silla de escritorio, solamente cambiaremos los materiales de algunas de las partes que la componen. La acción del cambio se llevará a cabo a través de elementos UI, al igual que las animaciones.

En este ejemplo solo vamos a modificar las partes de la silla que originalmente tenían un material de tela gris. Por lo tanto, en primer lugar, vamos a crear otros dos materiales (Assets > Create > Material), uno será de cuero negro y otro de tela azul, como se ve en la Figura 80.

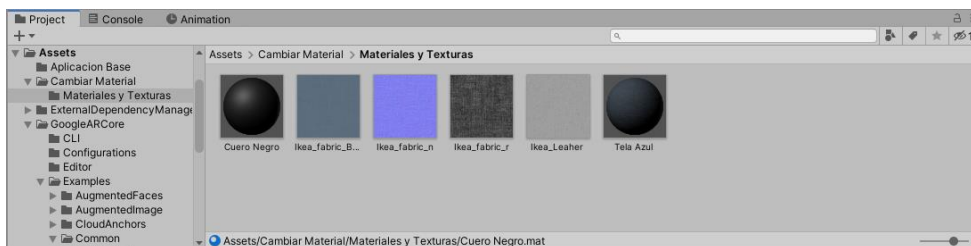


Figura 80. Nuevos materiales para la silla de escritorio.

Interacción para cambiar materiales

A continuación, crearemos un nuevo objeto de juego vacío que llamaremos *MaterialController* y le asignaremos un nuevo C# script con el nombre *ChangeMaterial*. Este componente lo programaremos con las funciones que realizarán el cambio en el material.

Vamos a empezar a modificarlo partiendo del código por defecto que ya tiene el script. En primer lugar, vamos a declarar tantas variables privadas de tipo *GameObject* como partes del modelo queramos modificar y una matriz (*array*) pública de tipo *Material*, tendrá un tamaño igual al número de materiales que queramos utilizar, en este caso 3.

```
private GameObject cabecero;
private GameObject lumbares;
private GameObject asiento;

public Material[] materials = new Material[3];
```

Crearemos el método **FindParts()** donde vamos a buscar las partes del modelo a las que queremos cambiar el nombre a través de su nombre y las almacenaremos en las variables privadas que habíamos declarado, como se ve en el siguiente código.

```
void FindParts()
{
    cabecero = GameObject.Find("Cabecero");
    lumbares = GameObject.Find("Lumbares");
    asiento = GameObject.Find("Asiento");
}
```

Finalmente, crearemos otras tres funciones: **ChangeTelaGris()**, **ChangeTelaAzul()** y **ChangeCuero()**. Dentro de cada una de ellas accederemos al componente *MeshRenderer* de cada parte del modelo y cambiaremos su material por uno de los materiales que contiene la matriz pública que habíamos declarado. En este caso el material de tela gris se corresponde con el elemento 0 de la matriz, la tela azul con el 1 y el cuero con el 2.

```
public void ChangeTelaGris()
{
    FindParts();
    cabecero.GetComponent<MeshRenderer>().material = materials[0];
    lumbares.GetComponent<MeshRenderer>().material = materials[0];
    asiento.GetComponent<MeshRenderer>().material = materials[0];
}

public void ChangeTelaAzul()
{
    FindParts();
    cabecero.GetComponent<MeshRenderer>().material = materials[1];
    lumbares.GetComponent<MeshRenderer>().material = materials[1];
    asiento.GetComponent<MeshRenderer>().material = materials[1];
}

public void ChangeCuero()
{
    FindParts();
    cabecero.GetComponent<MeshRenderer>().material = materials[2];
    lumbares.GetComponent<MeshRenderer>().material = materials[2];
    asiento.GetComponent<MeshRenderer>().material = materials[2];
}
```

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Debemos tener en cuenta el orden de los elementos a la hora de asignar los materiales a los valores de la matriz desde el Inspector. Para este ejemplo el script debería quedar configurado como se muestra en la Figura 81.

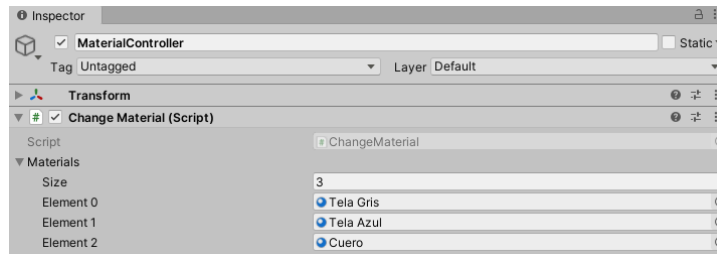


Figura 81. Vista del Inspector para MaterialController.

Ahora ya solo nos queda agregar al Canvas los elementos UI que activaran las funciones del script anterior. En este caso vamos a utilizar también botones, pero no contendrán texto, sino que serán imágenes de la silla con cada material aplicado.

Primero, vamos a crear tres botones como hicimos en el apartado anterior y eliminamos el objeto de texto que tienen como hijo. Importamos las imágenes arrastrándolas hasta la carpeta que contiene los assets de esta escena. Para poder utilizarlas como botones debemos ir a las **Import Settings** en el Inspector, establecer el **Texture Type** en **Sprite (2D and UI)** darle a **Apply**.

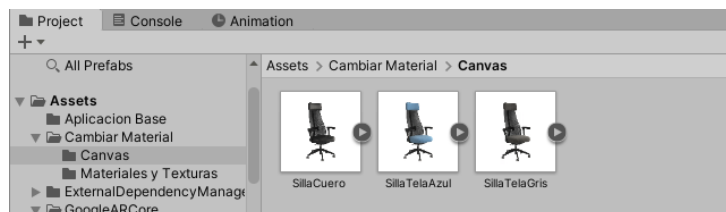


Figura 82. imágenes de las sillas tipo Sprite.

Después arrastramos cada imagen hasta la opción **Source Image** dentro del componente **Image** de cada botón. Es recomendable que el tamaño de las imágenes se ajuste al que tendrán en el Canvas, en este caso 250x250 pixeles.

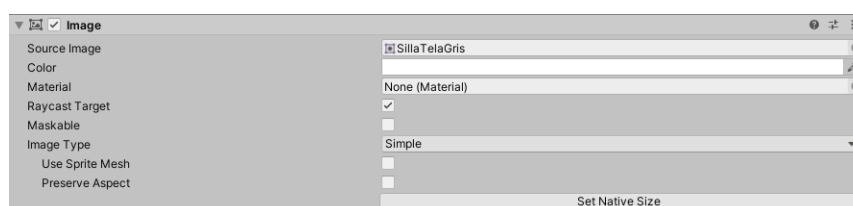


Figura 83. Componente Image de uno de los botones.

A continuación, vamos a asignar las funciones a los botones como hicimos para las animaciones en el apartado anterior. Arrastramos el objeto *MaterialController* hasta un nuevo evento **On Click()** y buscamos en el script *ChangeMaterial* el método que corresponda a cada botón.

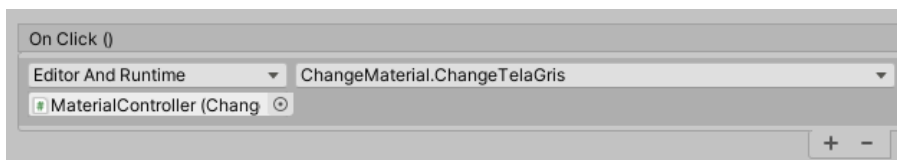


Figura 84. Configuración On Click para cambiar a Tela Gris.

Finalmente, vamos a instalar la aplicación como en otras ocasiones. Antes vamos a añadir esta escena a las **Scenes in Build**, deseleccionando las anteriores. También cambiaremos el **Product Name** por "Cambiar Materiales".

En la Figura 85 se muestran algunas dos capturas de esta aplicación en ejecución con la silla con diferentes materiales aplicados.

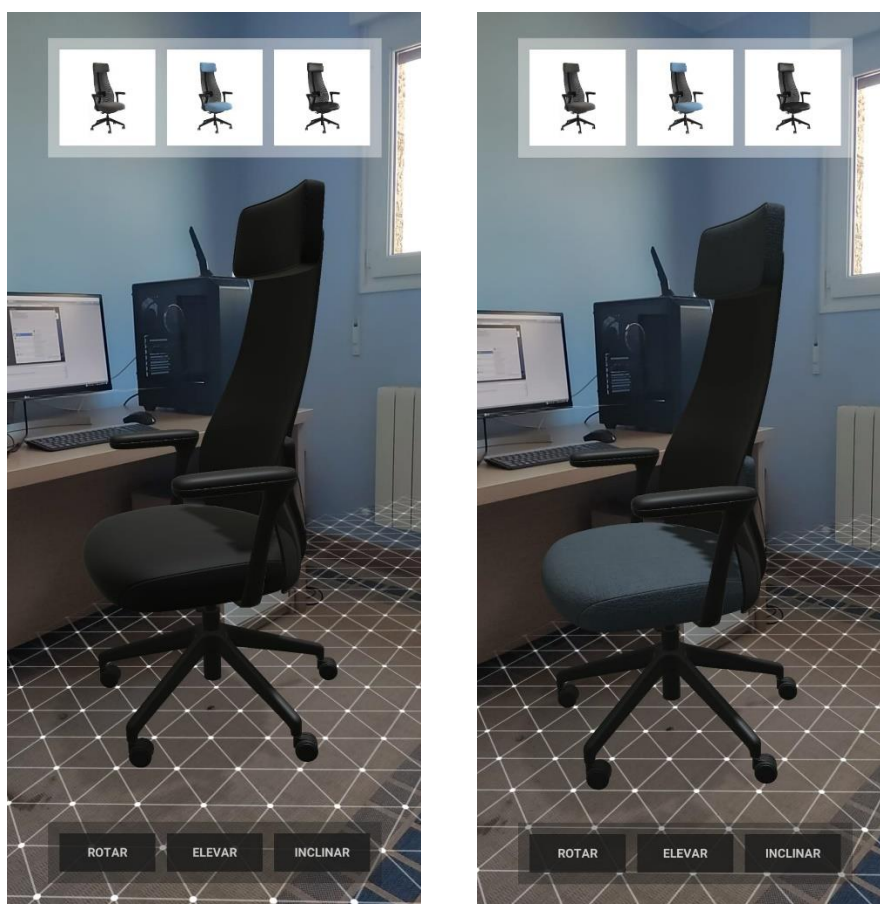


Figura 85. Capturas de la silla de escritorio con materiales diferentes

5.4 Mejoras en la interfaz

En este punto del tutorial ya contamos con una aplicación totalmente funcional para mostrar nuestros modelos de productos, con animaciones y cambios de material. Por lo tanto, podríamos considerar que ya hemos alcanzado nuestros objetivos, sin embargo, en este último apartado de la Aplicación Demostrador vamos a mostrar cómo realizar algunas mejoras en su interfaz de cara a conseguir una mejor experiencia de usuario.

