Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Universidad de Valladolid

Escuela de Ingenierías Industriales

Grado en Ingeniería de Diseño Industrial y Desarrollo del Producto

Programación de una aplicación Android para la visualización de modelos 3D en realidad aumentada

Autor: Javier Álvaro Arias Canseco

Tutor: David Escudero Mancebo

Departamento de Informática

Área de Ciencia de la Computación

e Inteligencia Artificial

Valladolid, Enero de 2020

En este proyecto se presenta una aplicación que permite a cualquier usuario que haya modelizado objetos en 3D, tener la oportunidad de ver su trabajo en realidad aumentada con todas las ventajas que esto conlleva. Comenzando con la introducción y aprendizaje sobre realidad aumentada, herramientas y aplicaciones, para posteriormente, paso a paso, mostrar cómo incorporar un modelo 3D propio dentro de una aplicación de realidad aumentada, que permite integrar el modelo en el mundo real, empleando el teléfono móvil. Con el programa Android Studio, la plataforma de google ARCore y código programado en Java, podemos lograr que tanto profesionales como novatos en este ámbito logren conocer y usar la realidad aumentada.

This project presents an application that allows any user who has modeled 3D objects, have the opportunity to see their work in augmented reality with all the advantages that this entails. Starting with the introduction and learning about augmented reality and its tools, later step by step clearly show how you can incorporate your own 3D model to an application that, through our mobile device, introduce it into the real world . With the Android Studio program, the Google ARCore platform and code programmed in Java, we can ensure that both professionals and newbies in this area get to know and use augmented reality.

Realidad aumentada - 3D - Android - Aplicación

Augmented reality - 3D - Android - Application

CONTENIDO

1. INTRODUCCIÓN

Hay ciertas tecnologías que se han establecido ya en nuestras vidas y no podemos ignorar su fuerza. Este es el caso de los ordenadores, internet, teléfonos móviles… Por ello siempre es interesante estar atentos de hacia donde se dirigen estos nuevos caminos, para saber qué nuevos logros o investigaciones nos depara el futuro. Este es el caso del tema a tratar en este trabajo.

El mundo relacionado con el 3D se está ganando nuestro interés, desde la gran cantidad de programas de modelado que hay hasta el auge del sector de la impresión 3D. Cada vez más personas, tanto profesionales como usuarios sin un conocimiento profundo sobre el tema, se lanzan a participar en este campo. Por ello parece interesante tenerla en cuenta.

Otro avance que seguro que en algún momento va a cambiar nuestras vidas o por lo menos en la forma que interactuamos con el mundo, es la realidad aumentada. Cada día se ven más aplicaciones que aparecen en nuestros móviles que utilizan esta nueva tecnología. Es cierto, que a día de hoy sigue en crecimiento, pero es indudable el potencial que tiene y lo que puede llegar a ser.

Teniendo estos dos campos y uno de los problemas en mente que diseñadores, arquitectos o usuarios del modelado 3D tienen, surge una idea. El problema que muchas personas pueden tener al hacer un modelo en 3D es la percepción que podamos recibir de este, tanto en relación con el propio modelo como de este con su entorno. Esto es algo que siempre pasará en el ordenador ya que nunca podremos ser cien por cien conscientes de la relación que buscamos en la forma si la vemos en 2D. De ahí surge la tecnología que es la impresión 3D, gracias a la cual podemos realizar con rapidez prototipos e interactuar con ellos. Sin embargo, esto tiene sus costes, sus limitaciones e impedimentos temporales. ¿Se podría lograr mejorar la percepción de los modelos 3D de alguna otra forma, tanto para su creación como para su venta? Quizá con el uso de la herramienta mencionada, la realidad aumentada.

1.1. Objetivos

El objetivo de este documento es presentar una introducción a la realidad aumentada y a su uso.

Primero queremos hacer un estudio de utilidades de estas tecnologías. Para ello, analizaremos la potencial importancia de la realidad aumentada en el futuro, además de profundizar en la actual, al realizar un estudio del arte de esta tecnología y un estudio de las aplicaciones que nos podemos encontrar a día de hoy que la usan.

Segundo. Enseñaremos a usar y descargar algunos de los software más usados para la programación de la realidad aumentada, además de poder probar ejemplos de dichas herramientas con nuestro propio hardware (ordenador y teléfono móvil).

Tercero. Después de haber visto las herramientas a usar, y haber hecho un inciso en los fundamentos de la programación, pasaremos a hacer un tutorial para programar nuestras propias aplicaciones de realidad aumentada. En estas aplicaciones conseguiremos visualizar nuestros propios modelos y animaciones 3D introduciéndolos en el mundo real a través de un dispositivo móvil.

1.2. Visión general

Se pueden distinguir tres grandes fases a la hora de entender la continuidad del documento.

La primera, es una investigación sobre la realidad aumentada (RA). En esta parte definimos la realidad aumentada, analizamos la importancia presente y futura de la RA, analizando la importancia de la experimentación así pudiendo casi afirmar el crecimiento gradual que conllevar. También investigamos el estado del arte y un estudio de mercado de las aplicaciones que ya se encuentran disponibles que usan esta tecnología.

En la segunda parte, mostramos los fundamentos y diferentes conceptos de la programación en realidad aumentada, enseñando a usar herramientas como entornos de desarrollo integrado que utilizaremos en la creación de nuestras aplicaciones.

La tercera parte, está compuesta por una serie de tutoriales para que el lector pueda programar diferentes aplicaciones en las que pueda introducir sus propios modelos y animaciones 3D y visualizarlos en su dispositivo usando lo aprendido de la realidad aumentada.

Como podemos ver, la tercera fase es la parte principal y finalidad del documento, llevando al lector desde el mostrar los fundamentos de la realidad aumentada hasta que pueda programar sus propias aplicaciones.

2. REALIDAD AUMENTADA

A pesar de que la realidad aumentada (RA) fue conceptualizada por primera por Ivan Shuterland (padre de la informática gráfica [1]) en 1968, hasta hace relativamente poco el campo se ocupó principalmente de los retos de ingeniería asociados con el desarrollo de hardware y software RA.

Debido a que la RA es un medio tan convincente con unos usos de tanto potencial, conviene conocer tanto sus antecedentes, como el desarrollo y los campos de aplicación de esta tecnología.

En este capítulo veremos una visión completa de la RA, desde su propia definición y origen hasta un análisis del estado del arte, sus limitaciones como tecnología y su aplicación en un estudio del mercado actual.

2.1. Definición

La realidad aumentada se define como el conjunto de tecnologías destinadas a que el usuario visualice parte del mundo real al que se le ha añadido información gráfica a través de un dispositivo tecnológico por el que dicho usuario recibe la información completa.

Mediante estos dispositivos de la realidad aumentada se añaden una parte sintética virtual a la real, es decir, añaden información virtual a la información física ya existente, de forma que elementos físicos reales se combinen con información virtual. La realidad aumentada se diferencia de la realidad virtual [2] en que en esta segunda los modelos 3D se integran en otros escenarios 3D y no en escenarios reales.

Esta característica puede ser la más importante al referirnos en términos de realidad aumentada (RA), sin embargo también hay quienes añaden más, para referirnos a la totalidad del RA. Así por ejemplo Ronald Azuma define que además de la combinación de elementos reales y virtuales, también tiene que ser interactiva en tiempo real y registrada en 3D. [3]

Figura 1. Continuo de la virtualidad readaptado por Azuma

En la Figura 1 podemos ver un gráfico de lo que Milgram y Kishino llamaron Virtuality Continuum (1994/1995), traducido al español como el Continuo de la Virtualidad [4]. Según esta, la RA es una parte de la zona más general llamada Realidad Mixta. Mientras que en las Realidades Virtuales se puede añadir objetos reales a los virtuales o sustituir el medio ambiente circundante por uno virtual, la RA se lleva a cabo en el mundo real, añadiendo la información virtual a esta realidad.

Según la definición de Azuma (mencionada anteriormente) un sistema de realidad aumentada: [3]

-combina objetos reales y virtuales

-está registrada en 3D

-es interactiva en tiempo real

Hay aspectos que en esta definición son importantes de mencionar. En primer lugar, no se limita a determinadas tecnologías de visualización como aparatos que son solo una pantalla en la cabeza. Ivan Sutherland aunque sentó el concepto de realidad aumentada también creó pantallas de este estilo pero eran dispositivos de realidad virtual.

Figura 2. "La espada de Damocles" considerada el primer sistema de pantalla montada en la cabeza (HMD) de Realidad Virtual

Tampoco la definición se limita al sentido de la vista, ya que potencialmente (si se puede tecnológicamente) se puede aplicar a todos los sentidos.

Por último, la eliminación de objetos reales por la superposición de los virtuales también es considerada como realidad aumentada.

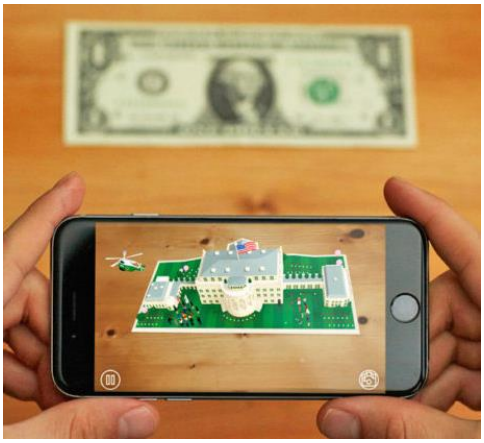


Figura 3. La eliminación de objetos reales a través de reconocimiento de patrones también es realidad aumentada

La RA supone la incorporación de datos e información en un entorno real a partir del reconocimiento de patrones que se realiza a través de un software. Lo que lo convierte en una herramienta interactiva de gran potencial.

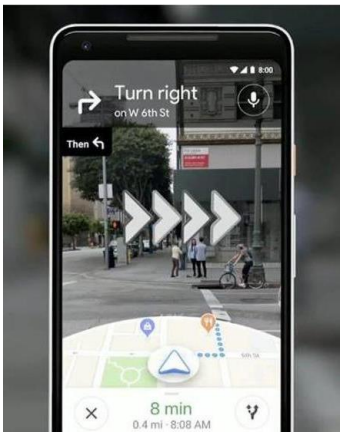


Figura 4. Ejemplo de realidad aumentada en un dispositivo móvil

La propia historia nos ha enseñado el avance tecnológico en el último siglo de los ordenadores, haciéndose cada vez más pequeños y menos costosos, normalizándose ya en la actualidad el llevarlos constantemente encima. Ahora empieza una revolución similar cuyo principio fundamental será el transformar la forma en la que accedemos a la información.

Es cierto que esta tecnología está en sus primeras fases de desarrollo, aunque en estos últimos años se está viendo un gran auge y crecimiento en la aplicación de la RA en sectores como publicidad, medicina, industria, videojuegos, arquitectura, diseño, etc.

Pero aún en sus primeras fases de desarrollo no podemos apartarlo de un posible imaginario común del futuro simplemente porque en la actualidad no esté todavía extendido. Este avance lento surge de que las exigencias tecnológicas para la RA son mucho más altas que para entornos virtuales o realidad virtual, por lo que su maduración como tecnología ha sido más lenta. Sin embargo, los componentes claves necesarios para construir un sistema de RA se han mantenido igual desde el trabajo pionero de Ivan Sutherland.

Para cerciorarnos de que esta tecnología va a tener un lugar en el futuro vamos a analizar y exponer lo importante que es la experimentación para el auge de este tipo de invenciones y como así, gradualmente irá formando parte de nuestra vida.

2.2. La importancia de la RA en el futuro. Importancia de la experimentación en la RA según J. Edward Swan y J.L. Gabbard o por qué gradualmente la realidad aumentada se verá cada vez más inmersa en nuestras vidas

En “Survey of user-based experimentation in augmented reality” de J. Edward Swan y J.L.Gabbard realizan un estudio de cómo se van realizando cada vez más publicaciones e investigaciones sobre el tema de la realidad aumentada. [5]

En el estudio hacen énfasis en la importancia de la experimentación, dado que es uno de los puntos más importantes para la mejora de la tecnología y su implantación gradual en la sociedad.

Según J.L. Gabbard los métodos tradicionales persona-ordenador (HCI) como el análisis de dominios, de las necesidades del usuario, de tareas o el desarrollo de casos de usos, se aplicaban con éxito para determinar el qué debía ser mostrado al usuario. Lo que estos enfoques no nos dicen, y lo que, hasta la fecha no se ha investigado lo suficiente, es el cómo la información debe ser presentada a los usuarios.

Solo tras la experimentación basada en el usuario para afrontar los retos del diseño de interfaz (tales como los inherentes a la dificultad de la percepción de la escena visual virtual y el mundo real combinados o los problemas asociados de las técnicas de interacción de los usuarios con sus móviles), irá evolucionando la realidad aumentada hasta que sea introducida, desarrollada y adoptada por la mayoría como una tecnología más de nuestro sistema de visualizar y entender la información.

Para comprender lo necesaria que es la experimentación basada en el usuario en la RA haremos un resumen del estudio realizado por J. Edward Swan y J.L. Gabbard [5]. En dicho estudio examinan la información basada en el usuario recopilada hasta la fecha.

El estudio es un mecanismo de cómo entienden ellos que se puede comprender mejor el alcance de la experimentación basada en el usuario hasta el presente y el potencial futuro de la RA. Ciertamente es una referencia útil para aquellos que deseen llevar a cabo la investigación basada en el usuario de la RA, ya que no sólo proporciona un único punto de entrada para un conjunto representativo de los estudios basados en el usuario de la RA, sino que también indica implícitamente áreas de investigación que aún no han sido examinadas desde la perspectiva del usuario.

Swan y Gabbard revisaron sistemáticamente la comunicación de los congresos más importantes en la investigación de la realidad aumentada. En concreto, se revisaron los artículos de 1998 hasta 2004 de los siguientes congresos:

- Simposio Internacional de Realidad Mixta y Aumentada (ISMAR) actuaciones entre 1998 y 2004. El simposio se realizó con los siguientes nombres:

> IEEE / ACM Taller Internacional sobre la Realidad Aumentada (Iwar) en 1998 y 1999
>
> IEEE / ACM Simposio Internacional de Realidad Aumentada (ISAR) en 2000 y 2001,
>
> Simposio Internacional de Realidad Mixta (ISMR) desde 1999 a 2001
>
> Simposio Internacional sobre la Realidad Aumentada Mixta y de 2002 a 2004

- Simposio Internacional sobre Wearable Computers (ISCA) actas de 1997 a 2004

- IEEE Realidad Virtual (VR) procedimientos de 1995 a 2004

- Presence: Teleoperadores y entornos virtuales publicaciones en revistas desde 1992 hasta 2004

De aquí sacaron la información y realizaron la siguiente tabla la cual muestra las numerosas publicaciones relacionadas con la realidad aumentada, publicaciones relacionadas con HCI y experimentos basados en el usuario.

Publicación	Años	Publicaciones Totales	RA-Publicaciones relacionadas	HCI-Publicaciones relacionadas	Experimentos con el usuario
ISMAR	1998-2004	181	181	14	9
ISWC	1997-2004	170	28	12	5
IEEE VR	1995-2004	301	24	3	3
Presence	1992-2004	452	33	9	4
Total		**1104**	**266**	**38**	**21**

Tabla 1. Resumen numérico de experimentos basados en el usuario relacionados con la RA

La columna del HCI, son publicaciones relacionadas con la RA; todas aquellas publicaciones sobre el HCI no relacionadas han sido descartadas en el recuento.

Hay que tener en cuenta que como se muestra en la Tabla 1, este estudio se realizó solo hasta el 2004, y desde entonces ha habido un crecimiento mucho mayor de la realidad aumentada. Aun así como podemos ver, hasta esa fecha ha habido no pocas publicaciones sobre la realidad aumentada. Incluso en esas fechas ya había publicaciones sobre la interacción con el usuario.

A lo largo de estos años los artículos han ido en aumento y quizá un análisis de este tipo hoy en día sería difícilmente tratable. Lo que si podemos ver fácilmente es el crecimiento que ha tenido desde el año del estudio; podemos afirmar casi con seguridad que es una tecnología que ciertamente con la experimentación e investigación va a ir gradualmente en aumento, a tal punto de que ya hemos comenzado a ver y usar esta tecnología en la actualidad de forma masiva.

2.3. Antecedentes. Estado del arte

Al paso de los años, investigadores y desarrolladores encuentran más y más áreas que podrían beneficiarse de la RA. Los primeros sistemas en aplicarse fueron en los militares, y los de aplicación industrial y médica, pero no tardaron en empezar a utilizarse en el uso comercial y ocio. A continuación vamos a ver los diferentes campos de aplicación del RA.

Aplicaciones en sistemas de información personal

Hollerer y Feiner consideran que uno de los mayores mercados potenciales para la RA podría ser en la computación personal. [6]

Por ejemplo, la RA podría integrarse en el teléfono y el correo electrónico con superposiciones al mundo real, gestionar la información personal relacionándola con lugares o personas físicas, proporcionar una interfaz de control unificado para todo tipo de electrodomésticos… en definitiva, un enorme potencial para la creación de aplicaciones sobre la computación personal.

Dicha plataforma también podría ser aprovechada para agencias de marketing, pudiendo ofrecer cupones a peatones o colocar vallas publicitarias virtuales. A día de hoy algunas empresas ya comienzan a utilizar en su packaging RA para llamar la atención frente a otras marcas.

Figura 5. Etiquetas que funcionan como patrón de reconocimiento para aplicaciones de RA

En el campo de la navegación tiene una fuerte acogida las aplicaciones de realidad aumentada. Este quizá sea el más reconocible a día de hoy, ya que algunos GPS ya funcionan con esta tecnología. Se podría usar ya no solo como guía sino también como informador al usuario de diferentes atascos, señalización o posibles accidentes en las diferentes carreteras.

Figura 6. Ejemplo de GPS con RA

En el turismo ya ha habido algunos avances en los cuales se han reconstruido de forma digital un lugar del patrimonio cultural de una ciudad. De esta forma los visitantes pueden ver y aprender sobre la arquitectura antigua y costumbres a la par que lo visualizan.

Figura 7. Vlahakis presentó un proyecto, ARCHEOGUIDE, que reconstruía patrimonio de Olimpia, Grecia [7]

Aplicaciones industriales

Diseño, montaje y mantenimiento son áreas típicas en las que la RA puede ser de utilidad. Estas actividades se pueden aumentar tanto en entornos coorporativos como militares.

En el campo del diseño sobretodo se está utilizando la realidad aumentada en la industria automovilística. SpaceDesign trabajó en una aplicación que permitía la visualización y modificación de la curvatura del coche y del diseño del motor. [8]

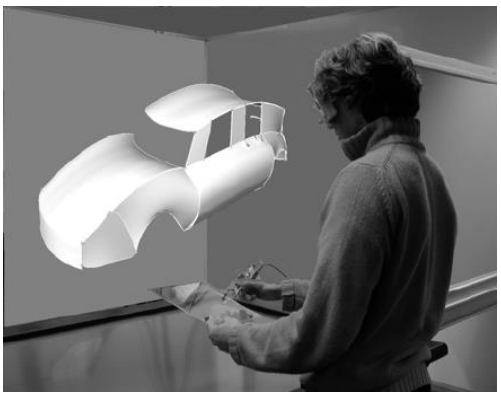


Figura 8. SpaceDesign MR workspace

Volkswagen tiene la intención de usar RA para comparar las imágenes de prueba de choque calculada y la real. [9]

También se utilizan los datos de los automóviles de DaimlerChrysler para crear una simulación donde se pueda abrir la puerta de un coche de concepto virtual y experimentar en su interior. Se basa en la combinación de un esqueleto real del coche y por encima de este los añadidos digitales. [10]

Figura 9. Aplicación en la que el usuario puede interactuar con el coche físico y ver o modificar los acabados virtuales

Otra aplicación que no pertenece a la industria del automóvil sería la presentada por Collet y MacDonald, la cual se encarga de la visualización de datos por el sensor de un robot. De esta forma, pequeños robots como la aspiradora Roomba podrían visualizar con sus sensores y crear trayectorias por todas las extensiones de la vivienda. [11]

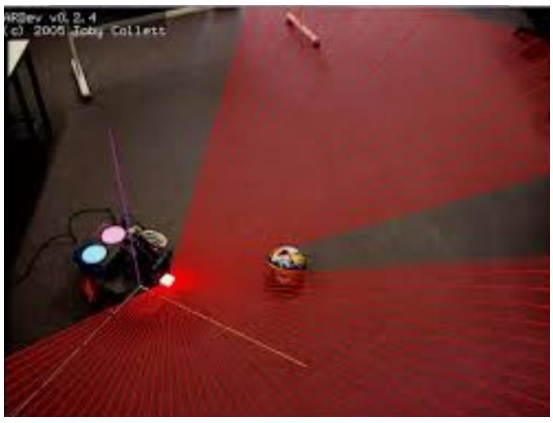


Figura 10. Visualización de datos de un robot (Collet y MacDonald, 2006)

En el campo del montaje de nuevo vuelve a aparecer la industria de la automoción, ya que BMW está experimentando con RA para mejorar los procesos de soldadura en sus coches. [12]

Otras empresas en procesos de producción utilizan la RA para superponer diagramas esquemáticos y la documentación adjunta directamente sobre el espacio de los operarios así facilitando el trabajo. Como por ejemplo los montadores de la empresa Airbus, que se apoyan en la RA para el montaje de sistemas de agua [13]. Sin embargo, estos sistemas todavía no están muy afianzados en estos trabajos ya que dificulta la realización de otras tareas, como la movilidad en esos espacios o la comunicación entre compañeros.

Figura 11. Visualización con RA de sistema de agua Airbus

También la RA puede ser utilizada en la construcción, pudiendo tener la posibilidad de controlar y programar el progreso individual de cada operario con el fin de poder gestionar grandes proyectos de construcción complejos. Feiner puso un ejemplo [6], en el cual se podía tener una representación de la visión general de toda la escena de la construcción ya que los trabajadores utilizan su HMD y además estos pueden con ellos ver su progresión y lo que les toca hacer en cada momento.

El ámbito de mantenimiento de maquinarias o estructuras complejas requiere una gran cantidad de personal formado. La aplicación de la RA está resultando útil en esta tarea. Honda y Volvo ordenaron estos sistemas para ayudar a sus técnicos a conocer los coches y la información de reparación.

La gran parte de la investigación de la RA en las universidades y empresas son el resultado de la financiación militar. Estas se basan en las mejoras de los sistemas de navegación, de comunicación, la reparación y el mantenimiento o la medicina de emergencia. También se investiga en específico para usos militares, que se posea información de escenarios de combate a gran escala, la simulación

del enemigo a tiempo real o que los soldados en tierra tengan datos del reconocimiento de vehículos aéreos no tripulados. En estos casos se busca que el nivel de información que se le muestre al usuario no sea crítico si no el necesario. Este punto también se extrapola al uso más comercial de las aplicaciones de RA. [14]

Aplicaciones médicas

La medicina podría beneficiarse de la RA para mejorar las rutas de visitas de enfermeras y médicos, incluso de recibir información directamente en las gafas como propuso Hasvold en 2002 [15].

Para ser aplicada en el campo de la cirugía se necesitaría más innovación y mejora de los sistemas, sin embargo, en la cirugía laparoscópica ya se ha documentado las oportunidades que ofrece y se han realizado simulaciones en las cuales el cirujano puede visualizar una simulación de los cortes en el paciente.

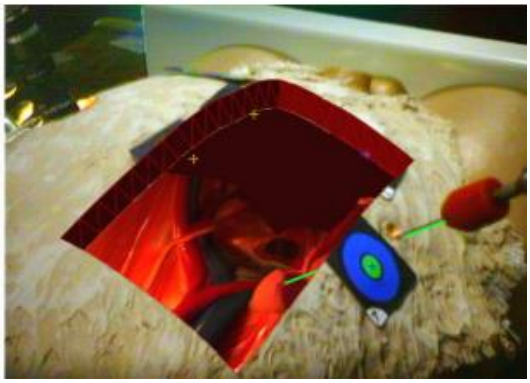


Figura 12. Visualización de la simulación de una operación en cirugía laparoscópica

Además de enfoques didácticos o de organización en hospitales también se están probando en la medicina aplicaciones de RA en la exploración médica. Se trata de superponer la imagen interna de la parte del cuerpo en cuestión a la externa que vemos utilizando ultrasonidos o tomografía computerizada. Esto podría mejorar la captura de la imagen médica, donde además de agilizar el proceso de la obtención de dichas imágenes, también desaparecerían problemas como la colocación y calibración de las cámaras. El médico podría llevar unas gafas que con resonancia magnética podría proporcionar los puntos de vista de la manipulación de herramientas ocultas bajo el tejido y superficies.

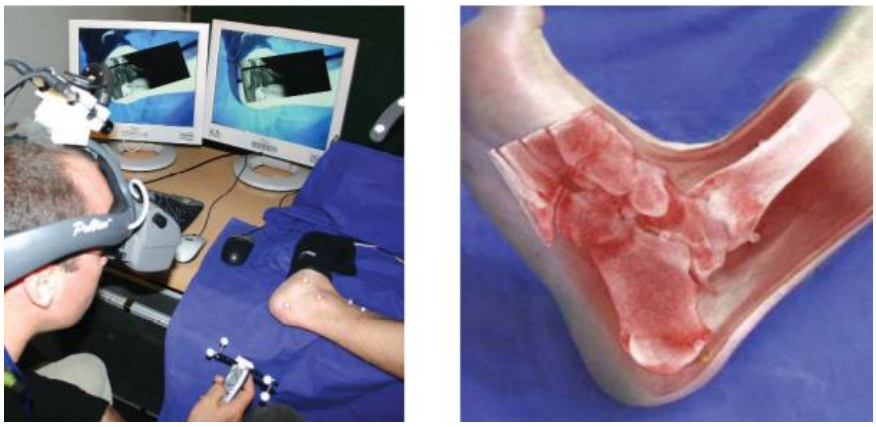


Figura 13. Aplicación de RA en la que se superpone los tejidos internos a los externos en una exploración médica

Aplicaciones en entretenimiento

Al igual que la realidad virtual, la RA se puede aplicar a la industria del entretenimiento. De hecho, si nos salimos de campos de investigación más especializados, las aplicaciones destinadas a usuarios convencionales que podemos descargar en la actualidad, la gran mayoría son de ocio. Este campo es el del público más amplio lo que también les sirve a anunciantes la colocación de publicidad de sus productos.

Hoy en día donde podemos ver muchos ejemplos de la aplicación de la RA es en las retransmisiones deportivas. Piscinas, campos de fútbol, pistas de carreras o cualquier otro entorno deportivo puede ser preparado fácilmente para el uso de aplicaciones de RA. Un ejemplo es el sistema FoxTrax, encargado de resaltar la ubicación del disco de hockey en las retransmisiones por su dificultad al verlo moverse a tanta velocidad. Otro es en las carreras de coches también por la dificultad de su seguimiento, la RA hace que se señalicen con mayor facilidad. También en partidas de billar para marcar la trayectoria de las bolas o señalizaciones en los campos de fútbol.

Sin embargo, quizá donde más se ha exprimido este concepto sea en la industria de los videojuegos. Ya cuentan muchos juegos con la aplicación de la RA, algunos pioneros basados en la ARToolKit (una biblioteca que permite la creación de RA) como “AquaGaunglet”, “ARHockey” o “AR-Bowling” entre otros muchos. En este sector ya ha habido bastante avance llegando al punto de que casi las únicas aplicaciones de RA que podemos encontrar en las tiendas virtuales de aplicaciones de nuestros móviles son juegos.

Aplicaciones en el mundo empresarial

La aplicación de RA en espacios como oficinas, en colaboraciones durante reuniones, gestión pública o planificación de proyecto puede ser de gran utilidad.

Cuando muchas personas tiene que ver, discutir o interactuar con un mismo modelo 3D de forma simultánea como podría hacerse en una reunión para la creación del diseño de un nuevo producto o la planificación de un proyecto arquitectónico, se podría utilizar para ello la aplicación de la RA. De esta forma se podría ahorrar el tiempo y el dinero de hacer prototipos o maquetas y se podrían realizar en entornos de colaboración actividades yuxtapuestas que de otra forma serían imposibles de realizar simultáneamente.

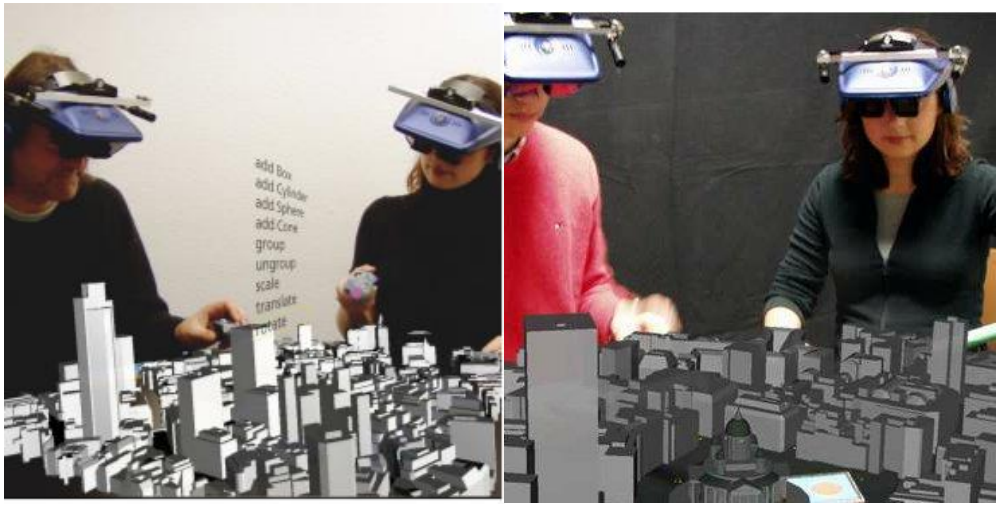


Figura 14. Visualización de planes urbanos o diseño industrial en un entorno colaborativo. En la imagen podemos ver el proyecto ARTHUR [16]

Aplicaciones en educación

En el ámbito didáctico también tiene cabida la RA. Kaufmann creó la herramienta Construct3D para aprender matemática y geometría [17].

