SISTEMA DE REALIDAD AUMENTADA PARA LA VISUALIZACIÓN DE MODELOS REALES

Universidad Politécnica de Valencia

Departamento de Sistemas Informáticos y Computación

Trabajo Final De Máster

Máster Universitario en Inteligencia Artificial,

Reconocimiento de Formas e Imagen Digital

Enrique Trilles Andreo

Supervisado por:

M. Carmen Juan Lizandra

Septiembre 2014

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

Capítulo 1

INTRODUCCIÓN

1.1.- Motivación

En los últimos años, la popularidad de las aplicaciones y herramientas que se apoyan en la realidad aumentada ha experimentado un elevado crecimiento. La Realidad Aumentada (RA) permite generar y ofrecer al usuario una gran cantidad de contenido e información adicional. Son muchos y muy variados los trabajos que existen relacionados con este campo [VAN 10]. Además de ser un tema de gran interés no solo dentro del campo científico, sino del técnico, educacional y del entretenimiento.

Como se podrá observar en el siguiente apartado, existen ya una gran cantidad de casos en los que se han aplicado con éxito tecnologías y herramientas basadas en RA. Mediante la utilización de diversos tipos de marcadores (Figura 1.1), se puede realizar el visionado de todo tipo de modelos 3D y animaciones. Por otra parte, se ha observado que los modelos que se muestran en los marcadores no pueden cambiarse por otros en tiempo real. Esto es debido a que, por defecto, dichos modelos deben compilarse junto a la aplicación en el momento de su programación. Hecho que provoca que, tanto la diversidad de modelos para visualizar, como la utilidad de las aplicaciones de visionado de modelos 3D, sean bastante reducidas.

Figura 1.1 – Marcador de Realidad Aumentada y efecto de visualización

Al observar algunas herramientas de desarrollo y modelado 3D, se puede apreciar la existencia de los llamados importadores y exportadores de ficheros. Subproceso que se encargan de abrir y guardar modelos en ficheros de diversos formatos. Sabiendo de la existencia de estas herramientas, se plantea la idea de utilizar dichos importadores en una herramienta basada en realidad aumentada, para poder mostrar y cambiar de forma dinámica cualquier tipo de modelo, en cualquier momento.

El poder cargar cualquier modelo dentro de un listado de ficheros, podría acarrear un problema grave, relacionado con la memoria interna del dispositivo móvil sobre el que se ejecute una aplicación de estas características. Aunque a día de hoy muchos dispositivos móviles

permiten ampliaciones de la memoria interna mediante tarjetas de memoria, esto no siempre tiene que ser así. Es por ello que surge la idea de intentar separar la aplicación de visionado de modelos de la base de datos que contenga los modelos en dos máquinas diferentes.

Con estas dos ideas en mente, se va a desarrollar un par de aplicaciones que sean capaces de comunicarse entre sí para listar y descargar ficheros, además de abrirlo e interpretarlos para mostrarlos por pantalla mediante una herramienta basada en realidad aumentada.

1.2.- Objetivos del proyecto

El objetivo del proyecto es el de crear un par de aplicaciones, una a modo de servidor, y otra a modo de cliente, que sean capaces de comunicarse y transmitir ficheros entre ellas para llevar a cabo el visionado de objetos 3D mediante realidad aumentada.

El servidor debe ser capaz de escanear una carpeta a nuestra elección, listar los ficheros que se encuentran en la misma, y enviárselos al cliente bajo demanda del mismo. Además, como factor imprescindible, el contenido de la carpeta debe poder ser modificable en todo momento, haciendo la herramienta lo más dinámica posible, y con el añadido de que no sea necesario volver a iniciar o a compilar la aplicación para su correcto funcionamiento. Por otra parte, el cliente debe ser capaz de obtener el listado de dicho directorio y visualizarlo por pantalla. Además, el usuario deberá poder elegir qué ficheros desea descargar. La aplicación debe permitir también el visionado de uno o varios modelos, a elección del usuario, y mostrarlos a la vez por pantalla con la ayuda de un único marcador. Todo esto teniendo en cuenta, por supuesto, las limitaciones a nivel de hardware del dispositivo sobre el que se ejecute la aplicación.

Para el desarrollo del sistema se ha decidido hacer uso de las siguientes herramientas y tecnologías: Unity 3D, Qualcomm Vuforia, MonoDevelop y el kit de desarrollo HelixToolkit.

Para cumplir los objetivos del proyecto, se han seguido estos pasos:

1) Sistema de conexión a red local mediante modelo cliente-servidor.
   Se va a hacer uso de un sistema de conexión basado en sockets TCP, programado mediante el lenguaje C#. En total, se generarán un par de librerías, una para cada aplicación, que contengan todo lo necesario para permitir la conexión y comunicación entre las dos aplicaciones.
2) Aplicación servidor.
   Desarrollada con la ayuda de Visual Studio 2013, esta aplicación hará uso de un sistema basado en multiproceso para permitir una conexión con el cliente lo más rápida y efectiva. Otro aspecto a tener en cuenta es el de poder listar los elementos de un directorio externo, a petición del cliente, sin que entorpezca la modificación de dicho directorio durante la ejecución de la aplicación.

3) Aplicación cliente.
   Esta aplicación se va a desarrollar con el entorno de desarrollo de Unity, y el sistema operativo de destino va a ser Android. El objetivo es que esta aplicación pueda ser ejecutada en dispositivos móviles sin ningún tipo de problema. La aplicación debe ser capaz de conectarse a un red local LAN y comunicarse con el servidor para solicitar y recibir ficheros con los modelos 3D que se van a visualizar. Para visualizar dichos modelos, se va a desarrollar un sistema de importación de los mismos, basándose de los importadores contenidos en el kit de desarrollo de HelixToolkit.

Para finalizar, se realizarán dos estudios del visor implementado. Un estudio se centrará en los tiempos de carga de diferentes modelos tridimensionales, así como de formatos de fichero, en varios dispositivos. El segundo estudio consistirá en analizar los fotogramas por segundo de la aplicación a medida que se van aumentado el número de vértices y polígonos en pantalla. El objetivo es evaluar el rendimiento de la aplicación en dichos dispositivos, así como intentar apreciar cuántos modelos se pueden visualizar sin que el funcionamiento de los dispositivos se vea afectado.

1.3.- Estructura

El resto de capítulos presenta la siguiente estructura:

Capítulo 2: Estado del arte y situación de la Realidad Aumentada.

Capítulo 3: Presentación de las herramientas, aplicaciones y librerías usadas, así como una introducción a los diferentes términos que se utilizarán a lo largo de todo el documento.

Capítulo 4: Explicación detallada y paso por paso de todas las tareas llevadas a cabo para la realización de la aplicación.

Capítulo 5: Explicación de las pruebas llevadas a cabo sobre la aplicación implementada, así como los resultados obtenidos. Extracción de conclusiones a las mismas.

Capítulo 6: Conclusiones y posibles ampliaciones y/o mejoras de la aplicación desarrollada.

Capítulo 2

ESTADO DEL ARTE

Desde los primeros casos en los que empieza a aparecer la realidad aumentada, hasta los últimos años, muchas han sido los campos de desarrollo y áreas investigación que han intentado hacer uso de esta tecnología. Han sido creadas todo tipo de herramientas de ayuda a profesionales en campos como la medicina o reparación de máquinas de carácter industrial, hasta detección de enfermedades y patologías, educación, así como entretenimiento y videojuegos. A continuación se presentan diversos casos y estudios que han sido presentados en los últimos años.

En un survey realizado sobre el estado de la realidad aumentada [VAN 10], sus autores nos presentan los diferentes tipos de dispositivos y visualizadores existentes para producir efectos de realidad aumentada. A parte de explicar en qué consisten cada uno de ellos, realiza una tabla comparativa mostrando sus características más destacables, así como los problemas y molestias de los mismos (Figura 2.2).

Los dispositivos utilizados para el proyecto entran dentro de la categoría de dispositivos de mano. Según Van Krevelen & Poelman [VAN 10], estos dispositivos son los que van a permitir una mayor inserción de las tecnologías de realidad aumentada en el mercado. Esto es debido a los bajos costes de producción y facilidad de uso, además de contar con una gran aceptación del público general. Una aplicación existente del uso de estos dispositivos es la presentada en una PDA que se utiliza a modo de asistente para la navegación [ZHOU 08] (Figura 2.1).

Figura 2.1 – PDA como sistema de apoyo a navegación

Dentro del campo de la educación, se pueden encontrar casos como el presentado por Mehmet Kesim [MEHM 12], en el que se hace un estudio de las diversas técnicas basadas en realidad aumentada, y su posible aplicación a la hora de mejorar y combinar diversas técnicas de enseñanza en entornos cotidianos. Se asegura que entre los diversos dispositivos de RA existentes, las tablets y smartphones podrían convertirse en los aparatos más importantes para propagar diversas técnicas y aplicaciones basadas en RA. Durante la *Escola d'Estiu* que se realiza en la Universidad Politécnica de Valencia, en el año 2010 se llevó a cabo una actividad en la que una serie de niños participaron en un juego basado en RA [JUAN 10] en el que se les enseñaba acerca de los animales en peligro de extinción. El objetivo del estudio era evaluar la viabilidad del uso de sistemas basados en RA para la enseñanza mediante un juego de carácter didáctico (Figura 2.3).

Positioning	Head-worn				Hand-held	Spatial		
Technology	**Retinal**	**Optical**	**Video**	**Projective**	**All**	**Video**	**Optical**	**Projective**
Mobile	+	+	+	+	+	–	–	–
Outdoor use	+	±	±	+	±	–	–	–
Interaction	+	+	+	+	+	Remote	–	–
Multi-user	+	+	+	+	+	+	Limited	Limited
Brightness	+	–	+	+	Limited	+	Limited	Limited
Contrast	+	–	+	+	Limited	+	Limited	Limited
Resolution	Growing	Growing	Growing	Growing	Limited	Limited	+	+
Field-of-view	Growing	Limited	Limited	Growing	Limited	Limited	+	+
Full-colour	+	+	+	+	+	+	+	+
Stereoscopic	+	+	+	+	–	–	+	+
Dynamic refocus (eye strain)	+	–	–	+	–	–	+	+
Occlusion	±	±	+	Limited	±	+	Limited	Limited
Power economy	+	–	–	–	–	–	–	–
Opportunities	Future dominance	Current dominance			Realistic, mass-market	Cheap, off-the-shelf	Tuning, ergonomics	
Drawbacks		Tuning, tracking	Delays	Retro-reflective material	Processor, Memory limits	No see-through metaphor	Clipping	Clipping, shadows

Figura 2.2 – Tabla de comparación de pantallas de visualización de RA

El estudio demostró que el sistema basado en RA puede mejorar la enseñanza, además de haber resultado mucho más entretenido y divertido de utilizar que el juego original.

Figura 2.3 – Cubos de RA frente a los cubos del juego original

Existen varios estudios en los que se ha analizado la relevancia de la RA como herramienta para la enseñanza. Como por ejemplo en Furió et al. [FURIO 01]. Furió et al. Desarrollaron un juego cuyo objetivo era transmitir conocimiento sobre multiculturalidad, solidaridad y tolerancia. En dicho estudio se analiza la viabilidad de usar estos métodos basados en RA, así como su impacto en los niños. Los resultados del estudio dieron como conclusión que, por una parte, no hubo diferencias en cuanto a aprendizaje entre los niños que usaron un juego tradicional y los que usaron un juego para iPhone con Realidad Aumentada. Pero por otra parte, quedó patente que casi todos los niños preferían el juego de RA sobre el juego tradicional (Figura 2.4).

Figura 2.4 – Imágenes del juego y dispositivo

Otro estudio que también se llevó a cabo dentro de este ámbito consistía en comprobar si usar diferentes tipos de dispositivos para el aprendizaje mediante juegos interactivos era determinante en el factor de aprendizaje [FURIO 02]. El objetivo del juego desarrollado en este caso era reforzar en los niños los conocimientos relacionados con el ciclo del agua. Se obtuvo como resultado que las diferencias entre peso y tamaño de las pantallas de los dispositivos utilizados no era determinante a la hora de aumentar o disminuir el factor de aprendizaje (Figura 2.5). Por otra parte, el juego basado en RA

obtuvo una gran aceptación entre los niños del estudio, sugiriendo que juegos de este tipo pueden ser un método muy apropiado como refuerzo y ayuda a la enseñanza.

Figura 2.5 – Dispositivos usados en el juego

Otro estudio dentro del campo de la enseñanza, busca analizar el impacto de la realidad aumentada a la hora de motivar a los estudiantes [DISE 13]. El estudio se realizó en una escuela de arte de Madrid, donde se compararon dos métodos de enseñanza; basado en diapositivas y basado en sistema de RA. El estudio demostró que el uso del sistema de RA mantuvo a los estudiantes más motivados y concentrados en la temática sobre la que se impartió clase que con el método convencional de enseñanza (Figura 2.6). En las encuestas que se realizaron a los estudiantes estos afirmaron que el método de RA resultaba más interesante y entretenido que el método de diapositivas convencional.

Figura 2.6 – Modelo de RA para el estudio de motivación en clases de arte

Dentro del campo de le mecánica industrial, se presenta un proyecto en el que se hace uso de dispositivos de RA para crear una aplicación experimental que pueda ser de ayuda a los profesionales que se dedican a la reparación de máquinas industriales pesadas, ya sea en entornos abiertos o cerrados [HEND 11]. Mediante la utilización de un casco de visión con detectores de posición, se desea añadir información extra, así como etiquetas y funcionalidades de las diferentes piezas o máquinas sobre las que se estén realizando operaciones de mantenimiento (Figura 2.7). Las conclusiones

obtenidas tras el estudio de dicha herramienta indican que la información extra proporcionada por la herramienta ayudó a reducir el tiempo de reparación de las máquinas, sobre todo en la parte de búsqueda de elementos y procesos a realizar.

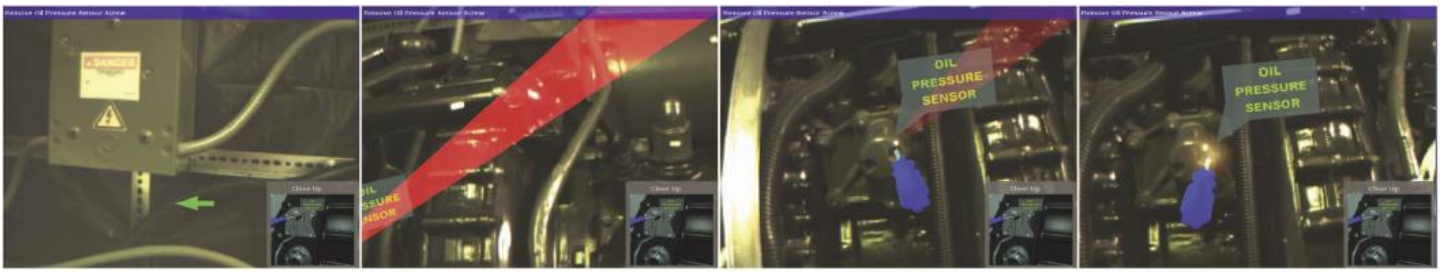

Figura 2.7 – Capturas de pantalla de la herramienta de apoyo al mantenimiento

También existen casos dentro del campo de la medicina donde se ha hecho uso de herramientas basadas en realidad aumentada. El caso que se presenta a continuación consiste en un simulador virtual desarrollado para el entrenamiento de cirujanos. De todos los componentes que conforman el sistema, se va a destacar el uso del sistema de visualización, que consiste en unas gafas de visión con posibilidad de visionado mediante RA (Figura 2.8). Con este simulador se espera entrenar al personal médico pertinente antes de pasar a realizar pruebas con pacientes reales, reduciendo los costes de entrenamiento y los riesgos de cara al paciente. Después de una exhaustiva evaluación del sistema, superó con creces las expectativas de los desarrolladores, aunque aún tendrían que mejorar la precisión de la máquina. Esto es debido al tamaño de las partes del cuerpo humano, que son muy pequeñas y requieren de un equipo de muestreo muy preciso y concreto.