Todos los cambios que realizaremos en este apartado serán aplicados sobre elementos UI y no afectarán, en esencia, al funcionamiento de la aplicación, por lo que podrían omitirse si así se desea.

Antes de empezar a modificar la interfaz vamos a crear una copia de última escena como hemos hecho en otras ocasiones. La llamaremos "Aplicación Demostrador" y la guardaremos en una carpeta con el mismo nombre, donde también almacenaremos los nuevos assets de la escena final.

En primer lugar, vamos a modificar los botones ya existentes, empezando por lo que modifican el material de algunas partes del modelo. Anteriormente eran tres botones cuadrados separados, cada uno con un cambio de material diferente. Lo que haremos ahora será colocar los tres botones en el mismo punto de manera que al pulsar uno se cambia el material y aparece el botón con el material correspondiente. Por ejemplo, si está activado el botón de Tela Gris la silla tiene este material, al pulsar el botón, se desactiva este y aparece el siguiente material, que también se aplica a la silla.

En la Figura 86 podemos ver el aspecto de los nuevos botones. Al igual que los demás, han sido diseñados en Adobe Illustrator y exportados como imágenes en formato PNG, directamente con el tamaño que tendrán en el Canvas.

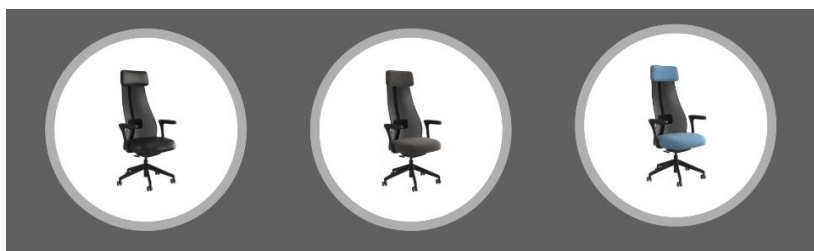


Figura 86. Nuevo diseño de los botones para los cambios de material.

Una vez importadas estas imágenes en Unity debemos establecer el **Texture Type** en **Sprite (2D and UI)**, como vimos en el apartado anterior. Después arrastramos cada imagen al **Source Image** del botón correspondiente y los colocamos todos en la misma posición en el Canvas. En este caso irán en la parte inferior de la pantalla.

Ahora debemos programar estos botones para que se alternen y realicen los cambios de material de manera correcta. Para ello vamos a crear un nuevo script llamado *CanvasController*, que será un componente del Canvas. Lo haremos así porque utilizaremos este mismo script para controlar también otros elementos UI.

Una vez abierto el script en Visual Studio vamos a empezar declarando cuatro nuevas variables públicas: tres de tipo *GameObject* para los botones y otra de tipo *ChangeMaterial* para el script del mismo nombre que configuramos en el apartado anterior.

```
public GameObject botonTelaGris;
public GameObject botonTelaAzul;
public GameObject botonCuero;

public ChangeMaterial ChangeMaterial;
```

Finalmente crearemos tres nuevas funciones públicas: **OnClickTelaGris()**, **OnClickTelaAzul()** y **OnClickCuero()**. En cada una de ellas se hará una llamada a una función de cambio de material del script *ChangeMaterial* y después se activan y desactivan los botones correspondientes.

```
public void OnClickTelaGris()
{
    //Se cambia el material a Tela Azul
    ChangeMaterial.ChangeTelaAzul();
    //Se activa el botón Tela Azul y se desactivan los otros dos
    botonTelaGris.SetActive(false);
    botonTelaAzul.SetActive(true);
    botonCuero.SetActive(false);
}

public void OnClickTelaAzul()
{
    //Se cambia el material a Cuero
    ChangeMaterial.ChangeCuero();
    //Se activa el botón Cuero y se desactivan los otros dos
    botonTelaGris.SetActive(false);
    botonTelaAzul.SetActive(false);
    botonCuero.SetActive(true);
}

public void OnClickCuero()
{
    //Se cambia el material a Tela Gris
    ChangeMaterial.ChangeTelaGris();
    //Se activa el botón Tela Gris y se desactivan los otros dos
    botonTelaGris.SetActive(true);
    botonTelaAzul.SetActive(false);
    botonCuero.SetActive(false);
}
```

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Ahora son estas funciones del Canvas las que debemos asignar a cada uno de los botones, en vez de las del *MaterialController*.

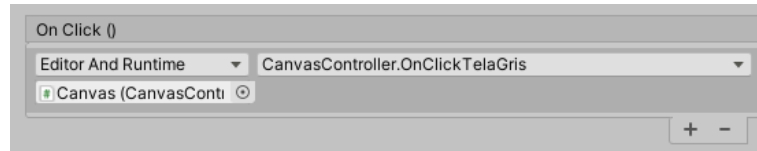


Figura 87. Configuración final On Click para cambiar a Tela Gris.

Por otra parte, también se ha cambiado el diseño de los botones de las animaciones por unos iconos circulares que irán dispuestos alrededor del botón de cambio de material, como se en la Figura 88.

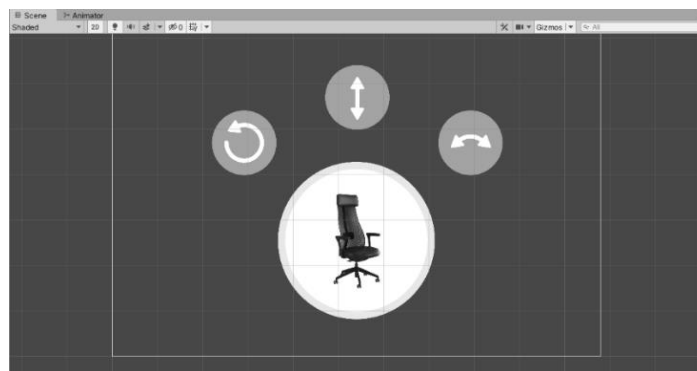


Figura 88. Parte inferior del Canvas definitivo.

Con esta nueva distribución hemos conseguido concentrar todos los elementos UI que teníamos en una zona, dejando suficiente espacio en el resto de la pantalla para ver y manipular el modelo en tiempo de ejecución.

Ahora vamos a añadir un nuevo elemento al Canvas de tipo *Toggle*, que llamaremos *Toggle Planes*. Se trata de una especie de conmutador que varía entre dos estados y que nos permitirá activar o desactivar la visualización de los planos detectados. Con esto pretendemos conseguir un mayor realismo en la integración del modelo en el entorno real, ya que las mallas triangulares de los planos pueden destacar demasiado.

Los objetos de juego de este tipo contienen a su vez otros dos elementos que son imágenes: el *Background* y el *Checkmark*, siendo el segundo hijo del primero. El *Background* es el fondo del elemento y el *Checkmark* aparece y desaparece dependiendo de si se encuentra activado o desactivado. En este caso se han cambiado las imágenes del *Toggle* para conseguir un icono que represente la función que ejerce. En la Figura 89 podemos verlo en sus dos estados y ubicado en la parte superior derecha del Canvas.

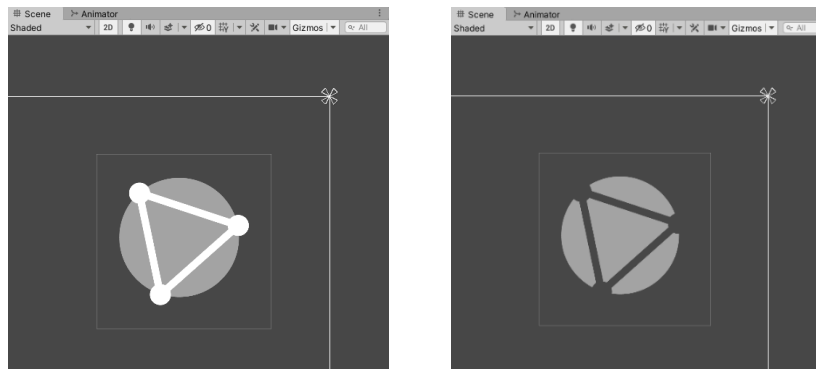


Figura 89. Toggle Planes activado (izq.) y desactivado (dcha.).

Para poder activar y desactivar la visualización de los planos detectados debemos realizar algunas modificaciones en el script *DetectedPlaneGenerator* del objeto de juego *PlaneGenerator*. Lo que haremos realmente será modificar el material con el que se renderizan los planos, siendo transparente cuando el *toggle* está desactivado. De esta forma desaparecerán aparentemente todas las instancias de planos, es decir, instancias del prefab *DetectedPlaneVisualizer*. Sin embargo, también debemos evitar que se creen nuevas instancias de planos mientras esté desactivado.

Antes de empezar a modificar el script vamos a abrir el prefabricado mencionado para asignarle una nueva etiqueta, que nos ayudará a localizar todas las instancias de este objeto. En la ventana Inspector abrimos la opción desplegable **Tag**, le damos a **Add Tag...** y añadimos una nueva etiqueta llamada “plane” a la lista.

Ahora, dentro del script vamos a añadir tres nuevas variables. Dos de ellas serán públicas, de tipo material y que almacenaran el material transparente y que contiene la malla triangular. La tercera variable será booleana, inicialmente verdadera y nos servirá permitir o no la búsqueda de nuevos planos.

```
public Material transparent;
public Material grid;

bool search = true;
```

Después vamos a añadir una nueva función llamada **OnTogglePlanes()** con una variable booleana de entrada llamada *toggle*, ya que cambiará de estado al cambiar el del elemento UI. Dentro de esta función, en primer lugar, se establece que la variable *search* es igual a *toggle*. Después, se buscan todos los objetos de juego con la etiqueta “plane” y se almacenan dentro de una nueva variable del mismo nombre. El material asignado a estos objetos será *grid* o *transparent*, dependiendo del estado del *toggle*.