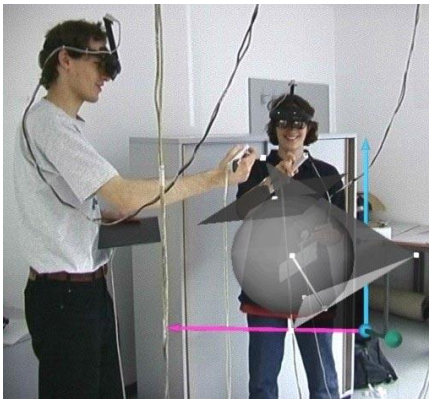


Figura 15. Visualización de Construct3D

También se han creado programas como MARIE, realizado para aprender conceptos de ingeniería y apoyar su aprendizaje [18].

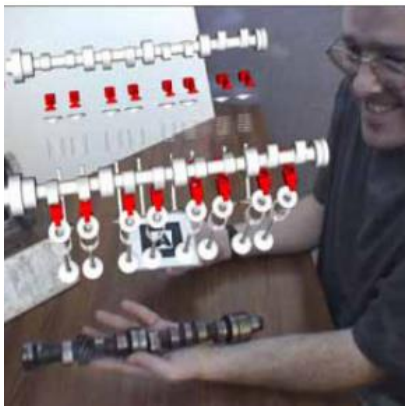


Figura 16. Visualización de MARIE

Hasta el MIT está trabajando en realizar juegos interactivos en RA para mejorar el aprendizaje. Además ya podemos encontrar una aplicación de la BBC que nos permite ver en 3D monumentos históricos y aprender sobre ellos.

Como hemos visto en el estudio del arte, la mayoría de las aplicaciones se realizan en campos muy especializados y forman parte de trabajos que se siguen investigando o que son solamente presentaciones de proyectos. Esto puede surgir de las distintas limitiaciones de la RA.

Y es que la RA se enfrenta a desafíos técnicos relacionados, por ejemplo, con el peso de las gafas o aparatos de visión, el costo, el uso de energía, la fatiga del ojo, la alta resolución, la profundidad de color, luminancia, contraste, campo de visión o la profundidad de enfoque.

Quizá la limitación del uso de la RA a través de gafas o aparatos de visión se su portabilidad y el uso al aire libre. La mayoría de los cascos y aparatos de RA, como hemos visto en algunas imágenes antes mostradas, son engorrosos, ya que el usuario tiene que portar el pc, las baterías, los sensores y pantallas. Las conexiones ente las diferentes partes tiene que resistir tanto el movimiento como los condicionantes del exterior. El problema del brillo de las pantallas y el contraste por la iluminación del exterior también es un factor a tener en cuenta, aunque MicroVision ha desarrollado pantallas láser que superan esta dificultad.

El seguimiento en entornos sin preparación sigue siendo un reto, pero los enfoques híbridos son cada vez más pequeños. De ahí a que en la última época veamos que ya hay teléfonos móviles capaces de hacer las calibraciones necesarias para empezar a usar algunas aplicaciones de RA.

2.4. Realidad aumentada en dispositivos móviles

Como hemos visto en las limitaciones de la RA, podemos ver que quizá donde menos empieza a haber es en el ámbito relacionado con los dispositivos móviles. Es cierto, que se necesitan avances tecnológicos y más investigación para aplicarlos consecuentemente a las ramas de la medicina o la ingeniería, sin embargo, en estos dispositivos ya se ha empezado a ver un avance con la mejora de la calibración y el aumento de su capacidad informática.

Así mismo, los dispositivos móviles ya están completamente arraigados a nuestro día a día y han llegado a un gran público para que empresas cada vez se interesen más en aplicar la RA en sus aplicaciones.

Haciendo un análisis de las aplicaciones a disposición de los usuarios en las tiendas virtuales de aplicaciones, podemos encontrar aplicaciones que ya usan la realidad aumentada.

BBC Civilizations AR

Como hemos visto con anterioridad la educación es uno de los puntos fuertes en los que se puede aplicar la RA. En esta aplicación se nos permite rotar, escalar y localizar diferentes artefactos históricos en RA.

Figura 17. Visualización de la app: BBC Civilizations AR

ARuler

Una aplicación que hace mediciones. Mide tanto distancias como ángulos, además de calcular volúmenes y áreas.

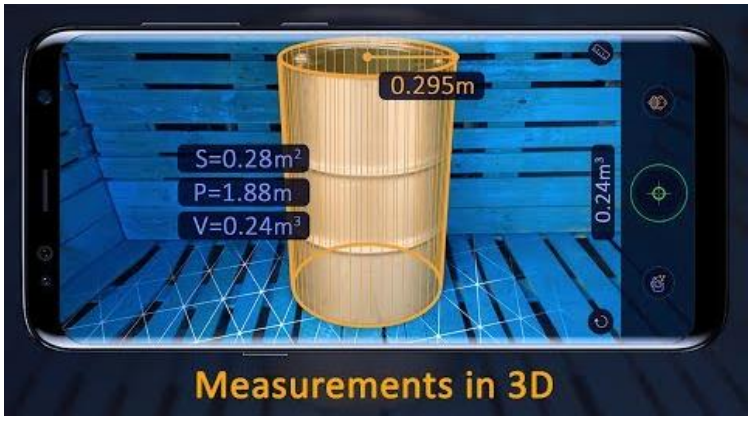


Figura 18. Visualización de la app:ARuler

MagicPlan

Esta aplicación te da, al ir clicando en los diferentes puntos de la sala en la que te encuentras, el plano creado con tus indicaciones.

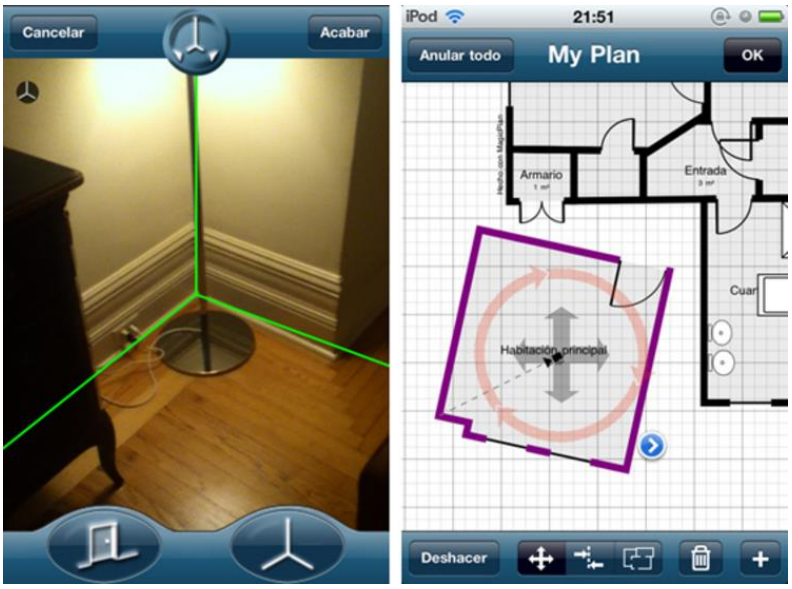


Figura 19. Visualización de la app: MagicPlan

SketchAR

En esta aplicación proyectará en un folio el dibujo que el usuario haya elegido.

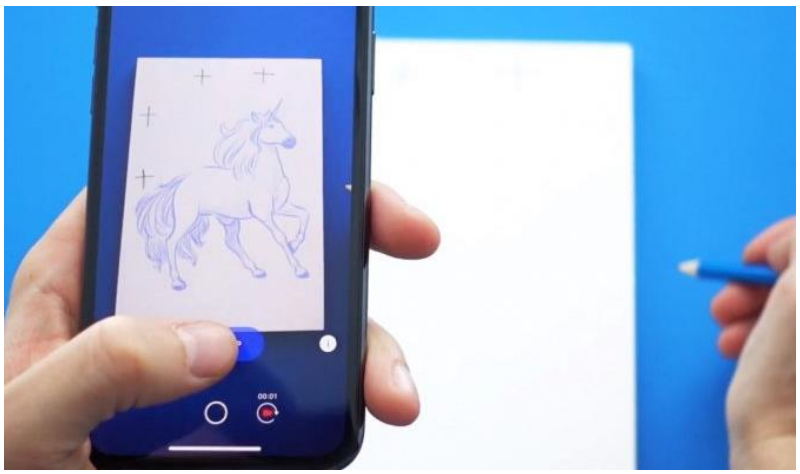


Figura 20. Visualización de la app:SketchAR

Quiver

Se trata de una aplicación que hace “salir” del papel lo dibujado en una serie de folios que pone en disposición la aplicación para que imprimas. Fundamentalmente es una aplicación para niños, los cuales pueden imprimirse folios de entretenimiento o didácticos.

Figura 21. Visualización de la app: Quiver

Ink Hunter

Esta aplicación logra poner una serie de tatuajes en el cuerpo de la persona que aparezca en la pantalla. Puedes rotarlos y escalar los dibujos e incluso utilizar tus propios diseños.

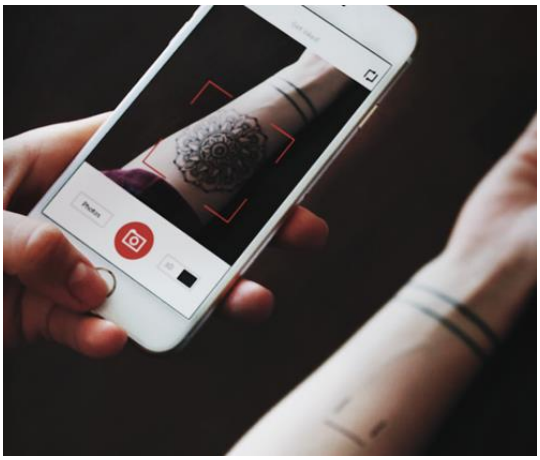


Figura 22. Visualización de la app: Ink Hunter

Augemented car finder

Esta aplicación busca tu coche (o cualquier punto que hayas marcado) y lo marca con una enorme flecha que te va indicando en qué lugar lo dejaste.

Figura 23. Visualización de la app: Augmented car finder

Ikea Place

Se trata de un catálogo de los productos de Ikea que puedes colocar en tu casa.

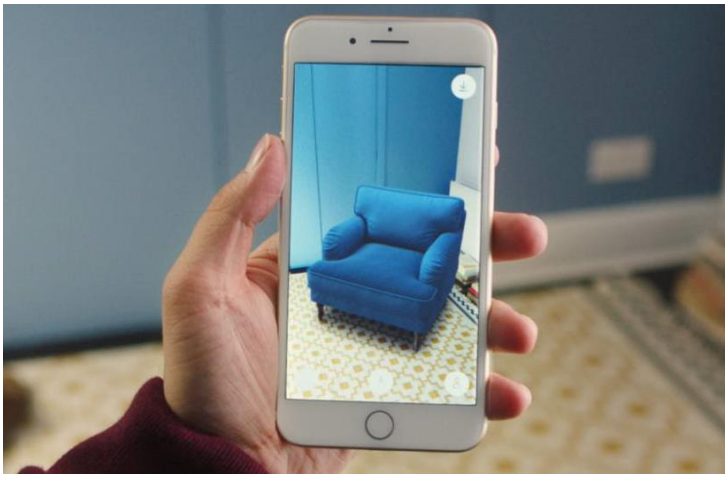


Figura 24. Visualización de la app: Ikea Place

Holo

Es una aplicación que nos permite tomar personajes tanto del mundo ficticio como del real y arrastrarlos al y soltarlos en tu entorno inmediato.

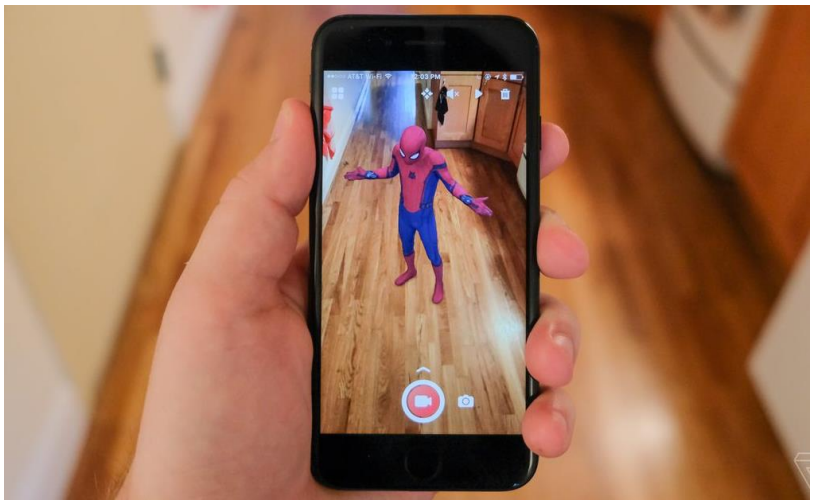


Figura 25. Visualización de la app: Holo

WallaMe

Esta aplicación permite dejar mensajes en todo el mundo para que otros usuarios lo puedan leer, al igual que tú también puedes ver lo suyos.

Figura 26. Visualización de la app: WallaMe

ROAR

Es una aplicación que te da la información, como el precio o los ingredientes, de los productos solo con apuntarles con la cámara.

Figura 27. Visualización de la app: ROAR

SkyView

Esta aplicación logra proyectar una cuadrícula de las constelaciones, estrellas y otros objetos estelares conocidos que aparezcan en la pantalla.

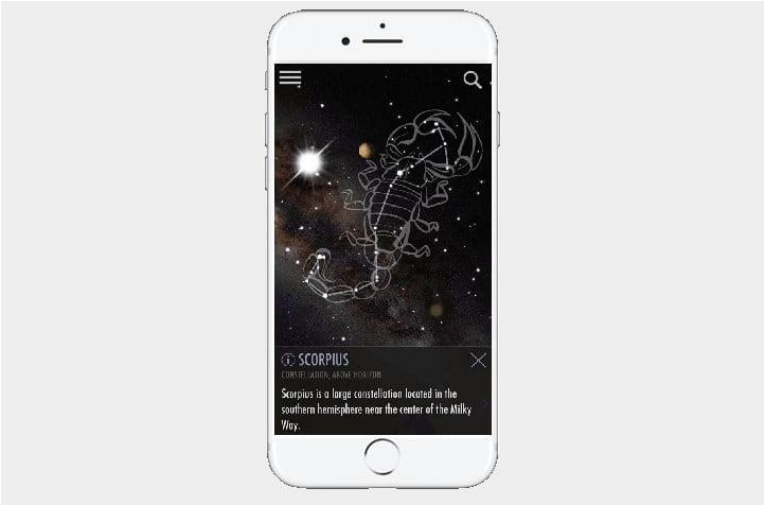


Figura 28. Visualización de la app: SkyView

El resto de aplicaciones que no son las mostradas anteriormente son mayoritariamente videojuegos o están destinadas al entretenimiento. Algunos son tan conocidos como Pokemon Go y otras están programados por empresas independientes o de bajo presupuesto.

2.6. Conclusiones

En este capítulo hemos presentado una introducción a la realidad aumentada. Hemos comenzado por definirla para posteriormente explicar su importancia y el potencial en el futuro de esta tecnología. Posteriormente hemos profundizado en un análisis del estado del arte y un estudio de mercado. Después de dicho estudio de las aplicaciones nos encontramos con que no hay ninguna aplicación destinada a ciertos campos específicos como el del diseño industrial. En dicho campo nos interesaría una aplicación capaz de mostrar uno o varios modelos en 3D o incluso animaciones 3D en el entorno cercano, pudiéndolos modificar de lugar y de escala.

Por esta razón, nos disponemos en la siguiente parte del documento a intentar entender y programar una aplicación de realidad aumentada con dichas características.

3. REALIDAD AUMENTADA EN ANDROID

3.1. ARCore

Explicación del funcionamiento y conceptos de ARCore

ARCore es la plataforma de Google para construir experiencias de realidad aumentada. Mediante diferentes interfaces de programación de aplicaciones (API), como puede ser Android Studio o Unity, ARCore permite que el dispositivo móvil detecte su entorno, comprenda el mundo e interactúe con la información. [19]

ARCore utiliza tres capacidades clave para integrar contenido virtual con el mundo real tal y como se ve a través de la cámara de su teléfono:

-El seguimiento de movimiento permite que el teléfono entienda y rastree su posición en relación con el mundo.

-La comprensión ambiental permite que el teléfono detecte el tamaño y la ubicación de todo tipo de superficies: superficies horizontales, verticales y anguladas como el suelo o paredes.

-La estimación de luz le permite al teléfono estimar las condiciones de iluminación actuales del entorno.

Para explicar el funcionamiento de ARCore primero expondremos conceptos fundamentales.

Concepto: Rastreo de movimiento;

A medida que un teléfono se mueve de posición, ARCore utiliza un proceso llamado odometría (estudio de la estimación de la posición del dispositivo durante la navegación) y mapeo concurrente para comprender dónde está el teléfono en relación con el mundo que lo rodea. ARCore detecta características visualmente diferentes en las imágenes que va capturando la cámara llamadas puntos de características y utiliza estos puntos para calcular su cambio de ubicación. La información visual se combina con mediciones inerciales de la IMU (Unidad de Medición Inercial) del dispositivo para estimar la posición y orientación de la cámara en relación con el mundo a lo largo del tiempo.

Al alinear la posición y orientación de la cámara virtual que representa su contenido 3D con la de la cámara del dispositivo, se puede renderizar contenido virtual desde la perspectiva correcta. La imagen virtual renderizada se puede superponer sobre la imagen obtenida de la cámara del dispositivo, haciendo que parezca que el contenido virtual es parte del mundo real.

Concepto: Entendimiento ambiental;

ARCore mejora constantemente la comprensión del entorno del mundo real mediante la detección de puntos de características y planos. Busca grupos de puntos de características que parecen estar en superficies horizontales o verticales comunes, como tablas o paredes, y pone estas superficies a disposición de su aplicación. Se puede usar esta información para colocar objetos virtuales que descansan sobre superficies planas.

Debido a que ARCore utiliza puntos de características para detectar planos, las superficies planas sin textura, como una pared blanca, pueden dar problemas a la hora de interpretar sus planos.

Concepto: Estimación de luz;

ARCore puede detectar información sobre la iluminación del entorno y proporcionar la intensidad promedio y la corrección de color de una imagen determinada. Esta información permite iluminar los objetos virtuales en las mismas condiciones que el entorno que los rodea, lo que aumenta la sensación de realismo.

Concepto: La interacción del usuario;

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

Concepto: Puntos orientados;

Los puntos orientados permiten colocar objetos virtuales en superficies en ángulo. Cuando realiza una prueba de impacto que devuelve un punto de características, ARCore observará los puntos de características cercanos y los usará para intentar estimar el ángulo de la superficie en el punto de característica dado. ARCore luego devolverá una posición y orientación que tenga en cuenta ese ángulo.

Concepto: Anclas y rastreables;

Las posiciones y orientaciones pueden cambiar a medida que ARCore mejora su comprensión de su propia posición y su entorno. Cuando queramos colocar un objeto virtual, debemos definir un ancla para asegurarse de que ARCore rastrea la posición del objeto a lo largo del tiempo. Muchas veces crea un ancla en función de la posición devuelta por una prueba de éxito en la interacción del usuario.

El hecho de que las posiciones y orientaciones puedan cambiar significa que ARCore puede actualizar la posición de los objetos ambientales como planos y puntos de características con el tiempo. Los planos y puntos son un tipo especial de objeto llamado rastreable. Como su nombre indica, estos son objetos que ARCore rastreará con el tiempo. Se pueden anclar objetos virtuales a rastreables específicos para garantizar que la relación entre su objeto virtual y el rastreable permanezca estable incluso cuando el dispositivo se mueva. Esto significa que si coloca una figura virtual en su escritorio, si ARCore luego ajusta la postura del punto asociado con el escritorio, el modelo seguirá apareciendo encima de la mesa.

Sabiendo estos conceptos podemos entender que ARCore funciona fundamentalmente rastreando la posición del dispositivo móvil mientras se mueve y construye a base de los puntos que recibe del mundo su propia comprensión de este.

Además de detectar esos puntos clave, puede detectar superficies planas y estimar la iluminación promedio en el área de su alrededor. De esta forma es como ARCore comprende el mundo y lo utiliza para poder colocar objetos 3D que se integren en el mundo real.

Conociendo los conceptos y el funcionamiento de ARCore nos será algo más fácil entender los siguientes procedimientos para la creación de nuestro propia aplicación, la cual usará ARCore.

Descargar ARCore

ARCore se descarga como cualquier otra aplicación del teléfono móvil. Si el dispositivo no carece de los requisitos de dicha aplicación, será cuestión de ir al App Store buscar el nombre y tener suficiente espacio en el móvil como para que se descargue.

Puede que solo descargándolo ya venga programado para que funcione correctamente sin necesidad de añadir ningún comando, pero si no es así, habría que añadir el comando adb siguiente:

```
adb install -r -d arcore-preview.apk
```

ARCore va a ser el encargado de analizar las superficies que aparezcan en la cámara en el momento de usar la aplicación, señalando en nuestro caso los planos que encuentre de tal forma que podamos interactuar con ellos e introducir nuestro propio objeto 3D en dicho entorno.

3.2. Android Studio

Como entorno de desarrollo utilizaremos el programa Android Studio 3.1 o una versión superior con una plataforma Android SDK Platform 7.0 (nivel API 24) o una también superior.

Android Studio es el entorno de desarrollo integrado oficial para el desarrollo de apps para Android. A continuación expondremos las funciones básicas del programa. [20]

Los proyectos en Android Studio

Un proyecto es todo aquello que contiene tu lugar de trabajo para una app; desde el código fuente hasta los recursos. Al comenzar un proyecto nuevo Android Studio crea la estructura necesaria para todos los archivos y los hace visibles en la ventana Proyecto. En caso de que no aparezca podremos abrirla en:

Ver>Ventanas de herramientas>Proyecto

Concepto: Java

Java es un lenguaje de Programación Orientada a Objetos. Su sintaxis deriva en gran medida de C y C++. En Android Studio se puede programar en varios lenguajes, los más usado son tanto Java como Kotlin. En nuestro caso usaremos la programación en Java.

Concepto: Entorno de desarrollo integrado

Llegados a este punto hemos de explicar qué es un entorno de desarrollo integrado (IDE). Es una aplicación informática la cual facilita a los programadores el desarrollo de software, es decir, donde vamos escribir el código con el que trabajaremos y compilarlo,

Concepto: Módulo

Un módulo es un conjunto de archivos fuente y opciones de compilación que te permite dividir el proyecto en unidades discretas de funcionalidad. En un proyecto puedes tener uno o varios y se pueden usar con dependencia. Cada módulo se puede compilar, probar y depurar de forma independiente. Entre los tipos de módulos se incluyen los siguientes: módulos de apps para Android, módulos de biblioteca y módulos de Google App Engine.

Estructura del proyecto;

Cada proyecto de Android Studio incluye uno o más módulos con archivos fuente y archivos recurso.

De manera predeterminada Android Studio muestra los archivos de proyecto en la vista de proyecto de Android (La primera pestaña de la figura). Esta vista está organizada en módulos para que puedas acceder rápidamente a los archivos fuente clave del proyecto.

Se puede ver todos los archivos de compilación en el nivel superior de Secuencias de comando de Gradle (es un sistema de construcción) y cada módulo de app contiene mínimo las siguientes carpetas:

-manifests: Contiene el archivo AndroidManifest.xml.

-java: Contiene los archivos fuente Java.

-res: Contiene todos los recursos sin código, como diseños XML, secuencias de comando de IU y también imágenes de mapa de bits.

Posteriormente si se compila o se arranca el programa el Gradle puede cambiar la forma inicial de cómo tenemos dispuesta la ventana.

Como podemos ver en la imagen tenemos una carpeta que tendrá relevancia en nuestro proyecto que es la carpeta “sampledata”, en la cual se encontrarán nuestros archivos .obj o .fbx y donde se creará el archivo .sfa. La importancia de estos archivos la veremos más adelante.

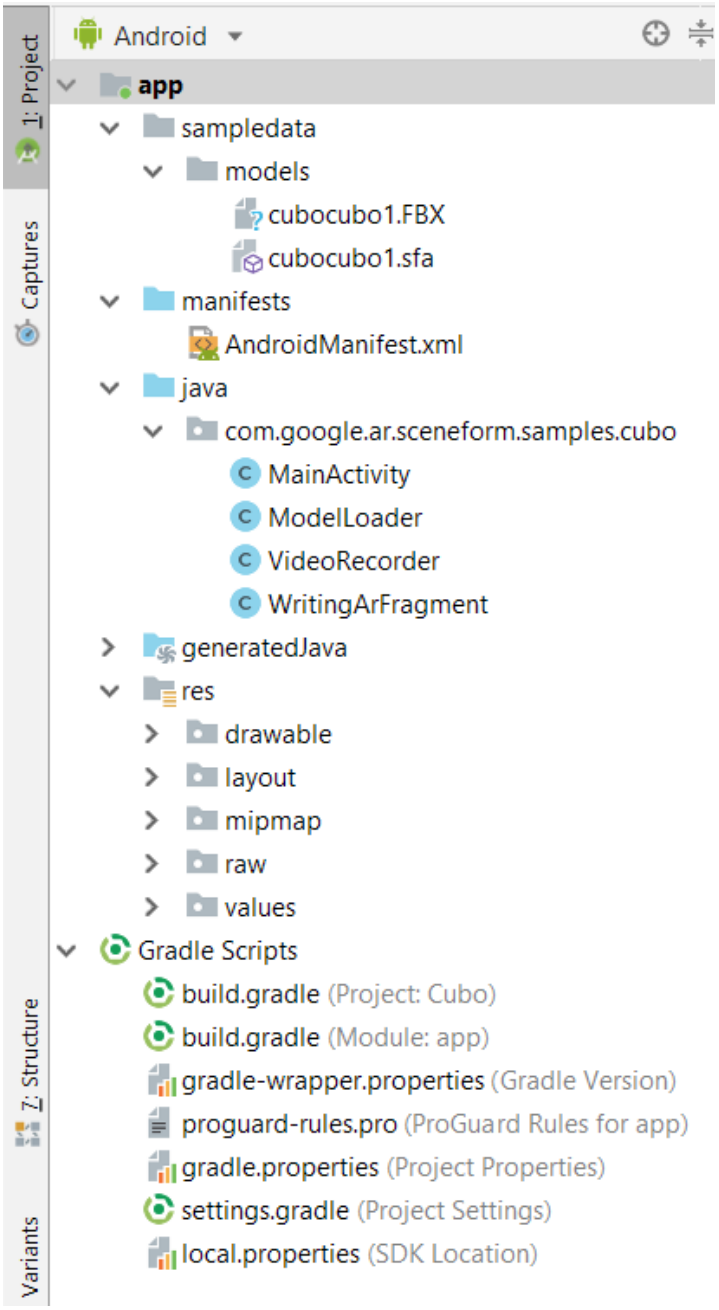
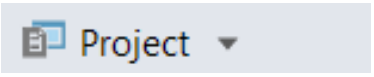


Figura 29. Ventana de Proyecto

Estructura real de un proyecto; 

Si queremos ver la estructura real de un proyecto seleccionaremos Project en el menú despegable de la ventana Proyecto, donde antes tendríamos Android. Aquí podremos ver incluso archivos que se encuentran ocultos en la pestaña de Android. A continuación pondremos el formato de Project, pero la explicación de la función sigue siendo la misma que en la ventana Android.

-build: Contiene resultados de compilación.

-libs: Contiene bibliotecas privadas. Este directorio puede desaparecer si se usa un sampledata y en este es donde aparecerán los archivos, que en nuestro ejemplo serán usados.

-src: Contiene todos los archivos de código y recursos para el módulo en los directorios siguientes:

-androidTest: Contiene el código para las pruebas de instrumentación que se ejecutan en un dispositivo Android.

-main: Contiene los archivos de conjuntos “principales”: el código y los recursos de Android compartidos por todas las variantes de compilación.

-AndroidManifest.xml: Describe la naturaleza de la aplicación y cada uno de sus componentes.

-java: Contiene fuentes de código Java.

-res: Contiene recursos de aplicación.

-drawable: en esta carpeta es donde estarán las imágenes para los iconos que usaremos en la aplicación.

-layout: Contiene el activity_main.xml, el cual tendremos que abrir si queremos modificar la interfaz de la aplicación

-raw: En esta carpeta es donde se creará en nuestro ejemplo el .sfb

-assets: Contiene el archivo que debe compilarse en un archivo .apk.

-build.gradle (module): Este archivo define las configuraciones de compilación específicas para el módulo, no como el bul.gradle (project) el cual es esencial ya que define la configuración de compilación para todo el proyecto. En nuestro caso aquí modificaremos el código para añadir las rutas a nuestros archivos .sfa, .fbx, .obj, etc.

Interfaz del usuario

La interfaz de Android Studio se distribuye en varias áreas lógicas que si sabemos localizar luego el proceso de programación o modificación de la aplicación nos va a ser mucho más sencillo.