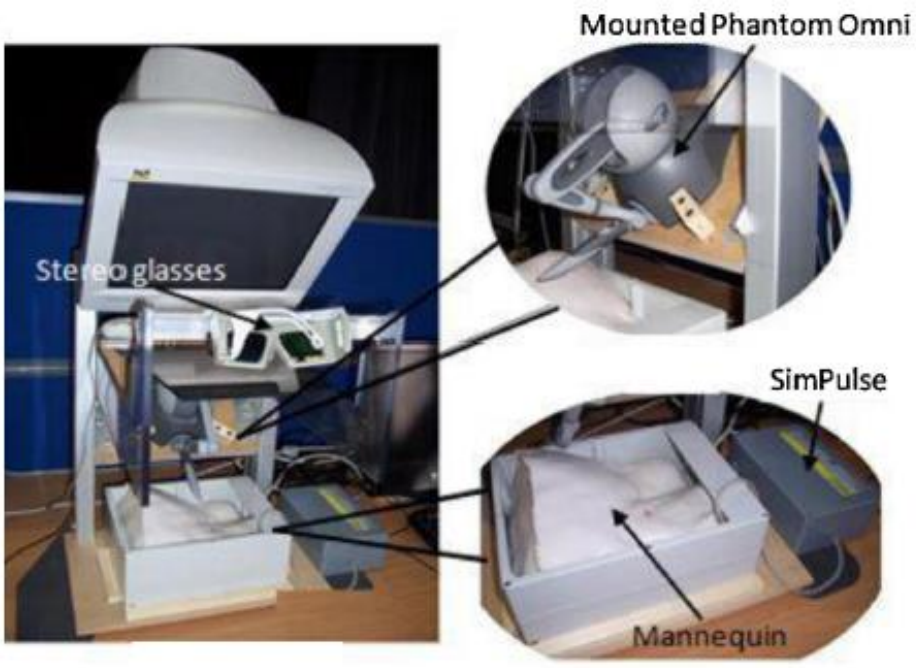

Figura 2.8 – Aspecto del simulador virtual de entrenamiento

Dentro del campo de la psicología se puede hablar de otro caso, en el que se ha realizado un estudio sobre el tratamiento de fobia a los animales pequeños e insectos [JUAN 05]. En este caso, se plantea el uso de sistemas basados en RA para simular algunos de los temores de dichos pacientes, como puede ser el temor hacia las arañas o las cucarachas (Figura 2.9). En el estudio llevado a cabo sobre el tratamiento de fobias hacia los animales pequeños, se pudo apreciar como en todos los pacientes que participaron en las pruebas se redujo de manera considerable el temor hacia dichos animales a los que tenían miedo. Los resultados mostraron que pacientes que en un principio eran incapaces de acercarse a estos animales, después del tratamiento fueron capaz de matar muchos de ellos, enfrentándose a sus temores.

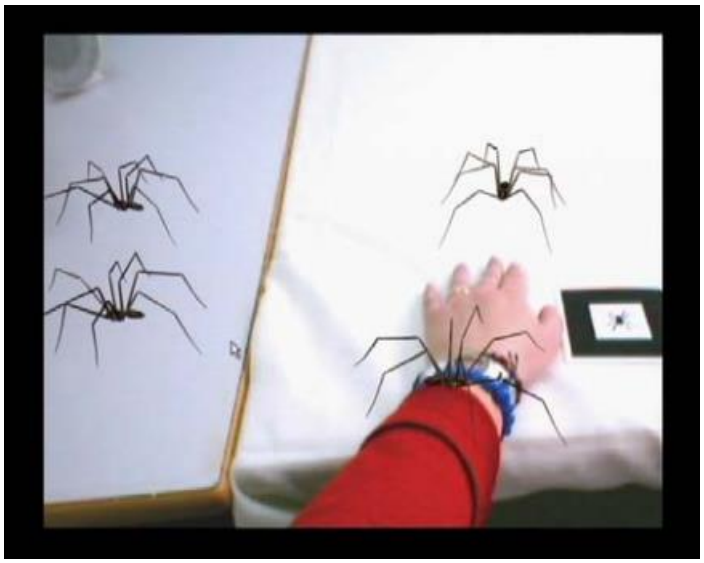

Figura 2.9 – Visualización de arañas mediante RA

Dentro del campo del entretenimiento, se va a destacar el caso de la saga de videojuegos para Playstation Vita llamada Invizimals, del estudio español Novarama [NOVA 14]. Dicho juego gira en torno la realidad aumentada, para dar vida a una serie de criaturas que el jugador podrá capturar y entrenar para luchar contra otros jugadores (Figura 2.10).

Esta saga de juegos ha tenido una gran aceptación entre el público, y se cuenta como uno de los juegos más vendidos e importantes de la consola portátil de Sony, hasta el punto que en la actualidad la empresa desarrolladora tiene firmado un contrato de exclusividad con Sony, pasando a convertirse en una empresa que desarrolla en exclusividad para Playstation.

Figura 2.10 – Captura de pantalla del juego Invizimals de PSVita

Capítulo 3

HERRAMIENTAS Y TECNOLOGÍAS

En este apartado se va a realizar una presentación de las herramientas y tecnologías que se han utilizado y forman parte, de forma directa o indirecta, con el desarrollo del sistema que se ha implementado en el presente proyecto.

3.1.- Unity 3D

La herramienta Unity es un sistema de desarrollo de videojuegos multiplataforma desarrollado por Unity Technologies [UNIT 14]. Incluye un motor gráfico propio así como un entorno de desarrollo integrado. Unity se utiliza para desarrollar principalmente videojuegos, aunque no se limita únicamente a esto, pudiéndose desarrollar todo tipo de aplicaciones gráficas. A día de hoy, Unity permite desarrollar aplicaciones para más de quince plataformas, entre las que se pueden incluir Android, iOS, así como todas las plataformas de videojuegos más importantes de empresas como Microsoft, Sony y Nintendo. Destacar además, que en el caso de Nintendo Wii U, Unity es además la plataforma de desarrollo base. Unity se encuentra en el mercado en dos versiones: Una versión gratuita llamada Unity Free, y otra versión completa y de pago denominada Unity Pro. Tanto Unity como Unity Pro incluyen el entorno de desarrollo, tutoriales, ejemplos de proyectos y de contenido, soporte a través de foros, wiki, y las actualizaciones futuras de la versión principal que se esté utilizando.

La primera versión de Unity se lanzó en la Conferencia Mundial de Desarrolladores de Apple en 2005. Fue construido solamente para funcionar y generar proyectos en los equipos de la plataforma Mac y obtuvo el éxito suficiente como para continuar con el desarrollo del motor y herramientas. Una nueva versión de Unity (Unity 3), fue lanzada en septiembre de 2010 y se centró en empezar a introducir más herramientas que los estudios de alta gama por lo general tienen a su disposición, con el fin de captar el interés de los desarrolladores más grandes, mientras que proporciona herramientas para equipos independientes y más pequeñas que normalmente serían difíciles de conseguir en un paquete asequible. La última versión de Unity (Unity 4), fue lanzada a finales de 2012, e incluye añadidos como animación de mecanismos, soporte para DirectX 11 y soporte para juegos en Linux.

Una de las propiedades más interesantes e importantes de Unity es su habilidad de poder exportar a múltiples plataformas un mismo proyecto, sin necesidad de realizar conversiones de código u operaciones muy complejas. Unity puede usarse junto a otras plataformas de desarrollo, como pueden ser: 3ds Max, Maya, Softimage, Blender, Modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks y Allegorithmic Substance. Otra de las propiedades más interesantes que ofrece Unity consiste en que los cambios realizados a los objetos creados con estos productos se actualizan

automáticamente en todas las instancias de ese objeto durante todo el proyecto, sin necesidad de tener que volver a realizar una importación de los mismos.

El motor gráfico de Unity soporta múltiples plataformas. Puede utilizar Direct3D (en Windows), OpenGL (en Mac y Linux), OpenGL ES (en Android y iOS), e interfaces propietarias en función de cada plataforma (Wii, Playstation, XBox). Unity también ofrece soporte para las siguientes tecnologías: Bump Mapping, Reflection Mapping, Parallax Mapping, Screen Space Ambient Occlusion (SSAO), sombras dinámicas mediante Shadow Mapping, así como renderizado de texturas y efectos de post-procesado en tiempo real.

Para la creación de shadders, Unity hace uso de ShaderLab language. Este lenguaje soporta diversos métodos de programación de shadders, además de dar soporte a otros lenguajes, como GLSL y Cg. Una de las características más importantes es la de permitir en un mismo shadder diversas variaciones del mismo, permitiendo al entorno poder escoger cuál de dichas variaciones es la más correcta y óptima en función del procesador gráfico (GPU) de la plataforma para la que se está desarrollando, ofreciendo así una gran compatibilidad.

A la hora de programar scripts y aplicaciones, Unity se apoya en Mono, una implementación libre y de código abierto de .NET Framework. Los lenguajes de programación que puede soportar Unity son C#, Boo y UnityScript (un lenguaje desarrollado para Unity basado en ECMAScript).

3.2.- MonoDevelop y .NET Framework

MonoDevelop es un entorno de desarrollo integrado libre y gratuito, diseñado principalmente para C# y otro lenguajes de programación basados en .NET como Nemerle, Boo, Java y, a partir de la versión 2.2, Python. MonoDevelop comenzó como una adaptación de otro entorno de desarrollo llamado SharpDevelop, con la intención de dar mayor soporte a la herramienta GTK+ o GimpToolkit. Dicha herramienta consiste en un conjunto de librerías multiplataforma para el desarrollo de interfaces gráficas de usuario [MONO 14].

El entorno de desarrollo integrado de MonoDevelop incluye manejo de clases, ayuda incorporada, opciones para auto completar código, un diseñador de interfaces integrado llamado Stetic, soporte para proyectos y un depurador de código integrado. Esta plataforma ofrece soporte completo para Linux, Windows y Mac, convirtiéndose en un entorno de desarrollo multiplataforma.

El framework de desarrollo .NET es una plataforma desarrollada por Microsoft que hace un especial énfasis en transparencia de redes, que presenta una independencia de plataforma a nivel de hardware y que permite un rápido desarrollo de aplicaciones. Basándose en este framework, Microsoft busca desarrollar una estrategia horizontal que sea capaz de integrar todos sus productos, abarcando desde

el sistema operativo hasta las herramientas más básicas, si como sus propias herramientas de mercado [NET 13].

Se puede considerar pues, que .NET es la respuesta de Microsoft al creciente mercado de los negocios en entornos web, como competencia a las diversas plataformas de desarrollo existentes, como puede ser Java de Oracle Corporation y los diversos entornos de desarrollo web basados en PHP. El objetivo de Microsoft es el de conseguir una manera rápida y lo más económica posible, a la vez que robusta y segura, para desarrollar aplicaciones. De esta forma, se intenta buscar una integración más rápida y ágil entre empresas y un acceso más simple y universal a todo tipo de información desde cualquier tipo de dispositivo.

Para poner en funcionamiento todo el Framework, .NET se apoya en el llamado Common Language Runtime [CLR 01]. La herramienta de desarrollo compila el código fuente de cualquiera de los lenguajes soportados por .NET (C++, C#, Visual Basic, entre otros) en un código intermedio, llamado Common Itermediate Language (CIL). Para generar dicho código, el compilador se basa en una especificación universal para determinar las reglas de traducción necesarias para crear el código CIL compatible. Para ejecutar el desarrollo inicial, es necesario realizar un segundo paso. Mediante un compilador en tiempo de ejecución (JIT) se genera el código máquina final necesario para realizar la ejecución final del programa. Dicho código generado se almacena en la memoria caché del ordenador, siendo recompilado de nuevo únicamente si se producen cambios en el código fuente.

Figura 3.2.1 – Diagrama básico de la Biblioteca de Clases Base

La biblioteca de clases base de .NET se encuentra dividida en cuatro sub-conjuntos clave: ASP .NET y servicios XML web, Windows Forms, ADO .NET y .NET (Figura 3.2.1). Dichas bibliotecas se encargan de manejar todas las operaciones básicas involucradas en el desarrollo de aplicaciones, incluyendo entre otras, acciones tales como: Interacción con dispositivos periféricos, manejo de bases de datos, administración de memoria, cifrado de datos, transmisión y recepción de datos, manejo y gestión de excepciones, y un largo etcétera.

3.3.- Vuforia Qualcomm y Realidad Aumentada

La realidad aumentada es el término que se usa para definir una visión a través de un dispositivo tecnológico, de forma directa o indirecta, de un entorno físico del mundo real, cuyos elementos se combinan con elementos virtuales para la creación de una realidad mixta en tiempo real. Consiste en un conjunto de dispositivos que añaden información virtual a la información física ya existente, es decir, se añade una parte sintética virtual a lo real. Ésta es la principal diferencia con la realidad virtual, puesto que no sustituye la realidad física, sino que añade datos informáticos sobreponiéndolos a los datos del mundo real.

Con la ayuda de la tecnología (por ejemplo, añadiendo la visión por computador y reconocimiento de objetos) la información sobre el mundo real alrededor del usuario se convierte en interactiva y digital. La información artificial sobre el medio ambiente y los objetos periféricos puede ser almacenada y recuperada como una capa de información en la parte superior de la visión del mundo real.

Los dispositivos de Realidad Aumentada pueden constar de varios sistemas de localización espacial y de un sistema de visualización para mostrar al usuario la información virtual que se añade a la real. Los sistemas de localización pueden llevar incorporados también sistemas de GPS, además de diversos tipos de sensores para que el dispositivo sea capaz de obtener tanto la posición como la orientación del usuario.

Los dos principales sistemas de visualización empleados son la pantalla óptica transparente (Optical See-through Display) y la pantalla de mezcla de imágenes (Video see-through Display). Tanto uno como el otro muestran imágenes virtuales que son mezcladas con la realidad o bien proyectadas directamente en la pantalla al usuario.