TRABAJO DE FIN DE GRADO
APLICACIÓN DE REALIDAD AUMENTADA

```
public void OnTogglePlanes(bool toggle)
{
    search = toggle;

    foreach (GameObject plane in GameObject.FindGameObjectsWithTag("plane"))
    {
        if (toggle)
        {
            plane.GetComponent<MeshRenderer>().material = grid;
        }
        else
        {
            plane.GetComponent<MeshRenderer>().material = transparent;
        }
    }
}
```

Finalmente, dentro de la función **Update()** vamos a introducir la parte del código que se encarga de crear nuevas instancias de planos dentro de una declaración lógica **if()** para que solo se lleve a cabo en el caso de que la variable *search* sea verdadera.

```
if (search)
{
    Session.GetTrackables<DetectedPlane>(_newPlanes,
    TrackableQueryFilter.New);
    for (int i = 0; i < _newPlanes.Count; i++)
    {
        GameObject planeObject =
            Instantiate(DetectedPlanePrefab, Vector3.zero,
            Quaternion.identity, transform);

        planeObject.GetComponent<DetectedPlaneVisualizer>().Initialize(_newPlanes[i]);
    }
}
```

Ahora tenemos que asignarle la función **OnTogglePlanes** del *DetectedPlaneGenerator* al elemento *Toggle Planes* en la configuración **On Value Changed (Boolean)**, como se ve en la Figura 90.

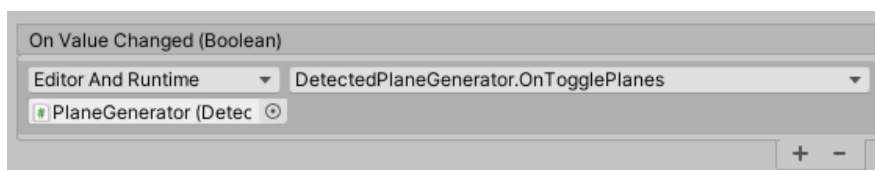


Figura 90. Configuración On Value Changed para el Toggle Planes.

También debemos recordar establecer los valores de las nuevas variables públicas del script. El material transparente será el *ShadowPlane* que utilizamos para los planos de los Prefabs de los modelos. El *PlaneGrid* lo podemos encontrar entre los assets del SDK.

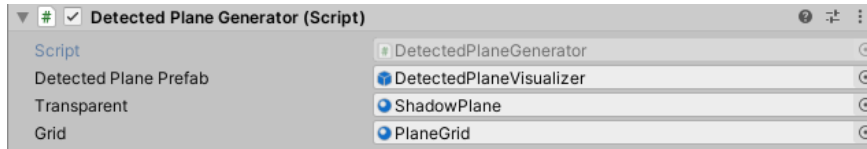


Figura 91. Variables públicas del *DetectedPlaneGenerator*.

A continuación, vamos a añadir otro elemento *toggle*, que servirá para activar y desactivar el resto de los elementos del Canvas. Lo que buscamos con esto es ofrecer la posibilidad de realizar una captura de pantalla sin que existan elementos UI que tapen una parte de la imagen. En este caso vamos a utilizar el logotipo de la Escuela de Ingenierías Industriales, por eso lo llamaremos *Toggle EII*. Cuando este *toggle* se encuentre desmarcado quedará como una pequeña marca de agua con la forma del logo en la esquina superior izquierda.

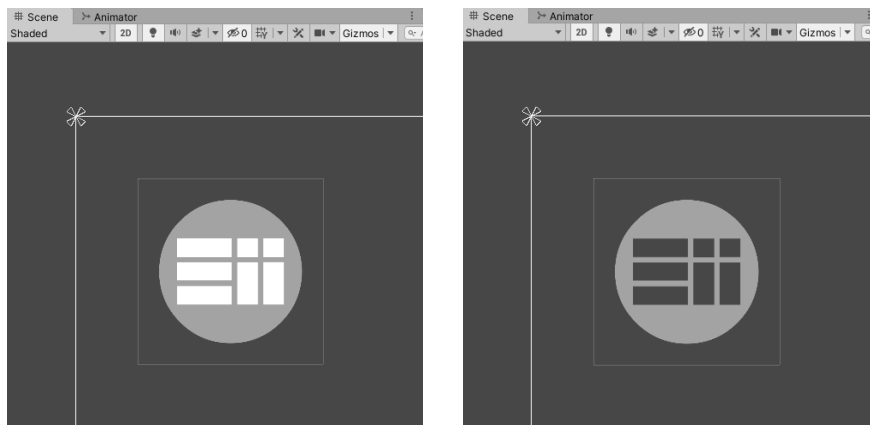


Figura 92. *Toggle EII* activado (izq.) y desactivado (dcha.).

Para programar el funcionamiento de este *toggle* vamos a volver al script *CanvasController*. En primer lugar, vamos a añadir a las variables que habíamos declarado antes otras para el resto de los elementos UI, es decir, los botones de animación y el *Toggle Planes*. En este caso hemos agrupado los tres botones de animación dentro de un elemento de tipo Panel (*GameObject > UI > Panel*) para trabajar más cómodamente. Por tanto, nos vale con declarar solo dos variables públicas.

```
public GameObject panelAnim;
public GameObject togglePlanes;
```

Ahora añadiremos una nueva función llamada **OnToggleEII()** con una variable booleana de entrada llamada *toggle*, al igual que hicimos antes para el otro elemento del mismo tipo. Dentro de esta función los elementos UI estarán activados o desactivados dependiendo del valor de la variable booleana *toggle*.

```
public void OnToggleEII(bool toggle)
{
    botonTelaGris.SetActive(toggle);
    botonTelaAzul.SetActive(toggle);
    botonCuero.SetActive(toggle);
    panelAnim.SetActive(toggle);
    togglePlanes.SetActive(toggle);
}
```

Al igual que hicimos antes, tenemos que asignarle la función **OnToggleEII** del *CanvasController* al elemento *Toggle EII* en la configuración **On Value Changed (Boolean)**.

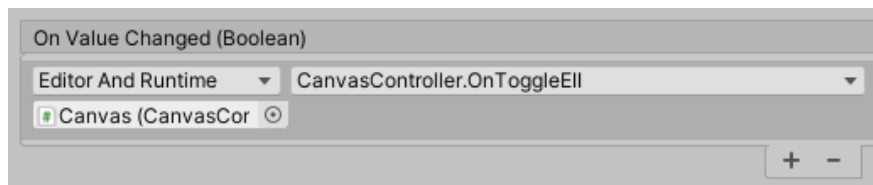


Figura 93. Configuración On Value Changed para el Toggle EII.

Para terminar la aplicación realizar a hacer dos cambios más. En primer lugar, haremos que el Canvas se active por primera vez justo en el momento en que se instancia el modelo sobre un plano. Para ello, vamos a realizar algunas modificaciones más en el script *CanvasController*.

En primer lugar, declaramos una nueva variable pública para el *Toggle EII*.

```
public GameObject toggleEII;
```

Después vamos a hacer que en la función **Start()** todos los elementos del Canvas estén desactivados.

```
void Start()
{
    botonTelaGris.SetActive(false);
    botonTelaAzul.SetActive(false);
    botonCuero.SetActive(false);
    panelAnim.SetActive(false);
    togglePlanes.SetActive(false);
    toggleEII.SetActive(false);
}
```


En una nueva función pública que llamaremos **ActivateCanvas()** se activarán de nuevo uno de los botones de cambio de material y el resto de los elementos.

```
public void ActivateCanvas()
{
    botonTelaGris.SetActive(true);
    panelAnim.SetActive(true);
    togglePlanes.SetActive(true);
    toggleEII.SetActive(true);
}
```

Ahora debemos realizar una llamada a esta función dentro del script *ObjectManipulator* del *ObjectGenerator*. Para ello debemos declarar en él una nueva variable para el script *CanvasController*.

```
public CanvasController CanvasController;
```

Después realizaremos una llamada a la función **ActivateCanvas()** justo después de que se lleve a cabo la instanciación del modelo.

```
...
var gameObject = Instantiate(ObjectPrefab, hit.Pose.position,
hit.Pose.rotation);
count++;
CanvasController.ActivateCanvas();
...
```

Para concluir vamos a añadir al Canvas un mensaje que aparecerá pasados 10 segundos desde el inicio de la aplicación si no se ha colocado el modelo en alguno de los planos. En este caso el mensaje ha sido importado como una imagen (Figura 94), pero podría hacerse directamente en Unity con un elemento UI de tipo *Text*.

**Toque un plano detectado
para colocar el modelo.**

Figura 94. Imagen del mensaje inicial.

También vamos a hacer que el mensaje aparezca con un movimiento descendente y se coloque en la parte superior de la pantalla. Para ello necesitamos crear en Unity una animación que haga este movimiento con un retraso de 10 segundos.

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Una vez importada la imagen vamos a añadirle una animación seleccionándola en la ventana Proyecto y dándole a **Create** en la ventana Animation (Windows > Animation para abrir esta ventana). Guardaremos esta animación en la carpeta de la Aplicación Demostrador con el nombre "ImageAnim" e inmediatamente se creará el asset de animación y un componente Animator en la imagen.

Ahora debemos colocar la imagen fuera del Canvas, en la parte superior. Para empezar a animar le damos al botón rojo (*keyframe recording mode*) de la ventana Animation, nos ubicamos en el segundo 1:30 en la línea de tiempo, colocamos la imagen en la posición correcta en el Canvas y le damos de nuevo al botón rojo para dejar de grabar. Después, seleccionamos los *keyframes* (fotogramas clave) que se han creado y los desplazamos en la línea temporal para que la animación comience en el segundo 10. Finalmente, nos ponemos de nuevo en el segundo 0 y añadimos un nuevo *keyframe* (botón *add keyframe*).