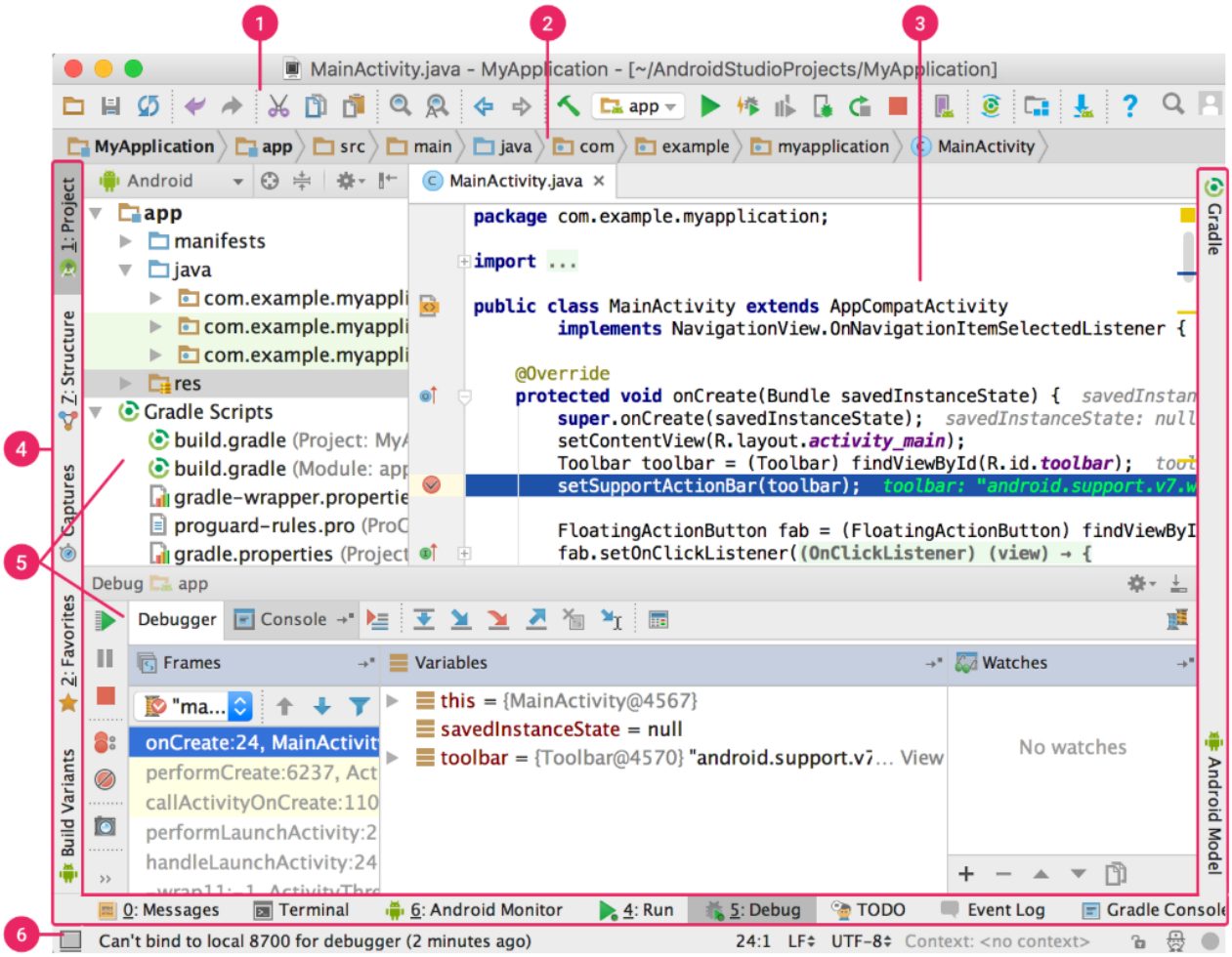


Figura 30. Visión de la interfaz de Android Studio

1_Barra de herramientas: Te permite realizar acciones rápidamente gracias a su disposición de opciones a través de iconos.

2_Barra de navegación: Te ayuda a ubicarte en el proyecto, es una vista en una sola línea de la estructura de la ventana de Proyecto.

3_Ventana de editor: Es el área donde más vamos a trabajar, es donde se crea y modifica el código. Cambia según el tipo de archivo abierto, pudiendo modificar código como también si abres un archivo XML pudiendo editar diseño.

4_Barra de la ventana de herramientas: Se encuentra al exterior de la ventana del entorno de desarrollo integrado y te permite expandir o contraer ventanas de diferentes herramientas individuales.

5_Ventanas de herramientas: Te permites hacer tareas como la administración proyectos o la búsqueda de archivos.

6_Barra de estado: se muestra el estado de tu proyecto además de advertencias y mensajes.

Descargar Android Studio

Para descargar el Android Studio en su versión 3.4, se puede hacer en el siguiente enlace:

https://developer.android.com/studio/index.html

4. TUTORIAL PARTE 1: Preparación del Hardware

Para poder arrancar la aplicación en nuestro dispositivo necesitaremos comprobar ciertos requisitos y cambiar algunos aspectos de este. Además necesitaremos descargar en nuestro ordenador algunas aplicaciones como la que será nuestro entorno de desarrollo. Todos los pasos que tenemos que llevar a cabo hasta poder ejecutar la aplicación de demostración son los siguientes.

4.1. Preparación de dispositivo

Aunque no hay grandes cambios que se tengan que hacer en el dispositivo en el que vamos a ejecutar la aplicación, si hay que cambiar ciertos aspectos de este para que funcione todo correctamente.

Comprobación de requisitos

El dispositivo que utilizaremos tiene que tener ciertos requisitos mínimos para poder ser utilizado en esta práctica, ya que tiene que soportar el programa ARCore el que es indispensable para el uso de la futura aplicación.

ARCore es la plataforma de Google que permite crear experiencias de realidad aumentada y por ello necesita de ciertos sensores como giroscopio, acelerómetro, etc, que quizá no todo dispositivo móvil posea.

Necesitaremos un smartphone real, ya que ARCore no soporta emuladores de android. Por lo que el dispositivo en cuestión debe de tener una versión de Android 7.0 o superior (incluso en algunos modelos puede que sea necesario versiones más nuevas).

Para confirmar si el dispositivo es compatible con ARCore se puede visitar el siguiente enlace en el que aparece un listado con los modelos compatibles:

https://developers.google.com/ar/discover/supported-devices

Opciones de desarrollador

Una vez comprobado que nuestro dispositivo es compatible con las aplicación, debemos pasar a hacer ciertos ajustes en el panel de control.

Debemos activar las opciones de desarrolladores del móvil en cuestión. En cada dispositivo, según la marca y modelo, cambia la forma de hacerlo. Pero generalmente todas se realizan de la siguiente forma:

Ajustes >

Acerca del teléfono / Sobre el teléfono >

Tocar un número concreto de veces un determinado campo: Tocar 7 veces el campo Número de compilación en Samsung, tocar 7 veces en versión MIUI en Xiami, etc.

Depuración usb

A continuación hay que activar el USB Debug, o si el móvil lo muestra en español, la Depuración USB. En este apartado cada modelo de dispositivo es diferente, pero la mayoría suelen hacer aparecer una notificación de este apartado cuando se conecta desde el ordenador por Android Studio.

5. TUTORIAL PARTE 2: Aplicación base

En este apartado descargaremos la aplicación soporte y haremos un resumen de los fundamentos del lenguaje de programación en el que vamos a trabajar.

5.1. Descarga de la aplicación soporte

Ejecutaremos la aplicación de demostración.

Como en anteriores apartados he mencionado, utilizaremos las muestras de código que ofrece Android para realizar nuestra aplicación.

Dichas muestras las podemos encontrar en el siguiente enlace:

https://github.com/google-ar/sceneform-android-sdk/releases

Descargamos el paquete sceneform, y posteriormente extraemos los archivos de su interior. En dichos archivos habrá una carpeta llamada “samples” donde aparecerán las demos.

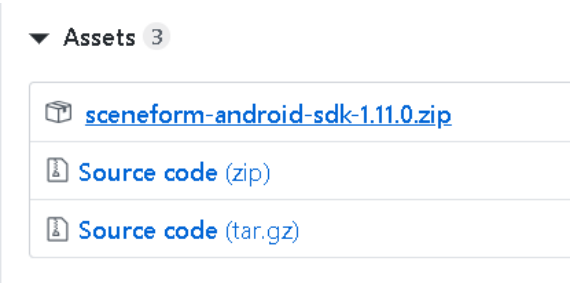


Figura 31. Paquete con las demos

De las demos disponibles, la que en principio nos interesa para el proyecto es la llamada “Hello Sceneform”. Posteriormente usaremos también el código de “Animation”.

Aquí estaremos utilizando un marco 3D llamado Sceneform con el que podemos crear aplicaciones con ARCore sin necesidad de OpenGL (que es un Interfaz de

Programación de Aplicaciones usada para escribir aplicaciones que produzcan gráficos tanto en 2D como en 3D, usándose para dibujar escenas tridimensionales complejas a partir de primitivas más simples).

Para probar dicha demo, deberemos tener abierto Android Studio y en el abrir el proyecto ejemplo “Hello Sceneform” que habremos descargado del anterior enlace, que estará ubicado en: sceneform-android-sdk/ directorio.

Una vez abierto el proyecto deberemos conectar nuestro dispositivo, ya que hemos decidido que utilizaremos uno físico y no un emulador para sacar mayor partido a la experiencia de realidad aumentada.

Si se han realizado todos los pasos puestos con anterioridad y tenemos el dispositivo conectado al ordenador podemos clicar en el botón de “Debug”.

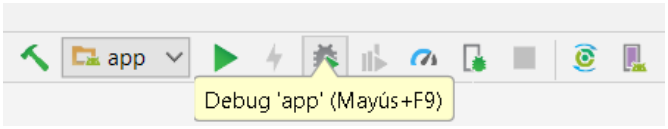


Figura 32. Botón Debug

Una vez hayamos clicado nos aparecerá una nueva pestaña con un listado con los dispositivos conectados. Seleccionamos el que vayamos a utilizar.

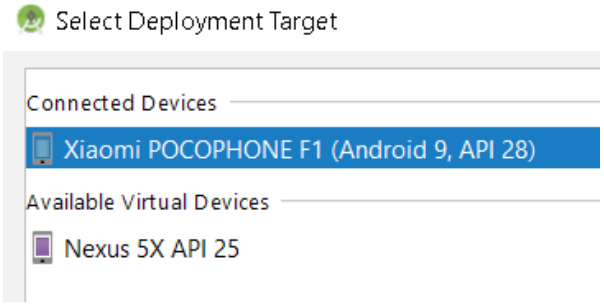


Figura 33. Listado de dispositivos disponibles

Después de tener conectado ya nuestro dispositivo podemos ya darle al botón “Run”, y se abrirá una página emergente para confirmar el dispositivo que vas a usar.

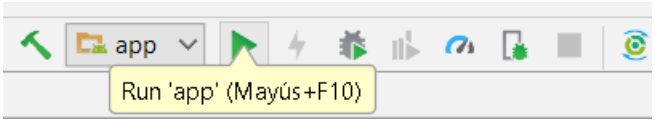


Figura 34. Botón Run

Luego Android Studio convertirá el proyecto en un APK y lo instalará en el dispositivo, abriéndose automáticamente en este una aplicación totalmente funcional.

Si no se llegara a abrir automáticamente o no se pudiese conectar el dispositivo, se podría guardar desde Android Studio el proyecto convertido en APK y pasarlo a este para posteriormente instalarlo.

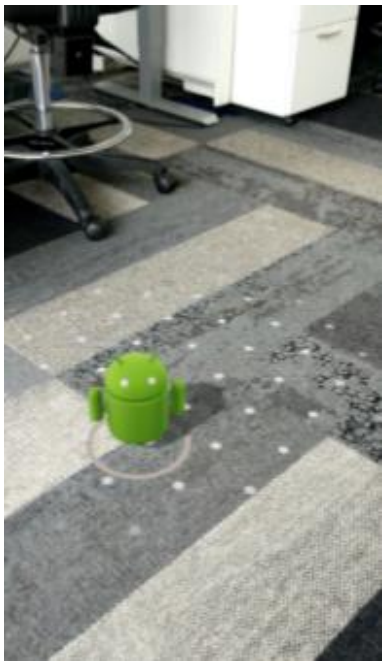


Figura 35. Visualización del ejemplo

Esta aplicación permite localizar las superficies planas gracias a ARCore, mostradas por una sucesión de puntos blancos, y colocar las figuras de demostración entre el el punto que haya tocado el usuario y el plano marcado, pudiendo también escalar, mover y rotar los modelos.

El código que utilizaremos está programado en Java. Este código lo da la empresa Android de demostración y como pauta de aprendizaje para usarlo de forma libre por los usuarios.

Como hemos visto en el apartado anterior, necesitaremos, si queremos modificar tanto el archivo obj como el código, tener descargado Android Studio.

5.2. Conceptos de programación

En este apartado intentaremos explicar algunos de los términos fundamentales en programación con la intención de sentar unas bases para lograr una mejor comprensión de los códigos escritos en Java. [21]

Concepto: Clase

Las clases se utilizan para definir las características genéricas o comportamientos de los objetos concretos que pertenezcan a dicha clase. Dicho de otra manera nos indica cómo es la estructura de un conjunto de datos. Ejemplo; en la clase guitarra podría tener características como el número de cuerdas.

Concepto: Objeto

Es las instancia de una clase. Son las características que distinguen un objeto del resto de ellos. Ejemplo; para las guitarras podría ser las características no genéricas, como puede ser el color, el modelo, etc.

Concepto: Variables

Cuando programamos tenemos que definir tipos de datos, ya que dependiendo de la naturaleza de estos datos necesitaremos definirlos de una forma o de otra. Para definirlos usaremos los siguiente tipos:

-boolean; nos devuelve True (1) o False (0). Ejemplo sintaxis:

```
boolean electrica=True;
```

-char; representa un único carácter. Ejemplo:

char marcaFender=”F”;

-int; para números enteros incluido el cero y negativos. Ejemplo:

int numeroCuerdas;

-float; para números con decimales. Ejemplo:

float frecuenciaMaxima;

-String; se usa para cadenas de caracteres. Los variables de los anteriores tipos son consideradas como datos primitivos, esta variable es de datos complejos. Ejemplo:

String nombreComprador = “Fernando García”

Concepto: Métodos

Son algoritmos asociados a objetos. Tras la recepción de un mensaje nos devuelve una acción, es decir genera una reacción que puede realizar cambios en las propiedades de los objetos o generar sucesos en el sistema. Los métodos podemos considerarlos reutilizables ya que los podemos llamar tantas veces como queramos sin tener que reescribirlo. Para declarar un método existen diferentes formas:

-public/private; definición del método. Si se define como private da solo acceso a dentro de la clase, es decir, no se podrá llamar desde fuera de la clase en donde esté declarado.

-dataTypeReturn; tipo de dato que devuelve el método. Podrán asignarse las variables que hemos visto con anterioridad, si el método no devuelve nada, será de tipo void.

-(); los paréntesis recogen la declaración de las variables de entrada al método.

-{}; dentro de los corchetes se encuentra el código del método. Aquí es donde se opera con las variables y se realizan las acciones necesarias del programa. Se finaliza con Return salvo en algunos casos como es el del dataTypeReturn.

Concepto: Estructuras condicionantes

Estas estructuras son las más usadas en la programación, a continuación veremos las que más se van a repetir a lo largo del código en el que trabajaremos.

-Condición ´if´; si se cumple la condición expuesta dentro de los paréntesis se activa la acción que hay dentro del bucle. Ejemplo sintaxis:

```
if(condición) {

}
```

-Condición ´if-else´; si no se cumple la condición del paréntesis del if, el programa pasará a leer el paréntesis del else. Si se cumple actuará de forma similar a la condición if anterior. Ejemplo sintaxis:

```
if(condición) {

}else {

{
```

-Bucle ´for´; hay varias condiciones que se encuentran dentro del paréntesis. La condición inicial nos indica lo que tiene que suceder para que se active el bucle. La condición de paso es la que permite que el bucle fluya. Y la condición final es la que hace que el bucle no se infinito. Ejemplo sintaxis:

```
for(condición inicial; condición final; condición de paso) {

{
```

-Bucle ´while´; el bucle ha de ser actualizado continuamente hasta que deje de cumplir alguna condición. Esta condición de paso se encuentra dentro de los corchetes y mientras se cumpla el bucle seguirá funcionando. Ejemplo sintaxis:

```
while(condición) {condición de paso}
```

-Bucle ´do-while´, con este bucle nos aseguramos que aunque la condición del bucle while no se cumpla, se activará por lo menos una vez lo que haya dentro de los corchetes. Ejemplo sintaxis:

```
do{

}while(condición)
```

5.3. Explicación del código base

Una vez entendidos los fundamentos del entorno de desarrollo Android Studio y comprendiendo el funcionamiento de la aplicación de demostración, pasaremos a ver cómo está programada dicha aplicación para que nos sea más fácil luego realizar las modificaciones necesarias para crear nuestra propia aplicación.

Como hemos mencionado en puntos anteriores, hay código en diferentes módulos y ventanas de Android Studio. Los más reseñables son el AndroidManifest.xml, el activityMain.java (en el caso del ejemplo llamado HelloSceneformActivity.java) y partes con código en el Gradle:app.

Como también hemos mencionado en puntos anteriores, el AndroidManifes.xml es donde se describe la naturaleza de la aplicación y sus componentes, es donde también se piden los permisos al dispositivo. En definitiva, define las características básicas y algunas funciones fundamentales de la aplicación. Más adelante aprenderemos a cambiar algunas de esas características como por ejemplo, el nombre de la aplicación. Ejemplo de sintaxis de la línea donde se cambia el nombre de la aplicación:

```
android:label="nombre de la aplicación"
```

En el código del Gradle:app aparecen las declaraciones de diferentes plugins de Android para el correcto funcionamiento de la aplicación y de la compilación. Cuando compilamos el código al haber importado o cambiado un modelo 3D, la parte final de este código se verá modificada, ya que en él se establecen las rutas de la búsqueda de los diferentes archivos .obj y .fbx y los creados a partir de la sincronización del Gradle como es el .sfa o el .sfb. Este es un ejemplo de la sintaxis de la parte del código que cambia al modificar los archivos y arrancar o compilar la aplicación:

```
sceneform.asset('sampledata/models/nuestroModelo.obj',
        'default',
        'sampledata/models/nuestroModelo.sfa',
        'src/main/res/raw/nuestroModelo')
```

Sin embargo, estas partes de la programación del programa son necesarias aunque secundarias, ya que salvo los ejemplos anteriores y en alguna línea más, son partes que no se modificarán ya que son iguales para cualquier aplicación de este tipo.

El grueso de la programación de la aplicación radica en el Activity Main (HelloSceneform). Por ello, nos vamos a detener más para analizar de qué se encargan las diferentes partes del código.

Iremos dividiendo en esas pequeñas partes el código y explicando la finalidad de cada una de ellas además de algún concepto de programación a mayores de los explicados en el apartado anterior para mejorar la legibilidad de comentarlo por completo. Si se prefiere ver el código entero sin explicación, está documentado en el apartado 7.5.

En la programación de Android Studio cuando colocamos dos barras inclinadas (//) lo siguiente a estas lo toma como una anotación o comentario y no tiene repercusión alguna en la compilación del código. Utilizaremos este método para señalizar con números (//1*, //2*…) las líneas que queramos comentar con mayor profundidad y así ubicarlas con mayor rapidez.

En esta primera línea de código nos indica la ruta del paquete. Como podemos ver en el entorno de desarrollo, las palabras separadas por un punto son las diferentes carpetas que están ubicadas dentro de la carpeta general java de la venta de Proyecto. Veremos más adelante como cambiar las direcciones de este paquete para poder tener varias aplicaciones en el dispositivo partiendo de la modificación del mismo código.

```
package com.google.ar.sceneform.samples.hellosceneform;
```

En todo código hay que importar diferentes bibliotecas. En este código podemos ver una serie de bibliotecas con diferentes funciones.

Línea 1* y su consecutiva: son bibliotecas necesarias para hacer aplicaciones Android, los activity son fundamentales en su programación.

Línea 2*: para poder acceder al manifiesto utiliza esta biblioteca.

Línea 3* y consecutiva: servicios de sistema operativo, comunicación de procesos.

Línea 4*: empleada para comunicarse entre actividades.

Línea 5*: para poder usar un actionBar. El actionBar se trata de un menú auxiliar de las aplicaciones Android que se ubica en la parte superior de cada actividad.

Línea 6*: esto activa los mensajes de error (Log) que explicamos en el apartado anterior.

Línea 7*: herramienta para colocar un objeto dentro de un contenedor. Usa Gravity.Center para mandar que lo ponga en el centro de este.

Línea 8*: reporta si hay alguna actividad del usuario.

Línea 9*: para informar a través de mensajes en la pantalla al usuario.

A partir de esta línea las bibliotecas son específicas de la realidad aumentada.

Línea 10*: fija orientación y localización en el mundo real a través de un ancla.

Línea 11*: define la intersección del rayo que se proyecta por el toque del usuario en la pantalla y el plano creado por el programa.

Línea 12*: define el plano que mejor se ajuste al mundo real.

Línea 13*: posiciona un nodo (Node) en un ancla (Anchor).

Línea 14*: renderiza un modelo 3D colocándolo en un nodo.

Línea 15*: un fragmento es una parte de la actividad. En nuestro caso el fragmento donde se muestra la realidad aumentada.

Línea 16*: nodo que se puede modificar.

```
import android.app.Activity;                                    //1*
import android.app.ActivityManager;
import android.content.Context;                                 //2*
import android.os.Build;                                        //3*
import android.os.Build.VERSION_CODES;
import android.os.Bundle;                                       //4*
import android.support.v7.app.AppCompatActivity;                //5*
import android.util.Log;                                        //6*
import android.view.Gravity;                                    //7*
import android.view.MotionEvent;                                //8*
import android.widget.Toast;                                    //9*
import com.google.ar.core.Anchor;                               //10*
import com.google.ar.core.HitResult;                            //11*
import com.google.ar.core.Plane;                                //12*
import com.google.ar.sceneform.AnchorNode;                      //13*
import com.google.ar.sceneform.rendering.ModelRenderable;//14*
import com.google.ar.sceneform.ux.ArFragment;                   //15*
import com.google.ar.sceneform.ux.TransformableNode;            //16*
```

En esta parte del código se definen las clases. “private static final” es una variable final, es decir, no puede cambiar su valor una vez asignada. Este tipo de variables es una constante de tiempo de compilación. También hay dos variables “private”, en la línea 1* definimos el fragmento de la aplicación donde ejecutar la realidad aumentada, mientras que en la línea 2* se define la variable del modelo 3D que se va a renderizar.

```
public class HelloSceneformActivity extends AppCompatActivity {
  private static final String TAG = HelloSceneformActivity.class.getSimpleName();
  private static final double MIN_OPENGL_VERSION = 3.0;

  private ArFragment arFragment;                                    //1*
  private ModelRenderable andyRenderable;                           //2*
```

El Override permite modificar un método ya existente. Y el SupressWarnings indica que las advertencias del compilador deben suprimirse siempre que se traten de los elementos anotados (también se suprimirán los elementos del programa contenidos en el elemento anotado).

```
@Override
@SuppressWarnings({"AndroidApiChecker", "FutureReturnValueIgnored"})
```

La forma usual de creación de una actividad es como se muestra en la línea 1*, ejecutándose el método original en la línea 2*. La línea 3* busca y comprueba si hay algún error en la actividad.

```
protected void onCreate(Bundle savedInstanceState) {              //1*
    super.onCreate(savedInstanceState);                           //2*

    if (!checkIsSupportedDeviceOrFinish(this)) {                  //3*
      return;
    }
```

La actividad es una ventana vacía en la cual iremos a partir de la línea 1* metiendo información. Al decir en esta línea "(R.layout.activity_ux)" le hemos introducido un requisito para usar realidad aumentada. En la línea 2* creamos un fragmento para meter en la actividad una vista en realidad aumentada. Cuando crea un Renderable, Sceneform carga sus recursos en segundo plano mientras regresa. El formato a partir de la línea 3* es un design pattern inventado por Android, en este punto construye el objeto 3D que va a renderizar.

```
setContentView(R.layout.activity_ux);                                    //1*
arFragment = (ArFragment)getSupportFragmentManager().findFragmentById
    (R.id.ux_fragment);                                                  //2*

ModelRenderable.builder()                                                //3*
    .setSource(this, R.raw.andy)
    .build()
    .thenAccept(renderable -> andyRenderable = renderable)
    .exceptionally(
        throwable -> {
          Toast toast =
              Toast.makeText(this, "Unable to load andy renderable",
                   Toast.LENGTH_LONG);
          toast.setGravity(Gravity.CENTER, 0, 0);
          toast.show();
          return null;
        });
```

Esta parte del código se ejecuta con la interacción del usuario, cuando este toca la pantalla (hace tap). "HitResult hitResult, Plane plane, MotionEvent motionEvent" son los parámetros que se envían a la función.

```
arFragment.setOnTapArPlaneListener(
    (HitResult hitResult, Plane plane, MotionEvent motionEvent) -> {
      if (andyRenderable == null) {
          return;
      }
```

En esta parte del código se crea el ancla (Anchor). En la línea 1* se indican las coordenadas del mundo real, que son la intersección de la trayectoria del "Tap" del usuario en la pantalla con el plano reconocido por el programa. En la línea 2* creamos el nodo, que es el componente gráfico que se puede manipular, colocado en el ancla. En la línea 3* el nodo cambiará de padre, que será la propia escena.

```
            Anchor anchor = hitResult.createAnchor();                          //1*
            AnchorNode anchorNode = new AnchorNode(anchor);                    //2*
            anchorNode.setParent(arFragment.getArSceneView().getScene());      //3*
```

En este punto vamos a crear el transformable y añadirlo al ancla creada con anterioridad. En la línea 1* el TransformableNode coge el sistema de transformación general. En la línea 2* se hace hijo con el ancla, es decir, el sitio de la intersección de la interacción del usuario con el plano calculado. En la línea 3* se indica que renderice en ese punto al modelo. Y por último en la línea 4* lo selecciona para que el usuario pueda interactuar con él.

```
            TransformableNode andy = new
                TransformableNode(arFragment.getTransformationSystem());       //1*
            andy.setParent(anchorNode);                                        //2*
            andy.setRenderable(andyRenderable);                                //3*
            andy.select();                                                     //4*
          });
}
```

El código restante es para comprobar que todo funciona correctamente utilizando mensajes internos (Log) y mensajes externos al usuario (Toast). Al ser una variable boolean devuelve un False y muestra un mensaje de error si el Sceneform no puede ejecutarse, y un verdadero si Sceneform puede ejecutarse en el dispositivo.

```
public static boolean checkIsSupportedDeviceOrFinish(final Activity activity) {
    if (Build.VERSION.SDK_INT < VERSION_CODES.N) {
      Log.e(TAG, "Sceneform requires Android N or later");
      Toast.makeText(activity, "Sceneform requires Android N or later",
          Toast.LENGTH_LONG).show();
      activity.finish();
      return false;
    }

    String openGlVersionString =
        ((ActivityManager)
            activity.getSystemService(Context.ACTIVITY_SERVICE))
            .getDeviceConfigurationInfo()
            .getGlEsVersion();
     if (Double.parseDouble(openGlVersionString) < MIN_OPENGL_VERSION) {
         Log.e(TAG, "Sceneform requires OpenGL ES 3.0 later");
         Toast.makeText(activity, "Sceneform requires OpenGL ES 3.0 or later",
             Toast.LENGTH_LONG)
             .show();
         activity.finish();
         return false;
     }
     return true;
  }
}
```

6. TUTORIAL PARTE 3: Inclusión de modelos 3D propios

Veremos a continuación los diferentes pasos en las posibles aplicaciones que podremos programar en realidad aumentada para el uso de nuestros propios modelos 3D.

6.1. Importar y modificar archivos 3D en una aplicación

Para crear ahora la aplicación personalizada que queremos, tendremos que aprender a introducir en la carpeta sampledata de nuestro proyecto nuestro archivo 3D.

Primero en el caso de tener, como en el ejemplo, un archivo ya predeterminado eliminaremos toda esta carpeta sampledata para posteriormente crear una e importar en esta nuestro archivo. Si dejamos la antigua funcionará correctamente ya que haremos las modificaciones necesarias en el código, sin embargo recomiendo eliminar todas las dependencias de la misma por mayor limpieza y facilidad para evitar un equívoco.

Podemos usar en Sceneform archivos 3D en los formatos OBJ, gITF y FBX. Para importar el archivo 3D primero hemos de cerciorarnos que en las carpetas de nuestra aplicación tengamos creada una carpeta *Sampledata.*

Para ello, deberemos ir a la carpeta *App* en nuestro proyecto en Android Studio y ver si está dicha carpeta creada en su interior. Si no está creada podemos hacerlo dándole botón derecho a la carpeta “app”, luego clicando en “New” aparecerá la opción de crear “Sample Data Directory”.

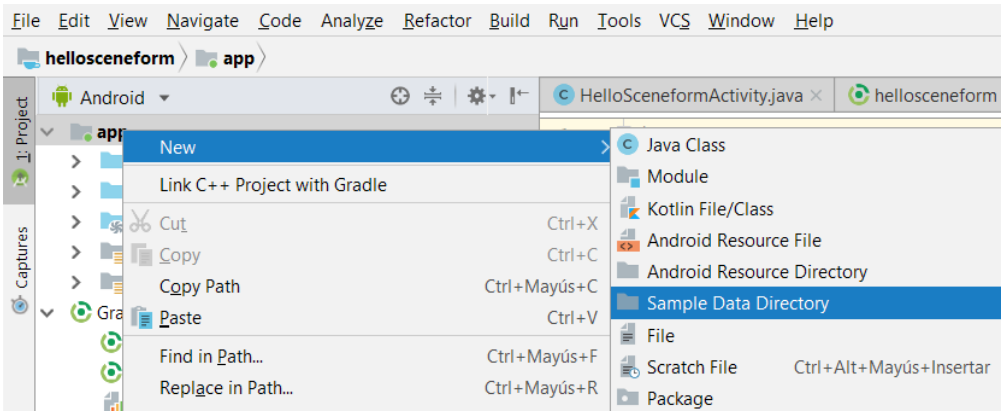


Figura 36. Sample Data Directory

El contenido que introduzcamos en esta carpeta sampledata formará parte del proyecto en Android Studio pero no se incluirá en la APK. Para evitar que se incluyan innecesariamente en la APK no se copiarán los archivos fuente en ninguna carpeta que no sea la indicada.

En las figuras de este apartado importaremos un archivo denominado “Letras” (este no es más que un archivo para el ejemplo). Recordar siempre que en Android Studio es mejor siempre trabajar con letras minúsculas y con números, ya que el resto de símbolos y valores suelen dar problemas en la compilación.

La carpeta indicada para copiar los archivos (.obj, .glbt, .fbx) y sus respectivas dependencias, si las tuviera (.mtl, .bin, .png, .jpg, etc), será la carpeta Sampledata.