Los sistemas de realidad aumentada pueden utilizar una o más de las siguientes tecnologías: cámaras digitales, sensores ópticos, acelerómetros, GPS, giroscopios, brújulas de estado sólido, RFID, etc. Los sistemas de cámaras basados en Realidad Aumentada requieren de una unidad CPU potente y una cantidad de memoria RAM suficiente para poder procesar las imágenes de dichas cámaras. La combinación de todos estos elementos se dan a menudo en muchos smartphones, que los convierten en una posible plataforma de Realidad Aumentada. Existen a día de hoy diversas herramientas para programar y desarrollar aplicaciones basadas en realidad aumentada, como pueden ser ARToolKit, ATOMIC Authoring tool, o la que se ha utilizado en el trabajo que se presenta, Vuforia Qualcomm.

Vuforia consiste en un kit de desarrollo de software SDK basado en realidad aumentada para dispositivos móviles que permite la creación de todo tipo de aplicaciones de realidad aumentada [VUFO 01]. El kit posee herramientas basadas en visión por computador para reconocer marcadores, buscar y posicionar imágenes o elementos 3D básicos en tiempo real. Estas capacidades de búsqueda y registro permiten a los desarrolladores posicionar y orientar todo tipo de objetos virtuales, como pueden ser modelos 3D o cualquier tipo de objeto multimedia en relación con el mundo

real. De esta forma, se puede conseguir recrear la sensación de que un objeto virtual se encuentra en un entorno real, al observarlo a través de la cámara de un dispositivo móvil. Vuforia también es capaz de detectar cambios en tiempo real en el posicionamiento de los marcadores o la cámara del dispositivo, aplicando las transformaciones necesarias al objeto virtual para mantener la sensación de que el objeto existe en la realidad en todo momento.

El kit de desarrollo de Vuforia soporta una gran variedad de imágenes 2D y 3D a modo de marcadores, además de ser capaz de detectar imágenes que no tienen por qué cumplir dicha función, así como la detección de múltiples marcadores. Otras características que posee el SDK pueden ser como la detección automática de oclusión parcial de los marcadores o la selección, cambio y configuración de los marcadores en tiempo real.

Vuforia ofrece soporte a diversos lenguajes de programación, como C++, Java, Objective-C y el entorno de desarrollo .NET a través de la herramienta Unity Game Engine. De esta forma, se ofrece soporte nativo al desarrollo de aplicaciones para Android y iOS, así como a muchas otras plataformas, a través del entorno de desarrollo de Unity. Cualquier aplicación de realidad aumentada desarrollada con Vuforia será compatible con gran rango de terminales móviles, que estén basados en iOS o Android, así como Windows Phone.

3.4.- HelixToolkit

El kit de desarrollo HelixToolKit consiste un conjunto de herramientas y librerías para trabajar con modelos y entornos 3D basados en Windows Presentation Foundation [WPF 45]. Este kit está siendo desarrollado en C# por el Massachusetts Institute of Technology (MIT) y posee una licencia de carácter Open Source [HELIX 14]. Aunque el kit posee una gran cantidad de métodos y librerías para trabajar con todo tipo de modelos 3D, para el presente trabajo solo se ha hecho uso del apartado de importadores, para poder procesar los ficheros de los modelos que se desea visualizar.

3.5.- Formatos de modelo y especificaciones

La información relacionada con los objetos y modelos tridimensionales que se desea visualizar en la aplicación va a estar almacenada dentro de ficheros. La aplicación va a ser capaz de trabajar con dos formatos de fichero diferentes: Wavefront (.obj) y STereoLithography (.stl).

A continuación se presentan dichos formatos.

Wavefront (.boj)

El formato OBJ es un formato de fichero para la definición de geometrías, y ha sido desarrollado por Wavefront Technologies para utilizar en su paquete de visualización avanzada de animaciones [OBJ 14]. Tanto el formato como su especificación son considerados de código abierto y ha sido adoptado por gran cantidad de aplicaciones y herramientas de gráficos 3D. Es considerado como un formato estándar.

El formato OBJ es un formato de almacenamiento de datos simple que representa la geometría de un modelo 3D únicamente. En el fichero se almacena la información relacionada con la posición de cada vértice, el mapeado de textura para cada coordenada, los vectores normales a cada vértice y las caras que conforman cada uno de los polígonos del modelo en forma de listado de vértices y de coordenadas de texturas. Los vértices se almacenan por defecto en sentido anti-horario, consiguiendo así que la declaración por defecto de todos los vectores normales de cada cara no sea necesaria. Aunque las medidas contenidas en los ficheros OBJ no poseen unidades de medida, es posible almacenar valores de escalado en el fichero.

Los ficheros OBJ no requieren de ninguna cabecera concreta. Cualquier línea del fichero que se quiera marcar como comentario debe comenzar con el carácter #. Los espacios en blanco se utilizan a modo de organización y facilidad de lectura. Cada línea del formato debe comenzar con un carácter de control y la información asociada a dicho carácter. Las líneas de texto son procesadas por orden hasta el final del fichero. A continuación se muestran los posibles caracteres de control, así como una breve descripción de cada uno de ellos (Figura 3.5.1):

- V, vt, vn: Caracteres para la definición de vértices, coordenadas de textura y vectores normales, respectivamente. Van acompañados de dos o tres cifras, representando las coordenadas (x, y, z) de cada uno (para las texturas, solo coordenadas x, y).
- f: Definición de una cara de un polígono. Va acompañado de tres pares de cifras, que se asocian a un vértice, una coordenada de textura y un vector normal. Dichas cifras se separan con el símbolo /.
- mtllib, usemtl: Definen el material del modelo completo. En el caso de mtllib, se asocia un fichero de formato mtl, con la información correspondiente al material.
- o, g: Con estos caracteres se representa, por una parte, el nombre asociado al modelo, y por otra, las diferentes agrupaciones que se realizan a nivel interno de los diferentes componentes del modelo.

```
# A 2 x 2 square mapped with a 1 x 1 square
# texture stretched to fit the square exactly.
mtllib master.mtl
v  0.000000 2.000000 0.000000
v  0.000000 0.000000 0.000000
v  2.000000 0.000000 0.000000
v  2.000000 2.000000 0.000000
vt 0.000000 1.000000 0.000000
vt 0.000000 0.000000 0.000000
vt 1.000000 0.000000 0.000000
vt 1.000000 1.000000 0.000000
# 4 vertices
usemtl wood
# The first number is the point,
# then the slash,
# and the second is the texture point
f 1/1 2/2 3/3 4/4
# 1 element
```

Figura 3.5.1 – Ejemplo de utilización del formato OBJ

El formato de fichero MTL siempre suele acompañar a un fichero OBJ. La finalidad de este fichero es la de definir todas las características del material del modelo. MTL representa Material Template Library y se considera como el estándar definido por Wavefront Technlogies para definir las propiedades de reflexión de la luz en una superficie en el momento del renderizado, mediante el modelo de iluminación de Phong. Aunque este formato se considere estándar, hoy en día no se encuentra actualizado, ya que no puede soportar completamente las últimas tecnologías de iluminación, como por ejemplo el Specular Mapping o Parallax Mapping.

En cuanto al formato de estos ficheros (Figura 3.5.2), se tiene un carácter de control para cada componente del modelo de iluminación: Ambiental (Ka), Difusa (Kd) y Especular (Ks) junto a su coeficiente (Ns) y transparencia del material (d, Tr).

```
newmtl Textured
Ka 1.000 1.000 1.000
Kd 1.000 1.000 1.000
Ks 0.000 0.000 0.000
d 1.0
illum 2
```

Figura 3.5.2 – Ejemplo de utilización del formato MTL

STereoLithography (.stl)

El formato STL se considera el formato estándar para muchas aplicaciones de modelado 3D asistidas por ordenador, o CAD. El formato fue creado por 3D Systems [STL 99]. Un fichero en formato STL define únicamente la geometría de un modelo tridimensional, sin tener ningún tipo de representación de color, textura, o cualquier otro tipo de atributo asociado a CAD. Un fichero con este formato puede estar codificado en formato ASCII o binario, siendo este último utilizado, ya que es el más compacto de los dos.

La estructura de un fichero STL consiste en un primer parámetro que define el número de triángulos que contiene el modelo, seguido de la definición de todos los triángulos, estructurados como: Vector normal, vértices primero, segundo y tercero. Debido a que los ficheros STL con los que se trabaja se encuentran en formato binario (Figura 3.5.3). Es necesario saber qué tipo de variables, y en qué orden, han sido codificadas. La información a nivel de byte queda representada de la siguiente manera:

Bytes	Data type	Description	
80	ASCII	Header. No data significance.	
4	unsigned long integer	Number of facets in file	
4	float	*i* for normal	+
4	float	*j*	
4	float	*k*	
4	float	*x* for vertex 1	
4	float	*y*	
4	float	*z*	
4	float	*x* for vertex 2	
4	float	*y*	
4	float	*z*	
4	float	*x* for vertex 3	
4	float	*y*	
4	float	*z*	
2	unsigned integer	Attribute byte count	

Figura 3.5.3 – Estructura de formato STL a nivel de byte

Aunque por norma general el valor de 'Attribute Byte count' no suele utilizarse, existen algunas aplicaciones que aprovechan este fragmento del fichero para codificar el color de cada polígono del modelo. Debido a que esta facultad depende del programa que carga y genera los ficheros STL, se ha decidido no hacer uso de esta propiedad, ya que se podrían producir efectos inesperados al intentar acceder a la información asociada a estos bytes.

Capítulo 4

DESARROLLO

En los siguientes apartados se va a exponer todos los aspectos del sistema desarrollado, así como los pasos que se han llevado a cabo en su realización. Se realizará una explicación, empezando desde una vista general, hasta los destalles más específicos en cada punto del proceso de desarrollo del mismo.

4.1.- Descripción general

El objetivo buscado es conseguir que una aplicación cliente (ARVisor) se comunique con otra aplicación servidor (PC) para solicitarle el listado de los ficheros con los modelos que ésta posea. El cliente también debe ser capaz de obtener dichos ficheros del servidor y visualizar dichos modelos en pantalla. La visualización de los modelos debe realizarse con la herramienta de realidad aumentada Vuforia, haciendo uso de la cámara del dispositivo que esté ejecutando la aplicación, así como de un marcador predefinido. Para finalizar, se estudiará el rendimiento de la aplicación a la hora de cargar diversos modelos, así como la ralentización que pueden experimentar los dispositivos al visualizar diversos modelos a la vez. El objetivo del estudio es observar el impacto que causa la aplicación desarrollada sobre el rendimiento de los dispositivos que la ejecuten.

4.2.- Herramientas para conexión cliente-servidor

Para poder conseguir que las dos aplicaciones que conforman el sistema a desarrollar se comuniquen entre ellas, es necesario saber qué se espera que realicen ambas aplicaciones. También va a ser necesario establecer en el tiempo el orden en el que pueden producirse las distintas acciones a realizar, para que el servidor sea capaz de satisfacerlas correctamente.

El servidor, en cuanto al apartado de comunicación se refiere, debe ser capaz de:

- Mantenerse a la espera hasta que un cliente se conecte a él.
- Recibir una petición de control, ya se descargar un listado o una serie de ficheros.
- Ser capaz de enviar uno o varios ficheros de forma continua.
- Ser capaz de realizar todo lo anterior en segundo plano, de forma que no pueda entorpecer con el funcionamiento habitual del ordenador en el que se ejecute.

Por otra parte, la aplicación cliente debe ser capaz de realizar las siguientes tareas relacionadas con la comunicación:

- Enviar peticiones de control al servidor, para empezar la comunicación con el mismo.
- Esperar a que el servidor le de confirmación de conexión en base a la petición solicitada.
- Ser capaz de recibir del servidor uno o varios ficheros de forma continua.
- Todas las acciones citadas deben estar reguladas por un controlador o *timeout* para evitar que la aplicación se cuelgue, ya que puede suponer un problema muy grave para el funcionamiento y rendimiento del terminal en el que se ejecute.

Para poder realizar la conexión entre aplicaciones a través de internet, se ha decidido hacer uso del protocolo de conexión de internet basado en TCP/IP mediante la utilización de sockets. El lenguaje C# da un soporte completo a esta tecnología mediante la inclusión de las librerías System.Net y System.Net.Socket, que son las que se van a utilizar para programar los métodos necesarios para llevar a cabo las tareas descritas anteriormente. En concreto, las clases utilizadas son las siguientes:

- TcpListener: La función que debe realizar esta clase es quedarse a la espera de una conexión entrante, que deberá generar la otra aplicación.
- TcpClient: Esta clase se encarga de establecer la conexión entre los dos clientes y ofrecer los servicios de red TCP.
- NetworkStream: Con esta clase se gestiona el flujo de datos subyacente para el acceso a través de la red.

Los métodos se han separado en función de qué parte del sistema los necesita. En total, se han creado dos clases distintas, una para la parte correspondiente al cliente y otra para la parte correspondiente al servidor (Figura 4.1). A continuación se muestra y explica el funcionamiento de las clases implementadas.

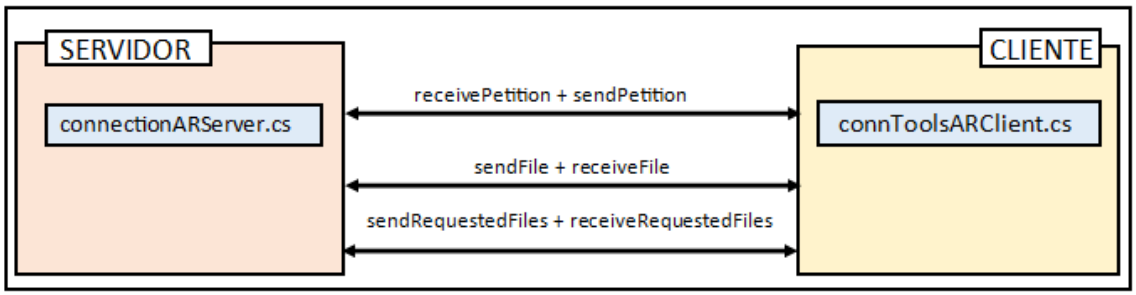

Figura 4.1 – Comunicación Cliente - Servidor

connectionARServer.cs

Esta clase se corresponde con la parte del servidor. Además de los métodos relacionados con la conexión, también se ha implementado un método extra para la creación del listado de ficheros. Todos los métodos se explican a continuación:

- *receivePetition(int sPort):* Este apartado se encarga de quedarse en espera hasta que llega una petición de conexión. Además, recibe una variable de control que le indicará qué acción se desea realizar; o enviar un fichero con

el listado de elementos, o enviar una serie de ficheros con los modelos que desea el cliente. Por motivos de seguridad, y para garantizar una mayor robustez de la aplicación, también se gestiona el caso en el que pueda recibirse una acción incorrecta. En este caso, se avisará al cliente del error. El servidor se quedará a la espera de una conexión entrante, la establecerá en el puerto indicado una vez la detecte, y enviará una señal de aceptación o rechazo en función de la orden recibida.