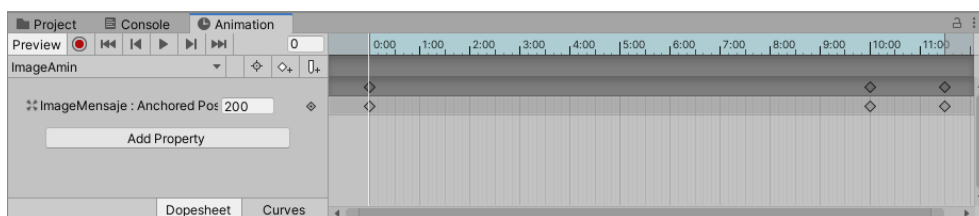


Figura 95. Animación para la imagen con el mensaje.



Figura 96. Posición inicial (izq.) y final (dcha.) del mensaje.

Para terminar con la animación debemos abrirla (desde la ventana Proyecto o Animator) y desmarcar la opción **Loop Pose** en el Inspector para evitar que se reproduzca una y otra vez. De esta forma, la animación se activará por defecto una sola vez al inicio.

Este elemento UI debemos programarlo para esté activo inicialmente y se desactive cuando se genera una instancia del modelo. Por lo tanto, vamos a realizar las últimas modificaciones en el script *CanvasController*.

Primero debemos declarar una variable pública para este elemento.

```
public GameObject imageMensaje;
```

Después, haremos que dentro de la función **Start()** este elementos sea el único que se encuentra activo.

```
void Start()
{
    ...

    imageMensaje.SetActive(true);
}
```

Finalmente, dentro de la función **ActivateCanvas()** desactivaremos el mensaje.

```
public void ActivateCanvas()
{
    ...

    imageMensaje.SetActive(false);
}
```

Una vez hecho esto podemos guardar definitivamente el script (en el Apéndice 2 se puede encontrar el código completo del script).

Debemos recordar asignarle el valor correspondiente a cada una de las variables públicas que hemos declarado, como se muestra en la Figura 97.



Figura 97. Variables públicas del CanvasController.

En esta sección hemos aprendido a configurar diferentes elementos UI para conseguir una buena interfaz y mejorar la experiencia de usuario. Ahora ya tenemos la aplicación lista para instalarla definitivamente en nuestro dispositivo.

5.5 Instalación de la Aplicación Demostrador

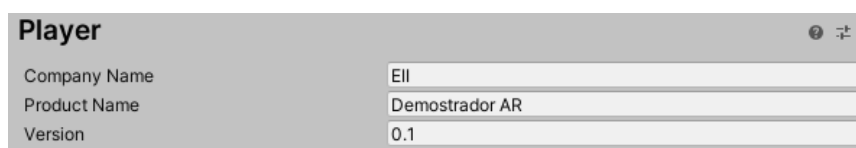
En este último apartado del tutorial vamos a ver de nuevo cómo podemos instalar nuestra aplicación, pero esta vez de una forma más detallada. Empezaremos estableciendo una identificación correcta para la aplicación. Después veremos cómo modificar el icono que la representa y la ventana de carga que aparece al ejecutarla. Finalmente la instalaremos en nuestro dispositivo.

Identificación de la aplicación

Cuando instalamos una aplicación en nuestro dispositivo, esta tiene una serie de características que la hacen única, es decir que la identifican. En el caso de existir varias aplicaciones con la misma información que las caracteriza, el dispositivo interpretará que son idénticas e instalará la más reciente, sobre escribiendo la versión más antigua.

En este tutorial hemos creado en total cuatro aplicaciones previas a la final: Aplicación Base, Modelos Propios, Modelos Animados y Cambiar Material. Las hemos ido instalando con diferentes nombres de producto para tener acceso a cada una de ellas por separado en nuestro teléfono.

Ahora vamos a definir el nombre que identificará nuestra aplicación final. Para ello iremos a las **Player Settings** y rellenaremos los tres primeros campos que aparecen. En **Company Name** podemos poner las siglas de la escuela "EII", en el **Product Name** introduciremos el nombre "Demostrador AR" y la **Versión** podemos mantenerla en 0.1.



Player	
Company Name	EII
Product Name	Demostrador AR
Version	0.1

Figura 98. Configuración para identificación de la aplicación.

De esta forma también se define automáticamente el **Package Name** como "com.EII.DemostradorAR", que podemos ver en la sección **Other Settings**.

Iconos

Otro de los aspectos característicos de nuestra aplicación que podemos modificar es el icono con el que aparece representada en nuestro dispositivo. En las **Player Settings** podemos encontrar una sección **Icon** donde podemos modificar los iconos, que pueden ser de tres tipos: *Adaptative*, *Round* y *Legacy*. Los dos últimos

tipos son para API 25 y versiones anteriores. Los iconos adaptativos son para API 26 y posteriores, y constan de dos partes: el fondo (*background*) y el primer plano (*foreground*).

Si desplegamos la opción **Legacy Icons**, veremos que hay una imagen asignada para las 6 resoluciones del icono. Esta imagen es la correspondiente al ejemplo *ObjectManipulation* del SDK y es la que se ha aplicado a los iconos de todas las aplicaciones que hemos instalado hasta ahora.

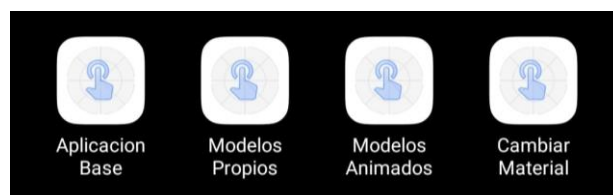


Figura 99. Iconos de las aplicaciones anteriores.

Para modificarlo podemos importar una imagen del tamaño adecuado y arrastrarla hasta todos los huecos para sustituir a la imagen que existía por defecto. En este caso utilizaremos el logotipo de la Escuela de Ingenierías Industriales como imagen para el icono.



Figura 100. Legacy Icons para la aplicación Demostrador.

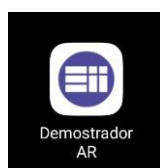


Figura 101. Icono de la Aplicación Demostrador.

Pantalla de inicio

Cuando iniciamos una aplicación que ha sido creada en Unity, normalmente se carga una pantalla de inicio que contiene el logotipo del programa. Esta pantalla puede ser modificada en la sección **Splash Image**, también en las **Player Settings**. Sin embargo, el nivel de personalización esta pantalla de presentación dependerá de nuestra suscripción a Unity. En el caso de la suscripción a Unity Personal no podremos desactivar por completo esta pantalla ni quitar el logotipo de Unity.

Podemos modificar varios aspectos como el *Splash Style* para que el logo se vea claro sobre oscuro u oscuro sobre claro, el tipo de animación, añadir más logos, la posición y duración de los logos, el fondo, etc.

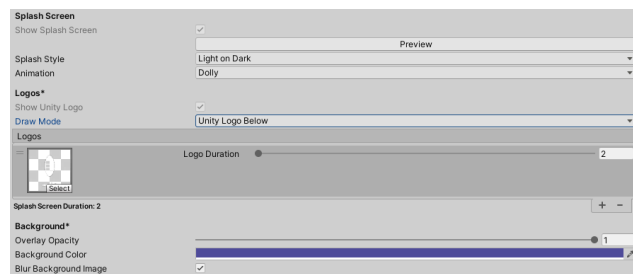


Figura 102. Configuración de la Splash Image.

Para esta aplicación tan solo vamos a añadir un nuevo logo, que será también el de la escuela, pero esta vez en blanco, con el nombre de la aplicación y con márgenes a ambos lados para que se ajuste bien al tamaño de la pantalla. También vamos a cambiar el color de fondo por un azul muy parecido al del logo de EII. El resultado de esta pantalla de inicio se muestra en la Figura 103.

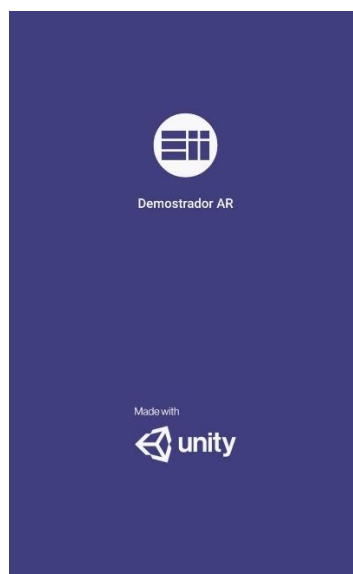


Figura 103. Vista de la pantalla de carga de la aplicación.

APLICACIÓN DEMOSTRADOR

Para terminar definitivamente el tutorial vamos a instalar la aplicación en nuestro dispositivo. Lo haremos de forma similar a otras ocasiones, sin embargo, cuando le demos al botón **Build and Run** y aparezca la ventana para guardar el archivo .apk, lo guardaremos con el nombre "Demostrador AR".

Con esto ya tenemos totalmente lista nuestra aplicación para visualizar nuestros propios modelos, con la posibilidad de manipularlos, animarlos, cambiar sus materiales y, además, con una interfaz atractiva e intuitiva.

En la Figura 104 se pueden ver algunas capturas de la aplicación final en ejecución.