Una vez copiada en ella el archivo 3D a visualizar, haremos click derecho encima del objeto que acabamos de copiar y seleccionaremos “Import Sceneform Assets” para comenzar el proceso de importación.

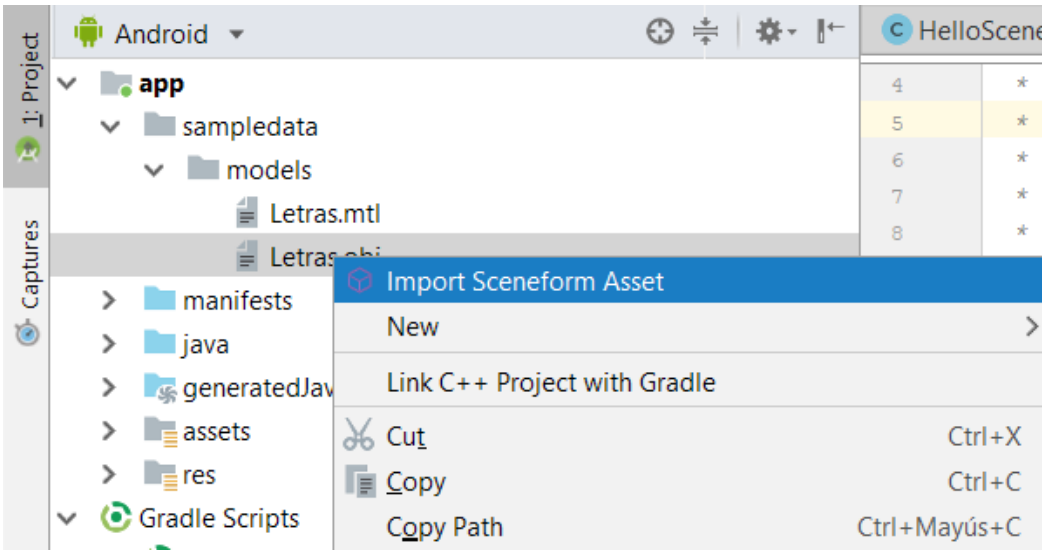


Figura 37. Import Sceneform Asset

Aparecerá una ventana emergente en la cual podremos seleccionar donde queremos que se generen los archivos .sfa y .sfb. Dejaremos estos valores en predeterminado, siendo para el .sfa la carpeta “sampledata/models/” y para el .sfb la carpeta “src/main/assets/”. La modificación de estas rutas es totalmente una decisión personal, ya que en lo único que se verá afectado es en la modificación del código.

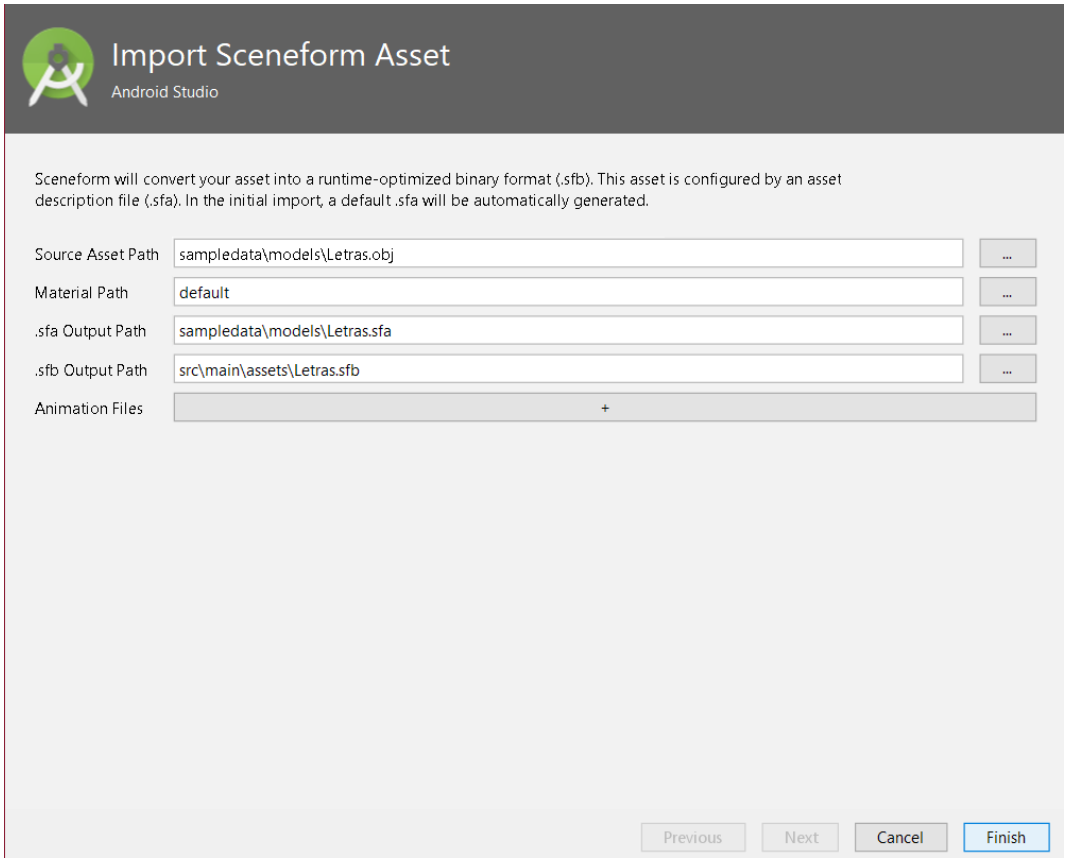


Figura 38. Rutas para la generación del .sfa y .sfb

Seleccionaremos el botón de Finish para comenzar la importación y creación de estos archivos. Se generarán los archivos .sfa y .sfb y el código en el Gradle.

Podremos comprobar en el código creado en el Gradle en la ventana “app” si se ha generado el código con las rutas de las carpetas correctamente.

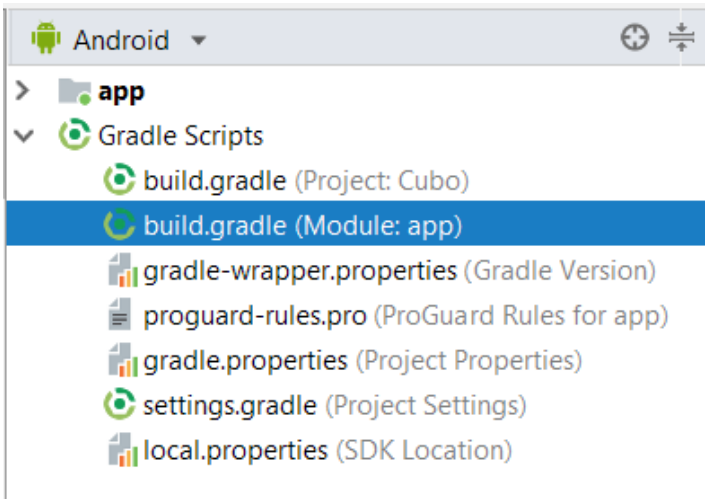


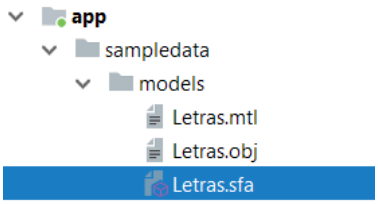
Figura 39. Acceso al Gradle en el buscador con doble clic a la ventana app

```
53      implementation "com.android.support:appcompat-v7:28.0.0"
54  }
55
56  apply plugin: 'com.google.ar.sceneform.plugin'
57
58  sceneform.asset('sampledata/models/andy.obj',
59          'default',
60          'sampledata/models/andy.sfa',
61          'src/main/res/raw/andy')
62
63
64
65  sceneform.asset('sampledata/models/Letras.obj',
66          'default',
67          'sampledata/models/Letras.sfa',
68          'src/main/assets/Letras')
```

Figura 40. Código resultante de la importación de archivos

Además podremos ver el código del antiguo modelo de ejemplo de la aplicación encima del nuevo, que podemos eliminar si deseamos.

Podemos comprobar si los archivos .sfa y .sfb se han generado correctamente fijándonos en sus rutas y buscándolos en el directorio de carpetas de Android Studio.




```
sceneform.asset('sampledata/models/Letras.obj',
        'default',
        'sampledata/models/Letras.sfa',
        'src/main/assets/Letras')
```

Figuras 41 y 42. Archivo .sfa creado correctamente en la carpeta indicada en el código

En el caso de que no se hayan creado en la carpeta adecuada se puede repetir todo el proceso o modificar manualmente. Para ello, sería dirigirse a las carpetas y mover el archivo de una a otra. Por si el usuario no tiene ubicados las carpetas puede hacer clic derecho encima del archivo que quiera mover y seleccionar a "Show in Explorer" abriéndole así directamente la carpeta.

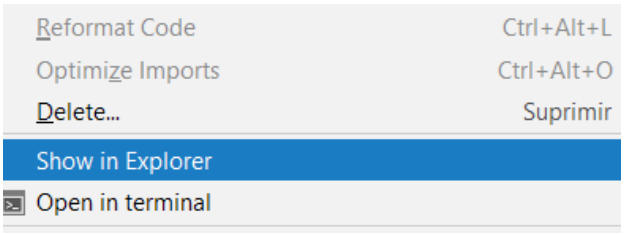


Figura 43. Show in Explorer

Posteriormente puede cortar el archivo (en este caso el .sfb) y pegarlo en la carpeta que haya escrito en el código.


```
sceneform.asset('sampledata/models/Letras.obj',
        'default',
        'sampledata/models/Letras.sfa',
        'src/main/assets/Letras')
```

Figuras 44 y 45. Archivo .sfb creado correctamente en la carpeta indicada en el código

Una vez creados los archivos .sfa y .sfb podremos abrir, haciendo doble clic en uno de ellos, una nueva ventana con las características del modelo, donde podemos modificar fácilmente en el código que aparece en dicha ventana.

Podemos cambiar diferentes características del material del modelo, ya sea el color (remarcado de la línea 10 a la 14 en la Figura 46) como la opacidad, la textura metálica o rugosa entre otras.

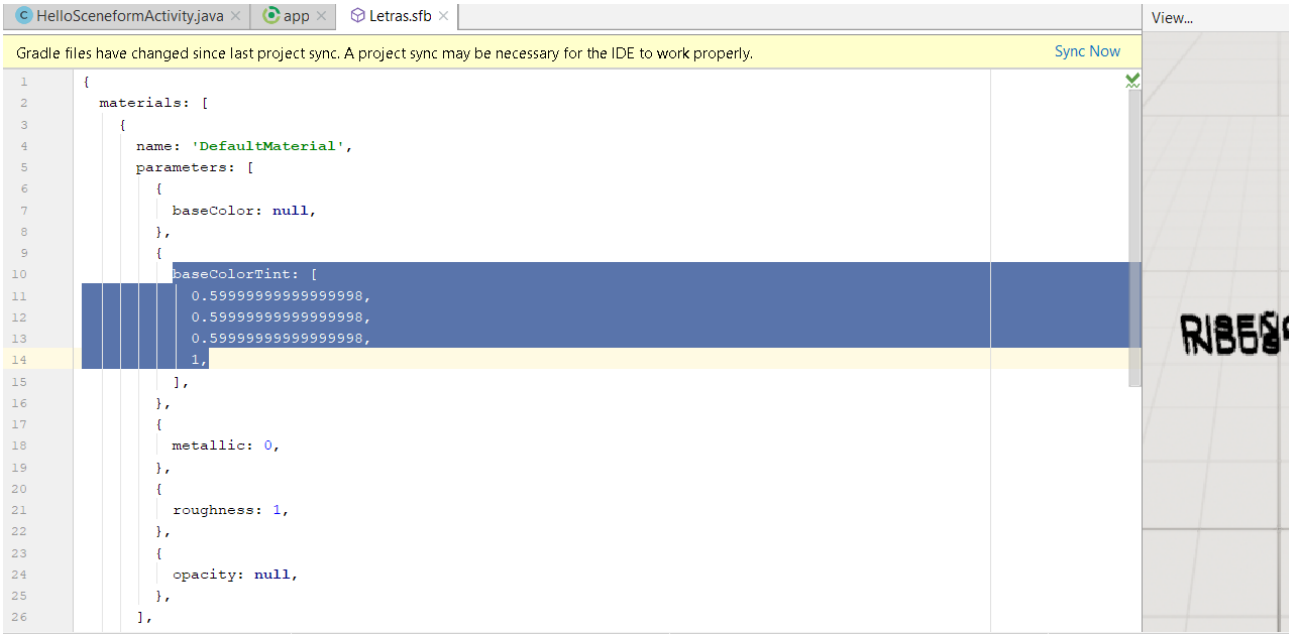


Figura 46. Características del modelo y visualizador del mismo

Todas las características son manipulables cambiando su valor.

Quizá la característica a modificar que más nos interesa sea la Escala, ya que Android Studio puede llegar a modificar la escala real del modelo dependiendo del Software con el que haya sido hecho o en el formato en el que esté.

Sin embargo, modificar la escala a nuestro gusto no es problema ya que solo es cambiar el número que va a continuación como el resto de características (Línea 39 de la Figura 47).

A veces, dependiendo del modelo, esta línea no aparece automáticamente, por lo que nos tocará añadirla en dichos casos. Se añade el término “scale:” seguido del número que marcará el tamaño que queramos para nuestro modelo tal como se ve en la Figura 47.

Figura 47. Resto de características del modelo

Una vez creado los archivos .sfa y .sfb tendremos que volver al código del MainActivity para sustituir el nombre del viejo modelo por el nuevo.

```
private ArFragment arFragment;
private ModelRenderable andyRenderable;
```

Figura 48. Nombre a cambiar en la definición de las clases

```
ModelRenderable.builder()
    .setSource(this, R.raw.andy)
    .build()
    .thenAccept(renderable -> andyRenderable = renderable)
    .exceptionally(
        throwable -> {
          Toast toast =
              Toast.makeText( context: this,  text: "Unable to load andy renderable", Toast.LENGTH_LONG);
          toast.setGravity(Gravity.CENTER,  xOffset: 0,  yOffset: 0);
          toast.show();
          return null;
        });
```

Figura 49. Nombre a cambiar en el métido ModelRenderable

```
arFragment.setOnTapArPlaneListener(
    (HitResult hitResult, Plane plane, MotionEvent motionEvent) -> {
      if (andyRenderable == null) {
        return;
      }

      // Create the Anchor.
      Anchor anchor = hitResult.createAnchor();
      AnchorNode anchorNode = new AnchorNode(anchor);
      anchorNode.setParent(arFragment.getArSceneView().getScene());

      // Create the transformable andy and add it to the anchor.
      TransformableNode andy = new TransformableNode(arFragment.getTransformationSystem());
      andy.setParent(anchorNode);
      andy.setRenderable(andyRenderable);
      andy.select();
    });
```

Figura 50. Nombre a cambiar en el método arFragment

Como podemos ver en la Figura 49, el método ModelRenderable utiliza unos comandos R.raw.nombre del archivo. Esto se puede utilizar siempre y cuando el .sfb esté en la carpeta raw.

Para ello habría dos opciones, cambiar la ruta del .sfb, o dejar el archivo en la carpeta assets o la elegida por el usuario y modificar el código como se hace en el apartado 7.2.

La primera opción en este caso sería la más rápida, ya que sería cambiar en la ventana de la Figura 52 la ruta de la salida del .sfb o a posteriori modificar el código de la Figura 51 con la nueva ruta: *src/main/res/raw/* como se muestra en dicha la Figura.

```
sceneform.asset('sampledata/models/letras.obj',
        'default',
        'sampledata/models/andy.sfa',
        'src/main/res/raw/letras')
```

Figura 51. Cambio de la ruta del archivo .sfb modificando el código de la ventana del Gradle: app

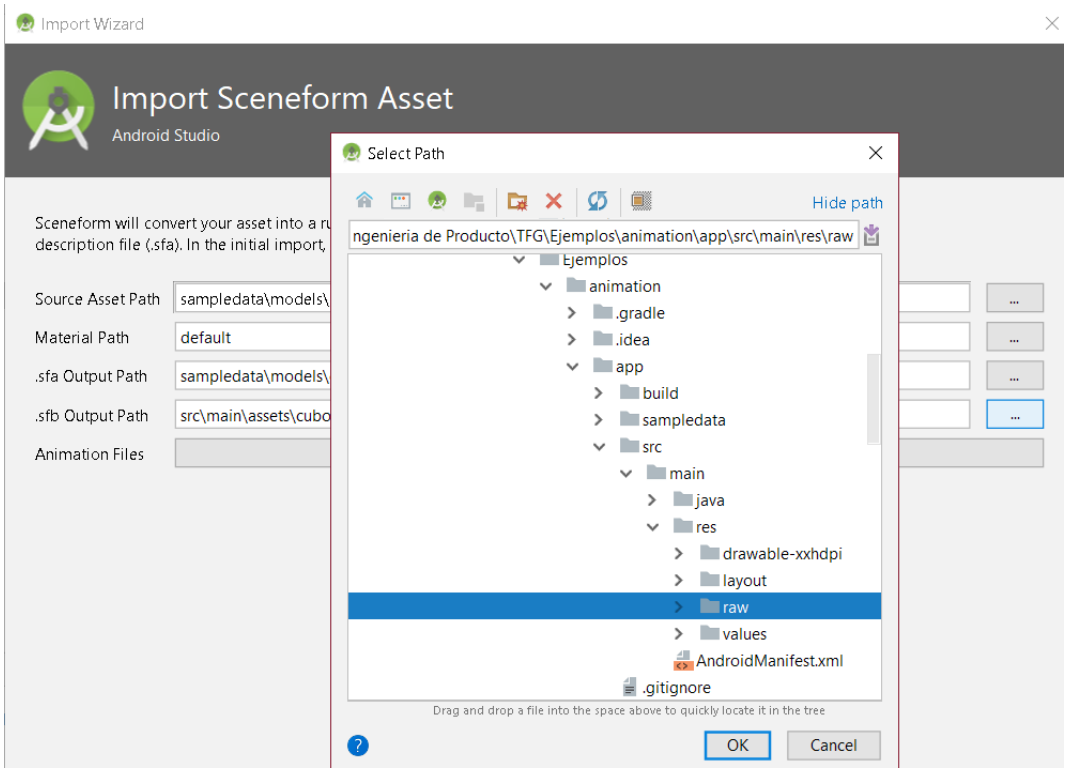


Figura 52. Cambio de ruta del archivo .sfb en la pestaña de Import Sceneform Asset

Por último habrá que sustituir los nombres de las clases antiguas por las nuevas tanto donde se definen como en el método ModelRenderable. Para que se muestren como en la Figura 53 y 54.

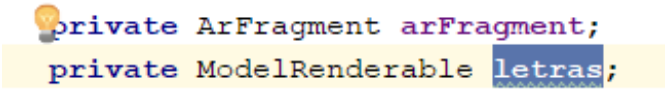


Figura 53. Cambio de nombre en la definición de clases

```
ModelRenderable.builder()
    .setSource(this, R.raw.letras)
    .build()
    .thenAccept(renderable -> letras = renderable)
    .exceptionally(
        throwable -> {
          Toast toast =
              Toast.makeText( context: this,  text: "Unable to load andy renderable", Toast.LENGTH_LONG);
          toast.setGravity(Gravity.CENTER,  xOffset: 0,  yOffset: 0);
          toast.show();
          return null;
        });
```

Figura 54. Cambio de las antiguas clases por las nuevas

Comprobadas las rutas, los códigos y las características del modelo podremos arrancar la aplicación.

Para ello deberemos de cerciorarnos de tener conectado el dispositivo y que haya sido reconocido por el software (el dispositivo tiene que tener los requisitos especificados en el apartado 4.1 de este documento). Una vez hecho esto, clicamos en el botón Run (Figura 3) y la aplicación arrancará automáticamente.

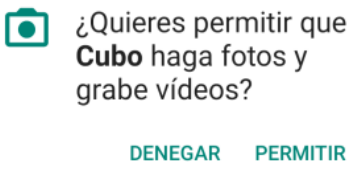


Figura 55. Permisos de la aplicación "Cubo"

Después de pedir los permisos necesarios al usuario ya podrá probar la aplicación.

El sistema ARCore reconoce los planos a los que apunta con la cámara el usuario. Este solo tiene que apuntar con la cámara en una dirección y balancear suavemente el dispositivo para que aparezcan unos pequeños círculos blancos que definen el plano y si el usuario desea usar dicho plano tocar en cualquier parte de este para que aparezca en ese mismo punto el modelo 3D que hemos introducido.

Este código además permite la modificación parcial del tamaño y el movimiento y giro total del modelo por el plano.

Figuras 56. Ejemplo de visualización en Smartphone de un modelo 3D en formato obj

A continuación veremos qué líneas añadir al código para poder poner varios modelos 3D en una misma aplicación.

6.2. Aplicación con varios modelos 3D

Para tener en la misma aplicación la posibilidad de ver varios modelos 3D tendremos que realizar el procedimiento de importar archivos que hemos visto en el apartado anterior con todos los modelos.

Importados los modelos añadiremos la biblioteca a mayores de las del código: *android.net.Uri*

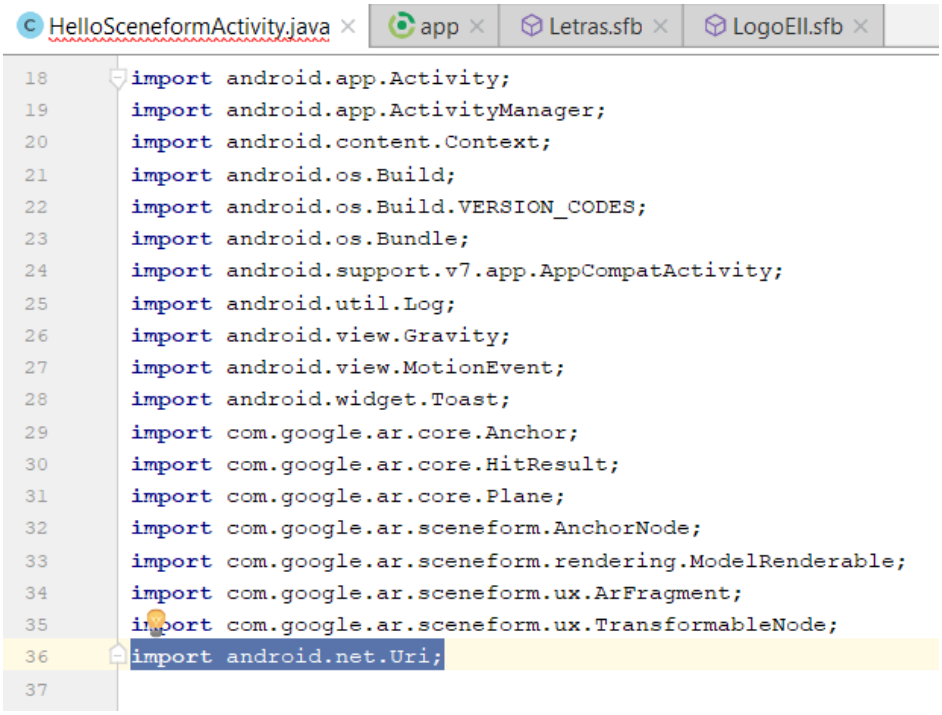


Figura 57. Bibliotecas necesarias del código

Definiremos ahora las diferentes variables (eliminando las antiguas). En el caso del ejemplo se está realizando con los modelos: “letras” y “logo” (estos archivos al igual que los utilizados en los ejemplos anteriores son para introducirnos de una manera más sencilla en el ejemplo y entender donde usar o modificar los nombres de nuestros archivos en el código).

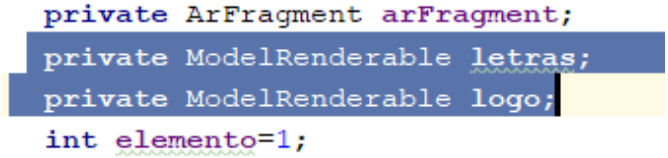


Figura 58. Definir clases

Podemos aprovechar y crear además una nueva clase int, llamada de la forma que más le convenga al programador de valor “1” (ver Figura 58). Usaremos más adelante esta clase para un contador, con el que lograremos que aparezcan los diferentes modelos.

Ahora nos iremos al método ModelRenderable para cambiar el código.

Lo primero es sustituir por: *Uri.parse (“nombre del archivo.sfb”)* el comando “R.raw.andy”. Este comando busca el archivo en la carpeta Raw, por lo que sería buen momento para comprobar si el archivo .sfb creado con la importación de nuestro modelo está colocado en la carpeta correspondiente, es decir la que haga referencia el código de la ventana app.

```
ModelRenderable.builder()
    .setSource(this, Uri.parse ("Letras.sfb"))
    .build()
    .thenAccept(renderable -> andyRenderable = renderable)
    .exceptionally(
        throwable -> {
          Toast toast =
              Toast.makeText( context: this,  text: "Unable to load andy renderable", Toas
          toast.setGravity(Gravity.CENTER,  xOffset: 0,  yOffset: 0);
          toast.show();
          return null;
        });
```

Figura 59. Ejemplo de uso del Uri.parse

Nos tenemos que cerciorar de que dentro de dicho comando ModelRenderable, no quede mencionada ninguna de las antiguas clases del ejemplo del código viejo. En el caso de la Figura 59 podemos ver que aún quedaba “andyRenderable”. Tendremos que cambiar esta por uno de las dos clases creadas anteriormente en el caso de la Figura 60, “letras”.

```
ModelRenderable.builder()
    .setSource(this, Uri.parse ("Letras.sfb"))
    .build()
    .thenAccept(renderable -> letras = renderable)
    .exceptionally(
        throwable -> {
          Toast toast =
              Toast.makeText( context: this,  text: "Unable to load andy renderable", Toast.
          toast.setGravity(Gravity.CENTER,  xOffset: 0,  yOffset: 0);
          toast.show();
          return null;
        });
```

Figura 60. Modificación del nombre de las clases viejas por las nuevas

Estas líneas de código tienen que repetirse para todos los modelos que se quieran usar.

El comando arFrangment es el encargado de transformar el nodo que ha dado el usuario al tocar el dispositivo en el punto donde se coloca el modelo 3D. Por ello, nos iremos a dicha parte del código para añadir unas líneas (Figura 61) que hagan que en el primer toque salga uno de los modelos y si el usuario da otro toque salga el siguiente modelo.

```
TransformableNode andy = new TransformableNode(arFragment.getTransformati
andy.setParent(anchorNode);
  if (elemento%2==0) {
      andy.setRenderable(letras);
  }else{
      andy.setRenderable(logo);
  }
  elemento++;
andy.select();
```

Figura 61. Líneas de código nuevo

Las nuevas líneas se encargan de ir aumentando con cada toque del usuario el valor de la clase “elemento” que habíamos creado anteriormente, y divide dicho valor entre dos sabiendo así cuál de los dos modelos es el primero y cuál el

segundo y a partir de este punto si sigue añadiendo saldrán alternativamente hasta que decida el usuario de la aplicación.

Hay que recordar que aquí también hay que cambiar las clases del antiguo código y sustituirlas por las nuevas.

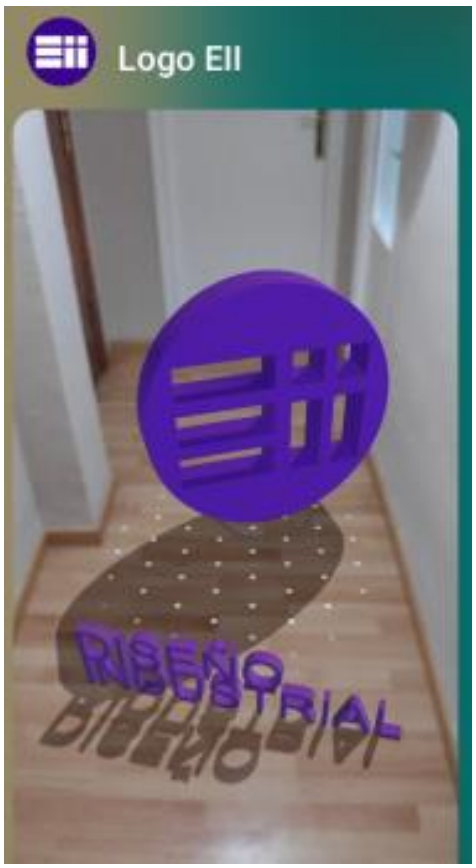
```
arFragment.setOnTapArPlaneListener(
  (HitResult hitResult, Plane plane, MotionEvent motionEvent) -> {
    if (letras == null) {
      return;
    }

    // Create the Anchor.
    Anchor anchor = hitResult.createAnchor();
    AnchorNode anchorNode = new AnchorNode(anchor);
    anchorNode.setParent(arFragment.getArSceneView().getScene());

    // Create the transformable andy and add it to the anchor.
    TransformableNode andy = new TransformableNode(arFragment.getTransformationSystem());
    andy.setParent(anchorNode);
      if (elemento%2==0) {
          andy.setRenderable(letras);
      }else{
          andy.setRenderable(logo);
      }
      elemento++;
    andy.select();
  });
```

Figura 62. Código arFragment totalmente modificado

Una vez realizado este código ya se podrá arrancar la aplicación para probar los cambios efectuados.



Figuras 63. Ejemplo de visualización en Smartphone de varios modelo 3D

Ya que hemos visto cómo podemos tener diferentes moldeos 3D en una misma aplicación, ahora nos dispondremos a programar una aplicación donde consigamos una animación 3D.

6.3. Aplicación de un modelo 3D con animaciones

Lo primero que necesitaremos es un modelo animado (un modelo 3D con una animación incluida en el formato), el cual podemos crear en los diferentes programas en los que trabaje el usuario. En nuestro caso trabajaremos con AutoDesk 3ds Max, ya que exporta en formato FBX que es el que menos problemas al importar archivos en Android Studio.

Podemos crear el modelo 3D en el propio programa o introducirlo si ya lo tenemos creado de otro software. En el ejemplo crearemos un cubo y sobre él añadiremos tres diferentes animaciones.