- sendFile(int sPort, String path, String name): Este método realiza la tarea de enviar un único fichero al cliente. Se establece una nueva conexión con el cliente en el puerto indicado para transferencia de datos y se le envía el fichero solicitado. Solo se usará en el caso de que solo sea un fichero el que se desea enviar. En caso contrario, es mejor utilizar la siguiente instrucción.
- sendRequestedFiles(int sPort, ind dPort, String path): Este método se ocupa de enviar una serie de ficheros al cliente. Para este caso, se establece una primera conexión para recibir el listado de ficheros que desea el cliente. Con una segunda conexión se irán enviando los ficheros solicitados según se vayan recibiendo las peticiones de los mismos.
- createXMLfromDirectory(String path, String name): El objetivo de este método es revisar un directorio indicado en la variable path, para localizar todos los ficheros que se encuentran en el mismo y listarlos en un nuevo fichero de tipo xml. El fichero xml que se creará contendrá todos los ficheros de un directorio, así como la extensión de cada uno de ellos, para facilitar el filtrado de los mismos en un paso posterior. De dicha acción se encargará el cliente en un paso posterior. Para la creación de este fichero se ha optado por utilizar la librería nativa de C# System.Xml, que contiene todo lo necesario para crear dichos ficheros.

A la hora de establecer una conexión mediante TCP y enviar o recibir ficheros, se puede dar el caso de que se produzca algún tipo de error o desconexión de red. Esto se debe normalmente a la calidad de la señal wifi y de la red a la que se encuentren conectados los dispositivos. Por defecto, el framework de .NET detecta estos casos en tiempo de ejecución como excepciones, provocando el detenimiento prematuro de las aplicaciones implicadas. Debido a esto, se hace necesario gestionar dichas excepciones mediante las instrucciones try y catch de C#. En caso de que se produzca algún tipo de excepción relacionada con la conexión, se debe cerrar y reiniciar de forma segura todos los puertos y sockets utilizados, para no provocar cuelgues y problemas graves a ninguna de las aplicaciones

connToolsARClient.cs

De forma análoga al servidor, esta clase contiene todos los métodos necesarios para la gestión de la conexión de la aplicación cliente. Los métodos son los siguientes:

- sendPetition(int value, String sIP, int sPort): Con este método se intenta realizar una conexión con el servidor. En caso de llevarlo a cabo, se envía una señal de control en función de la acción que se desea realizar. A continuación, el cliente espera una confirmación por parte del servidor, para asegurar que la señal se ha recibido correctamente y se puede seguir con el siguiente paso sin ningún problema. En caso de no ser así se devuelve una señal de error para que el cliente actúe en consecuencia.
- receiveFile(String sIP, int sPort, String path, String name): Este método se encarga de intentar establecer una conexión con el servidor en la dirección IP y puerto indicados, para acto seguido quedarse a la espera de recibir un único fichero y almacenarlo en memoria.
- receiveRequestedFiles(String sIP, int sPort, int dPort, String path, String[] fList): De forma similar al método anterior, esta instrucción intenta establecer una conexión con el servidor en la dirección IP y puerto indicados, para enviarle al mismo un listado con los ficheros que desea recibir del servidor. Además, también intentará establecer una segunda conexión en el puerto indicado por dPort, para recibir por este canal todos los ficheros a modo de stream de datos.

Al igual que sucede en el servidor, en las instrucciones del cliente también es necesario gestionar mediante las instrucciones try y catch las posibles excepciones que pueden producir los sockets de conexión debido a algún error asociado a la conexión o transferencia de datos.

Además de todas las precauciones ya indicadas hasta ahora, en la aplicación cliente es necesario aplicar una restricción extra, debido a la naturaleza y funcionamiento de los sockets a la hora de intentar establecer una conexión. Cuando un socket intenta realizar una conexión, la ejecución de la aplicación se queda a la espera durante un tiempo indefinido hasta que el socket consigue realizar dicha conexión. No se puede permitir que esto suceda en el cliente, debido a los problemas de memoria y recursos que conlleva. Para solucionar este problema, se aplica el siguiente fragmento de código al establecimiento de conexión:

```
TcpClient client = new TcpClient();
IAsyncResult result = client.BeginConnect(serverIP, serverPort, null, null);
int serverTimeout = 5000; // Espera de 5 segundos
bool success = result.AsyncWaitHandle.WaitOne(serverTimeout, true);
if (!success){
    client.Close();
    throw new Exception("Failed to connect server.");
}
```

Figura 4.2.1 – Código de conexión asíncrono y gestión de timeout

El funcionamiento del mismo es el siguiente (Figura 4.2.1): El establecimiento de conexión se realiza de forma asíncrona (se genera automáticamente un nuevo proceso) mediante la llamada BeginConnect. A continuación, se asocia dicho proceso a un reloj

con un tiempo de parada o timeout de cinco segundos mediante la instrucción AsyncWaitHandle.WaitOne. Lo que ocurrirá es lo siguiente: Si el proceso (el establecimiento de llamada) se realiza con éxito y finaliza antes de los cinco segundos de timeout, la variable success tomará el valor verdadero; si lo que termina antes es el contador del reloj, la variable tomará el valor falso. En caso de haber saltado el timeout, se generará una excepción, que ya se encargará de gestionar correctamente el bloque catch. De esta forma se consigue controlar con éxito el tiempo que se quedará a la espera la aplicación mientras se intenta establecer una conexión con el servidor.

4.3.- Aplicación servidor

La aplicación servidor ha sido creada para utilizarse desde un PC con acceso a una conexión a internet. Primero se explicará el comportamiento de la misma de cara al usuario, a través de la interfaz de la misma:

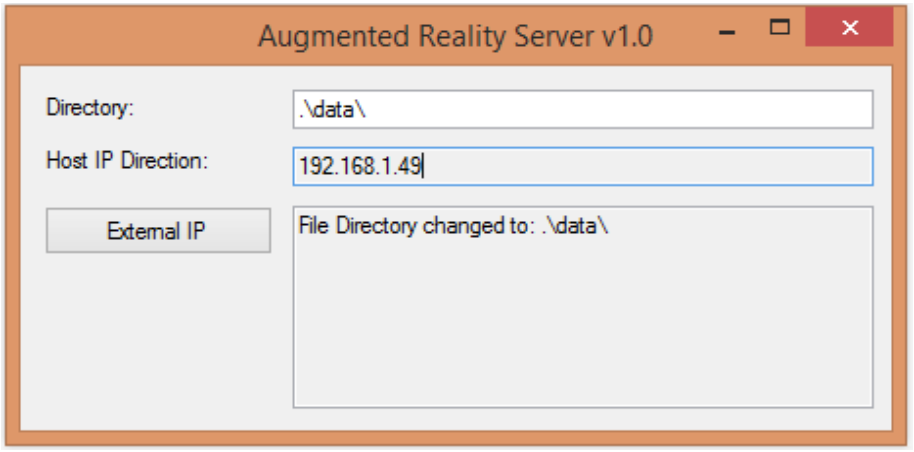

Figura 4.3.1 – Interfaz de la aplicación servidor del sistema implementado

El funcionamiento de la herramienta es el siguiente (Figura 4.3.1): El campo Directory se utiliza para establecer el directorio en el que se encuentran los ficheros con los modelos. Dichos ficheros serán a los que tendrá acceso el cliente a la hora de solicitar su listado y su posterior descarga. En el campo Host IP Direction se encontrará la dirección IP de la red de área local LAN. La finalidad de este campo es la de obtener dicha dirección de forma sencilla para el usuario para que pueda configurarla en el cliente. Esto se verá más adelante. La aplicación posee un cuadro de texto que funciona a modo de registro, para comunicar si se ha producido algún cambio en las direcciones IP o el directorio. Por último, el botón de External IP obtiene la dirección IP externa de la máquina que está ejecutando el servidor. Indicar también que el programa es capaz de recordar la última ruta de directorio establecida, para no tener que estar configurándola de nuevo cada vez que se ejecuta la aplicación.

En cuanto al funcionamiento interno de la herramienta, a continuación se explican los métodos y sub-clases implementados para el funcionamiento de la misma:

- getLocalIPAddress(): Este método obtiene la dirección IP local de la máquina en la que se ejecuta. Para obtener dicha dirección, se listan todas las IPs que hay asociadas a la máquina. Dada la forma en la que Windows gestiona las direcciones IP, la primera dirección asociada al protocolo de conexión IP se corresponde con la dirección local de la misma. Dicha dirección se devuelve en formato IPv4.
- getExternalIPAdress(): Así como la IP local es relativamente fácil de obtener, no ocurre lo mismo con la dirección externa. Para obtener lo más rápido posible dicha dirección, se ha decidido hacer uso de una de las múltiples páginas web existentes cuya finalidad es la de devolver un fichero html con dicha dirección cada vez que se intenta acceder a ella. La página que se ha decidido usar es:

  http://ipecho.net/plain

  Esta página devuelve en su fichero html únicamente la dirección IP externa, por lo que facilita su posterior obtención al no poseer ningún fragmento de texto ni etiquetas características del lenguaje html. Así pues, la función realizará una petición a ésta página web, y buscará en el fichero html que devuelve la dirección IP para mostrarla por pantalla.
- launchServer(): Éste es el método que se encargará de controlar toda la parte de conexión del servidor. Se apoya en la clase connectionARServer explicada en el apartado anterior.

Básicamente, las tareas del servidor funcionan de la siguiente manera: El servidor espera a que algún cliente se conecte al mismo y le haga una petición (Figura 4.3.2). Pueden haber dos tipos de peticiones: Listado y Descarga.

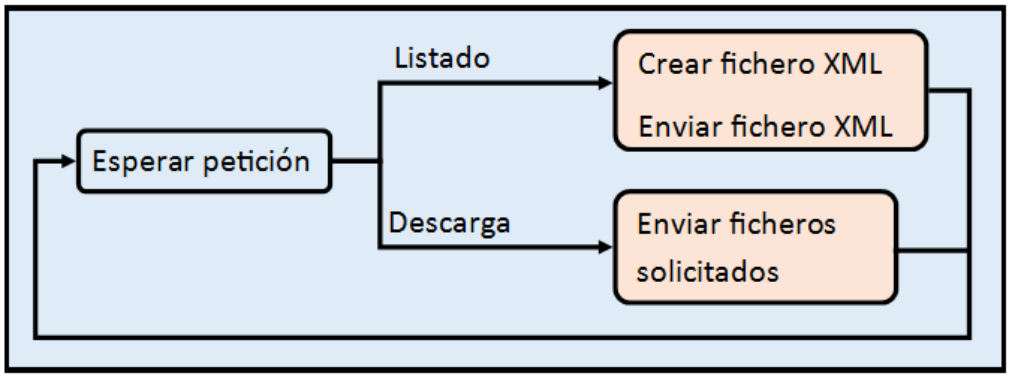

Figura 4.3.2 – Diagrama de flujo para la máquina de estados del servidor

En el primer caso, el servidor creará un fichero XML con el listado de todos los ficheros que se encuentren en ese momento en la carpeta especificada en el directorio y acto seguido se lo enviará al cliente. En el segundo caso, el servidor se quedará esperando a que el cliente le diga qué ficheros desea descargar para ir enviándoselos al mismo. Después de haber completado cualquiera de las dos tareas, el servidor volverá a quedarse a la espera de una nueva petición, hasta que la aplicación se cierre.

Para finalizar, indicar que el método launchServer se arranca al inicializar la aplicación, de la siguiente manera:

```
objServer = new serverT();
sT = new Thread(objServer.launchServer);
sT.IsBackground = true;
sT.Start();
while (!sT.IsAlive) ; // Esperar a que el proceso se ponga en marcha
```

Figura 4.3.3 – Fragmento de código para la creación de procesos y configuración

Este código se corresponde con la creación e inicialización por defecto de procesos que se encuentra en la documentación de sockets de C# y .NET (Figura 4.3.3). Destacar únicamente la instrucción Thread.IsBackGround, que le va a indicar al proceso principal (padre) que el subproceso (hijo) debe terminarse y cerrarse a la vez que el programa principal. De esta forma se asegura que el subproceso no se quedará en estado zombi en el caso de que la aplicación se cierre.

4.4.- Aplicación cliente

Para desarrollar la aplicación cliente del sistema, se ha decidido utilizar el entorno de desarrollo Unity, gracias a las herramientas y facilidades que ofrece a la hora de exportar y compilar aplicaciones gráficas para dispositivos móviles. El sistema operativo para dispositivos móviles por el cual se ha optado ha sido Android. Por otra parte, para poder incorporar las funcionalidades correspondientes a realidad aumentada, se ha utilizado la extensión para Unity de la herramienta Vuforia. A continuación se explicará con detalle todo el proceso de desarrollo de la aplicación cliente.

4.4.1.- Visualización y Vuforia

La primera tarea realizada ha sido la creación del entorno de realidad aumentada de la aplicación, así como la cámara que deberá mostrar los modelos tridimensionales. La web de Vuforia ofrece para su descarga el paquete de Unity necesario, que contiene todos los componentes que hacen falta para crear aplicaciones de realidad aumentada. Además, el marcador de seguimiento que se desea utilizar, así como las herramientas necesarias para su correcto funcionamiento, deben agregarse también al proyecto. Para agregar estos dos paquetes al proyecto de Unity se deben seleccionar desde el menú contextual de Unity las opciones correspondientes:

Assets > Import Package > Custom Package > Vuforia-Unity + Marker

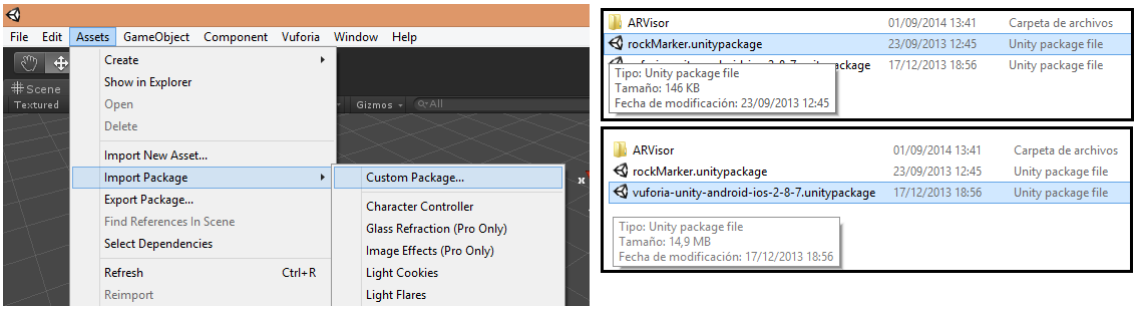

Figura 4.4.1 – Menú contextual e importación de recursos de Unity

De esta forma quedan agregados todos los recursos de Vuforia en Unity, así como el marcador a utilizar (Figura 4.4.1). El siguiente paso será crear y configurar las instancias de los componentes que se van a utilizar.