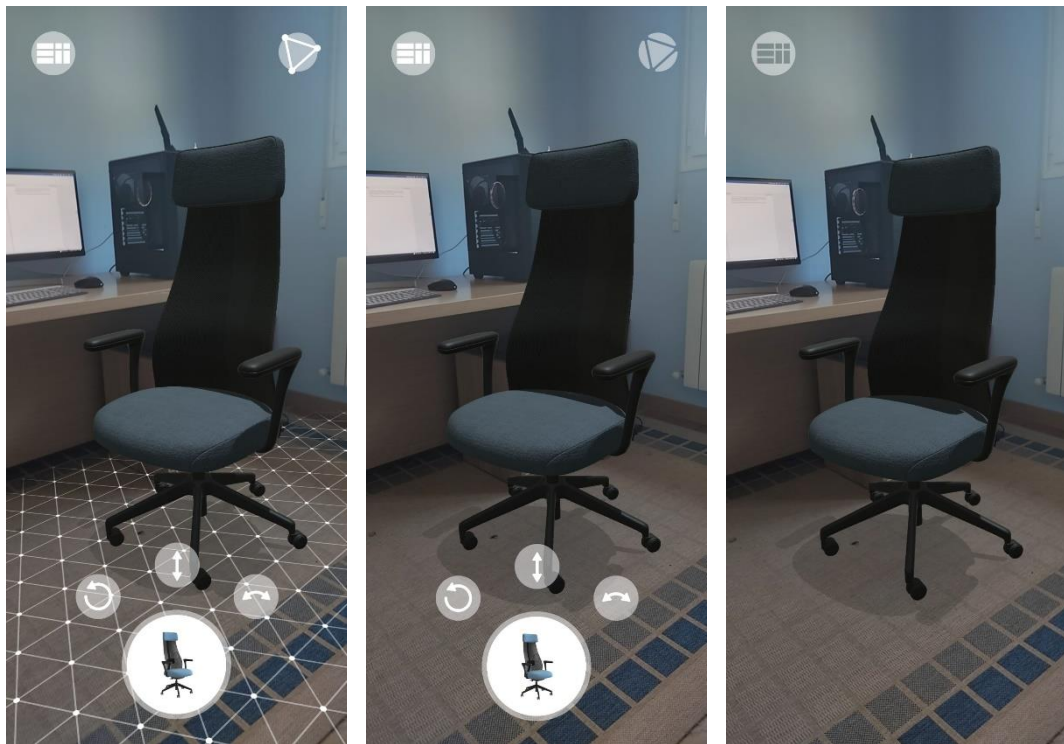


Figura 104. Capturas de la Aplicación Demostrador en funcionamiento.

5.6 Usabilidad

La utilidad de un sistema, en tanto que medio para conseguir un objetivo, tiene una componente de funcionalidad (utilidad funcional) y otra basada en el modo en que los usuarios pueden usar dicha funcionalidad. Es esta componente la que nos interesa ahora.

Podemos definir la usabilidad como la medida en la cual un producto puede ser usado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado [48].

La efectividad se refiere la precisión y la plenitud con las que los usuarios alcanzan los objetivos especificados.

Por eficiencia se entenderán los recursos empleados en relación con la precisión y plenitud con que los usuarios alcanzan los objetivos especificados.

Con satisfacción nos referimos la ausencia de incomodidad y la actitud positiva en el uso del producto. Se trata, pues, de un factor subjetivo.

La usabilidad es por lo tanto un aspecto esencial en cualquier aplicación, desde el comienzo del proceso de desarrollo hasta las últimas acciones antes de que el producto vea la luz. Es por ello por lo que el presente trabajo ha perseguido en todo momento el objetivo de conseguir una aplicación sencilla e intuitiva, que cumpliera con solvencia los objetivos propuestos.

Desde la manera de instanciar y manipular los modelos, hasta la disposición final de los elementos de la interfaz de usuario, siempre se ha buscado simplificar al máximo el funcionamiento de la aplicación. Se han eliminado todos los elementos y funciones que resultaban superfluos y que, por tanto, dificultan la usabilidad. De este modo se ha tratado de favorecer al máximo la experiencia del usuario.

Por otra parte, la aplicación aquí desarrollada, por su naturaleza, puede llegar a ofrecer una gran versatilidad en su uso. Desde un principio fue orientada hacia el mundo del diseño industrial. Fue concebida como una aplicación para mostrar productos en realidad aumentada. Sin embargo, los usos de esta aplicación pueden ser tantos como modelos 3D se quieran utilizar.

Teniendo en cuenta las características de esta aplicación se puede comprender que el estudio de la usabilidad puede ser de un gran de interés. Es por ello que se ha realizado una prueba de usabilidad que ayudará a valorar en qué medida la aplicación cumple con sus objetivos de una forma efectiva, eficiente y satisfactoria.

Prueba de usabilidad

En este apartado veremos cómo se ha llevado a cabo la valoración de la usabilidad de la aplicación, así como los resultados obtenidos en la prueba.

El método que vamos a utilizar es conocido como Escala de Usabilidad de un Sistema o SUS (System Usability Scale). Se trata de una métrica estandarizada para medir la usabilidad de un sitio web u otro sistema interactivo. Es una de las herramientas favoritas de los investigadores de UX (experiencia de usuario) por su simplicidad y precisión [49]. Diferentes pruebas y test han demostrado que los resultados obtenidos a partir de este método suelen ser muy confiables y acertados.

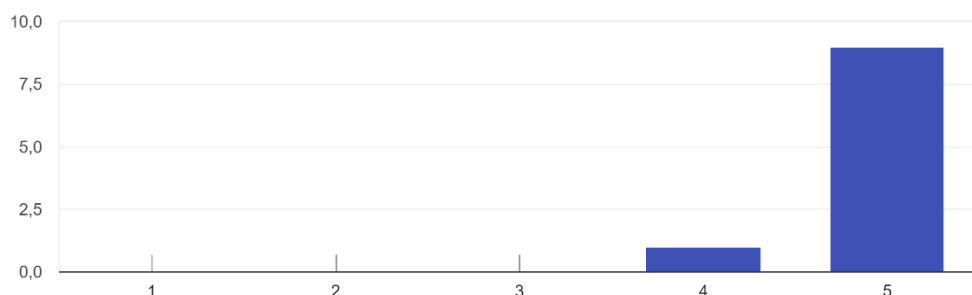
SUS utiliza un breve cuestionario de 10 preguntas que debe cumplimentarse una vez concluida la prueba de usabilidad. Los usuarios responden a cada pregunta en una escala de 5 puntos, que va desde “Totalmente en desacuerdo” hasta “Totalmente de acuerdo”. De estas respuestas puede obtenerse una puntuación de usabilidad para la aplicación en cuestión.

Respecto a las condiciones de las pruebas de usabilidad cabe decir que se han realizado de manera remota, poniendo a disposición de los usuarios el archivo .apk de la aplicación. De esta forma se ha podido comprobar también que se comporta de forma correcta en otros dispositivos Android. Además, se ha aportado una explicación muy breve sobre el funcionamiento de la aplicación, sin entrar en detalles para mantener la objetividad.

Debido a la naturaleza de la aplicación y las diferentes posibilidades que puede ofrecer para otros usos, se ha elegido un total de 10 personas para realizar la prueba. Todos ellos estudiantes de ingeniería o arquitectura, relacionados de alguna manera con el mundo del diseño 3D, por lo que se deduce un potencial interés en el uso de una aplicación como esta.

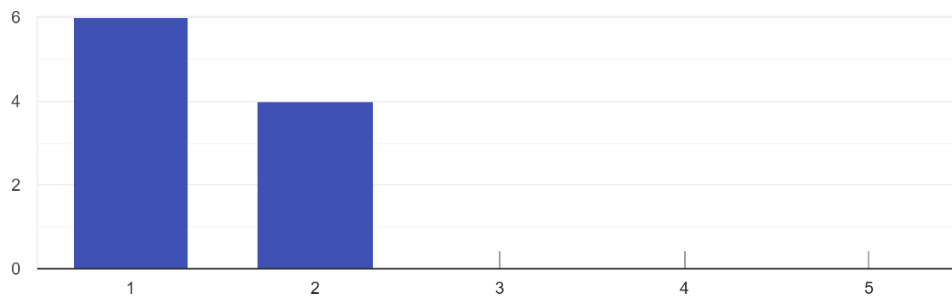
A continuación, se muestran las 10 preguntas del cuestionario (que son las originales que estipula el método con alguna ligera variación), junto con las respuestas obtenidas:

1. Creo que usaría esta aplicación para mostrar mis modelos 3D.

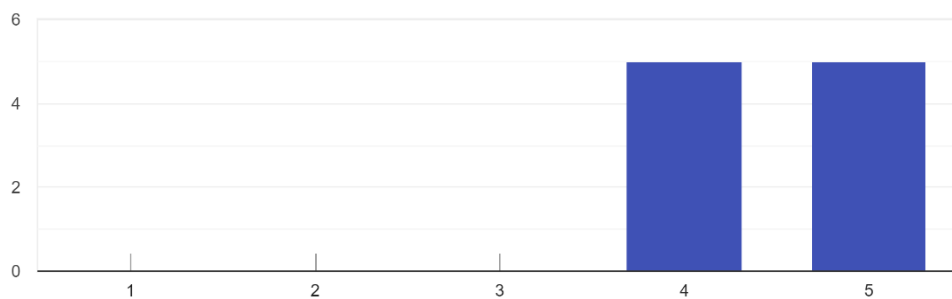


TRABAJO DE FIN DE GRADO
APLICACIÓN DE REALIDAD AUMENTADA

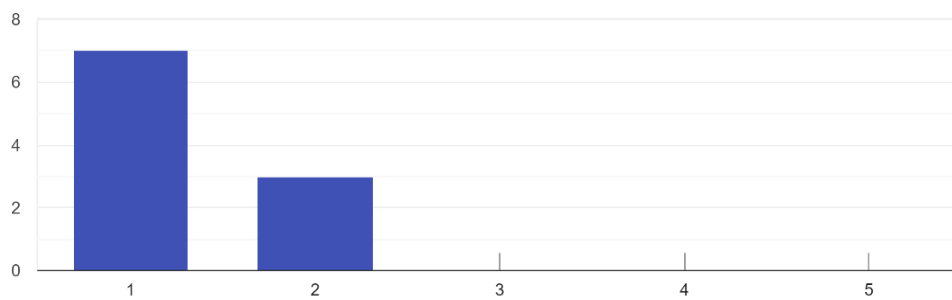
2. Encuentro esta aplicación innecesariamente compleja.



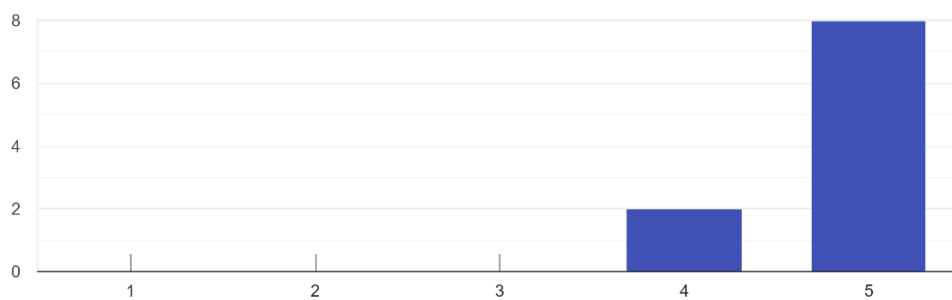
3. Creo que la aplicación fue fácil de usar.