Una vez introducido o creado el modelo que queramos animar estamos en disposición de realizar la animación que deseemos. En estos programas tipo 3D Studio controlaremos el tiempo de la animación con la línea de tiempo, que en el caso de este software está en la parte inferior de la pantalla.

Figura 64. Línea de tiempo en 3D Max

Aquí nosotros tendremos que realizar conscientemente todas las diferentes animaciones que queramos que realice nuestro modelo, tomando nota de la referencia de cuando empieza y termina cada una de ellas.

Con el modelo ya animado iremos a “Export” y utilizaremos la herramienta “Game Exporter”.

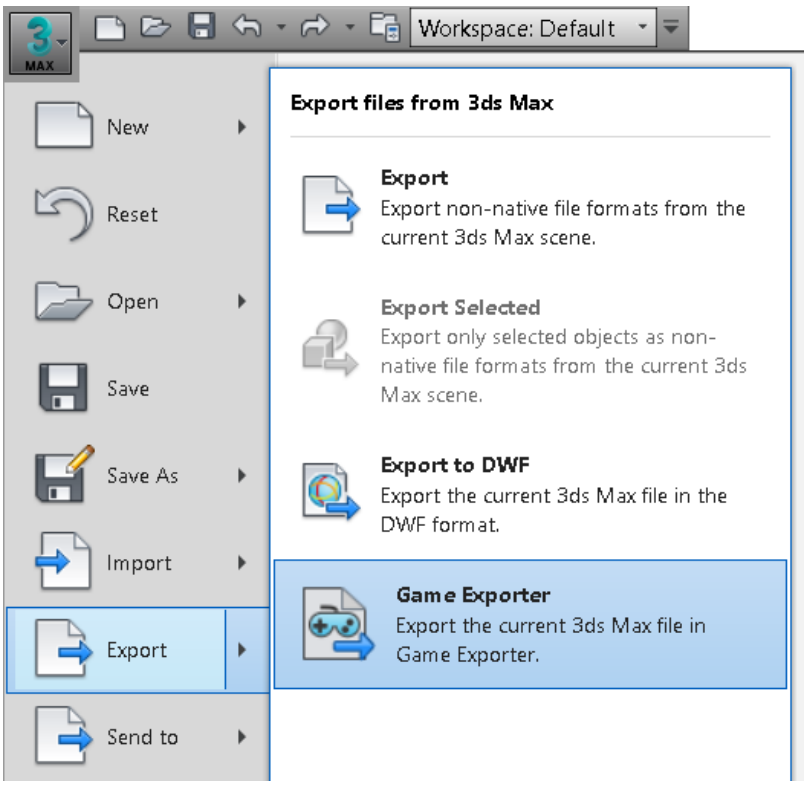


Figura 65. Panel donde aparece Game Exporter

Esta herramienta es la que nos permite dividir la animación completa que aparece en la línea de tiempo en el número de fragmentos que nosotros queramos.

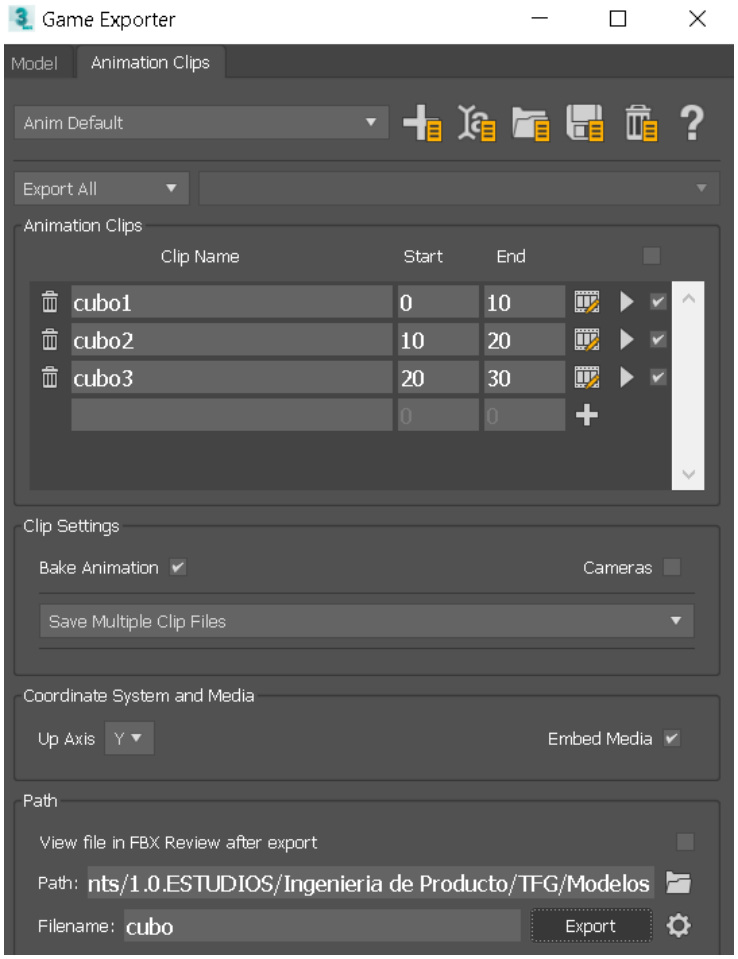


Figura 66. Game Exporter

Como podemos ver en la Figura 66 añadiremos en el apartado Animation Clips tantas divisiones de la animación completa como animaciones individuales queramos tener, solo tendremos que marcar el tiempo en el que empieza y termina en la línea de tiempo.

En el apartado Clip Setting deberemos poner la pestaña en “Save Multiple Clip Files” para que se guarden en diferentes archivos cada una de las animaciones. Y ya podríamos exportar los archivos en el formato FBX.

Iremos a la carpeta en la que hayamos exportado los archivos y comprobaremos que están todos. En el caso del ejemplo “cubo” tendría que haber tres archivos: cubocubo1.fbx, cubocubo2.fbx y cubocubo3.fbx. Esta denominación surge de la unión de nombre del proyecto “cubo” al nombre que le hayamos asignado a las animaciones, en este caso “cubo1, cubo2, cubo3”.

Moveremos estos archivos a la carpeta Sampledata como hemos hecho en los puntos anteriores. Ahora importaremos solo el primero de los archivos, ya que Android Studio extraerá el modelo 3D de este y en la pestaña de importación añadiremos las animaciones.

Para abrir la pestaña de importación tenemos que hacer clic derecho en el primero de los archivos y darle a “Import Sceneform Assest”.

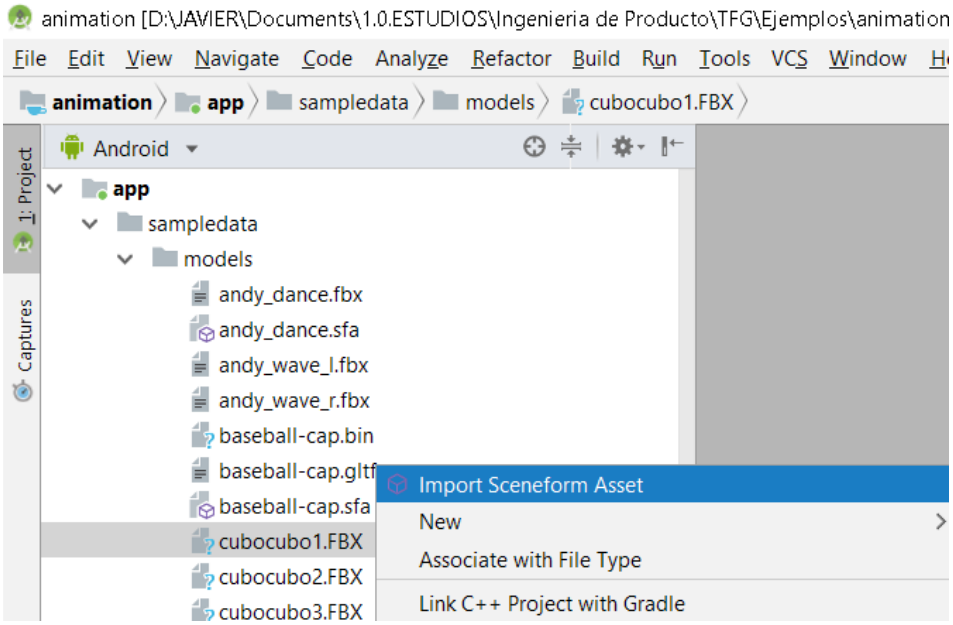


Figura 67. Panel donde se encuentra Import Sceneform Assest

Aparecerá de nuevo la misma pestaña que en el apartado anterior cuando importábamos un modelo 3D, sin embargo ahora tendremos que añadir las animaciones.

Para ello tendremos que clicar en el símbolo “+” en el último apartado de la pestaña “Animation Files”. Se abrirá entonces un espacio en blanco en el cual al clicar en el símbolo “…” te permitirá buscar los archivos de las animaciones, que tendrás en el sampledata.

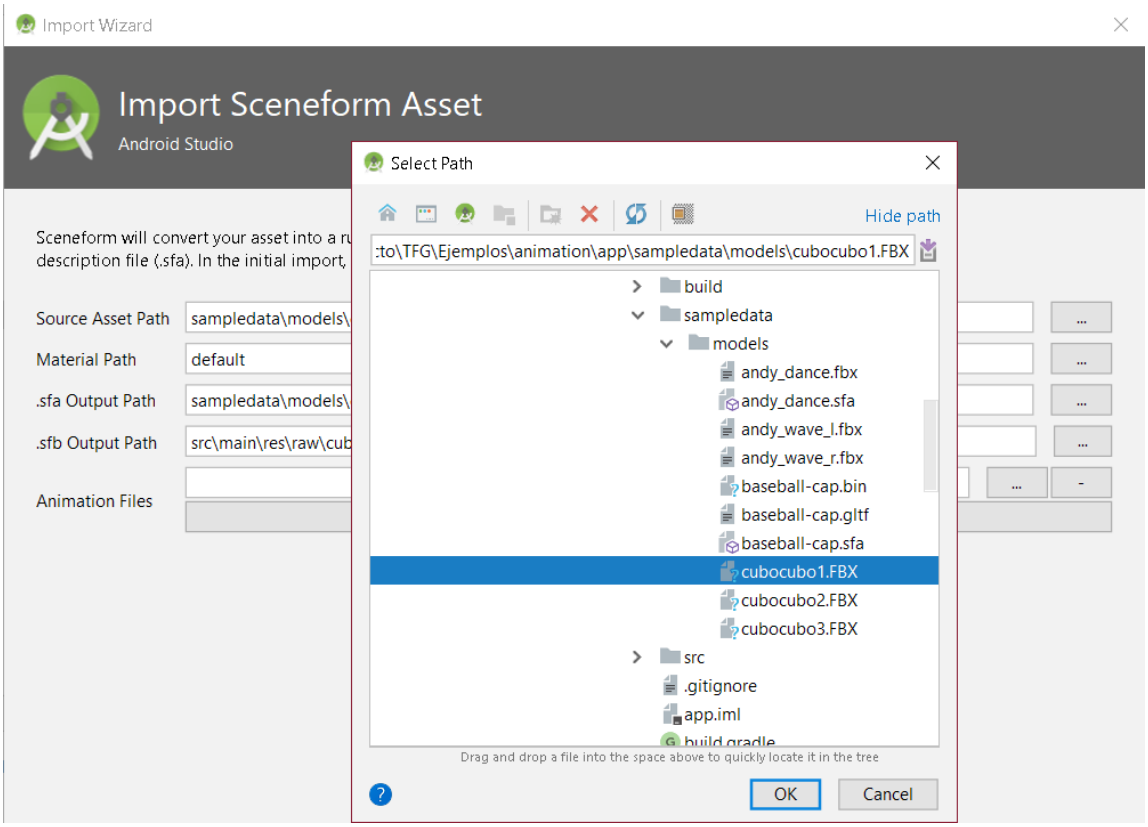


Figura 68. Selección del primer archivo FBX

Recordar que aquí hay que añadir todos los archivos, el primero que hemos usado para abrir esta pestaña de Importación también. En el caso del ejemplo se añaden los tres archivos.

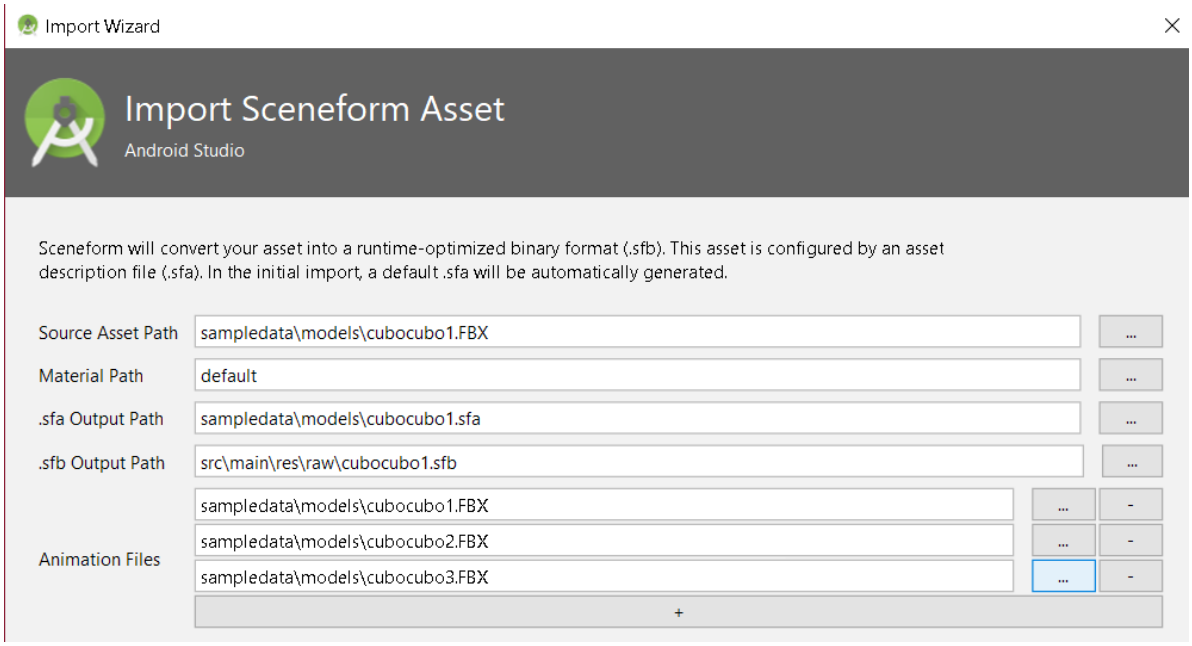


Figura 69. Apariencia de la pestaña cuando se han seleccionado todos los archivos FBX

En la selección de la carpeta para la creación del archivo .sfb en este caso no podemos usar cualquier carpeta como hacíamos en el punto 7.2 al usar el método Uri, ya que en el caso de las animaciones tenemos que usar el método R.

Por ello, debemos tener el archivo .sfb en la carpeta Raw, y así mismo cambiar la ruta en el código. Eso se hace como se muestra en el apartado 7.1 ilustrado en las Figuras 51 y 52, modificando el código en la ventana Gradle: app o cambiando la carpeta al momento de importar el archivo (como también podemos ver correctamente cambiada en la Figura 52).

Seleccionados todos los archivos que contienen las animaciones podemos darle a finalizar y se creará el código pertinente.

Podemos abrir el código del archivo .sfa o .sfb haciendo doble clic en él. Así podemos comprobar si se han cargado correctamente las animaciones y el orden de estas. Si al probar la aplicación vemos que las animaciones están desordenadas podremos cambiar el orden en este código.

cubocubo1.sfb

```
{
  animations: [
    {
      clips: [
        {
          name: 'cubo3',
        },
      ],
      path: 'sampledata/models/cubocubo3.FBX',
    },
    {
      clips: [
        {
          name: 'cubo1',
        },
      ],
      path: 'sampledata/models/cubocubo1.FBX',
    },
    {
      clips: [
        {
          name: 'cubo2',
        },
      ],
      path: 'sampledata/models/cubocubo2.FBX',
    },
  ],
  materials: [
```

Figura 70. Código del archivo .sfb con animaciones desordenadas

Al igual que en el código .sfb de los modelos 3D en este código también podemos modificar las características físicas del objeto.

Normalmente la adicción de animaciones en un modelo hace que no aparezca la línea de código de la escala que como explicamos en puntos anteriores se puede añadir manualmente y darle un valor numérico.

```
76      collision: {},
77      file: 'sampledata/models/cubocubo1.FBX',
78      name: 'cubocubo1',
79      recenter: 'root',
80      scale: 5,
81    },
82    version: '0.54:2',
83  }
```

Figura 71. Línea de código para la modificación de la escala

Viendo si se han creado correctamente estos archivos comprobaremos si se ha creado correctamente el Gradle. En la ventana app, podremos ver al final del código como se ha creado una parte de este que interrelaciona los diferentes archivos.

```
sceneform.asset('sampledata/models/cubocubo1.FBX',
        'default',
        'sampledata/models/cubocubo1.sfa',
        'src/main/res/raw/cubocubo1',
        ['sampledata/models/cubocubo3.FBX', 'sampledata/models/cubocubo1.FBX', 'sampledata/models/cubocubo2.FBX'])
```

Figura 72. Código creado en el Gradle: app

Ahora nos dirigiremos al código del MainActivity en el cual tendremos que cambiar el nombre del archivo en el modelLoader. Siendo “cubocubo1” el archivo principal de nuestro proyecto y la forma de denominar es R.raw.nombre del archivo, en el ejemplo quedaría como en la Figura 73.

```
modelLoader = new ModelLoader( owner: this);

modelLoader.loadModel(ANDY_RENDERABLE, R.raw.cubocubo1);
modelLoader.loadModel(HAT_RENDERABLE, R.raw.baseball_cap);

arFragment.setOnTapArPlaneListener(this::onPlaneTap);

arFragment.getArSceneView().getScene().addOnUpdateListener(this::onFrameUpdate);
```

Figura 73. Modificación del código en el modelLoader

Con estos cambios podremos arrancar nuestra nueva aplicación. Una vez buscado el plano deseado el usuario tocara la pantalla del dispositivo colocando el modelo 3D. En ese momento el botón que aparece en la pantalla cambiará de color mostrando que está activo, cada vez que el usuario lo pulse mostrará una de las animaciones creadas.

Figura 74. Visualización de la aplicación Cubo

Ahora que hemos visto como programar diferentes aplicaciones vamos a ver a continuación como poder tenerlas instaladas todas a la vez en un mismo dispositivo.

7. TUTORIAL PARTE 4: Instalación en dispositivo

Para la correcta instalación e identificación de las aplicaciones cada una de ellas al instalarse tiene una serie de características únicas que hace que su apk se instale correctamente. Si esta información la comparten varias aplicaciones el teléfono entenderá que es la misma y desinstalará la más antigua. O en el caso de tener diferentes ID pero las mismas características y nombres, que el usuario no sepa diferenciarlas.

Por ello, vamos a ver cómo cambiar esta información e incluso la estética de la aplicación con Android Studio para que el usuario pueda lograr tener todas las aplicaciones que programe en su dispositivo y las pueda diferencias y usar sin problemas.

7.1. Cambiar ID de la aplicación

Esto quizá sería lo más importante ya que si no cambiamos el ID de nuestra aplicación y usamos el mismo código de una misma aplicación base, solo podremos tener instalado a la vez una de las aplicaciones.

Para que esto no suceda empezaremos por cambiarle el nombre a la última carpeta del paquete predeterminado por el nombre de nuestra aplicación en el AndroidManifest.

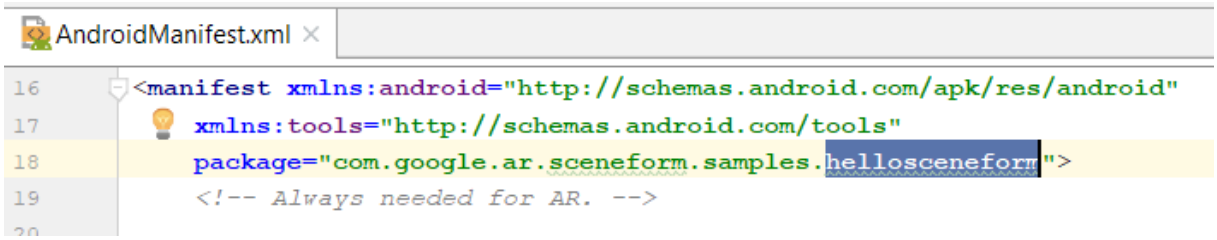


Figura 75. Nombre del paquete predeterminado

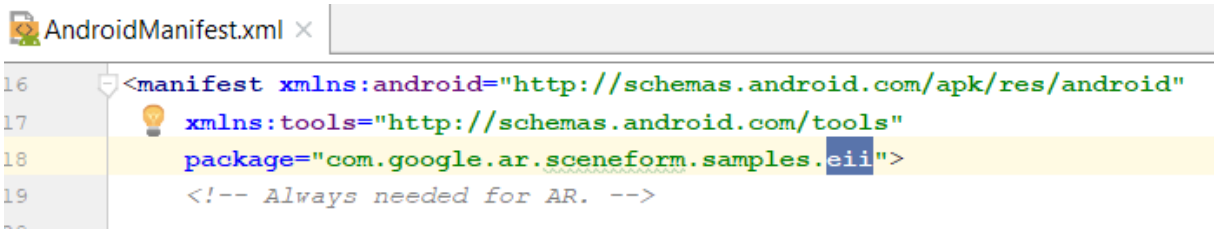


Figura 76. El nombre del paquete ya cambiado

Luego iremos a la ventana del Gradle: app, y en ella cambiaremos de la misma forma applicationId.

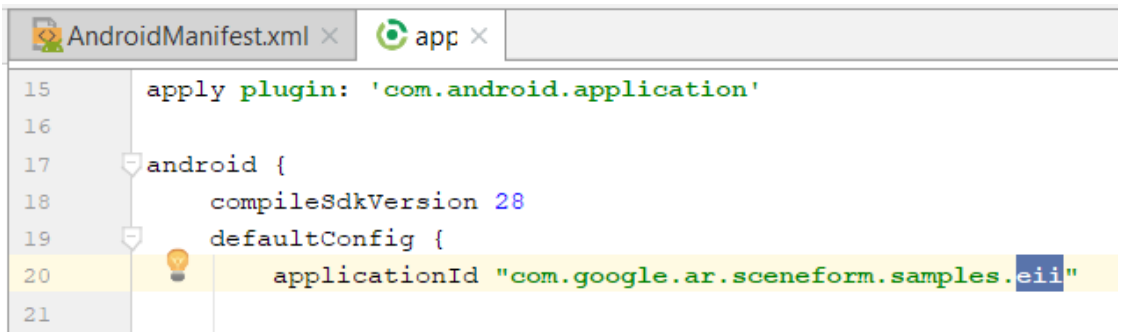


Figura 77. Nombre cambiado en la línea del applicationId

Para cambiar ahora el nombre del paquete en el código del HelloSceneformActivity deberemos seleccionar toda la ruta y una vez marcada darle a Control+Shift+R.

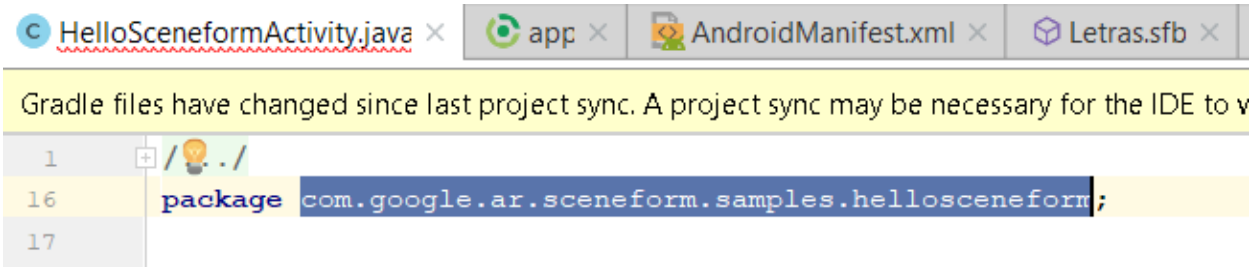


Figura 78. Selección de la ruta del package

Se abrirá en ese momento una pestaña en la que aparecerán arriba de esta dos espacios en blancos con una lupa en el comienzo de la línea. En el primer espacio estará escrita la ruta que hemos marcado, en el segundo espacio, ahora en blanco, deberemos escribir la nueva ruta. Esta nueva ruta será la que hemos escrito anteriormente, que para no complicarnos y cambiar el nombre de todas o algunas de las carpetas hemos decidido solo cambiar de nombre a la última.

Una vez escrita la nueva ruta clicaremos en “Replace All”.

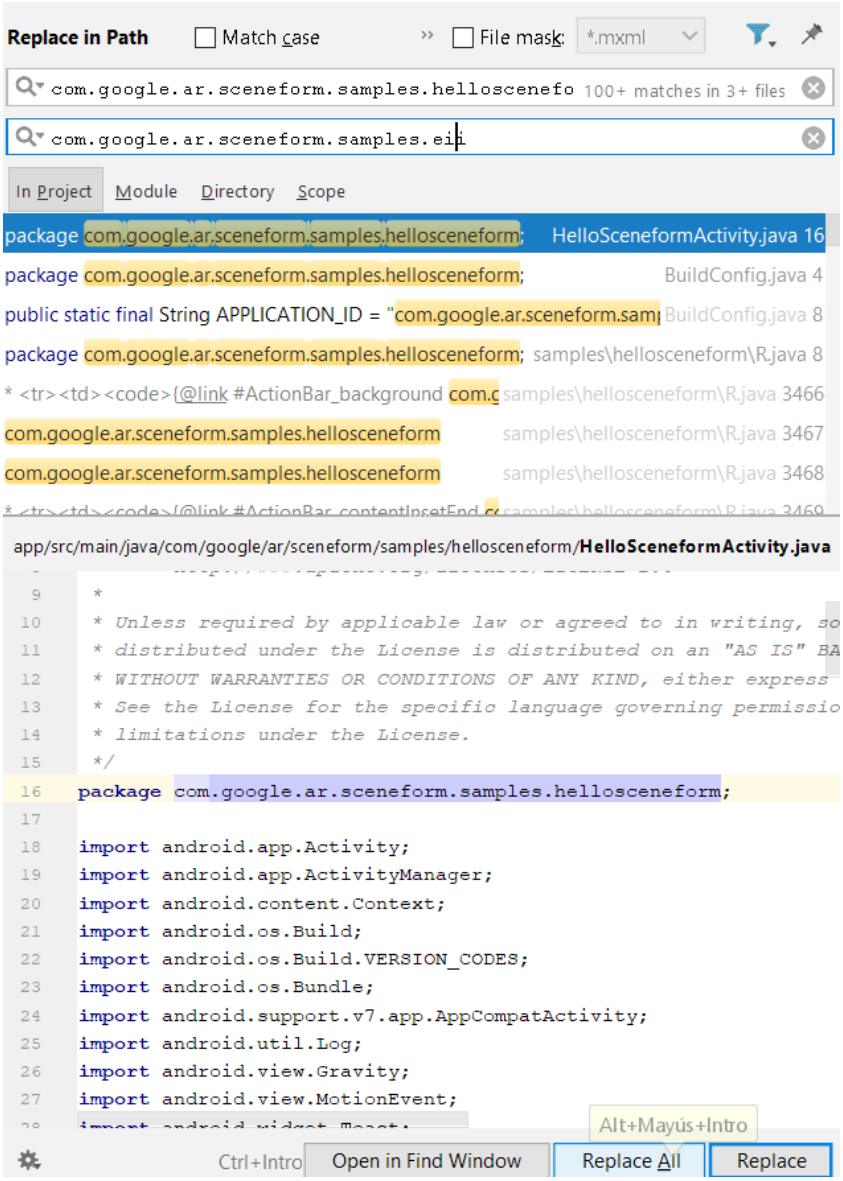


Figura 79. Pestaña en la cual cambiaremos la ruta del package

Ahora nos dirigiremos al buscador donde, dentro de la carpeta java, se encuentra una carpeta con la ruta y en ella el código HelloSceneformActivity.

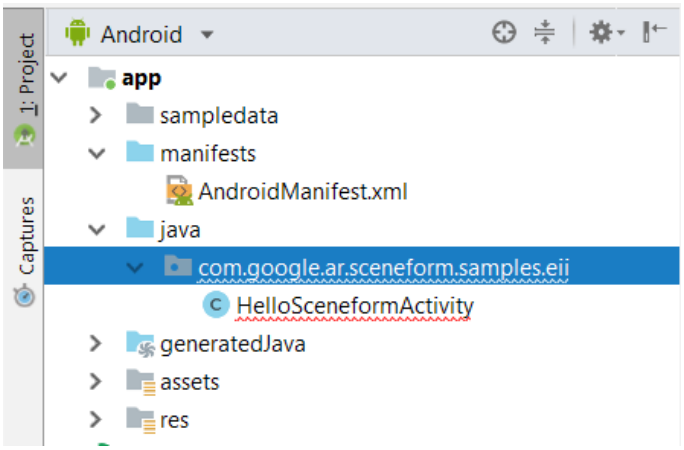


Figura 80. Selección de la carpeta de la ruta

Si dicha carpeta tiene por nombre la ruta antigua tendremos que cambiar el nombre a la última carpeta. Para ello la desglosaremos en diferentes carpetas haciendo clic derecho y dándole a “Hide Empty Middle Packages”.