De los diversos componentes que contiene el paquete Vuforia. Para la aplicación creada solo se necesitan dos, que se explican a continuación:

- ARCamera: Al contrario que las aplicaciones gráficas de Unity convencionales, una aplicación con realidad aumentada necesita usar una cámara de visualización concreta. La propiedad de este componente es la de ser capaz de reconocer cualquier cámara del dispositivo sobre el que se ejecute la aplicación. De esta forma el dispositivo será capaz de observar el entorno, y reconocer el marcador sobre el que se desea mostrar el modelo 3D.
- ImageTarget: Este componente representa el marcador de realidad aumentada. Cualquier objeto y modelo de Unity que se asocie a un ImageTarget se mostrará por pantalla cuando la aplicación detecte el marcador asociado al mismo.

Respecto al paquete que contiene el marcador de seguimiento, indicar que ha sido creado desde la web de desarrollo de Vuforia Qualcomm, ya que en estos momentos, solo la herramienta que se ofrece en su página web puede convertir una imagen en un marcador válido para Vuforia. La herramienta es de uso gratuito y se puede acceder desde la misma web de desarrollo [VUFO 02].

Para crear los elementos necesarios, hay que seleccionarlos y arrastrarlos a la pestaña de jerarquías del proyecto de Unity (Figura 4.4.2). Una vez creada tanto la ARCamera como el ImageTarget, es necesario configurarlo correctamente. Primero hay que configurar el marcador. Para ello, en la pestaña de comportamiento de ImageTarget hay que asociar a este objeto el marcador de seguimiento en la casilla de ImageTargetBehaviour. Segundo, es necesario configurar la cámara. En la pestaña de comportamiento de ARCamera, se establece el detector de visualización como auto para que detecte la camera de manera automática. En cuanto al marcador, se escoge de la misma manera que con la imagen objetivo en el menú DataSet.

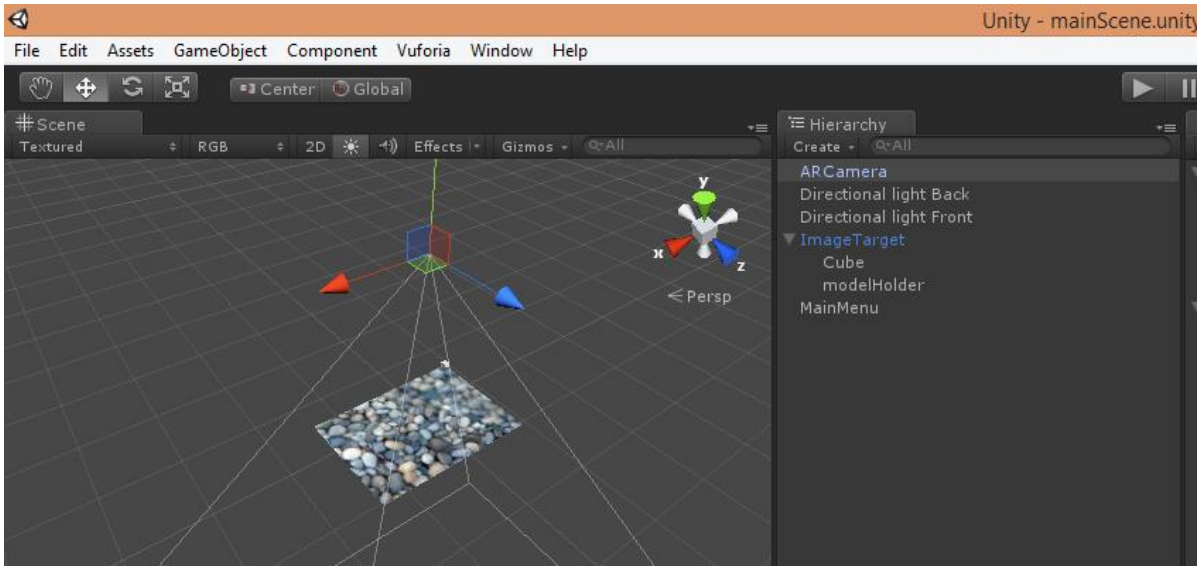

Figura 4.4.2 – Jerarquía de objetos y estado de la aplicación en Unity

A parte de los objetos ya explicados, se agrega al editor un par de luces ambientales, un objeto vacío asociado al ImageTarget, y un último objeto vacío que no se va asociado a ningún otro elemento.

La función de las luces ambientales es la de iluminar los modelos cuando se muestren por pantalla. Sin ninguna luz, los modelos se verían totalmente negros. Con una segunda luz, que tendrá una intensidad menor que la primera, se consigue que las sombras y partes no iluminadas del modelo no se queden totalmente a oscuras. De esta forma se consigue que el modelo quede bien iluminado y puede apreciarse en su totalidad.

El objeto vacío asociado al marcador, de nombre modelHolder, se utiliza a modo de nodo raíz para que la aplicación almacene a partir de él los modelos que se cargarán durante la ejecución de la aplicación. Como ejemplo, si durante una ejecución se desea tener cargado tres modelos y mostrarlos a través de la cámara, estos quedarían de la siguiente manera (Figura 4.4.3):

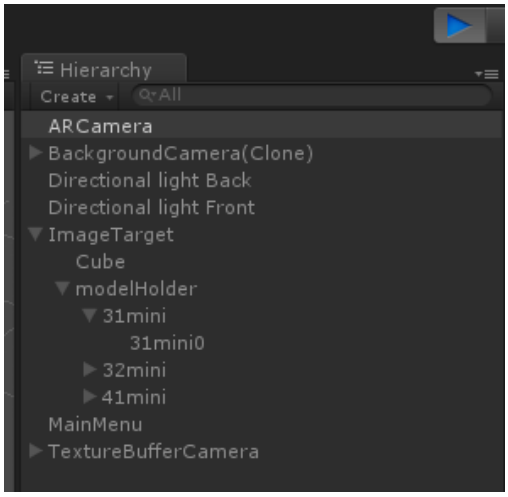

Figura 4.4.3 – Estado de la jerarquía durante una ejecución

Respecto al último objeto vacío, de nombre mainMenu, su función es la de almacenar los scripts que se encargan de hacer funcionar el menú de la aplicación, y que se explican en el siguiente apartado.

4.4.2.- Implementación de menús

El comportamiento de los menús se ha dividido en dos scripts: Uno que gestione el comportamiento y aparición de los botones y sub-menús, y otro que se ocupe de las acciones que se deben realizar al interactuar con dichos botones y sub-menús. Para esta parte del código de la aplicación se ha optado por utilizar UnityScript, ya que el propio Unity ofrece un soporte a nivel nativo de este lenguaje.

MenuSystem.js

Este script se encarga de gestionar todas las operaciones y acciones relacionadas con las animaciones y operaciones para mostrar u ocultar los diferentes menús de la aplicación.

Para gestionar el estado de los menús, se ha optado por utilizar una máquina de estados, con una variable de control para mantener la posición dentro de la máquina. Un menú puede encontrarse en cualquiera de los siguientes estados (Figura 4.4.4):

- Abierto: El menú o submenú se encuentra abierto y activo.
- Cerrado: El menú o submenú se encuentra cerrado y desactivado. Es el estado por defecto.
- Abriéndose: Estado de transición del estado Cerrado al estado Abierto. En este estado se actualiza la posición del menú y su valor de visualización alpha. Cuando alpha == 1, se pasa al estado Abierto.
- Cerrándose: Estado de transición del estado Abierto al estado Cerrado. En este estado se actualiza la posición del menú y su valor de visualización alpha. Cuando alpha == 0, se desactiva el menú y se pasa al estado Cerrado.

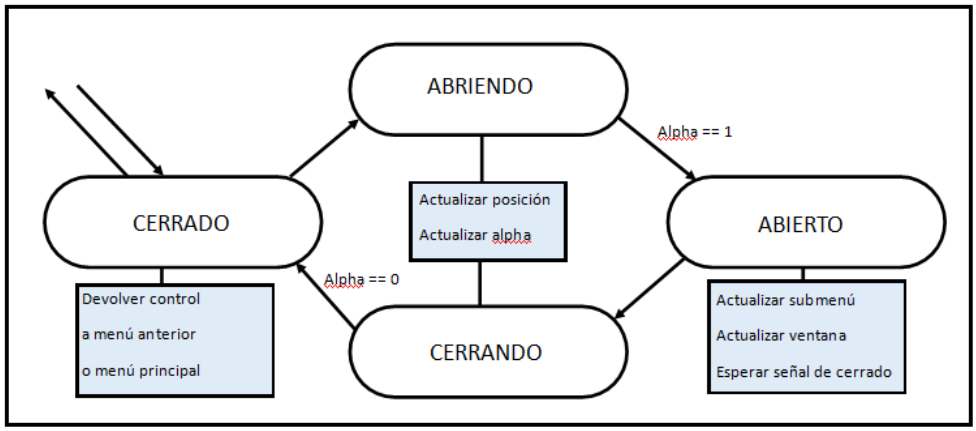

Figura 4.4.4 – Máquina de estados para el comportamiento de los menús

Un menú en el estado Cerrado debe liberar los recursos de memoria que haya estado utilizando, actualizar el estado global de la aplicación y devolver el control o al menú principal o al menú que lo hubiera invocado antes. Como excepción al comportamiento de este estado, dejar aclarado que el botón principal que activa todos los submenús y controla el funcionamiento de toda la aplicación, nunca puede llegar al estado cerrado, ya que esto implicaría dejar desactivada toda la aplicación.

Durante los estados de Abriendo y Cerrando, la aplicación va actualizando de manera progresiva tanto la posición como el valor de visualización (alpha), causando un efecto animado a modo de 'aparición' del menú que se encuentre en este estado. Una vez el valor de alpha haya llegado a uno de los dos extremos, en función de si se está abriendo o cerrando el menú, el control pasará al estado correspondiente.

Cuando un menú se encuentra en el estado activo, la aplicación puede reaccionar de dos maneras: Por una parte, si el menú contiene algún submenú, se le pasará el control del estado al nuevo submenú, bloqueando el actual hasta que el nuevo se desactive. Por otra parte, si el menú tiene asociado un panel o ventana de acción, se le pasa el control de la aplicación al script de comportamiento (explicado en el siguiente apartado) y se bloquea el menú actual hasta que se finalicen todas las acciones de la ventana.

A nivel de funcionamiento, la aplicación consta de un único botón en la esquina de la pantalla, que actuará a modo de menú principal. Dicho botón activará a su vez un sub-menú con cuatro nuevos botones. Dichos botones han sido nombrados de la siguiente manera: UpdateList, DownloadItems, ShowItems y ConfigConnection. Dichos botones activan a su vez una ventana de acción diferente, cada uno de ellos. El comportamiento de dichas ventanas y su función dentro de la aplicación se explicarán en apartados posteriores.

MenuBehaviour.js

Este script se va a encargar de crear y controlar las acciones de las diferentes ventanas de la aplicación. Como se ha comentado, estas ventanas se activarán según el estado del menú, y tendrán que devolver el control de la aplicación al sistema de menús una vez terminen las acciones que debe realizar cada una de ellas. En total, la aplicación consta de cuatro paneles, cuya función se explica a continuación:

- *Actualización de listados:* Solicita a la aplicación servidor el listado con los ficheros disponibles, según se haya configurado el servidor en su aplicación.
- *Lista de descargas:* Muestra por pantalla el listado de los ficheros con formato válido obtenidos en el paso anterior. Permite seleccionar los que se desee descargar, y realiza la solicitud de descarga al servidor.
- *Lista de visionado:* Muestra por pantalla los ficheros con que se han descargado en el paso anterior. Permite seleccionar los que se deseen

mostrar, y realiza la importación de los mismos para mostrarlos por pantalla a través del marcador y la cámara.

- *Menú de configuración:* Permite configurar la dirección IP local donde se encuentra conectada la aplicación servidor.

A la hora de programar el comportamiento de cada ventana, hay que tener en cuenta dos factores. Primero, como ocurre en toda aplicación gráfica, las ventanas e información que se visualizan por pantalla se están actualizando constantemente mediante una función de actualizado. Segundo, cualquier intento de conexión con el servidor debe realizarse fuera del bucle de actualización del apartado gráfico. Debido a esto, la máquina de estados de transición entre aplicaciones debe actualizarse para reflejar estas características. Así pues, la máquina de estados quedaría de la siguiente manera:

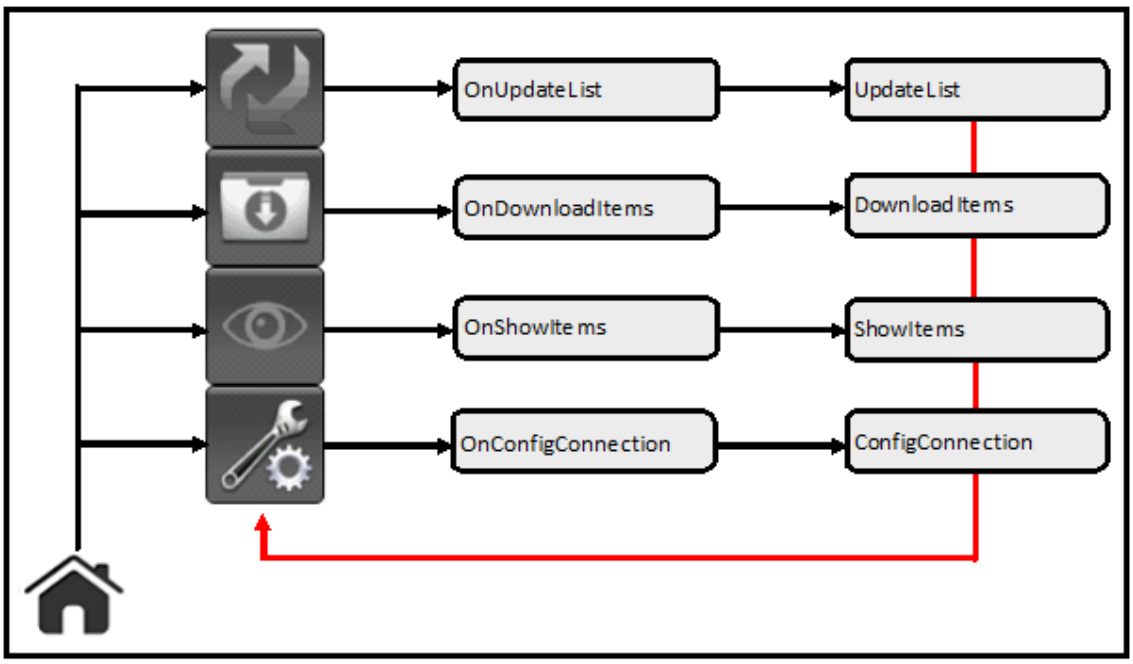

Figura 4.4.5 – Máquina de estados para la transición entre ventanas de la aplicación

En total, el script de MenuBehaviour tiene programados ocho estados de la máquina de estados de la aplicación (Figura 4.4.5). Las acciones que debe realizar cada nodo de la máquina de estados vienen explicadas a continuación:

- OnUpdateList: Este método intenta realizar una petición al servidor para obtener y descargar el listado de ficheros disponibles para su posterior descarga.
- UpdateList: Este método mostrará por pantalla si la acción de descargar el listado de ficheros ha tenido éxito o no.
- OnDownloadItems: La función de este método es la de abrir el listado de ficheros obtenido en el paso anterior y cargar en un vector de texto todos los nombres que se encuentren en el listado.