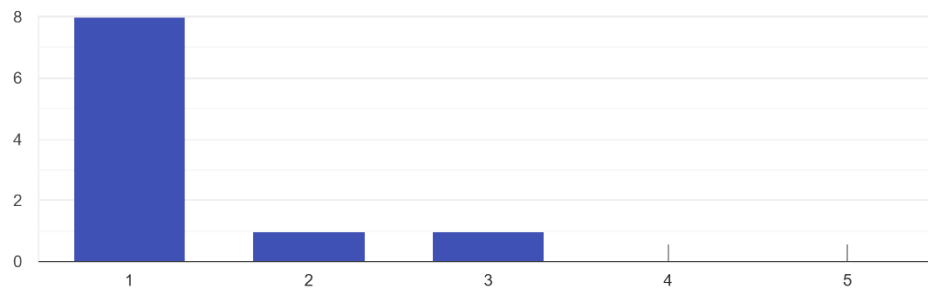
4. Creo que necesitaría ayuda de una persona con conocimientos técnicos para usar esta aplicación.



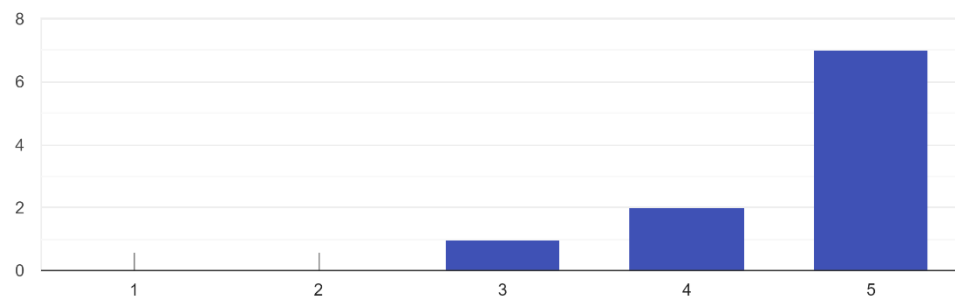
5. Las funciones de esta aplicación están bien integradas.



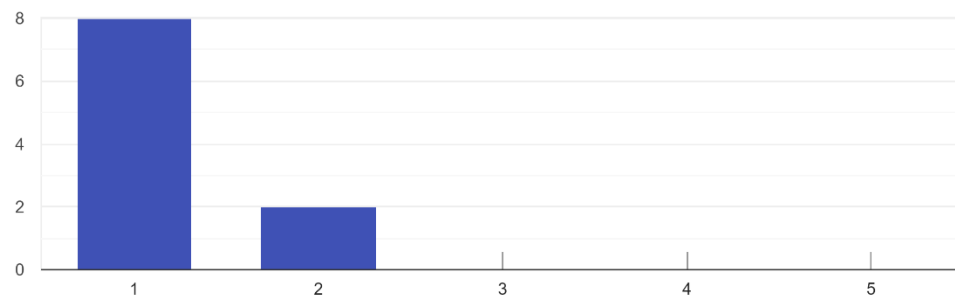
6. Creo que la aplicación es muy inconsistente.



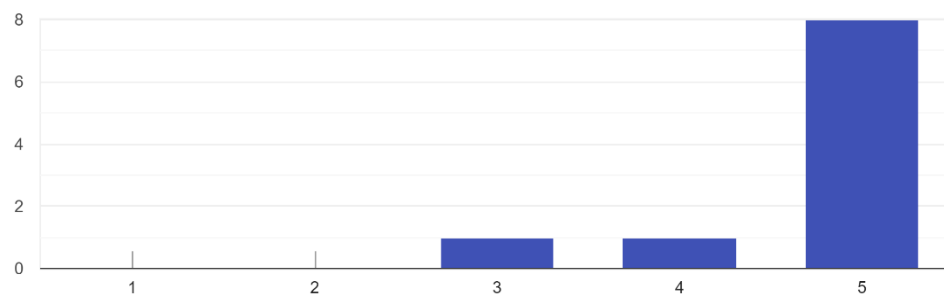
7. Imagino que la mayoría de gente aprendería a usar esta aplicación de forma muy rápida.



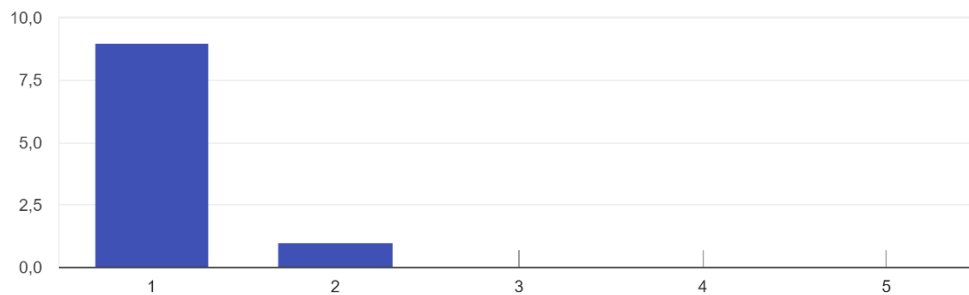
8. Encuentro que la aplicación es muy difícil de usar.



9. Me siento confiado al usar esta aplicación.



10. Necesité aprender muchas cosas antes de ser capaz de usar esta aplicación.



Una vez que contamos con todas las respuestas del test podemos pasar a realizar la medición. Para obtener los resultados, debemos seguir la siguiente pauta para cada una de las respuestas al test: las preguntas impares (1,3,5,7 y 9) tomarán el valor asignado por el entrevistado, y se le restará 1. Para las preguntas pares (2,4,6,8,10), el valor será de 5 menos el valor asignado por los usuarios. Una vez obtenido el número final, se lo multiplica por 2,5.

Por ejemplo, los resultados obtenidos en el segundo entrevistado son:

5, 1, 4, 1, 4, 1, 3, 1, 5, 1.

Asignando los nuevos valores según el algoritmo, obtenemos su puntaje SUS:

$$((5-1)+(5-1)+(4-1)+(5-1)+(4-1)+(5-1)+(3-1)+(5-1)+(5-1)+(5-1)) * 2,5 = 36 * 2,5 = 90$$

Debemos hacer lo mismo con los resultados de todos los usuarios y después calcular la media de sus puntajes SUS. El máximo teórico de la escala es 100 y el resultado medio de todos los entrevistados que se ha obtenido para nuestra aplicación es de 93,25. Según el gráfico de la Figura 105 esta marca supera "lo mejor imaginable" en lo que se refiere a la usabilidad de nuestro sistema.

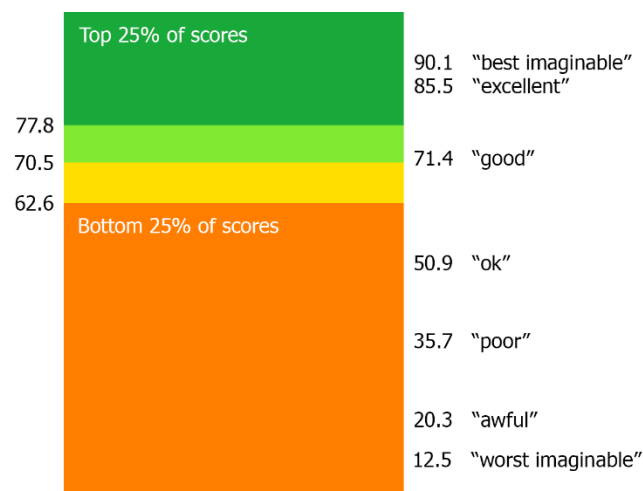


Figura 105. Gráfico de valoración SUS [49].

6. CONCLUSIONES

En el presente trabajo se ha abordado el tema de la realidad aumentada desde un enfoque amplio, pero persiguiendo unos objetivos concretos. Todos ellos se podrían sintetizar en uno solo, es decir, en el propósito de acercar esta tecnología, de hacerla más accesible, para la comunidad universitaria principalmente, pero también para todo tipo de profesionales que pudieran sacar partido a esta tecnología.

La persecución de los objetivos planteados al inicio de este documento ha llevado al desarrollo de una aplicación de realidad aumentada para visualizar productos. Esto puede verse como una contribución al mundo del diseño industrial, pero, por su versatilidad, esta aplicación puede ofrecer grandes posibilidades también en otros sectores, con otros usos muy diversos. Sin embargo, para llegar a la creación la aplicación Demostrador AR antes hemos dado otros pasos igualmente importantes.

En la primera parte realizamos un amplio estudio de la realidad aumentada que sirve como punto de partida y para sentar las bases de este trabajo. En él tratamos de dar una definición de la realidad aumentada, así como explicar su funcionamiento desde una punto de vista general. Después hicimos un repaso de toda su historia, empezando por sus orígenes y llegando a su estado del arte, con ejemplos de aplicaciones en los principales campos en los que esta tecnología tiene utilidad. Finalmente, se dieron algunas ideas de lo que podría llegar a ser su futuro, demostrando la gran potencialidad de esta tecnología.

Después, seguimos con una introducción a las tecnologías que después emplearíamos para crear la aplicación. Explicamos qué es Unity, cómo es su interfaz y cuáles son las principales alternativas para crear experiencias de RA con este software. También detallamos el funcionamiento de ARCore, el motor de realidad aumentada que empleamos en nuestra aplicación.

Una vez llegamos a la tercera parte, pasamos al desarrollo de nuestra aplicación. En forma de tutorial se detallan todos los pasos a seguir para construir la aplicación Demostrador AR. En primer lugar, se muestran los requisitos para el hardware y se explica cómo descargar e instalar el software. Después creamos un proyecto en Unity, con las configuraciones iniciales que son necesarias para implementar la realidad aumentada de ARCore. Luego añadimos los elementos necesarios para crear una aplicación base que permitiese la instanciación de modelos 3d sobre planos. Partiendo de esta continuamos añadiendo elementos para conseguir nuestra aplicación definitiva, que permite colocar modelos propios y activar animaciones y cambios de material con diferentes botones. Para finalizar realizamos una prueba para valorar su usabilidad.