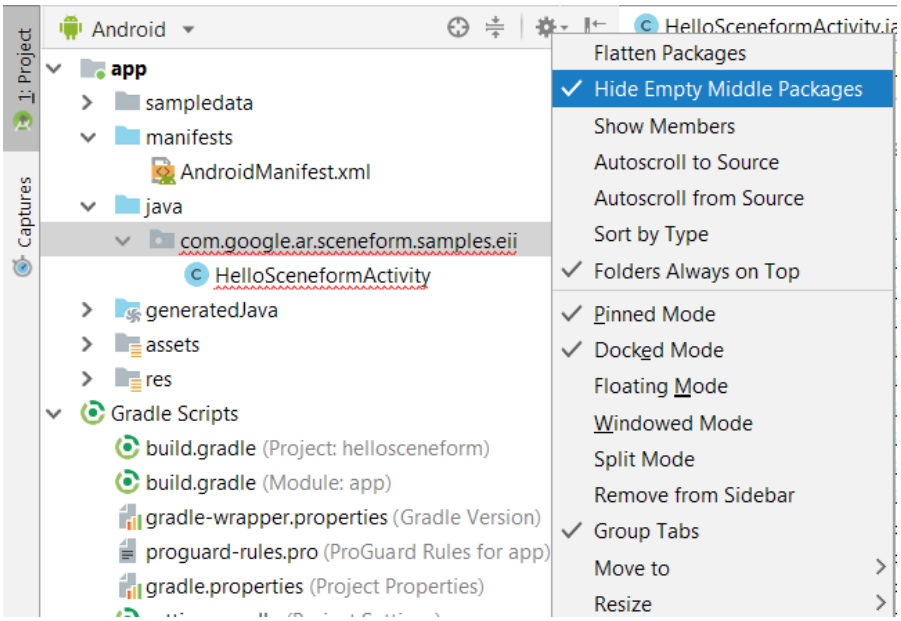


Figura 81. Botón para desplegar por individual las carpetas de la ruta

De esta forma, una vez separados podamos hacer clic derecho en la última carpeta, la cual deberá tener el nombre de la aplicación antigua.

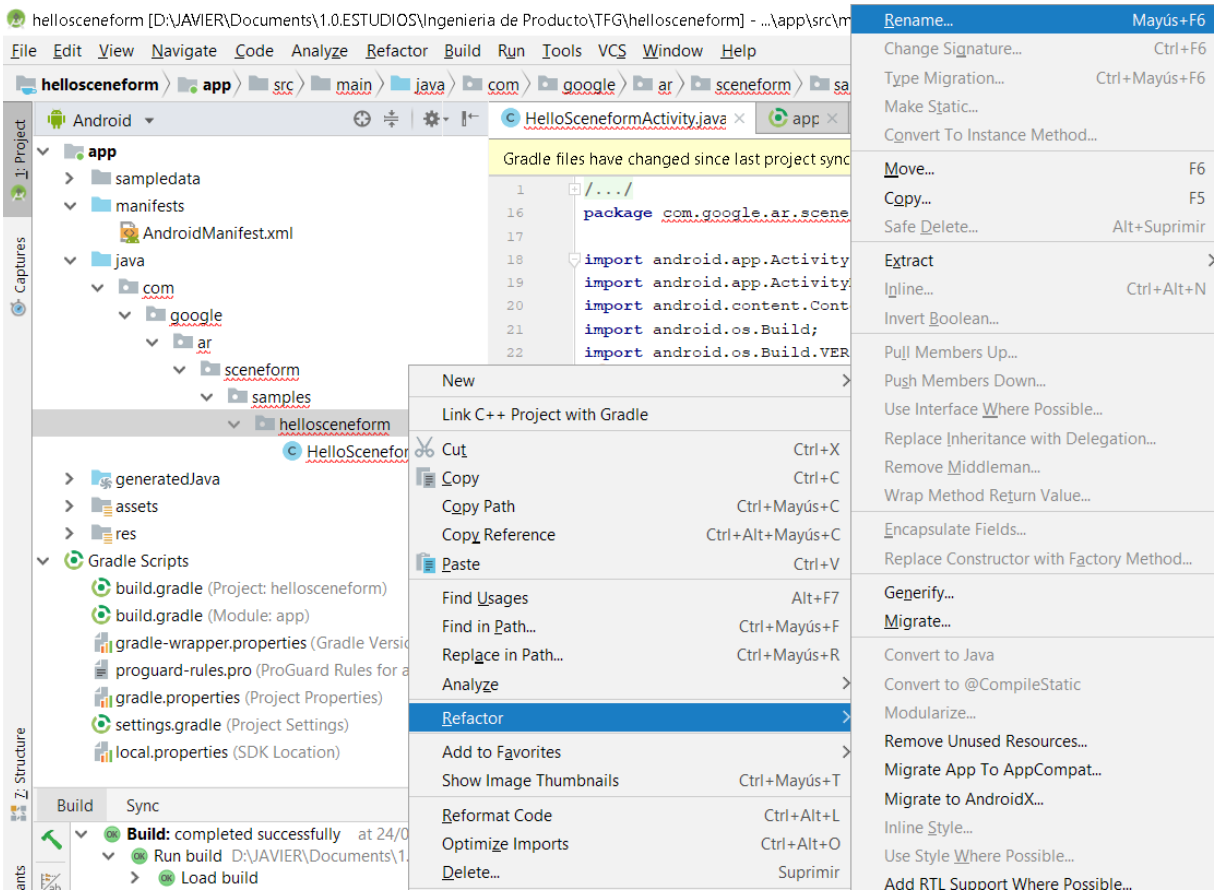


Figura 82. Botón para renombrar la carpeta

Le damos a “Refractor” y luego a “Rename” abriéndonos otra pestaña en la cual ya podremos cambiar el nombre.

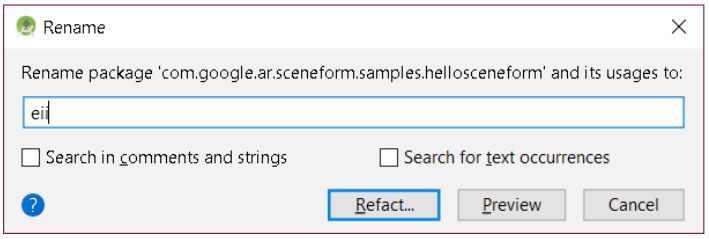


Figura 83. Pestaña donde se cambia el nombre a la carpeta

Una vez realizado estos pasos podemos arrancar la aplicación en nuestro dispositivo. Se instalará como otra apk y así se guardará como una aplicación diferente.

Sin embargo, si hemos usado este aplicación base para tener diferentes aplicaciones, estas tendrán el mismo nombre e icono, por lo que el usuario no sabrá diferenciarlas. Por ello, a continuación explicamos como cambiar estos aspectos.

7.2. Cambiar Nombre de la aplicación

Nos dirigiremos al Android Manifest en el cual podremos cambiar tanto el nombre de la aplicación como el nombre que verá el usuario debajo del logo una vez instalada esta en su dispositivo.

En el primero módulo “application” habrá una línea de código idéntica que en el “activity” que será: android: label=”nombre de la aplicación”. Es en estas dos líneas donde hay que cambiar el nombre de la aplicación.

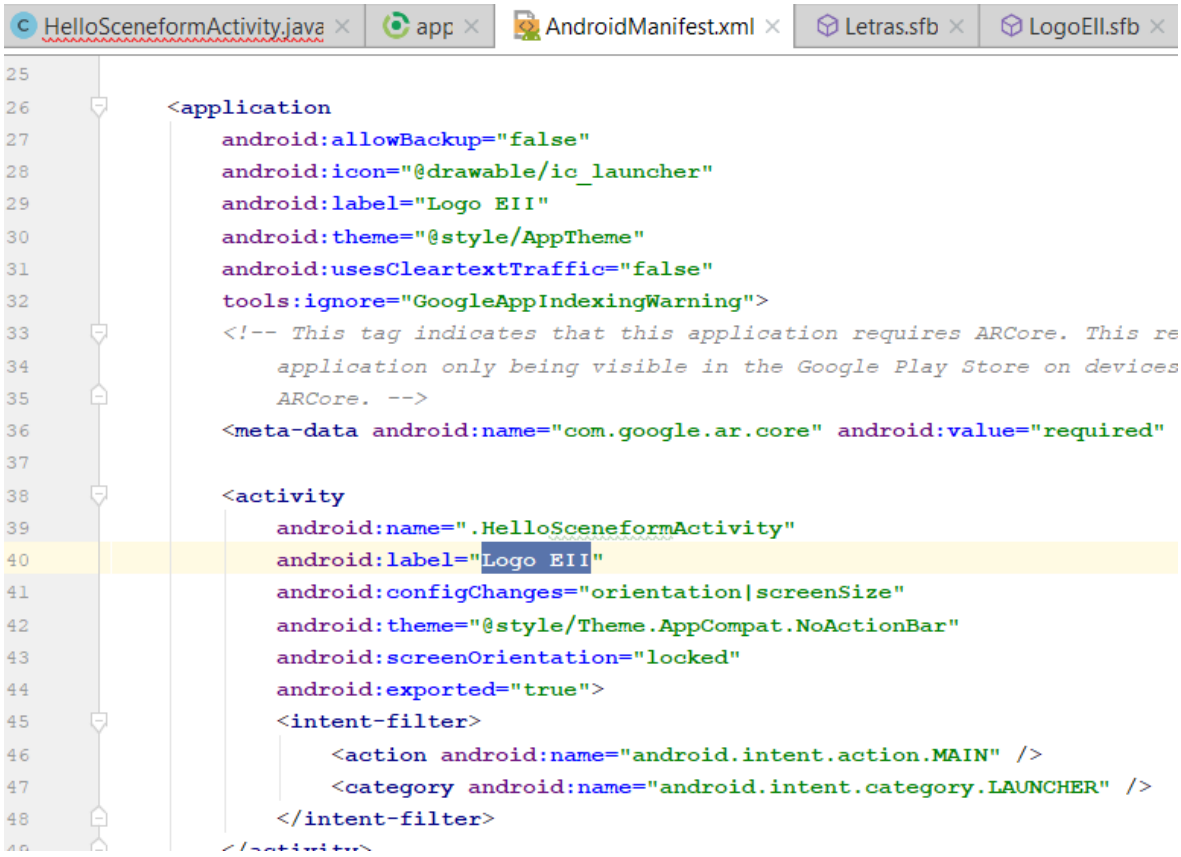


Figura 84. Línea que denomina el nombre de la aplicación

7.3. Cambiar Icono de la aplicación

Aunque Android Studio de los píxeles que utiliza al meter el png automáticamente, introducimos el icono predeterminado que usa en us aplicación para recrear nuestro icono con la mayor calidad posible.

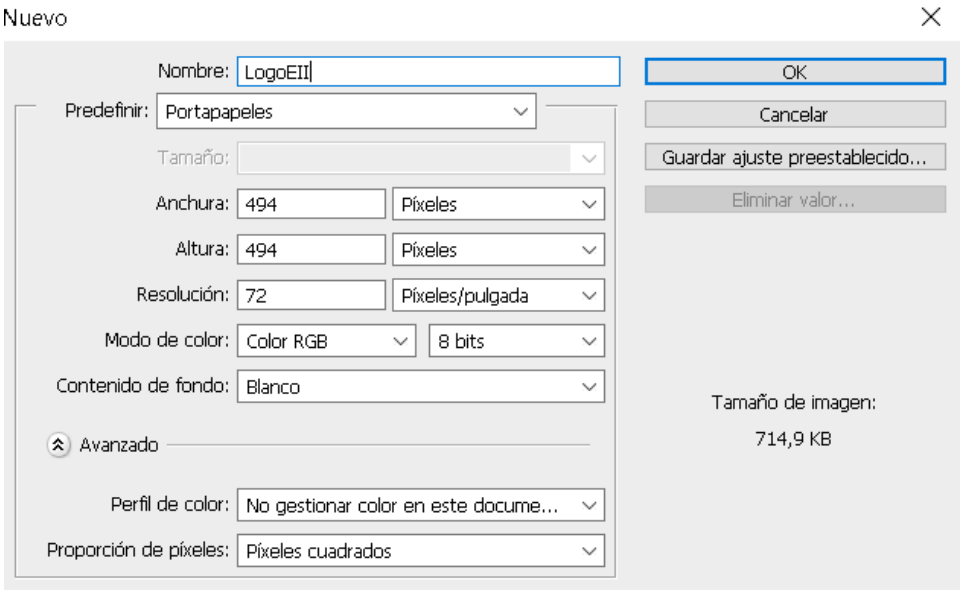


Figura 85. Píxeles del icono predeterminado de la aplicación

Al introducir el icono predeterminado nos da un tamaño de 494x494 píxeles. Así que con esa medida podremos crear nuestro icono, el cual tendremos que recordar exportarlo en el formato png.

Una vez tengamos nuestro icono listo nos dispondremos a cambiarlo en Android Studio. Para ello deberemos hacer clic derecho en el buscador, clicar en “New” y buscar “Image Asset”.

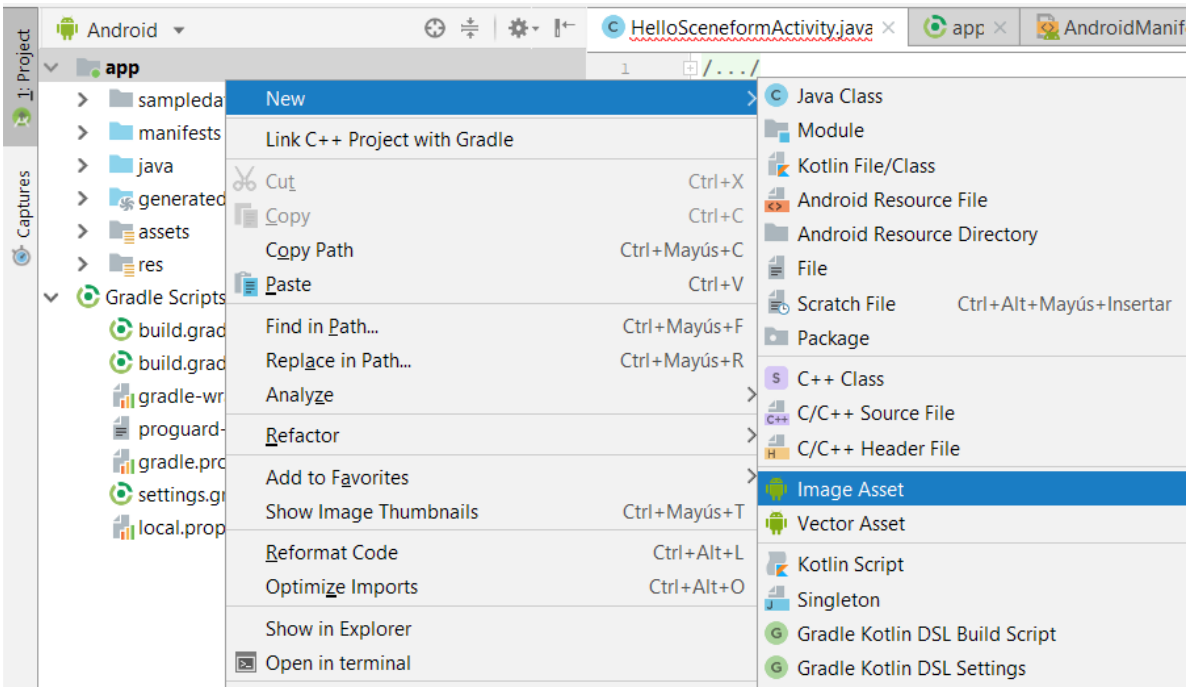


Figura 86. Image Asset

Aparecerá una pestaña en la cual podremos modificar las características de nuestro icono.

Aunque también podemos cambiar el nombre de la Image Asset no se recomienda. Si lo hacemos, deberemos modificar el código del AndroidManifest (en la línea: android: icon=”drawable/nombre del icono”), por lo que si lo dejamos en el predeterminado nos podemos ahorrar ese paso.

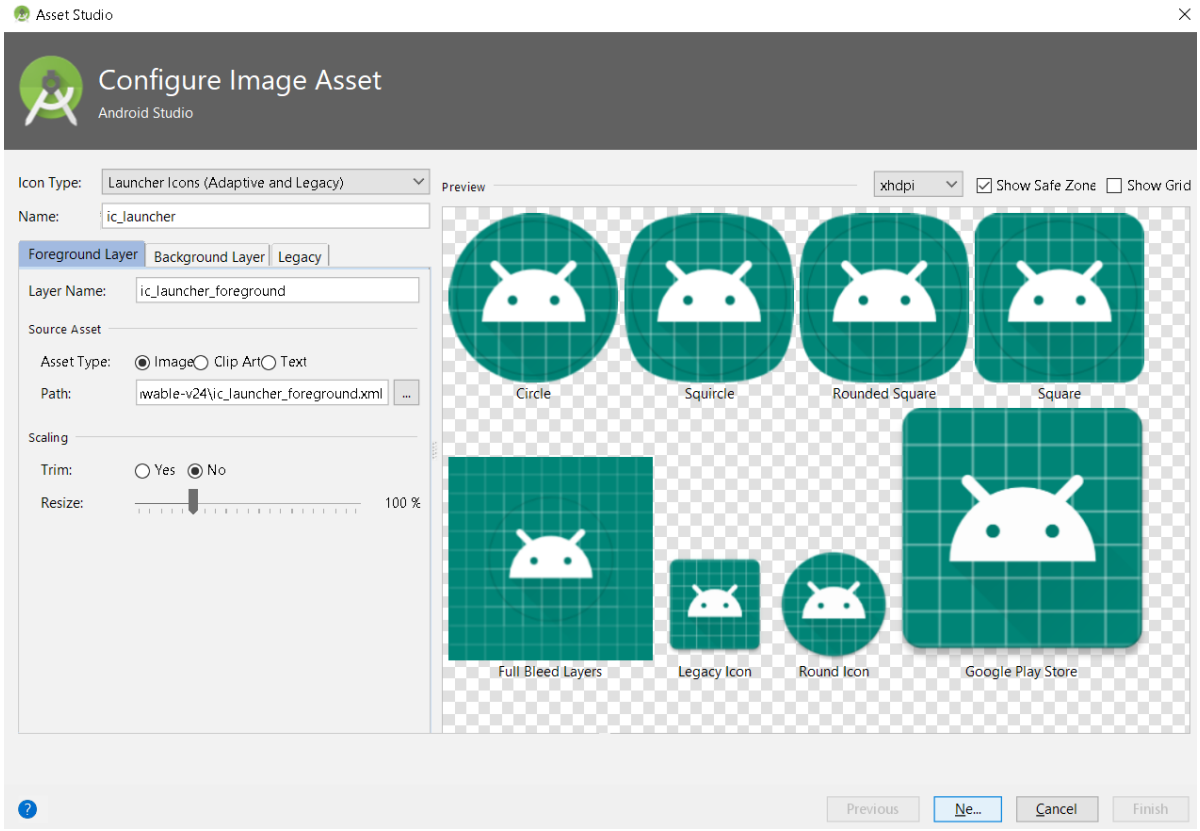


Figura 87. Editor de imágenes de Android Studio

En el apartado Source Asset podemos buscar el archivo png a introducir. Cabe recordar que en Android Studio siempre trabajamos con letras de la a a la z en minúsculas, y con números, si no, da errores de compilación.

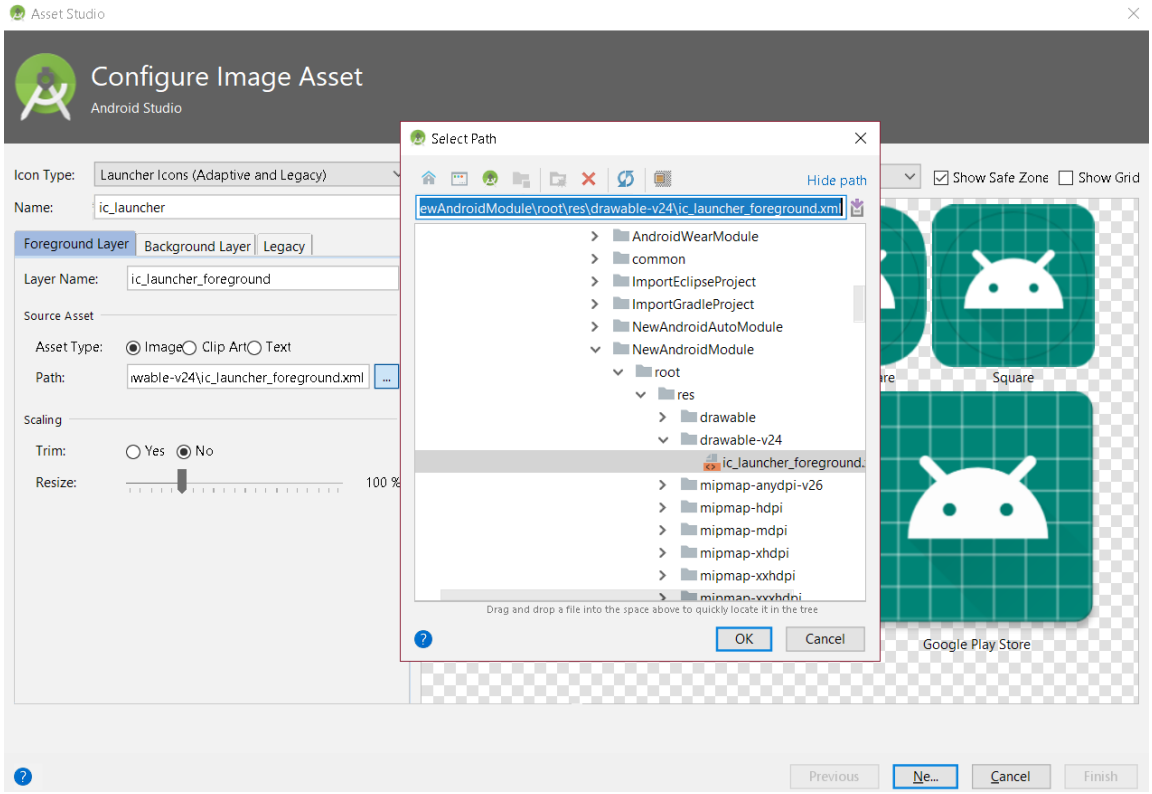


Figura 88. Selección de la imagen en el editor

Una vez seleccionada la imagen podremos ver como se ha creado la previsualización de los diferentes formatos de icono.

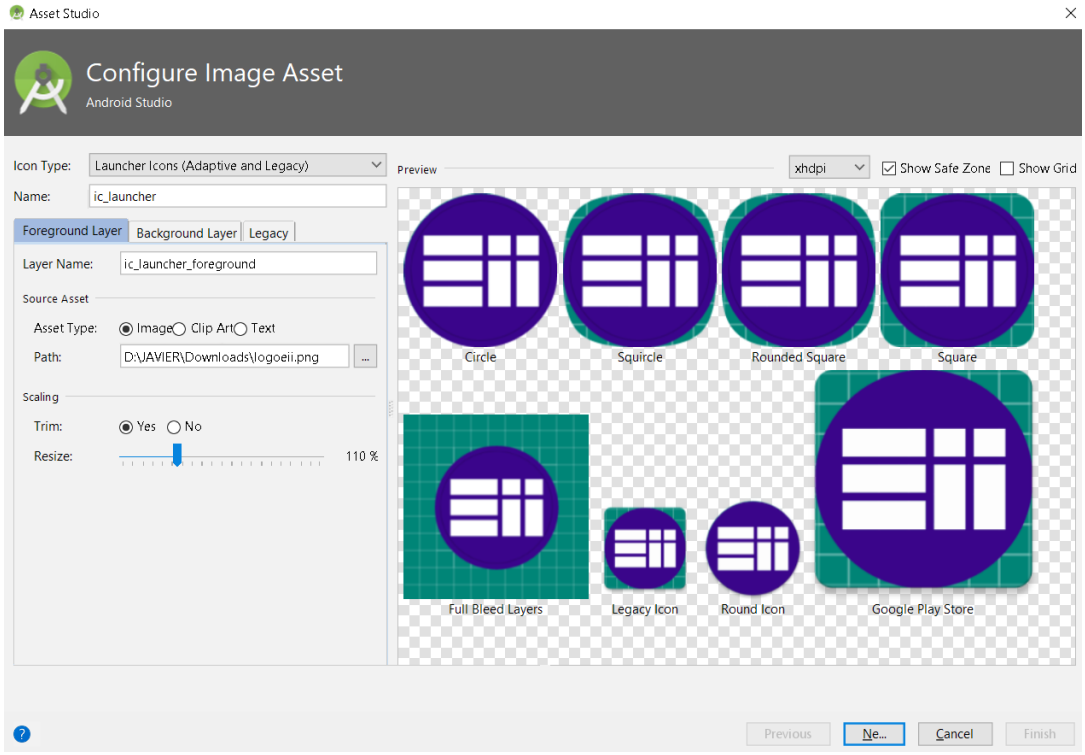


Figura 89. Imagen seleccionada cargada en el editor

Para que el icono se quede con el fondo transparente, es decir, que se vea en el formato png que nosotros hayamos creado. Tendremos que en el apartado Scaling del Background Layer bajar el tamaño a 0%.

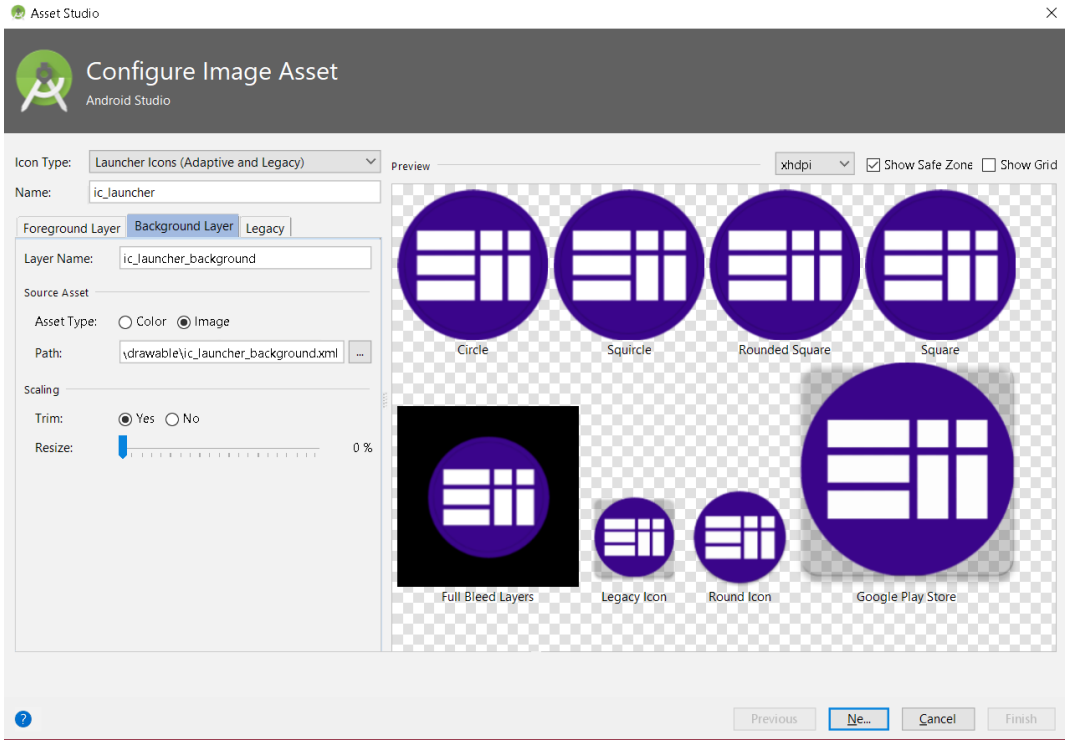


Figura 90. Modificaciones del Background Layer

Para cambiar la sombra que se queda en este momento nos iremos a la última pestaña, Legacy en la cual modificaremos el apartado Shape, para que la forma de la sombre tenga la forma que más nos convenga en cada caso.

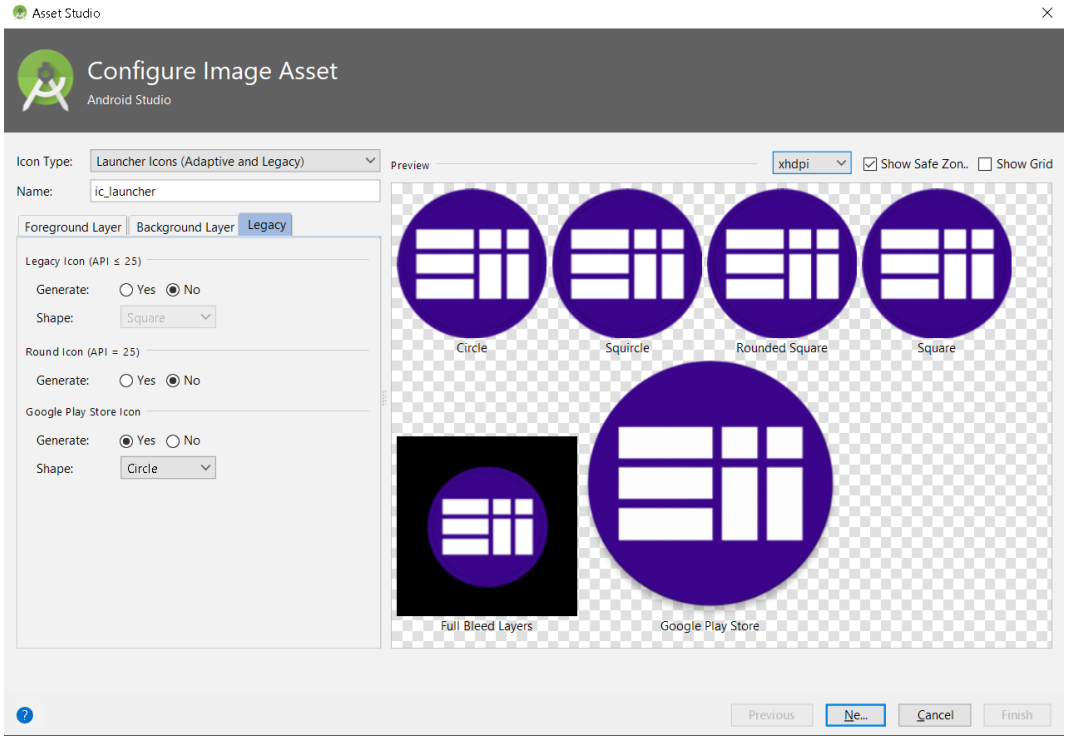


Figura 91. Modificaciones del Legacy

Suele ocurrir que acabado este proceso el icono antiguo no sea remplazado por el nuevo que acabamos de crear, por ello habrá que cambiarlo manualmente.