- DownloadItems: Con este método se muestra por pantalla el vector de texto con todos los ficheros que se encuentran en el servidor. Junto al nombre de cada fichero aparecerá una casilla qué se utiliza para elegir que ficheros se desean descargar. Al pulsar sobre el botón de aceptar que se encuentra en esta ventana, el programa revisará qué ficheros han sido seleccionados, creando un stream de datos con todos los nombres de los ficheros seleccionados. A continuación, realizará una petición al servidor para solicitar la descarga de dichos ficheros. Todos los ficheros descargados se almacenan temporalmente en una carpeta interna de la aplicación. El máximo de memoria para el almacenamiento de ficheros se ha establecido en 200MB; límite que se ha establecido internamente y que, por motivos de seguridad, se ha decidido no permitir que pueda ser modificado por el usuario.
- OnShowItems: Este método revisa los ficheros almacenados en la carpeta interna de descargas, y lista sus nombres en un vector de texto.
- ShowItems: Con el vector que contiene los nombres de los ficheros descargados, este método los mostrará por pantalla, junto a su casilla de selección correspondiente, que se utilizará para seleccionar qué modelos se quieren mostrar en el marcador de realidad aumentada. Al pulsar sobre el botón de aceptar, se revisan qué modelos han sido seleccionados de todos los disponibles, y se cargarán en el visor de modelos mediante la utilización de los importadores de modelos, que se explicarán en un apartado posterior.
- OnConfigConnection: Este método abrirá el fichero de configuración que tiene la aplicación, para mostrar los parámetros en el siguiente estado.
- COnfigConnection: Este método permite visualizar y editar los parámetros de configuración de la aplicación. En estos momentos, el único parámetro que se puede modificar corresponde con la dirección IP local, que deberá configurarse con la dirección IP local que ofrece la aplicación servidor, explicada anteriormente.

Recordar que todas las funciones y herramientas relacionadas con la conexión TCP con el servidor han sido explicadas en el apartado de herramientas para conexión cliente-servidor. A la hora de gestionar los errores que pueden provocar las excepciones de control en los métodos de conexión, se ha decidido mostrar por pantalla que no se ha podido establecer la conexión con el servidor, además de reiniciar todos los sockets y puertos. De esta forma se puede asegurar que no existirá un socket con una conexión mal cerrada, permitiendo que la aplicación siga su ejecución sin ningún contratiempo. Los tiempos de espera de todos los intentos de conexión se ha establecido en cinco segundos, por motivos de seguridad y de consumo, tanto de memoria como de batería del dispositivo móvil donde se ejecute la aplicación.

4.4.3.- Importador de modelos

El trabajo que debe llevar a cabo el importador de modelos es el de ser capaz de abrir un determinado fichero que contenga un modelos, y extraer toda la información del mismo. Dicha información será almacenada en una estructura intermedia, y a continuación indexada a un objeto interno de Unity. En el apartado de visualización y Vuforia se explicó la existencia de un objeto vacío de nombre modelHolder que será el que se utilizará a continuación para poder gestionar los modelos que se deseen cargar. El importador de modelos debe ser capaz de abrir e interpretar los dos formatos de fichero para modelos explicados anteriormente: El formato WaveFront OBJ y el formato STereoLithography STL.

La programación de todo el sistema de importación de modelos se ha realizado con el lenguaje de programación C#, y se ha estructurado de la siguiente manera:

- Point.cs y Point3D.cs: Se trata de dos estructuras de datos auxiliares, para facilitar la programación de los módulos superiores.
- MeshBuilder.cs: Estructura intermedia que contiene toda la información relativa al modelo que se desea cargar.
- ObjReader.cs: Esta clase contiene todo lo necesario para abrir e interpretar un fichero en formato WaveFront OBJ, así como las estructuras necesarias para almacenar la información asociada al modelo que se está leyendo.
- StLReader.cs: Esta clase contiene las herramientas necesarias para abrir e interpretar un fichero en formato STereoLithography STL, además de las estructuras necesarias para contener toda la información asociada al modelo que se está leyendo.
- externalReader.cs: La clase principal de todo el importado. Encapsula el resto de clases y métodos para utilizar todo el importador con la mayor facilidad posible en las capas superiores de la aplicación.

La estructura de dependencias de todas las clases implementadas se puede observar en la siguiente figura (Figura 4.4.6):

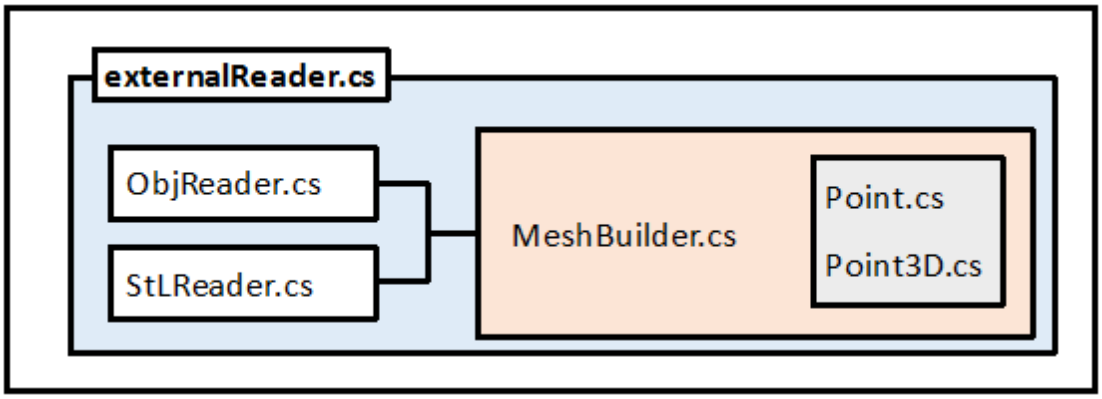

Figura 4.4.6 – Diagrama de dependencias del importador de modelos

A la hora de cargar el modelo dentro de Unity, hay que asociar sus componentes a un objeto interno llamado GameObject. Un GameObject es el componente básico, y por tanto el más importante, que puede existir en Unity. Se puede entender un GameObject como si fuera un contenedor, y en función de qué componentes se asocien a él, el GameObject puede representar cualquiera de los otros componentes que existen en Unity. De todos los componentes que pueden formar parte de un GameObject, se explican a continuación únicamente aquellos que van a hacer falta para la aplicación actual:

- Mesh: Un mesh consiste en una colección de vértices, lados y caras que definen la forma de un objeto tridimensional dentro de un dominio basado en modelado sólido y de gráficos por computador. En el caso de Unity, además de asociar los vértices y los triángulos que conforman el modelo, también es necesario asociar un vector normal, así como una coordenada de textura (en el caso de que existan) a cada vértice del mesh. Es muy importante recordar la importancia del orden en el que se definen los triángulos, así como la dirección de los vectores normales, ya que en caso de que estos datos no sean correctos, no se podría visualizar correctamente el modelo.
- Transform: Unity será incapaz de mostrar por pantalla un modelo si no tiene una transformación asociada al mismo. Una transformación define la posición, rotación y escala de un objeto a la hora de posicionarlo en una escena. Aunque la aplicación diseñada no va a generar ningún tipo de transformación especial o animación, es necesario definir dicha transformación, aunque sea con los valores por defecto de la misma.
- MeshFilter: La función de un MeshFilter dentro de Unity es la de asociar un mesh y asignarlo correctamente, para poder renderizarlo a continuación.
- MeshRenderer: Este componente del GameObject usará el MeshFilter que tenga asociado, así como cualquier componente de tipo transformación que posea, para renderizar el modelo en la posición deseada.
- Material: Este componente va a gestionar todas las propiedades del material que conforma el objeto. Este objeto va a afectar a la iluminación y aspecto del modelo que se desea cargar. Atributos como las componentes del modelo de iluminación y refracción de la luz sobre la superficie del modelo son especificados aquí (Componentes como luz ambiental, difusa, especular o transparencia, entre otros).

Límite de vértices por malla

Una particularidad muy importante que hay que tener en cuenta a la hora de crear una malla o mesh para el modelo, es que el número total de vértices que puede contener no debe superar en ningún momento la cota de 65.000. Si en algún momento llegara a suceder esto el modelo no podría ser cargado. Se trata de una restricción

interna de Unity que no se puede retirar de forma directa. No en vano, existe una forma de sortear dicha restricción.

Dada la naturaleza de los objetos de tipo GameObject, Unity permite anidar una cantidad indefinida del mismo objeto dentro de otro. Sabiendo esto, se puede hacer que, dado un modelo que sobrepase el límite permitido de vértices, éste se rompa en un número de modelos con una cantidad más reducida de componentes. Además, si dichos modelos 'hijo' son asociados al modelo principal 'padre', cualquier tipo de transformación u operación que se le aplique al padre, también se les aplicará de forma automática a los hijos (Figura 4.4.7). A la hora de partir el modelo, es necesario crear nuevos GameObject, con sus correspondientes componentes, y asociarlos al nodo que hará de padre de todo el modelo. Como ejemplo, una figura que sobrepase el límite, quedaría dividida de la siguiente manera, dentro de la jerarquía de objetos en tiempo de ejecución:

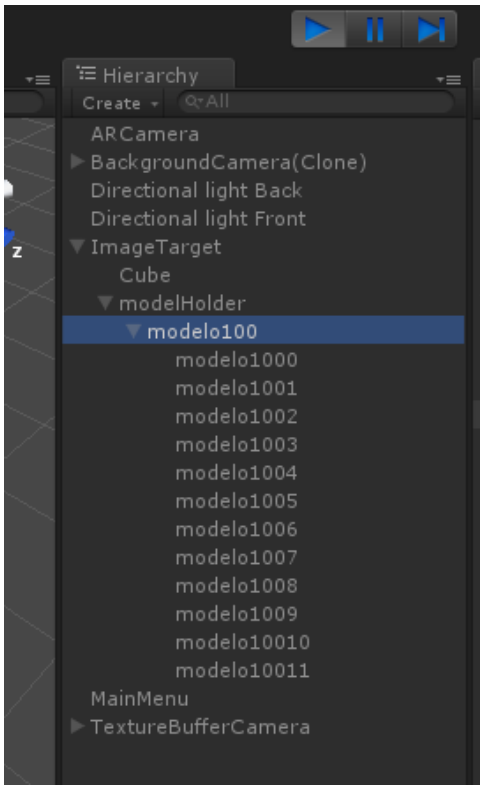

Figura 4.4.7 – Partición de un modelo en un conjunto de componentes reducido

En los siguientes apartados se pasará a explicar con detalle el funcionamiento de cada una de las clases y métodos que se han implementado, así como la forma en la que se asocian todos los componentes.

Point.cs y Point3D.cs

Dos clases auxiliares simples cuya finalidad es la de simplificar las operaciones con coordenadas en las clases siguientes. Point y Point3D van a representar vértices de dos y tres dimensiones, respectivamente. Se han implementado constructores y

destructores, así como las correspondientes sobrecargas en los operadores básicos: suma, resta e igualdad de valores. También hay implementado un conversor de tipos que transforma estas estructuras en variables de tipo Vector2 y Vector3, usadas por Unity a nivel interno en algunos aspectos de los objetos GameObject, que almacenan la información de los modelos.

MeshBuilder.cs

El objetivo de esta clase es la de almacenar todos los datos que se extraen de los ficheros de los modelos. Los datos serán almacenados directamente en las estructuras correctas para asignarlos directamente a los GameObject correspondientes de Unity. En concreto, esta clase se encarga de almacenar y dar forma al componente mesh del GameObject. Los datos requeridos para formar el mesh son los siguientes:

- Vértices: Lista de componentes de tipo Vector3 que va a contener el listado completo de vértices del modelo.
- Normales: Otra lista de componentes de tipo Vector3 que va a contener en este caso el listado completo de vectores normales del modelo. Hay que tener en cuenta que el tamaño de este vector y el de vértices deben coincidir, o habría un error en el modelo a cargar
- Triángulos: Consiste en un listado de enteros, que contiene los índices de los vértices que forman los diferentes triángulos de la malla del modelo. En caso de que a la hora de insertar un triángulo no se tuvieran todos los vectores normales, se deberán calcular con la información de los vértices y agregarse a la vez.
- Vértices de texturas: En el caso de que existiera un mapeado de textura para el modelo, también es necesario agregar las coordenadas de la textura (2D) para asociarlas con cada vértice. Una vez más, en caso de existir dicho mapeado, el número de coordenadas de textura debe coincidir con el número de vértices, o habría un error en el modelo a cargar.

Debido a la importancia del orden en el que se van agregando los componentes del mesh o malla del modelo, solo se permite agregar cualquiera de dichos componentes a través de un único método llamado AddTriangle, que irá añadiendo cada vértice, normal, coordenada de textura y triángulo, en el orden correcto. De no ser así, el modelo a la hora del visionado podría mostrar efectos indeseados.

Puede darse el caso excepcional en el que un modelo pueda tener algunas de sus caras definidas como un cuadrado o quad, en vez de usar triángulos. Para estos casos concretos, se ha creado un método extra de nombre AddQuad, que dividirá el cuadrilátero en dos triángulos, y los agregará a la malla, conservando la integridad de la estructura.

Para finalizar con todo el proceso, un método de nombre CreateMesh, se encargará de crear un objeto de tipo mesh limpio y asociar todos los elementos. Es este

mesh el que se acabará asociando al GameObject correspondiente como paso final del importador de ficheros.

ObjReader.cs

Esta clase se va a encargar de leer la información contenido en los ficheros Wavefront OBJ. La clase también realiza un almacenado parcial de todos los datos, y se los pasará a la clase MeshBuilder para que los organice de forma correcta. Los métodos que han sido implementados para esta clase se pueden dividir en tres conjuntos:

- Métodos de lectura: En este conjunto se listan todos los métodos encargados de leer el fichero del modelo, así como de interpretar, procesar y separar los componentes de cada línea del fichero. Recordar que además de procesar los ficheros OBJ, también se debe hacer lo propio con los ficheros de material MTL. También hay implementados métodos para separar por carácter de control y valores una línea de texto.
- Métodos de agregación: Aquí se listan todas las funciones de agregación de elementos. Los elementos que se pueden agregar son todos los vistos en el apartado de herramientas, en la sección del formato Wavefront. Muy importante recordar que en este punto de la ejecución se están agregando los elementos en las variables de almacenamiento parcial. El orden correcto de los elementos se establece en el siguiente bloque.
- Métodos de conversión: Una vez se han extraído todos los elementos que conforman el modelo del fichero, se deben reestructurar y almacenar en el formato correcto y de forma definitiva. El formato correcto es el establecido en MeshBuilder, y para transformar los datos temporales en los definitivos, se usan los métodos que se agrupan en este bloque. Dichos métodos serán BuildModel y CreateModel, y se encargan de utilizar los métodos de la clase MeshBuilder para generar la información correcta y, en última instancia, crear el GameObject con el resultado final del importador.