CONCLUSIONES

Como conclusión, podría decirse que este proyecto es una toma de contacto con la realidad aumentada, que ofrece los conocimientos para iniciarse en el estudio de esta tecnología en general, y en particular, también aporta las herramientas para la creación de aplicaciones en Unity basadas en experiencias de realidad aumentada.

Que este trabajo pueda servir de base para otros estudiantes ha sido también otro de los objetivos. La realidad aumentada tiene una gran proyección de futuro y, aunque todavía queda mucho por investigar, son de esperar grandes avances en los próximos años, que traerán consigo nuevas oportunidades para satisfacer necesidades. Este continuo cambio debe motivarnos seguir progresando en el desarrollo en esta tecnología, porque las posibilidades son inmensas y aún falta mucho por crear.

7. BIBLIOGRAFÍA

- [1] L. H. Lara and J. L. V. Benitez, "La realidad aumentada: una tecnología en espera de usuarios," *Rev. Digit. Univ.*, 2007.
- [2] J. I. Icaza, J. Luis De La Cruz, M. Muñoz, and I. Rudomín, "Realidad mixta," in *LAS MEGATENDENCIAS TECNOLÓGICAS ACTUALES Y SU IMPACTO EN LA IDENTIFICACIÓN DE OPORTUNIDADES ESTRATÉGICAS DE NEGOCIOS*, pp. 172–177.
- [3] R. Skarbez, M. Smith, and M. C. Whitton, "Revisiting Milgram and Kishino's Reality-Virtuality Continuum," *Front. Virtual Real.*, vol. 2, pp. 1–8, Mar. 2021, doi: 10.3389/frvir.2021.647997.
- [4] R. T. Azuma, "A survey of augmented reality," in *Presence: Teleoperators and Virtual Environments*, vol. 6, no. 4, 1997, pp. 355–385.
- [5] R. T. Azuma, Y. Baillet, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent Advances in Augmented Reality," in *IEEE Computer Graphics and Applications*, 2001, pp. 34–47.
- [6] "La Realidad Aumentada," *Innovae*. <https://www.innovae.eu/la-realidad-aumentada/>.
- [7] H. A. Cárdenas Ruiz, F. Y. Mesa Jiménez, and M. J. Suarez Barón, "Realidad aumentada (RA): aplicaciones y desafíos para su uso en el aula de clase," *Rev. Educ. y Ciudad*, no. 35, pp. 137–148, 2018.
- [8] "Augmented Reality – The Past, The Present and The Future," *Interaction Design Foundation (IXDF)*. <https://www.interaction-design.org/literature/article/augmented-reality-the-past-the-present-and-the-future>.
- [9] M. Isberto, "The History of Augmented Reality," *Colocation America*, 2018. <https://www.colocationamerica.com/blog/history-of-augmented-reality>.
- [10] "VideoPlace," *IDIS*. <https://proyectoidis.org/videoplacement/>.
- [11] E. Chocarro De Luis, B. Lainez, J. H. Busto Sancirian, and J. López Benito, "Aportaciones de la Realidad Aumentada en la inclusión en el aula de estudiantes con Trastorno del Espectro Autista," *Edmetic*, 2018, doi: 10.21071/edmetic.v7i2.10134.
- [12] X. Basogain, M. Olabe, K. Espinosa, C. Rouèche, and J. C. Olabe, "Realidad Aumentada en la Educación : una tecnología emergente." *Escuela Superior de Ingeniería de Bilbao, EHU, Bilbao*, pp. 2–3.

- [13] S. Singhal, S. Bagga, P. Goyal, and V. Saxena, "Augmented Chemistry: Interactive Education System," *Int. J. Comput. Appl.*, vol. 49, no. 15, pp. 2–6, 2012, doi: 10.5120/7700-1041.
- [14] "Construct3D - An Augmented Reality System for Mathematics and Geometry Education," *Interactive Media Systems, TU Wien*. <https://www.ims.tuwien.ac.at/projects/construct3d>.
- [15] "ARART." <http://www.arart.info/>.
- [16] "Realidad aumentada en arquitectura y construcción," *IAT*. <https://iat.es/tecnologias/realidad-aumentada/arquitectura/>.
- [17] "6 aplicaciones creativas de Realidad Aumentada en publicidad," *Xperimenta Cultura*. <https://xperimentacultura.com/realidad-aumentada-publicidad-aplicaciones-creativas-2/>.
- [18] "Airbus vuela más alto con ayuda de la tecnología de realidad mixta de Microsoft," *Centro de noticias Microsoft*. <https://news.microsoft.com/es-es/2019/06/18/airbus-vuela-mas-alto-con-ayuda-de-la-tecnologia-de-realidad-mixta-de-microsoft/>.
- [19] "Airbus y el Ejército del Aire español desarrollarán la inspección de aviones militares mediante drones y realidad aumentada," *Defence - Airbus*. <https://www.airbus.com/newsroom/press-releases/es/2019/05/airbus-and-spanish-air-force-to-develop-drone-and-augmented-reality-inspections-for-military-aircraft.html>.
- [20] "Cómo utiliza Thyssenkrupp la realidad aumentada en su área de mantenimiento," *Industria y Utilities | CIO*. <https://www.ciospain.es/industria-y-utilities/como-utiliza-thyssenkrupp-la-realidad-aumentada-en-su-area-de-mantenimiento>.
- [21] B. Díaz, "Así funciona la realidad aumentada con la que Volvo desarrollará sus vehículos," *Car and Driver*, 2019. <https://www.caranddriver.com/es/coches/planeta-motor/a60364/volvo-realidad-aumentada-desarrollo-vehiculos-seguridad/>.
- [22] "Realidad Aumentada en Medicina," *Internovam Blog*, 2018. <http://internovam.com/blog/realidad-aumentada-en-medicina/>.
- [23] X. Migelez, "Realidad aumentada, la herramienta que está cambiando la TV tal y como la conoces," *El Confidencial*. https://www.elconfidencial.com/television/programas-tv/2018-11-15/realidad-aumentada-la-herramienta-que-esta-revolucionando-la-television_1644974/.
- [24] B. Rodrigo L., "Realidad aumentada, un futuro lleno de posibilidades." <https://rodrigolbarnes.com/2016/12/13/realidad-aumentada/>.

- [25] C. Arce, "Realidad aumentada." Universidad Católica "Nuestra Señora de la Asunción," pp. 26–30, 2007.
- [26] "Unity Platform," *Unity*. <https://unity.com/products/unity-platform>.
- [27] "¿Qué es Unity y para qué puedo utilizarlo?," *IFP*, Aug. 21, 2020. <https://www.ifp.es/blog/que-es-unity-y-para-que-puedo-utilizarlo>.
- [28] "Unity (motor de videojuego)," *Wikipedia, la enciclopedia libre*. [https://es.wikipedia.org/wiki/Unity_\(motor_de_videojuego\)](https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)).
- [29] F. Huanay Martínez, "Tutorial Básico - UNITY." Universidad Nacional de Ingeniería, pp. 1–8.
- [30] "Cómo crear realidad aumentada en Unity," *IAT*. <https://iat.es/tecnologias/realidad-aumentada/unity/>.
- [31] "Marco de trabajo AR Foundation de Unity," *Unity*. <https://unity.com/es/unity/features/arfoundation>.
- [32] "Unity MARS," *Unity*. <https://unity.com/es/products/unity-mars>.
- [33] "Unity as a Library.," *Unity*. <https://unity.com/features/unity-as-a-library>.
- [34] M. Fuad and M. Dalby, "XR Interaction Toolkit Preview Package is here," *Unity Blog*, Dec. 17, 2019. <https://blog.unity.com/technology/xr-interaction-toolkit-preview-package-is-here>.
- [35] "Getting Started with Vuforia Engine in Unity," *VuforiaLibrary*. <https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html>.
- [36] "Unity overview of features," *ARCore | Google Developers*. <https://developers.google.com/ar/develop/unity>.
- [37] "ARCore overview," *Google Developers*. <https://developers.google.com/ar/discover>.
- [38] I. Ramírez, "ARCore: qué es, para qué sirve y móviles compatibles," *xataka android*, Apr. 25, 2021. <https://www.xatakandroid.com/realidad-aumentada/arcore-que-sirve-moviles-compatibles>.
- [39] "Fundamental concepts," *ARCore | Google Developers*. <https://developers.google.com/ar/discover/concepts>.
- [40] "ARCore supported devices," *Google Developers*. <https://developers.google.com/ar/devices>.
- [41] "Configure on-device developer options," *Android Developers*. <https://developer.android.com/studio/debug/dev-options>.

- [42] "Cloud Anchors overview for Unity," *ARCore | Google Developers*. <https://developers.google.com/ar/develop/unity/cloud-anchors/overview-unity>.
- [43] "Using Instant Preview," *ARCore | Google Developers*. <https://developers.google.com/ar/develop/unity/instant-preview>.
- [44] "Configure an ARCore session in Unity," *Google Developers*. <https://developers.google.com/ar/develop/unity/session-config>.
- [45] "Configuring the camera," *ARCore | Google Developers*. <https://developers.google.com/ar/develop/unity/camera-configs>.
- [46] "Using ARCore to light models in a scene," *Google Developers*. <https://developers.google.com/ar/develop/unity/light-estimation>.
- [47] "Event System," *Unity UI | 1.0.0*. <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/EventSystem.html>.
- [48] "¿Qué es la Usabilidad?," *Sidar*. <http://www.sidar.org/recur/desdi/traduc/es/visitable/quees/usab.htm>.
- [49] "SUS - The System Usability Scale," *TryMyUI*. <https://www.trymyui.com/sus-system-usability-scale>.
- "Unity User Manual (2019.4 LTS)," *Unity - Manual*. <https://docs.unity3d.com/2019.4/Documentation/Manual/index.html>.
- M. Lidon, "UNITY 3D," Primera Ed., S. A. Alfaomega Grupo Editor, Ed. MARCOMBO, S.A., Barcelona, España.
- J. Arias, "Programación de una aplicación Android para la visualización de modelos 3D en realidad aumentada." Universidad de Valladolid, 2020.