Si no se ha modificado iremos a buscar el icono antiguo a la carpeta “drawable” que se encuentra en la carpeta “res”. Una vez allí podemos eliminarlo desde Android Studio o buscar el archivo en las carpetas y eliminarlo desde ahí.

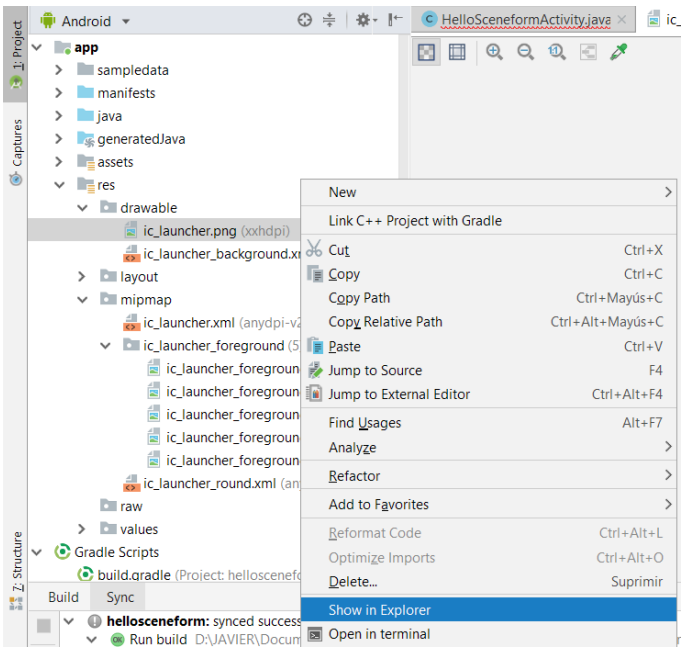


Figura 92. Búsqueda en los archivos del ordenador. Se puede eliminar desde el propio programa o decidir abrirlo en una carpeta

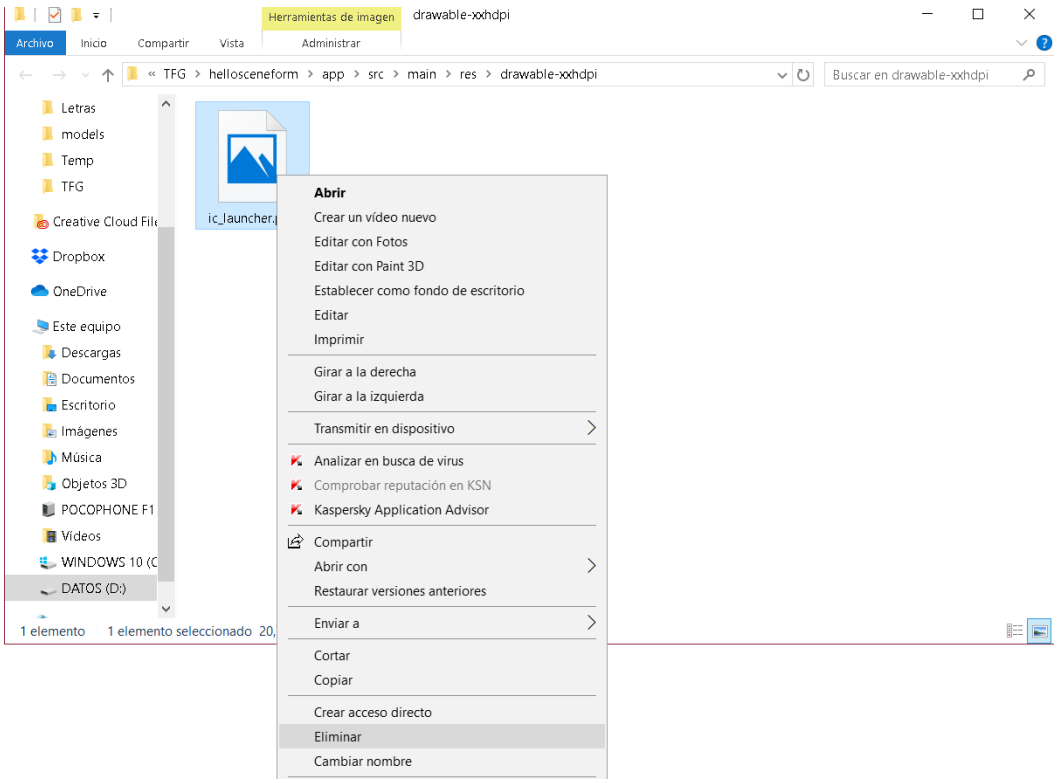


Figura 93. Una vez abierto en una carpeta

Una vez eliminado el archivo antiguo cogemos el archivo xxhdpi que queramos como icono (que ya habrá sido generado en el paso anterior) y lo pegamos en la carpeta donde anteriormente hemos eliminado el predeterminado.

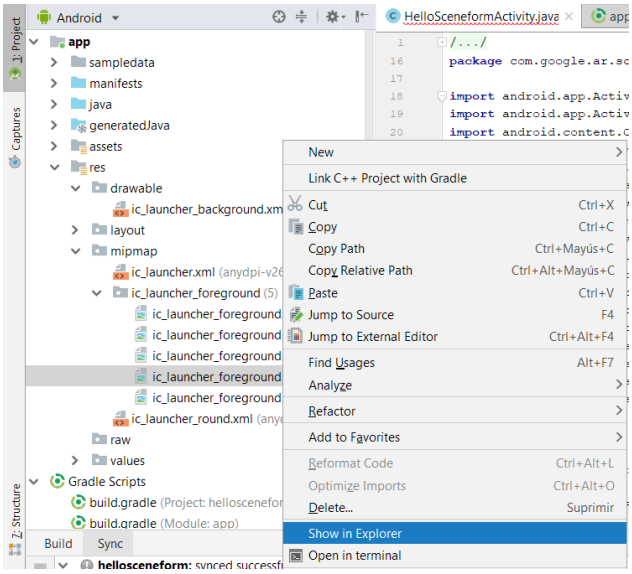


Figura 94. Buscamos el archivo que queremos directamente desde el programa

Habrá que tener en consideración que si hemos tenido que hacer estos pasos, ese archivo xxhdpi que queremos como icono puede que haya sido generado con otro nombre o con algún otro añadido (suele ser una barra baja y la carpeta en la que ha sido creado). Si es así, simplemente tendríamos que cambiarle el nombre al que hayamos elegido en un principio y moverlo a la carpeta drawable.

Llegados a este punto el dispositivo ya entenderá que se trata de otra aplicación al instalar el apk y el usuario podrá diferenciar por el icono y el nombre de este la aplicación.

Aun así, una vez abierta será igual que las demás, ya que el interfaz que usamos sigue siendo el de base. Quizá el programador al introducir un objeto quiere cambiar el color de este y que los botones (por ejemplo en la aplicación ejemplo del apartado 7.3) o las barras de carga y espera de permisos tengan una cierta concordancia entre ellas.

Por ello, veremos también como podremos modificar estas opciones en el Android Studio.

Primero nos tenemos que ir al archivo activity main, pero no al archivo con el que hemos estado trabajando hasta entonces, sino al del formato xml. El cual se encuentra en la carpeta “res”, dentro de otra llamada “layout”.

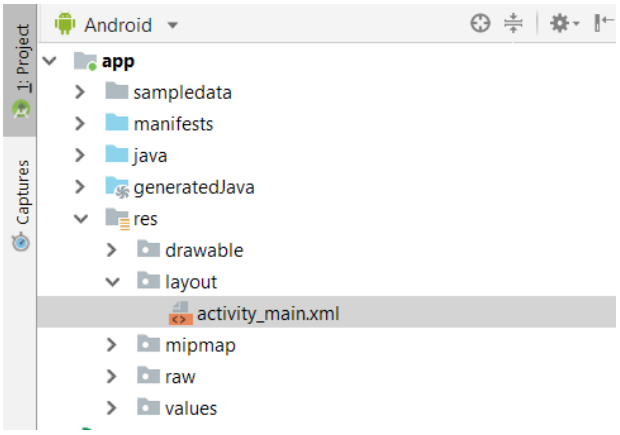


Figura 95. activity_main.xml

Una vez lo abramos comprobaremos que la pestaña es diferente a las abiertas anteriormente, ya que, por primera vez no se trata de un código sino de un editor de diseño con forma de la pantalla de nuestro dispositivo.

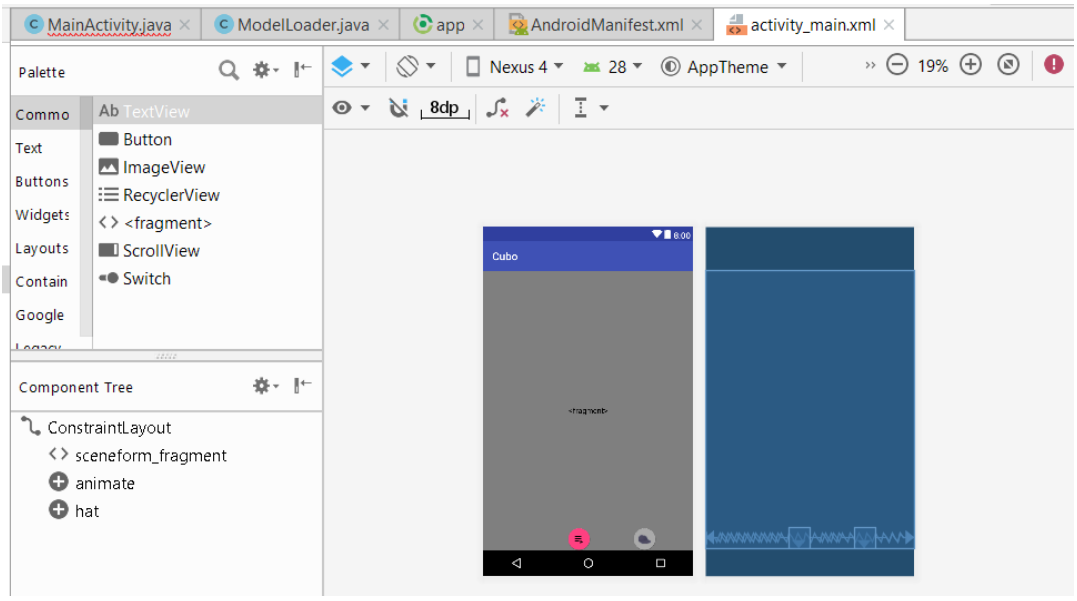


Figura 96. Editor usado en formatos XML

Si clicamos encima del botón aparecerá un panel a la derecha que contiene una serie de atributos que podremos modificar.

En ese panel podremos modificar la situación del botón de tal forma que siempre se respete las indicaciones puestas, ya que si lo hacemos manualmente en la previsualización de la pantalla, si el tamaño del móvil cambia, también cambiará la posición del botón

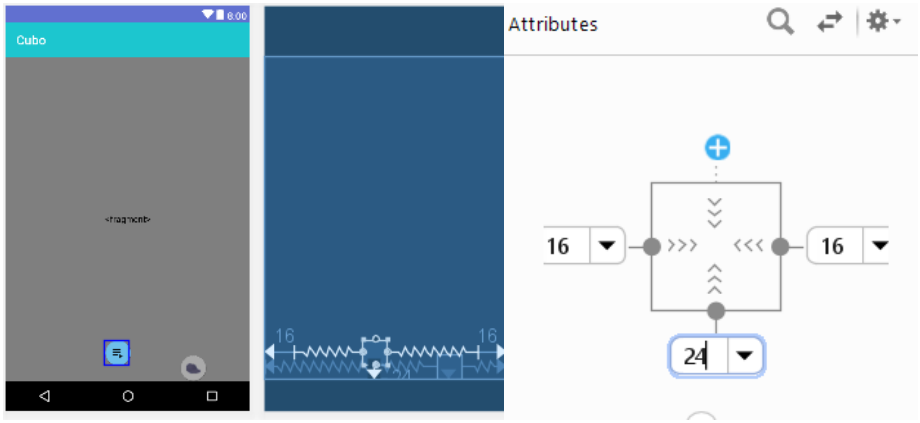


Figura 97. Herramienta para posicionar los diferentes botones en relación a la pantalla

En ese panel también podremos clicar en los tres puntos de la derecha que aparece en el apartado de color del botón, de tal forma que no solo podemos cambiar desde ahí el color del botón, sino que también podemos seleccionar una paleta de colores para el resto de la aplicación.

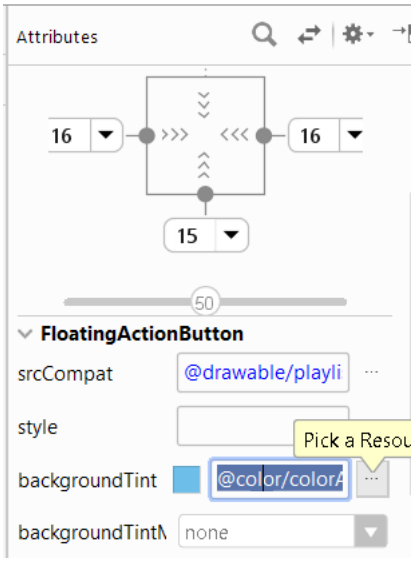


Figura 98. Botón de recursos de color

En la pestaña resultante podemos cambiar los colores de todas las partes de la aplicación. Los colores están asignados a las barras del título, otro a las barras de espera para pedir los permisos, otro a los botones, etc.

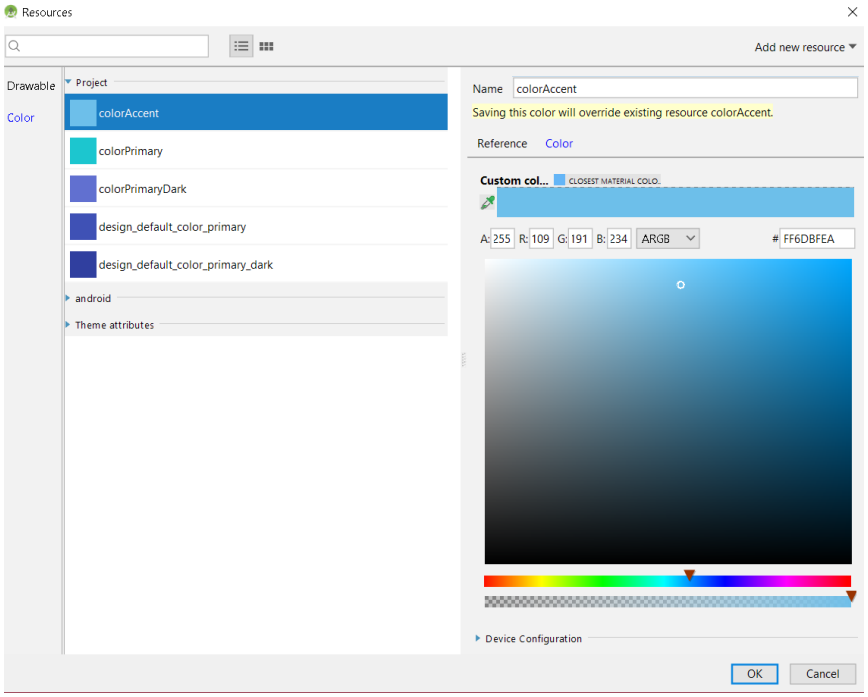


Figura 99. Ventana de recursos en relación a la paleta de colores

Podemos introducir en la carpeta drawable la imagen que nosotros queramos y luego en el apartado del panel del botón “srcCompact” podemos seleccionarla para que sea el icono de nuestro botón.

Figura 100. Botón de recursos de imágenes

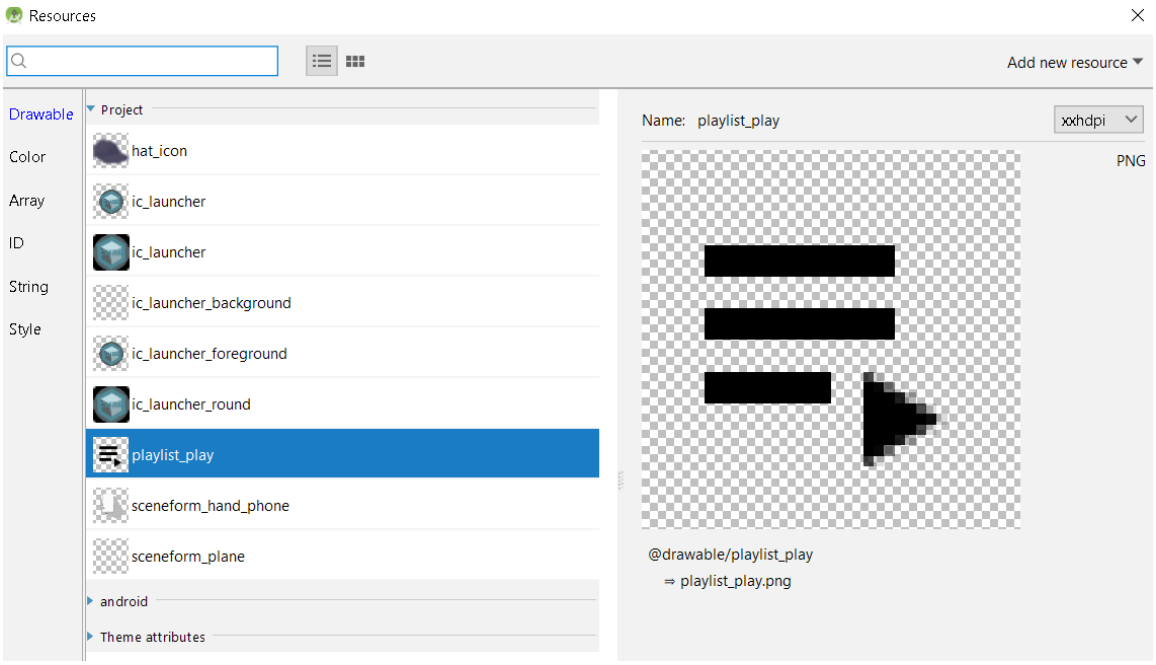


Figura 101. Selección de una imagen en la ventana de recursos

Ahora si arrancamos podremos ver que se descarga como una nueva aplicación y además de verse con un nombre y un icono nuevo también al abrirse se ve diferente gracias a los cambios en la estética del interfaz.

8. USABILIDAD

La usabilidad se refiere a la facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto con el fin de alcanzar un objetivo concreto.

La usabilidad del tipo de aplicaciones programadas en este documento es muy extensa, ya que dependerá de la utilidad que le quiera dar el programador. Los campos en los que se puede usar pueden ser desde el entretenimiento hasta el marketing pasando por el diseño industrial. En este último caso, puede tomar cierta relevancia ya que muchas veces no se poseen los medios (dinero, tiempo…) para realizar un prototipo o maqueta de un determinado producto. Esto es claramente una desventaja porque en estas situaciones estamos obligados a mostrar o vender el producto usando medios 2D, en los cuales quizá ciertas características o puntos fuertes pueden quedar reducidos por el soporte usado. No solo en la venta o muestra de nuestros productos puede ser usado este tipo de aplicaciones, sino también en el propio proceso de diseño como fase comprobante de diferentes características físicas como por ejemplo como se relaciona con su entorno o como le afecta la luz. En estos casos, solo se trataría de seguir los pasos de este documento y se podría hacer una aplicación de RA en la cual se importe el modelo 3D del producto para su investigación o muestra. Cuando probamos estas aplicaciones destinadas a este objetivo nos damos cuenta de la facilidad de su uso. Cualquier usuario medio podría utilizar este tipo de aplicaciones que hemos programado y se sentiría cómodo en su uso y satisfecho por su funcionalidad. Para cerciorarnos de ello realizaremos una prueba de usabilidad en las aplicaciones programadas. Un trabajo futuro podría ser también una prueba de usabilidad del tutorial de este documento.

8.1. Prueba de usabilidad

A continuación realizamos una prueba para medir la usabilidad en nuestro sistema interactivo. La herramienta metodológica que vamos a usar se denomina Escala de Usabilidad de un Sistema o SUS por sus siglas en inglés (System Usability Scale), la cual es uno de los métodos de usabilidad más utilizados en Experiencia de Usuario. [22]

Esta prueba se realizó a cinco usuarios de un rango de edad de 20 hasta 28 años. Se les hizo una breve explicación de la aplicación y se les dejó interactuar con ella. Luego se les presentó un test a realizar.

Condiciones de la prueba;

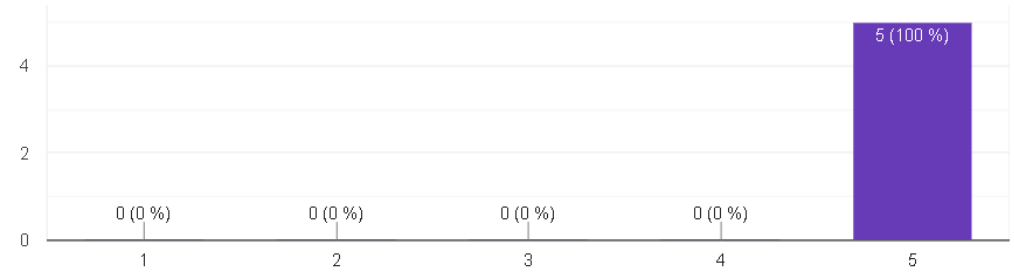
La información antes de entregarles a probar la aplicación a los usuarios fue una breve explicación en la que se mostraba el objetivo de la aplicación. El modo de uso no se explicó, solo se señaló que para hacer aparecer el modelo había que clicar en la pantalla y no se explicó nada más para que la prueba fuera lo más objetiva posible. Posteriormente se les dejó interactuar con la aplicación.

El dispositivo usado fue un Xiaomi Pocophone F1.

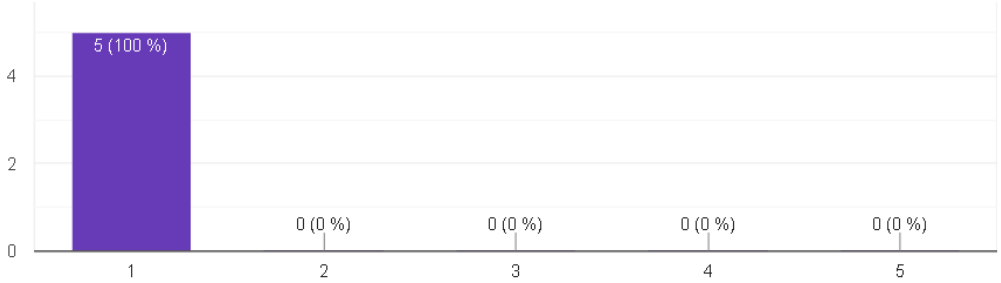
El lugar donde se realizó la prueba fue una habitación con una bombilla LED de 5W blanca colgada del techo que daba una luz central en la estancia. Los usuarios colocaron sus modelos en una mesa blanca que estaba justo debajo de la luz, y en el suelo de parqué, uno o dos metros alejado de la trayectoria perpendicular del foco de luz.

La escala de medición consistió en 10 preguntas, puntuadas del 1 al 5, siendo 1 “Total desacuerdo” y 5 “Total acuerdo”. Las preguntas fueron las siguientes:

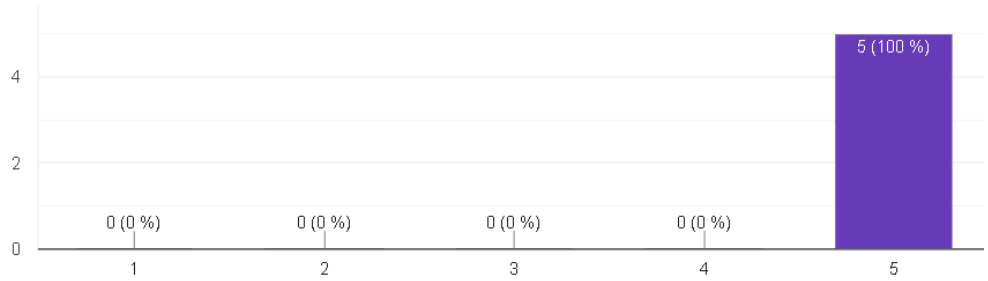
1. Creo que usaría esta aplicación para mostrar mis modelos 3D/me gustaría que usaran esta aplicación para enseñarme modelos 3D.



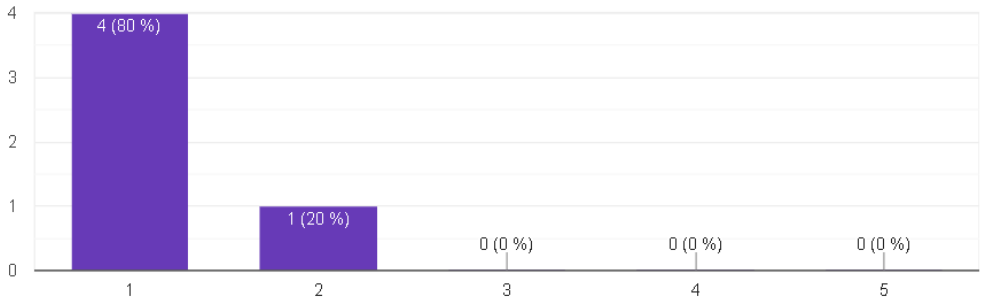
2. Encuentro esta aplicación innecesariamente compleja.



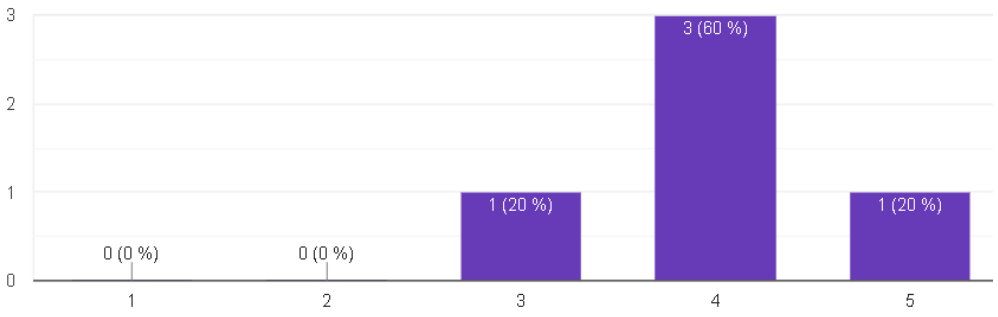
3. Creo que la aplicación fue fácil de usar.



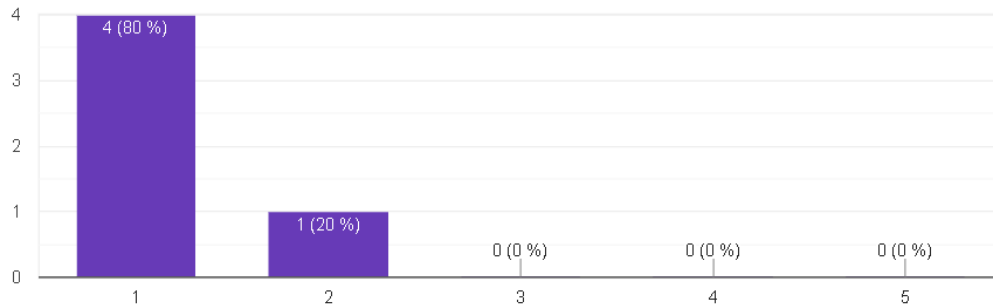
4. Creo que necesitaría de una persona con conocimientos técnicos para usar esta aplicación.



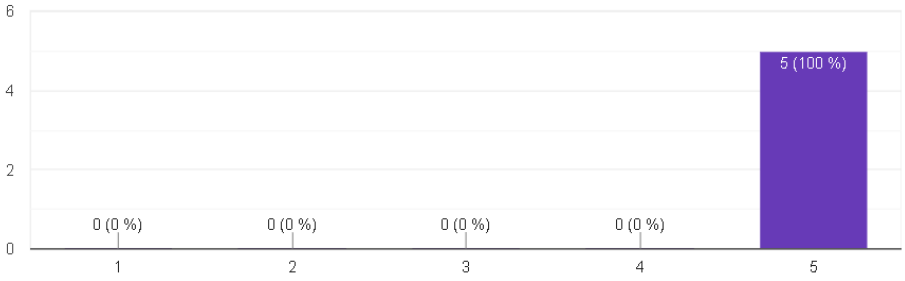
5. Las funciones de esta aplicación están bien integradas.



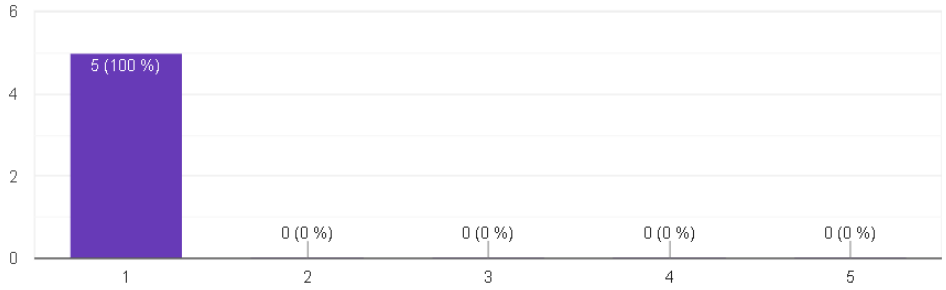
6. Creo que la aplicación es la aplicación es muy inconsciente.



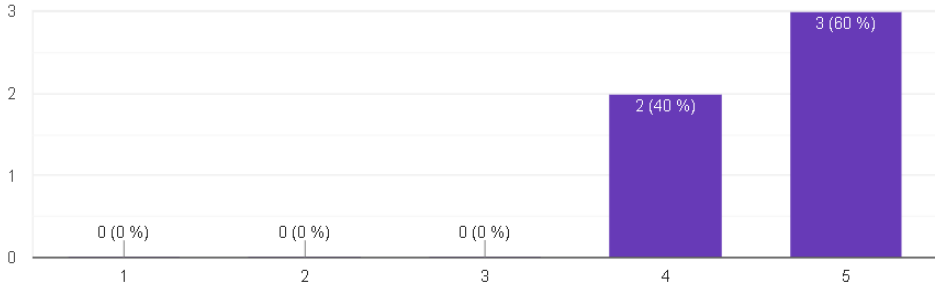
7. Imagino que la mayoría de la gente aprendería a usar esta aplicación de forma rápida.