Para finalizar con este apartado, indicar que, en caso de que no existiera un fichero MTL o éste no contuviera un material válido, el importador generaría un material por defecto de manera automática y lo asociaría al modelo. De esta forma, el modelo 3D se podría visualizar sin ningún problema en caso de no poseer dicho componente.

StLReader.cs

De forma similar al formato OBJ, la tarea de esta clase es la de hacer la misma tarea explicada en el apartado anterior, pero esta vez para el formato STL. Aun y así, este formato va a generar una dificultad añadida respecto al caso anterior, puesto que la codificación de este formato se realiza en binario. Sabiendo el orden en el que se han estructurado los datos, qué tipo de datos contienen, y su tamaño en bytes, se han

implementado una serie de métodos capaces de traducir la información contenida en bytes a su formato correspondiente.

Por otra parte, como el fichero estructura de forma ordenada directamente todos los triángulos, en este caso se puede hacer la inserción directa de los elementos a la estructura MeshBuilder.

Con toda esta información, se pueden separar los métodos implementados en los siguientes conjuntos:

- Métodos de lectura: En este conjunto se listan todos los métodos encargados de leer el fichero del modelo, así como de interpretar, procesar y separar los componentes de cada línea del fichero. Como el formato STL puede venir codificado en ASCII o binario, es necesario crear dos tipos de métodos de lectura; uno para cada codificación.
- Traductores: En caso de que el fichero STL se encuentre codificado en binario, es necesario tener una forma de traducir del binario cada bloque de bits. Es necesario que la clase sea capaz de convertir de binario a enteros de 16 y 32 bits, así como a cifras reales de tipo float (4 bytes), así como el bloque de cabecera de 80 bytes.
- Métodos de agregación: Con la línea de instrucción interpretada, se puede pasar al siguiente punto, que consiste en la agregación del elemento. Como el formato STL ya viene ordenado en función de los triángulos del modelo, se pueden agregar los datos directamente en el MeshFilter. El método de agregación necesario en este apartado es pues la agregación de triángulos.
- Métodos de conversión: Como la conversión al formato correcto de MeshBuilder ya se ha realizado, este apartado se encarga únicamente de crear el GameObject final y asociarlo con la información correcta, como paso final del proceso de importación.

Para este formato de fichero indicar que, al no existir una definición concreta del material del modelo a mostrar, se ha decidido usar un material establecido por defecto, para que la aplicación sea capaz de mostrarlo por pantalla sin ningún problema.

externalReader.cs

El objetivo de esta clase es la de encapsular el resto de código que conforma el importador de ficheros. Cuando esta clase reciba la ruta de un fichero, deberá comprobar su extensión, y en función del resultado, realizar una llamada a alguna de las otras dos clases. Si la extensión del fichero encontrada no se corresponde con alguna de las permitidas, o si existe algún error en la ruta de fichero, se creará un GameObject vacío.

4.5.- Funcionamiento global

Con todos los componentes que conforman las aplicaciones explicados, se pasa a enseñar el funcionamiento global del sistema. Para intentar hacer una explicación lo más clara posible, dicha explicación se realizará junto a una traza del funcionamiento del sistema.

1) La aplicación servidor del sistema debe encontrarse en ejecución en todo momento. Aunque no es imprescindible que se ejecute antes de arrancar la aplicación cliente. Sí que debe encontrarse en marcha antes de que comiencen los procesos de comunicación del cliente.
2) Lo primero que se debe hacer es configurar la dirección IP local en la aplicación cliente. Recordar que tanto el servidor como el cliente deben estar conectados a la misma red de área local LAN, o en caso contrario la comunicación no se podrá realizar de forma correcta. Una vez realizada la configuración inicial, se puede pasar a utilizar la aplicación. La aplicación cliente es capaz de guardar en memoria la última dirección IP indicada. Si las direcciones IP y la red local no varían en ejecuciones futuras, no debería ser necesario tener que reconfigurar de nuevo la aplicación (Figura 4.5.1). El problema de conexión de red se explicará con más detalle en el apartado 4.5.1.

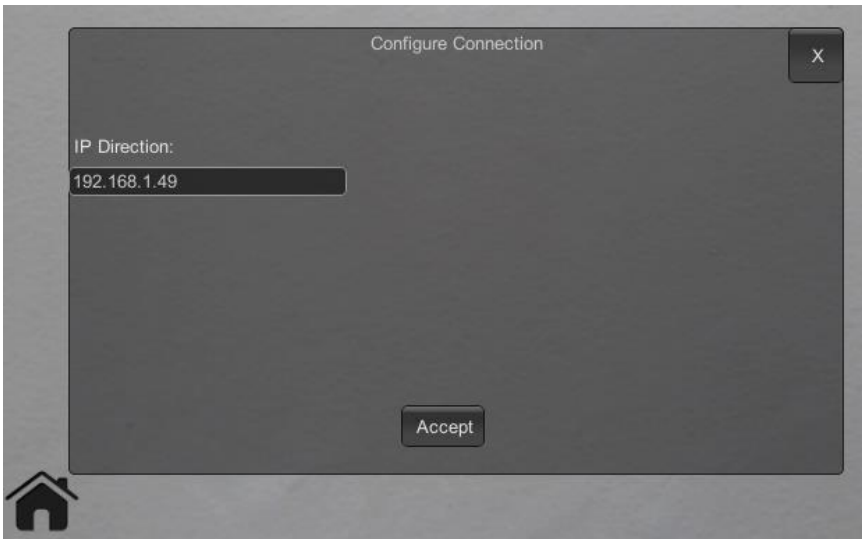

Figura 4.5.1 – Menú de configuración

3) Desde la aplicación servidor, se puede elegir una carpeta objetivo. Esa carpeta es la que se debe utilizar a modo de contenedor para almacenar los ficheros que se quieran visualizar en el cliente. Los ficheros que no se encuentren en esta carpeta no podrán ser accedidos en ningún momento por la aplicación (Figura 4.5.2).

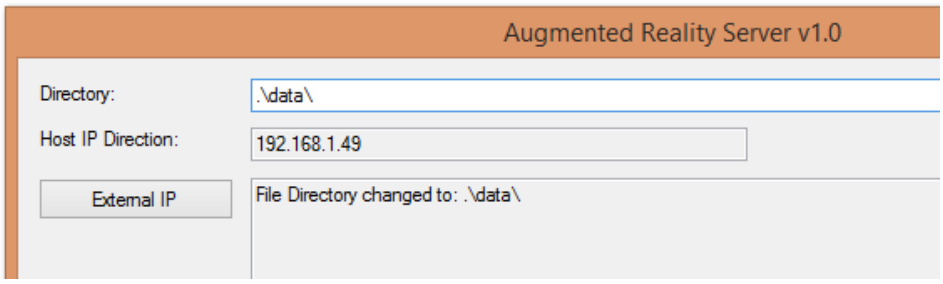

Figura 4.5.2 – Cambio de directorio en servidor

4) El primer botón del menú del cliente se utiliza para obtener el listado de los ficheros que se encuentran almacenados en la carpeta destino del servidor. Cuando el cliente realice una petición al servidor, éste creará un fichero en formato XML con todos los ficheros encontrados en la carpeta y se lo enviará al cliente.
5) El segundo botón del menú leerá el fichero XML obtenido en el paso anterior y mostrará su contenido por pantalla (Figura 4.5.3).

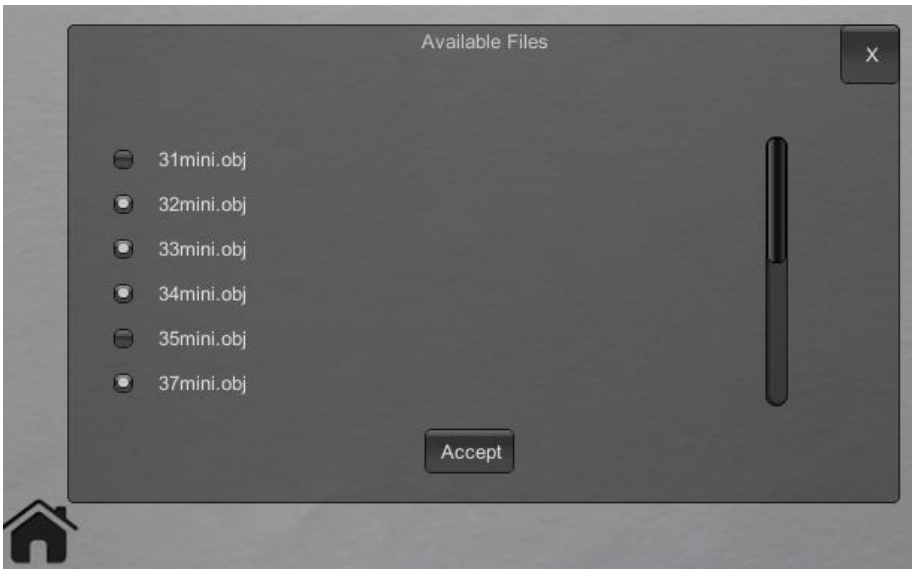

Figura 4.5.3 – Menú de descargar. Selección de ficheros

6) Desde la ventana que se muestra a continuación, se pueden seleccionar una serie de ficheros, y al usar el botón aceptar, se llevará a cabo una petición al servidor para descargar todos los ficheros.
7) Cuando el servidor recibe una petición de descarga múltiple de ficheros, se queda a la espera de recibir el listado de modelos que desea el cliente, y se los irá enviando conforme le lleguen los nombres de dichos ficheros. Cuando los ha enviado todos, el cliente puede pasar al siguiente paso.
8) El tercer botón de la aplicación buscará los ficheros descargados y almacenados en la memoria interna del dispositivo, y los mostrará por pantalla (Figura 4.5.4).

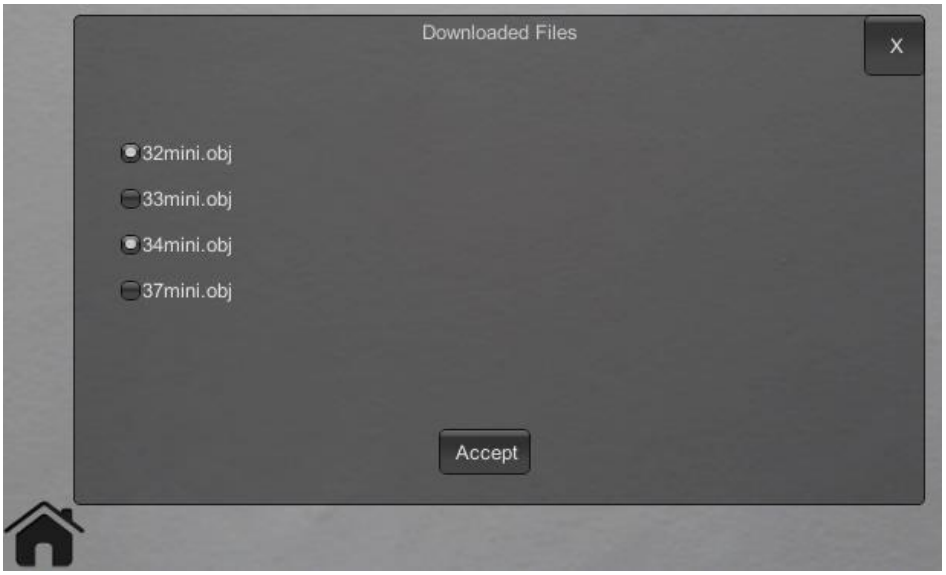

Figura 4.5.4 – Menú de visualización. Selección de modelos

9) De forma similar al paso 6, se puede seleccionar qué modelos se desean visualizar en el dispositivo cliente. Al pulsar el botón aceptar, se le pide al importador que abra y lea dichos ficheros.
10) El importador se encarga de ver la extensión de los ficheros a cargar, y usar el módulo correspondiente para llevar a cabo dicha tarea. Los modelos se cargan en las estructuras correspondientes y se muestran por pantalla con la ayuda del marcador de realidad aumentada (Figura 4.5.5) (Figura 4.5.6) (Figura 4.5.7) (Figura 4.5.8).