APÉNDICE 1: Código de la Aplicación Base

Código correspondiente a *ObjectManipulator.cs*:

```
namespace GoogleARCore.Examples.ObjectManipulation
{
    using GoogleARCore;
    using UnityEngine;

    public class ObjectManipulator : Manipulator
    {
        public Camera FirstPersonCamera;
        public GameObject ObjectPrefab;
        public GameObject ManipulatorPrefab;

        public int Instances;
        private int count = 0;

        protected override bool CanStartManipulationForGesture(TapGesture gesture)
        {
            if (gesture.TargetObject == null)
            {
                return true;
            }

            return false;
        }

        protected override void OnEndManipulation(TapGesture gesture)
        {
            if (gesture.WasCancelled)
            {
                return;
            }

            if (gesture.TargetObject != null)
            {
                return;
            }

            TrackableHit hit;
            TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon;

            if (Frame.Raycast(
                gesture.StartPosition.x, gesture.StartPosition.y, raycastFilter, out hit))
            {
                if (((hit.Trackable is DetectedPlane) &&
                    Vector3.Dot(FirstPersonCamera.transform.position - hit.Pose.position,
                        hit.Pose.rotation * Vector3.up) < 0) || count >= Instances)
                {
                    Debug.Log("Hit at back of the current DetectedPlane");
                }
            }
        }
    }
}
```

APÉNDICE 1: Código de la Aplicación Base

```
        else
        {
            var gameObject = Instantiate(ObjectPrefab, hit.Pose.position,
hit.Pose.rotation);
            count++;

            var manipulator =
Instantiate(ManipulatorPrefab, hit.Pose.position,
hit.Pose.rotation);

            gameObject.transform.parent = manipulator.transform;

            var anchor = hit.Trackable.CreateAnchor(hit.Pose);
            manipulator.transform.parent = anchor.transform;
            manipulator.GetComponent<Manipulator>().Select();
        }
    }
}
```

APÉNDICE 2: Código de la Aplicación Demostrador

Código correspondiente a *InstanceAnimator.cs*:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class InstanceAnimator : MonoBehaviour
{
    private GameObject instance;
    private Animator anim;

    void FindObject()
    {
        instance = GameObject.Find("SillaEscritorio");
        anim = instance.GetComponent<Animator>();
    }

    public void PlayRotar()
    {
        FindObject();
        anim.Play("Rotar", -1, 0f);
    }

    public void PlayElevar()
    {
        FindObject();
        anim.Play("Elevar", -1, 0f);
    }

    public void PlayInclinar()
    {
        FindObject();
        anim.Play("Inclinar", -1, 0f);
    }
}
```

APÉNDICE 2: Código de la Aplicación Demostrador

Código correspondiente a *ChangeMaterial.cs*:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ChangeMaterial : MonoBehaviour
{
    private GameObject cabecero;
    private GameObject lumbares;
    private GameObject asiento;

    public Material[] materials = new Material[3];

    void FindParts()
    {
        cabecero = GameObject.Find("Cabecero");
        lumbares = GameObject.Find("Lumbares");
        asiento = GameObject.Find("Asiento");
    }

    public void ChangeTelaGris()
    {
        FindParts();
        cabecero.GetComponent<MeshRenderer>().material = materials[0];
        lumbares.GetComponent<MeshRenderer>().material = materials[0];
        asiento.GetComponent<MeshRenderer>().material = materials[0];
    }

    public void ChangeTelaAzul()
    {
        FindParts();
        cabecero.GetComponent<MeshRenderer>().material = materials[1];
        lumbares.GetComponent<MeshRenderer>().material = materials[1];
        asiento.GetComponent<MeshRenderer>().material = materials[1];
    }

    public void ChangeCuero()
    {
        FindParts();
        cabecero.GetComponent<MeshRenderer>().material = materials[2];
        lumbares.GetComponent<MeshRenderer>().material = materials[2];
        asiento.GetComponent<MeshRenderer>().material = materials[2];
    }
}
```

TRABAJO DE FIN DE GRADO
APLICACIÓN DE REALIDAD AUMENTADA

Código correspondiente a *CanvasController.cs*:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CanvasController : MonoBehaviour
{
    public GameObject botonTelaGris;
    public GameObject botonTelaAzul;
    public GameObject botonCuero;
    public GameObject panelAnim;
    public GameObject togglePlanes;
    public GameObject toggleEII;
    public GameObject imageMensaje;

    public ChangeMaterial ChangeMaterial;

    //Estado del Canvas inicialmente
    void Start()
    {
        botonTelaGris.SetActive(false);
        botonTelaAzul.SetActive(false);
        botonCuero.SetActive(false);
        panelAnim.SetActive(false);
        togglePlanes.SetActive(false);
        toggleEII.SetActive(false);

        imageMensaje.SetActive(true);
    }

    //Elementos UI que se activan al instanciarse un objeto
    public void ActivateCanvas()
    {
        botonTelaGris.SetActive(true);
        panelAnim.SetActive(true);
        togglePlanes.SetActive(true);
        toggleEII.SetActive(true);

        imageMensaje.SetActive(false);
    }

    //Activacion-Desactivacion de elementos con el Toggle EII
    public void OnToggleEII(bool toggle)
    {
        botonTelaGris.SetActive(toggle);
        botonTelaAzul.SetActive(toggle);
        botonCuero.SetActive(toggle);
        panelAnim.SetActive(toggle);
        togglePlanes.SetActive(toggle);
    }
}
```


APÉNDICE 2: Código de la Aplicación Demostrador

```
//Funciones para los botones de cambio de material
public void OnClickTelaGris()
{
    //Se cambia el material a Tela Azul
    ChangeMaterial.ChangeTelaAzul();
    //Se activa el botón Tela Azul y se desactivan los otros dos
    botonTelaGris.SetActive(false);
    botonTelaAzul.SetActive(true);
    botonCuero.SetActive(false);
}

public void OnClickTelaAzul()
{
    //Se cambia el material a Cuero
    ChangeMaterial.ChangeCuero();
    //Se activa el botón Cuero y se desactivan los otros dos
    botonTelaGris.SetActive(false);
    botonTelaAzul.SetActive(false);
    botonCuero.SetActive(true);
}

public void OnClickCuero()
{
    //Se cambia el material a Tela Gris
    ChangeMaterial.ChangeTelaGris();
    //Se activa el botón Tela Azul y se desactivan los otros dos
    botonTelaGris.SetActive(true);
    botonTelaAzul.SetActive(false);
    botonCuero.SetActive(false);
}
}
```

Código correspondiente a *DetectedPlaneGenerator.cs*:

```
namespace GoogleARCore.Examples.Common
{
    using System.Collections.Generic;
    using GoogleARCore;
    using UnityEngine;

    public class DetectedPlaneGenerator : MonoBehaviour
    {
        public GameObject DetectedPlanePrefab;

        private List<DetectedPlane> _newPlanes = new List<DetectedPlane>();

        public Material transparent;
        public Material grid;

        bool search = true;

        public void OnTogglePlanes(bool toggle)
        {
            search = toggle;

            foreach (GameObject plane in
                GameObject.FindGameObjectsWithTag("plane"))
            {
                if (toggle)
                {
                    plane.GetComponent<MeshRenderer>().material = grid;
                }
                else
                {
                    plane.GetComponent<MeshRenderer>().material = transparent;
                }
            }
        }

        public void Update()
        {
            if (Session.Status != SessionStatus.Tracking)
            {
                return;
            }
        }
    }
}
```

APÉNDICE 2: Código de la Aplicación Demostrador

```
        if (search)
        {
            Session.GetTrackables<DetectedPlane>(_newPlanes,
TrackableQueryFilter.New);
            for (int i = 0; i < _newPlanes.Count; i++)
            {
                GameObject planeObject =
                    Instantiate(DetectedPlanePrefab, Vector3.zero,
Quaternion.identity, transform);

planeObject.GetComponent<DetectedPlaneVisualizer>().Initialize(_newPlanes[i]);
            }
        }
    }
}
```

TRABAJO DE FIN DE GRADO APLICACIÓN DE REALIDAD AUMENTADA

Código correspondiente a *ObjectManipulator.cs*:

```
namespace GoogleARCore.Examples.ObjectManipulation
{
    using GoogleARCore;
    using UnityEngine;

    public class ObjectManipulator : Manipulator
    {
        public Camera FirstPersonCamera;
        public GameObject ObjectPrefab;
        public GameObject ManipulatorPrefab;

        public int Instances;
        private int count = 0;

        public CanvasController CanvasController;

        protected override bool CanStartManipulationForGesture(TapGesture gesture)
        {
            if (gesture.TargetObject == null)
            {
                return true;
            }

            return false;
        }

        protected override void OnEndManipulation(TapGesture gesture)
        {
            if (gesture.WasCancelled)
            {
                return;
            }

            if (gesture.TargetObject != null)
            {
                return;
            }

            TrackableHit hit;
            TrackableHitFlags raycastFilter = TrackableHitFlags.PlaneWithinPolygon;

            if (Frame.Raycast(
                gesture.StartPosition.x, gesture.StartPosition.y, raycastFilter, out hit))
            {
                if (((hit.Trackable is DetectedPlane) &&
                    Vector3.Dot(FirstPersonCamera.transform.position - hit.Pose.position,
                        hit.Pose.rotation * Vector3.up) < 0) || count >= Instances)
                {
                    Debug.Log("Hit at back of the current DetectedPlane");
                }
            }
        }
    }
}
```

APÉNDICE 2: Código de la Aplicación Demostrador

```
        else
        {
            var gameObject = Instantiate(ObjectPrefab, hit.Pose.position,
hit.Pose.rotation);
            count++;
            CanvasController.ActivateCanvas();

            var manipulator =
            Instantiate(ManipulatorPrefab, hit.Pose.position,
hit.Pose.rotation);

            gameObject.transform.parent = manipulator.transform;

            var anchor = hit.Trackable.CreateAnchor(hit.Pose);

            manipulator.transform.parent = anchor.transform;

            manipulator.GetComponent<Manipulator>().Select();
        }
    }
}
```