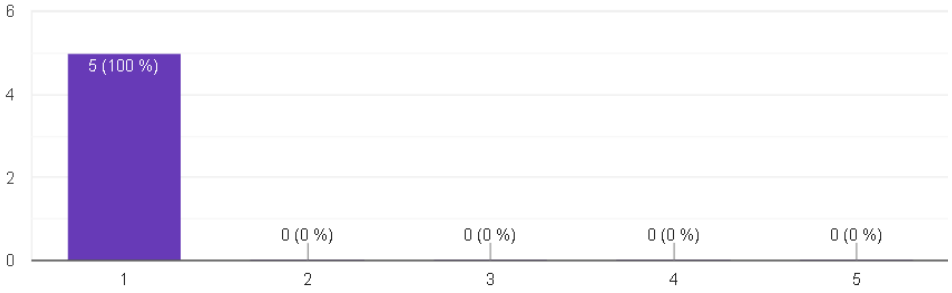
8. Encuentro que la aplicación es muy difícil de usar.



9. Me siento confiado al usar esta aplicación.



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



Luego de tener todas las respuestas del test, realizamos la medición. Consideramos lo siguiente: las preguntas impares tomarán el valor seleccionado por el usuario y le restaremos 1. Las preguntas pares, restaremos a 5 el número asignado por el usuario. Sumaremos estos diez datos y los multiplicaremos por 2,5. Así obtenemos el SUS de cada usuario. [22]

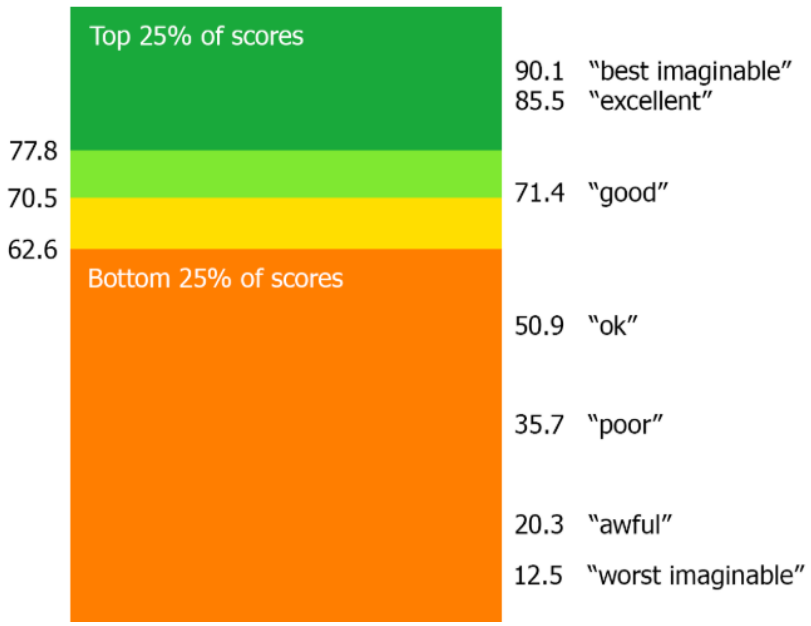


Figura 102. Gráfica de valoración del SUS [22]

El máximo teórico de la escala es 100. Con nuestros puntajes SUS hacemos la media entre ellos dividiendo entre 5, resultando un valor de 95.5. Lo que es, según el gráfico de la Figura 102, una marca que supera lo mejor imaginable en lo que se refiere a la usabilidad de nuestra aplicación.

9. CONCLUSIONES

En este trabajo se ha tratado la realidad aumentada desde diversos puntos de vista. Todos estos buscaban explorar el tema para que un lector, sin experiencia en la RA, lograra tener una visión global de ella.

La primera parte fundamentada en la explicación de la realidad aumentada desde los inicios de esta hasta la actualidad, supone un primer paso para sentar las bases del trabajo. En este punto aprendemos lo que es la realidad aumentada, a definirla, cómo de importante es y va a llegar a ser, entendiendo su potencial, y que a través de la experimentación va a lograr hacerse un hueco en nuestro día a día. También realizamos un estudio del arte para conocer hasta qué punto puede llegar a ser útil esta tecnología y un estudio de mercado para ver como de útil lo es ya. La conclusión de esta parte es que la realidad aumentada tiene un claro potencial y, aunque ya ha recorrido un largo camino, aún le queda más camino por recorrer.

Una vez dado el primer paso, nos disponemos a tratar ya el tema desde una parte más específica y técnica, llegando a conocer las herramientas que vamos a usar en el trabajo y aprender sobre ellas. En esta parte vemos el entono de desarrollo integrado con el que vamos a trabajar, Android Studio, y la plataforma de realidad aumentada que vamos a utilizar en nuestra aplicación, ARCore. Con esta información podemos probar ya una aplicación demostración que ofrece Android, entendiendo quizás ya mejor el objetivo del documento. Llegados a este punto aprendemos los conceptos más básicos del lenguaje de programación a usar, Java, y explicamos el código de la demostración para así, tener la base para poder hacer las modificaciones necesarias en este y crear nuestras propias aplicaciones.

Aquí es cuando llegamos a la parte del trabajo que realmente tiene un final claro, un objetivo; crear nuestra aplicación. A través de las explicaciones más claras y sencillas que se han podido mostrar, el lector puede seguirlas realizando diversas aplicaciones de realidad aumentada. Desde introducir nuestro propio modelo 3D en el mundo real a través de nuestro dispositivo, hasta lograr introducir animaciones o incluso varios modelos simultáneamente. No solo se ha enseñado un solo ejemplo sino que se han dado las pautas para la creación de multitud de combinaciones con los diferentes conceptos expuestos en las aplicaciones mostradas.

La conclusión más concisa es que se ha logrado el objetivo del trabajo. Además he aprendido no solo la creación de una aplicación en Android para la visualización de mis modelos 3D en realidad aumentada, si no también mejorado, entendido y profundizado en este mundo y esta tecnología de futuro.

10. BIBLIOGRAFÍA

[1] D. Escudero Mancebo, David. (2003) Fundamentos de Informática Gráfica, Editorial Ceysa.

[2] Olmedo H., Escudero, D., & Cardeñoso, V. (2015) Multimodal interaction with virtual worlds XMMVR: eXtensible language for MultiModal interaction with virtual reality worlds. Journal on Multimodal User Interfaces, 9 (3), 153-172.

[3] Azuma, RT (1997). Una encuesta de realidad aumentada. *Presencia: Teleoperadores y entornos virtuales*, *6* (4), 355-385.

[4] Milgram, P., Takemura, H., Utsumi, A., & Kishino, F. (1995, December). Augmented reality: A class of displays on the reality-virtuality continuum. In *Telemanipulator and telepresence technologies* (Vol. 2351, pp. 282-292). International Society for Optics and Photonics.

[5] Swan, J. E., & Gabbard, J. L. (2005, July). Survey of user-based experimentation in augmented reality. In *Proceedings of 1st International Conference on Virtual Reality* (Vol. 22, pp. 1-9).

[6] Feiner, S., MacIntyre, B., Höllerer, T., & Webster, A. (1997). A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. *Personal Technologies*, *1*(4), 208-217.

[7] Vlahakis, V., Ioannidis, M., Karigiannis, J., Tsotros, M., Gounaris, M., Stricker, D., ... y Almeida, L. (2002). Archeoguide: una guía de realidad aumentada para sitios arqueológicos. *IEEE Computer Graphics and Applications* , *22* (5), 52-60.

[8] M. Fiorentino, R. de Amicis, G. Monno, and A. Stork. Spacedesign: A mixed reality workspace for aesthetic industrial design. In [7], pp. 86–318.

[9] W. Friedrich. ARVIKA–augmented reality for development, production and service. In [7], pp. 3–6.

[10] M. Tönnis, C. Sandor, G. Klinker, C. Lange, and H. Bubb. Experimental evaluation of an augmented reality visualization for directing a car driver‟s attention. In [10], pp. 56–59.

[11] T. H. J. Collett and B. A. MacDonald. Developer oriented visualisation of a robot program: An augmented reality approach. In HRI'06: Proc. 1st Conf. on Human-Robot Interaction, pp. 49–56, Salt Lake City, Utah, USA, Mar. 2006. ACM Press. ISBN 1-59593-294-1.

[12] B. T. Schowengerdt, E. J. Seibel, J. P. Kelly, N. L. Silverman, and T. A. Furness III. Binocular retinal scanning laser display with integrated focus cues for ocular accommodation. In A. J. Woods, M. T. Bolas, J. O. Merritt, and S. A. Benton, editors, Proc. SPIE, Electronic Imaging Science and Technology, Stereoscopic Displays and Applications XIV, vol. 5006, pp. 1–9, Bellingham, WA, USA, Jan. 2003. SPIE Press.

[13] D. Willers. Augmented Reality at Airbus. http://www.ismar06.org/data/3a-Airbus.pdf, Oct. 22-25 2006. ISMAR‟06 industrial track

[14] Van Krevelen, D. y Poelman, R. (2007). Realidad aumentada: tecnologías, aplicaciones y limitaciones. *Vrije Univ. Amsterdam, Dep. Comput Sci* .

[15] P. Hasvold. In-the-field health informatics. In The Open Group Conf., Paris, France, 2002

[16] V. Buchmann, S. Violich, M. Billinghurst, and A. Cockburn. FingARtips: Gesture based direct manipulation in augmented reality. In [146], pp. 212–221.

[17] S. Kim and A. K. Dey. Simulated augmented reality windshield display as a cognitive mapping aid for elder driver navigation. In [119], pp. 133–142.

[18] C. Lindinger, R. Haring, H. Hörtner, D. Kuka, and H. Kato. Multi-user mixed reality system „Gulliver‟s World‟: a case study on collaborative edutainment at the intersection of material and virtual worlds. Virtual Reality, 10(2):109–118, Oct. 2006.

FitzGerald, E., Ferguson, R., Adams, A., Gaved, M., Mor, Y., y Thomas, R. (2013). Realidad aumentada y aprendizaje móvil: el estado del arte. *Revista internacional de aprendizaje móvil y mixto (IJMBL)* , *5* (4), 43-58.

Papagiannakis, G., Singh, G., & Magnenat-Thalmann, N. (2008). A survey of mobile and wireless technologies for augmented reality systems. *Computer Animation and Virtual Worlds*, *19*(1), 3-22.

Welch, G. y Foxlin, E. (2002). Encuesta de seguimiento de movimiento. *IEEE Computer graphics and Applications*, *22* (6), 24-38.

[19] Información sobre ARCore:

https://developers.google.com/ar/discover/

https://developers.google.com/ar/discover/concepts

https://elpoderdelandroideverde.com/arcore-que-es-y-para-que-sirve/

https://elandroidelibre.elespanol.com/2019/05/realidad-aumentada-arcore-mas-realista-mejoras-iluminacion.html

[20] Guía de Android Studio:

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

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

[21] Explicación de módulos, códigos, funcionamiento y otros:

https://developer.android.com

(Está página es la general de la que parten multitud de páginas de las que se ha sacado información. No menciono todas por la cantidad que son y que se ven reflejadas en el índice de esta página)

https://developers.google.com/ar/reference/java/sceneform/reference/packages

(Está página es la general de la que parten multitud de páginas de las que se ha sacado información. No menciono todas por la cantidad que son y que se ven reflejadas en el índice de esta página)

https://developers.google.com/ar/develop/java/quickstart

https://developers.google.com/ar/develop/java/sceneform/

https://javarevisited.blogspot.com/2017/01/how-public-static-final-variable-works.html

[22] Prueba de usabilidad

https://uxpanol.com/teoria/sistema-de-escalas-de-usabilidad-que-es-y-para-que-sirve/

https://www.trymyui.com/sus-system-usability-scale

(Basado en Bangor, Kortum, & Miller. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. Journal of Usability Studies 4(3), 114-123)

Páginas para información básica:

https://es.wikipedia.org/wiki/OpenGL

https://es.wikipedia.org/wiki/Android_Studio

APÉNDICE 1: Código de los ejemplos

Códigos del ejemplo del apartado 7.1: Introducir tu propio modelo 3D

En el apartado 7.1 veíamos como importar tu propio modelo 3D. En esta aplicación introducimos un objeto llamado “esfera1” y llamamos a la aplicación “Sphere”.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.google.ar.sceneform.samples.esfera">

    <uses-permission android:name="android.permission.CAMERA" />

    <uses-feature android:name="android.hardware.camera.ar"
android:required="true"/>

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="Sphere"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="false"
        tools:ignore="GoogleAppIndexingWarning">

        <meta-data android:name="com.google.ar.core" android:value="required" />

        <activity
            android:name=".HelloSceneformActivity"
            android:label="Sphere"
            android:configChanges="orientation|screenSize"
            android:theme="@style/Theme.AppCompat.NoActionBar"
            android:screenOrientation="locked"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

HelloSceneformActivity.java

```
package com.google.ar.sceneform.samples.esfera;

import android.app.Activity;
import android.app.ActivityManager;
import android.content.Context;
import android.os.Build;
import android.os.Build.VERSION_CODES;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.Gravity;
import android.view.MotionEvent;
import android.widget.Toast;
import com.google.ar.core.Anchor;
import com.google.ar.core.HitResult;
import com.google.ar.core.Plane;
import com.google.ar.sceneform.AnchorNode;
import com.google.ar.sceneform.rendering.ModelRenderable;
import com.google.ar.sceneform.ux.ArFragment;
import com.google.ar.sceneform.ux.TransformableNode;
import android.net.Uri;


public class HelloSceneformActivity extends AppCompatActivity {
  private static final String TAG =
HelloSceneformActivity.class.getSimpleName();
  private static final 116ndy116é MIN_OPENGL_VERSION = 3.0;

  private ArFragment arFragment;
  private ModelRenderable esfera1;


  @Override
  @SuppressWarnings({"AndroidApiChecker", "FutureReturnValueIgnored"})

  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    if (!checkIsSupportedDeviceOrFinish(this)) {
      return;
    }

    setContentView(R.layout.activity_ux);
    arFragment = (ArFragment)
getSupportFragmentManager().findFragmentById(R.id.ux_fragment);

    ModelRenderable.builder()
        .setSource(this, Uri.parse ("esfera1.sfb"))
        .build()
        .thenAccept(renderable -> esfera1 = renderable)
        .exceptionally(
            throwable -> {
              Toast toast =
```

```
                    Toast.makeText(this, "Unable to load 117ndy renderable",
Toast.LENGTH_LONG);
                toast.setGravity(Gravity.CENTER, 0, 0);
                toast.show();
                return null;
              });



      arFragment.setOnTapArPlaneListener(
        (HitResult hitResult, Plane plane, MotionEvent motionEvent) -> {
          if (esfera1 == null) {
            return;
          }

          Anchor anchor = hitResult.createAnchor();
          AnchorNode anchorNode = new AnchorNode(anchor);
          anchorNode.setParent(arFragment.getArSceneView().getScene());

          TransformableNode 117ndy = new
TransformableNode(arFragment.getTransformationSystem());
          117ndy.setParent(anchorNode);
          117ndy.select();
        });
  }


  public static boolean checkIsSupportedDeviceOrFinish(final Activity activity)
{
    if (Build.VERSION.SDK_INT < VERSION_CODES.N) {
      Log.e(TAG, "Sceneform requires Android N or later");
      Toast.makeText(activity, "Sceneform requires Android N or later",
Toast.LENGTH_LONG).show();
      activity.finish();
      return false;
    }
    String openGlVersionString =
        ((ActivityManager) activity.getSystemService(Context.ACTIVITY_SERVICE))
            .getDeviceConfigurationInfo()
            .getGlEsVersion();
    if (Double.parseDouble(openGlVersionString) < MIN_OPENGL_VERSION) {
      Log.e(TAG, "Sceneform requires OpenGL ES 3.0 later");
      Toast.makeText(activity, "Sceneform requires OpenGL ES 3.0 or later",
Toast.LENGTH_LONG)
          .show();
      activity.finish();
      return false;
    }
    return true;
  }
}
```

Gradle: app

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.google.ar.sceneform.samples.esfera"
        minSdkVersion 24
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {

    implementation "com.google.ar.sceneform.ux:sceneform-ux:1.8.0"

    implementation "com.android.support:appcompat-v7:28.0.0"
}

apply plugin: 'com.google.ar.sceneform.plugin'


sceneform.asset('sampledata/models/esfera1.obj',
        'default',
        'sampledata/models/esfera1.sfa',
        'src/main/assets/esfera1')
```

Códigos del ejemplo del apartado 7.2: Introducir varios modelos 3D simultáneamente

En el apartado 7.2 veíamos cómo introducir varios modelos 3D a la vez. En este ejemplo importamos dos modelos, uno llamado “Letras” y otro llamado “LogoEII”, la aplicación se llama “Logo EII”.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.google.ar.sceneform.samples.eii">

    <uses-permission android:name="android.permission.CAMERA" />

    <uses-feature android:name="android.hardware.camera.ar"
android:required="true"/>

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="Logo EII"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="false"
        tools:ignore="GoogleAppIndexingWarning">

        <meta-data android:name="com.google.ar.core" android:value="required" />

        <activity
            android:name=".HelloSceneformActivity"
            android:label="Logo EII"
            android:configChanges="orientation|screenSize"
            android:theme="@style/Theme.AppCompat.NoActionBar"
            android:screenOrientation="locked"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

HelloSceneformActivity.java

```
package com.google.ar.sceneform.samples.eii;

import android.app.Activity;
import android.app.ActivityManager;
import android.content.Context;
import android.os.Build;
import android.os.Build.VERSION_CODES;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.Gravity;
import android.view.MotionEvent;
import android.widget.Toast;
import com.google.ar.core.Anchor;
import com.google.ar.core.HitResult;
import com.google.ar.core.Plane;
import com.google.ar.sceneform.AnchorNode;
import com.google.ar.sceneform.rendering.ModelRenderable;
import com.google.ar.sceneform.ux.ArFragment;
import com.google.ar.sceneform.ux.TransformableNode;
import android.net.Uri;

public class HelloSceneformActivity extends AppCompatActivity {
  private static final String TAG =
HelloSceneformActivity.class.getSimpleName();
  private static final double MIN_OPENGL_VERSION = 3.0;

  private ArFragment arFragment;
  private ModelRenderable letras;
  private ModelRenderable logo;
  int elemento=1;

  @Override
  @SuppressWarnings({"AndroidApiChecker", "FutureReturnValueIgnored"})
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    if (!checkIsSupportedDeviceOrFinish(this)) {
      return;
    }

    setContentView(R.layout.activity_ux);
    arFragment = (ArFragment)
getSupportFragmentManager().findFragmentById(R.id.ux_fragment);

    ModelRenderable.builder()
        .setSource(this, Uri.parse ("Letras.sfb"))
        .build()
        .thenAccept(renderable -> letras = renderable)
        .exceptionally(
            throwable -> {
              Toast toast =
                  Toast.makeText(this, "Unable to load andy renderable",
```

```
Toast.LENGTH_LONG);
              toast.setGravity(Gravity.CENTER, 0, 0);
              toast.show();
              return null;
            });
    ModelRenderable.builder()
              .setSource(this, Uri.parse ("LogoEII.sfb"))
              .build()
              .thenAccept(renderable -> logo = renderable)
              .exceptionally(
                      throwable -> {
                          Toast toast =
                                  Toast.makeText(this, "Unable to load andy
renderable", Toast.LENGTH_LONG);
                          toast.setGravity(Gravity.CENTER, 0, 0);
                          toast.show();
                          return null;
                      });


      arFragment.setOnTapArPlaneListener(
        (HitResult hitResult, Plane plane, MotionEvent motionEvent) -> {
          if (letras == null) {
            return;
          }

          Anchor anchor = hitResult.createAnchor();
          AnchorNode anchorNode = new AnchorNode(anchor);
          anchorNode.setParent(arFragment.getArSceneView().getScene());

          TransformableNode andy = new
TransformableNode(arFragment.getTransformationSystem());
          andy.setParent(anchorNode);
            if (elemento%2==0) {
                andy.setRenderable(letras);
            }else{
                andy.setRenderable(logo);
            }
            elemento++;
          andy.select();
        });
  }

  public static boolean checkIsSupportedDeviceOrFinish(final Activity activity)
{
    if (Build.VERSION.SDK_INT < VERSION_CODES.N) {
      Log.e(TAG, "Sceneform requires Android N or later");
      Toast.makeText(activity, "Sceneform requires Android N or later",
Toast.LENGTH_LONG).show();
      activity.finish();
      return false;
    }
    String openGlVersionString =
        ((ActivityManager) activity.getSystemService(Context.ACTIVITY_SERVICE))
            .getDeviceConfigurationInfo()
```

```
            .getGlEsVersion();
    if (Double.parseDouble(openGlVersionString) < MIN_OPENGL_VERSION) {
      Log.e(TAG, "Sceneform requires OpenGL ES 3.0 later");
      Toast.makeText(activity, "Sceneform requires OpenGL ES 3.0 or later",
Toast.LENGTH_LONG)
          .show();
      activity.finish();
      return false;
    }
    return true;
  }
}
```

Gradle: app

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.google.ar.sceneform.samples.eii"

        minSdkVersion 24
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
    }

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
}

dependencies {

    implementation "com.google.ar.sceneform.ux:sceneform-ux:1.8.0"
    implementation "com.android.support:appcompat-v7:28.0.0"
}

apply plugin: 'com.google.ar.sceneform.plugin'


sceneform.asset('sampledata/models/Letras.obj',
        'default',
        'sampledata/models/Letras.sfa',
```

```
        'src/main/assets/Letras')

sceneform.asset('sampledata/models/LogoEII.obj',
        'default',
        'sampledata/models/LogoEII.sfa',
        'src/main/assets/LogoEII')
```

Códigos del ejemplo del apartado 7.3: Introducir un modelo animado 3D

En el apartado 7.3 veíamos cómo añadir animaciones continuas a un modelo 3D. En este ejemplo importamos una serie de archivos fbx que vimos en dicho apartado con los nombres de “cubocubo1”, “cubocubo2” y “cubocubo3” y la aplicación la hemos llamado “Cube”.

AndroidManifest.xml

```
<?xml                          version="1.0"                          encoding="utf-8"?>

<manifest                xmlns:android="http://schemas.android.com/apk/res/android"
          xmlns:tools="http://schemas.android.com/tools"
          package="com.google.ar.sceneform.samples.cubo">

    <uses-permission          android:name="android.permission.CAMERA"          />

    <uses-feature                          android:name="android.hardware.camera.ar"
android:required="true"/>

    <application
        android:allowBackup="false"
        android:icon="@drawable/ic_launcher"
        android:label="Cube"
        android:theme="@style/AppTheme"
        android:usesCleartextTraffic="false"
        tools:ignore="GoogleAppIndexingWarning">

        <meta-data  android:name="com.google.ar.core"  android:value="required"  />

        <activity                                          android:name=".MainActivity"
            android:label="Cube"
            android:configChanges="orientation|screenSize"
            android:exported="true">
            <intent-filter>
                <action                android:name="android.intent.action.MAIN"/>
                <category        android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

MainActivity.java

```
package com.google.ar.sceneform.samples.cubo;

import android.content.res.ColorStateList;
import android.os.Bundle;
import android.support.design.widget.FloatingActionButton;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.Gravity;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Toast;
import com.google.ar.core.Anchor;
import com.google.ar.core.HitResult;
import com.google.ar.core.Plane;
import com.google.ar.sceneform.AnchorNode;
import com.google.ar.sceneform.FrameTime;
import com.google.ar.sceneform.Node;
import com.google.ar.sceneform.SkeletonNode;
import com.google.ar.sceneform.animation.ModelAnimator;
import com.google.ar.sceneform.math.Quaternion;
import com.google.ar.sceneform.math.Vector3;
import com.google.ar.sceneform.rendering.AnimationData;
import com.google.ar.sceneform.rendering.ModelRenderable;
import com.google.ar.sceneform.ux.ArFragment;

public class MainActivity extends AppCompatActivity {

  private static final String TAG = "AnimationSample";
  private static final int ANDY_RENDERABLE = 1;
  private static final int HAT_RENDERABLE = 2;
  private static final String HAT_BONE_NAME = "hat_point";
  private ArFragment arFragment;
  private ModelLoader modelLoader;
  private ModelRenderable andyRenderable;
  private AnchorNode anchorNode;
  private SkeletonNode andy;
  private ModelAnimator animator;
  private int nextAnimation;
  private FloatingActionButton animationButton;
  private FloatingActionButton hatButton;
  private Node hatNode;
  private ModelRenderable hatRenderable;

  @Override
  @SuppressWarnings({"AndroidApiChecker", "FutureReturnValueIgnored"})
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    arFragment = (ArFragment)
getSupportFragmentManager().findFragmentById(R.id.sceneform_fragment);
```

```
    modelLoader = new ModelLoader(this);

    modelLoader.loadModel(ANDY_RENDERABLE, R.raw.cubocubo1);
    modelLoader.loadModel(HAT_RENDERABLE, R.raw.baseball_cap);

    arFragment.setOnTapArPlaneListener(this::onPlaneTap);


arFragment.getArSceneView().getScene().addOnUpdateListener(this::onFrameUpdate);

    animationButton = findViewById(R.id.animate);
    animationButton.setEnabled(false);
    animationButton.setOnClickListener(this::onPlayAnimation);

    hatButton = findViewById(R.id.hat);
    hatButton.setEnabled(false);
    hatButton.setOnClickListener(this::onToggleHat);
  }

  private void onPlayAnimation(View unusedView) {
    if (animator == null || !animator.isRunning()) {
      AnimationData data = andyRenderable.getAnimationData(nextAnimation);
      nextAnimation = (nextAnimation + 1) %
andyRenderable.getAnimationDataCount();
      animator = new ModelAnimator(data, andyRenderable);
      animator.start();
      Toast toast = Toast.makeText(this, data.getName(), Toast.LENGTH_SHORT);
      Log.d(
          TAG,
          String.format(
              "Starting animation %s - %d ms long", data.getName(),
data.getDurationMs()));
      toast.setGravity(Gravity.CENTER, 0, 0);
      toast.show();
    }
  }

  private void onPlaneTap(HitResult hitResult, Plane unusedPlane, MotionEvent
unusedMotionEvent) {
    if (andyRenderable == null || hatRenderable == null) {
      return;
    }
    Anchor anchor = hitResult.createAnchor();

    if (anchorNode == null) {
      anchorNode = new AnchorNode(anchor);
      anchorNode.setParent(arFragment.getArSceneView().getScene());

      andy = new SkeletonNode();

      andy.setParent(anchorNode);
      andy.setRenderable(andyRenderable);
      hatNode = new Node();

      Node boneNode = new Node();
```

```
      boneNode.setParent(andy);
      andy.setBoneAttachment(HAT_BONE_NAME, boneNode);
      hatNode.setRenderable(hatRenderable);
      hatNode.setParent(boneNode);
      hatNode.setWorldScale(Vector3.one());
      hatNode.setWorldRotation(Quaternion.identity());
      Vector3 pos = hatNode.getWorldPosition();

      pos.y -= .1f;

      hatNode.setWorldPosition(pos);
    }
  }

  private void onFrameUpdate(FrameTime unusedframeTime) {

    if (anchorNode == null) {
      if (animationButton.isEnabled()) {

animationButton.setBackgroundTintList(ColorStateList.valueOf(android.graphics.Co
lor.GRAY));
        animationButton.setEnabled(false);

hatButton.setBackgroundTintList(ColorStateList.valueOf(android.graphics.Color.GR
AY));
        hatButton.setEnabled(false);
      }
    } else {
      if (!animationButton.isEnabled()) {
        animationButton.setBackgroundTintList(
            ColorStateList.valueOf(ContextCompat.getColor(this,
R.color.colorAccent)));
        animationButton.setEnabled(true);
        hatButton.setEnabled(true);
        hatButton.setBackgroundTintList(
            ColorStateList.valueOf(ContextCompat.getColor(this,
R.color.colorPrimary)));
      }
    }
  }

  private void onToggleHat(View unused) {
    if (hatNode != null) {
      hatNode.setEnabled(!hatNode.isEnabled());

      if (hatNode.isEnabled()) {
        hatButton.setBackgroundTintList(
            ColorStateList.valueOf(ContextCompat.getColor(this,
R.color.colorPrimary)));
      } else {
        hatButton.setBackgroundTintList(
            ColorStateList.valueOf(ContextCompat.getColor(this,
R.color.colorAccent)));
      }
    }
```

```
  }

  void setRenderable(int id, ModelRenderable renderable) {
    if (id == ANDY_RENDERABLE) {
      this.andyRenderable = renderable;
    } else {
      this.hatRenderable = renderable;
    }
  }

  void onException(int id, Throwable throwable) {
    Toast toast = Toast.makeText(this, "Unable to load renderable: " + id,
Toast.LENGTH_LONG);
    toast.setGravity(Gravity.CENTER, 0, 0);
    toast.show();
    Log.e(TAG, "Unable to load andy renderable", throwable);
  }
}
```

Gradle: app

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.google.ar.sceneform.samples.cubo"
        minSdkVersion 24
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
        }
    }
    compileOptions {
        targetCompatibility 1.8
        sourceCompatibility 1.8
    }
}

dependencies {
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'

    implementation "com.google.ar.sceneform.ux:sceneform-ux:1.10.0"
    implementation "com.google.ar.sceneform:animation:1.10.0"
    implementation 'com.android.support:design:28.0.0'
}
```

```
apply plugin: 'com.google.ar.sceneform.plugin'

sceneform.asset('sampledata/models/cubocubo1.FBX',
        'default',
        'sampledata/models/cubocubo1.sfa',
        'src/main/res/raw/cubocubo1',
        ['sampledata/models/cubocubo3.FBX', 'sampledata/models/cubocubo1.FBX',
'sampledata/models/cubocubo2.FBX'])
```