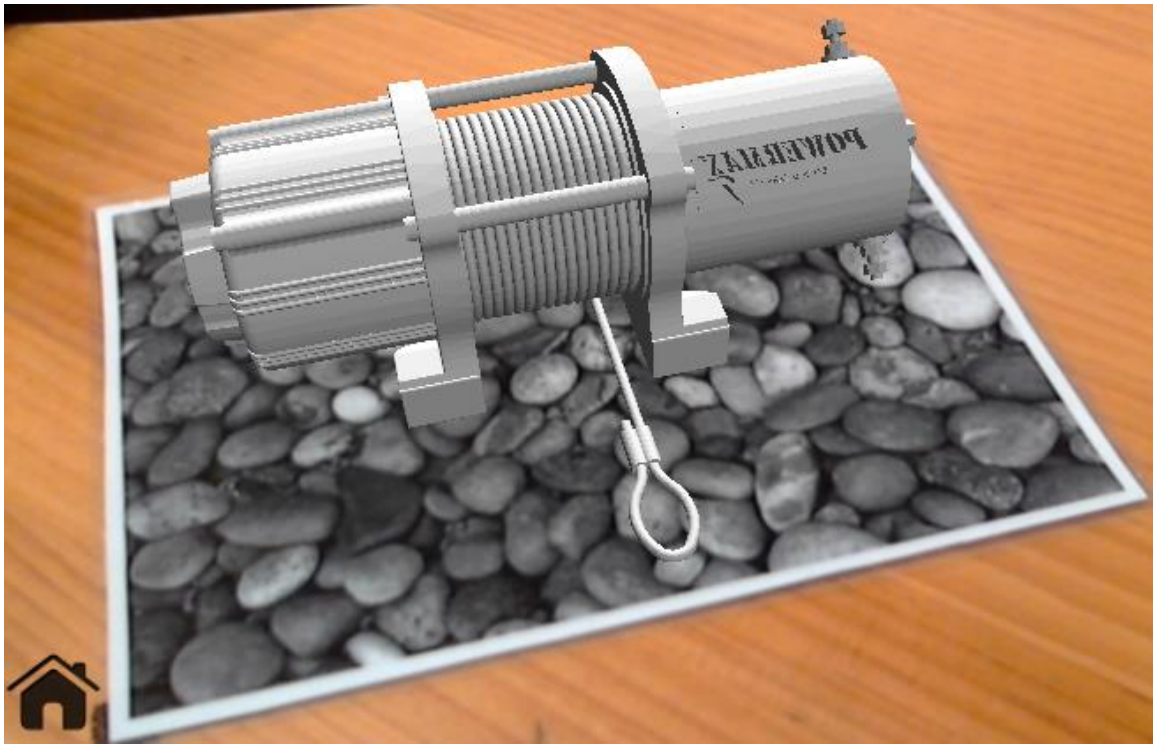

Figura 4.5.5 – Modelo A sobre marcador

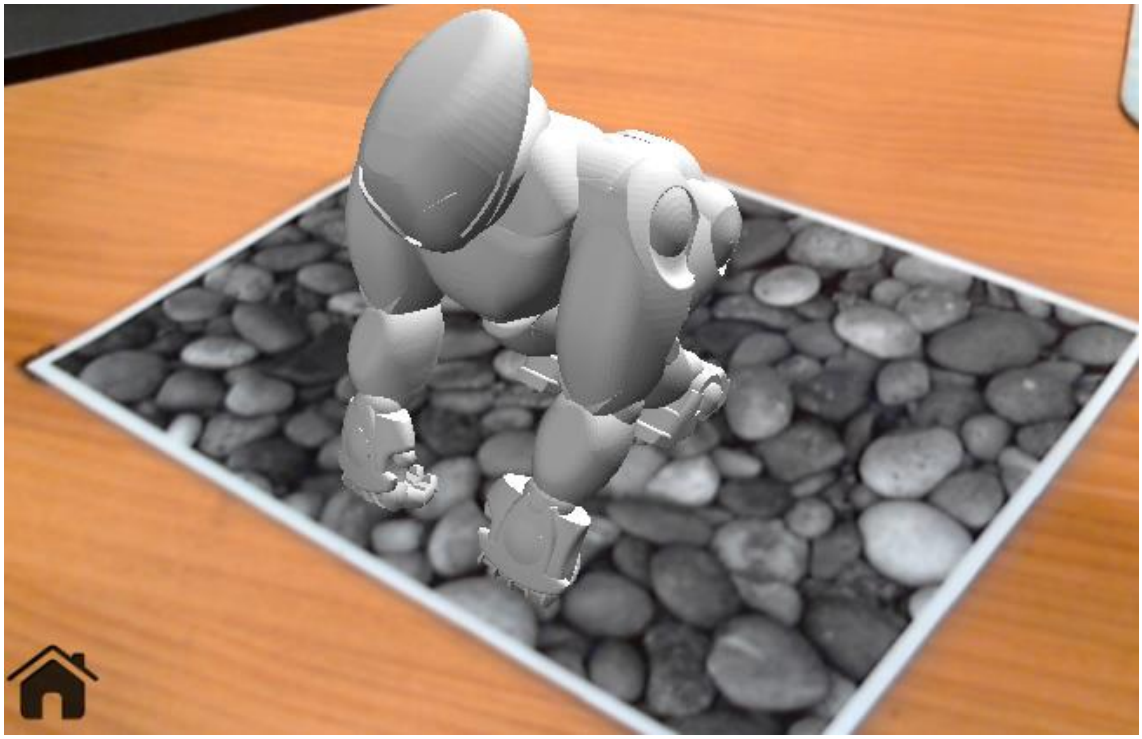

Figura 4.5.6 – Modelo B sobre marcador

11) Los pasos 4, 5 y 8 se pueden repetir tantas veces como se quiera.
12) Los ficheros que se encuentran en la carpeta del servidor pueden modificarse y borrarse sin necesidad de cerrar ninguna aplicación. También pueden añadirse nuevos ficheros sin ningún problema. Por supuesto, sí que será necesario repetir los pasos 4, 5 y 8 para que los cambios tengan efecto en el cliente.

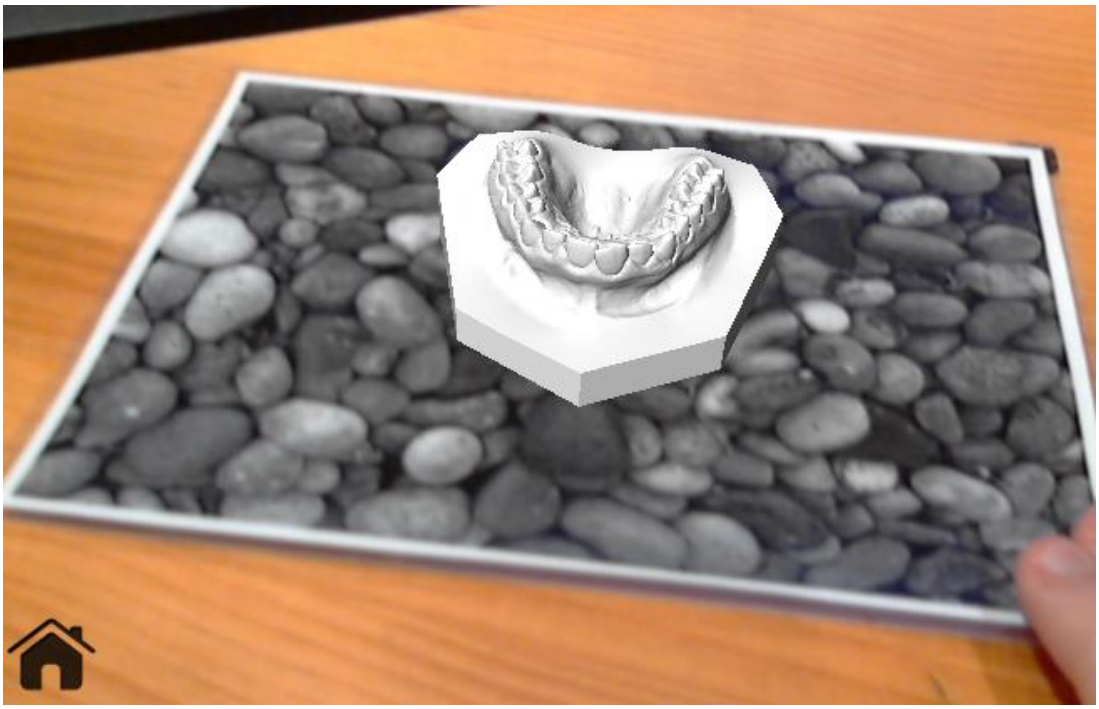

Figura 4.5.7 – Modelo C sobre marcador

Problemas encontrados

A la hora de implementar todo el sistema, se han ido encontrando diversos problemas, que se han tenido que resolver a medida que avanzaba el desarrollo de las aplicaciones. Se van a comentar a continuación dichos problemas, y cómo han sido abordados.

Los primeros problemas que se encontraron están relacionados con la conexión a internet de los sockets TCP. Un socket realiza una conexión directa a través de un puerto entre dos terminales. Para realizar dicha conexión se hace uso de las direcciones IP de las máquinas implicadas. Uno de los problemas surge a raíz de intentar establecer dicha conexión. En el caso de usar una red de área local LAN, el problema se reduce a realizar la configuración correcta de direcciones y puertos. Dejando el puerto fijo y oculto al usuario, esto deja de suponer una molestia. En cambio, configurar la dirección IP en el cliente es diferente y bastante más complejo. Normalmente, un dispositivo con la posibilidad de conectarse a internet tiene configurado por defecto la denominada IP dinámica. Esto es que, la dirección IP no tiene porqué ser siempre la misma. Para poder sortear este problema, es necesario generar una opción en la aplicación que permita configurar de forma manual la misma (lo cual es lo que se ha realizado). El segundo problema, y más grave, ocurre cuando se intentan conectar las dos aplicaciones si se encuentran en redes distintas. Para poder realizar esta conexión, hace falta utilizar la dirección IP pública del servidor. Por desgracia, además de esto, es necesario configurar el router en el que se encuentra conectado el servidor. Dicha configuración debe realizarla el administrador de dicha red. En el caso de que la red sea pública, esta configuración no se podrá llevar a cabo en la mayoría de los casos. Ya que este problema de conexión no es causado por la aplicación en sí misma, no va a ser posible resolverse con facilidad. Teóricamente, si esta configuración se llevara a cabo de forma correcta, no debería haber ningún problema extra para realizar la conexión entre las aplicaciones a través de internet.

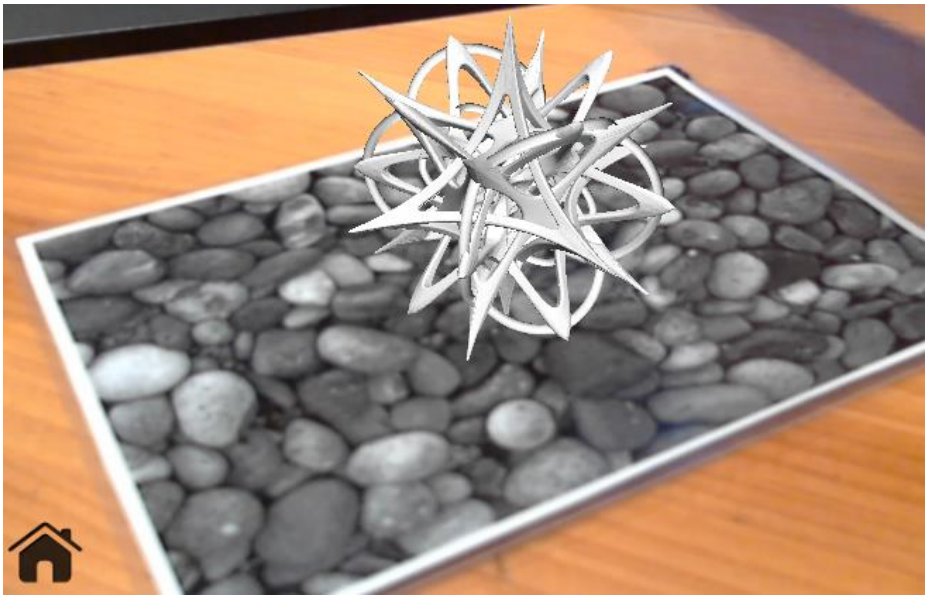

Figura 4.5.8 – Modelo D sobre marcador

El segundo tema problemático surge con los formatos de fichero que almacenan los modelos. En una primera instancia, se deseaba que la aplicación fuera capaz de cargar modelos almacenados en tres formatos: OBJ, STL y 3DS. Por desgracia, no existe una especificación completa, fiable y de código abierto para el tercer formato citado. Debido a esto, los cargadores que se han intentado probar y/o implementar generan resultados imprevisibles y en la mayoría de casos, no satisfactorios. Es por esto que se ha decidido no incluir este formato en la aplicación actual. Si en un futuro se liberara la especificación completa, o se consiguiera crear con éxito un cargador completo, incluirlo en el importador programado no debería suponer un gran esfuerzo por parte del programador. Esto es gracias a la forma modular que se le ha dado al importador de ficheros.

Capítulo 5

PRUEBAS Y RESULTADOS

Con las aplicaciones ya completas. Se deseaba comprobar el rendimiento y observar los tiempos de carga de los modelos. En concreto, se quería analizar la cantidad de modelos que se pueden llegar a cargar en el dispositivo sin que afecta al rendimiento del mismo. Para ello, se han realizado una serie de pruebas de carga sobre dos dispositivos móviles.

Los dispositivos móviles que se han utilizado para las pruebas funcionan sobre el sistema operativo Android. Los móviles usados han sido el BQ Aquaris 5 [ABQ5 16] y el LG Nexus 5 [NEX5 16]. Sus especificaciones técnicas se muestran a continuación (Figura 5.2.1):

	BQ Aquaris 5	**LG Nexus 5**
Medidas	142x71x9,9	137,2x69,2x8,6
Peso	170g	130g
Pantalla	5"	4.95"
Resolución	540x960. 220 PPI	1080x1920. 445PPI
Memoria	16GB	16GB
Procesador	Cortex A7	Qualcomm Snapdragon 800 MSM8974
Reloj / Núcleos	1.2GHz / Quad Core	2.3GHz / Quad Core
Memoria RAM	1GB	2GB
Sistema Operativo	Android 4.2	Android 4.4 KitKat
Conexión red	Wifi 802.11 b/g/n	Wifi 802.11 a/b/g/n/ac

Figura 5.2.1 – Tabla de especificaciones

Destacar que, a primera vista, la potencia del terminal Nexus 5 es mayor que la del Aquaris 5, y esto afectará a los resultados obtenidos en las pruebas.

5.1.- Pruebas realizadas

Para comprobar el rendimiento de la aplicación y de los terminales, se han llevado a cabo dos pruebas.

En la primera prueba, se han medido los tiempos de carga y visualizado de una serie de modelos tridimensionales en cada dispositivo. Las pruebas se han repetido varias veces en cada dispositivo, y se ha obtenido la media de todas las pruebas para cada modelo. Se han utilizado 13 modelos para cada prueba, y se ha lanzado cada prueba un total de 5 veces por dispositivo, aplicando la media aritmética a cada conjunto de muestras. Los resultados se han agrupado en función del formato de fichero.

En la segunda prueba, se ha medido el Frame Rate (FPS) en función del número de vértices de los modelos mostrados por pantalla. Para la realización de esta prueba se han usado los 26 modelos. Durante la visualización de cada modelo, se han adquirido los FPS de cada dispositivo durante intervalos de tiempo fijos, obteniendo la media aritmética de cada intervalo y modelo.

5.2.- Resultados obtenidos

A continuación se muestran las gráficas que representan los resultados que se han obtenido en las pruebas realizadas y las conclusiones extraídas de las mismas.

Tiempos de carga

Las mediciones se han realizado en función del número de vértices y del tiempo que ha tardado cada terminal en cargar dichos modelos (Figura 5.2.2) (Figura 5.2.3).

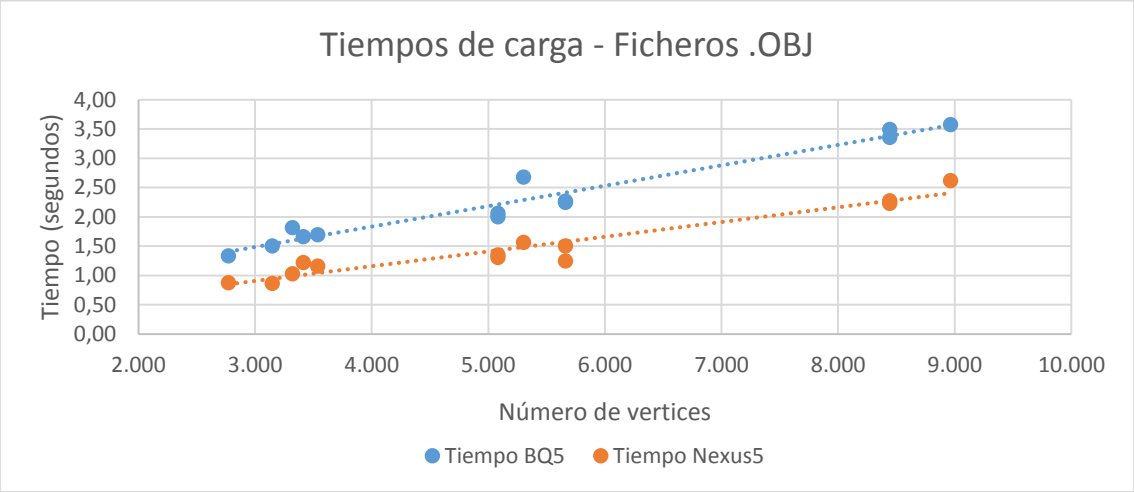

Figura 5.2.2 – Tiempos de carga OBJ

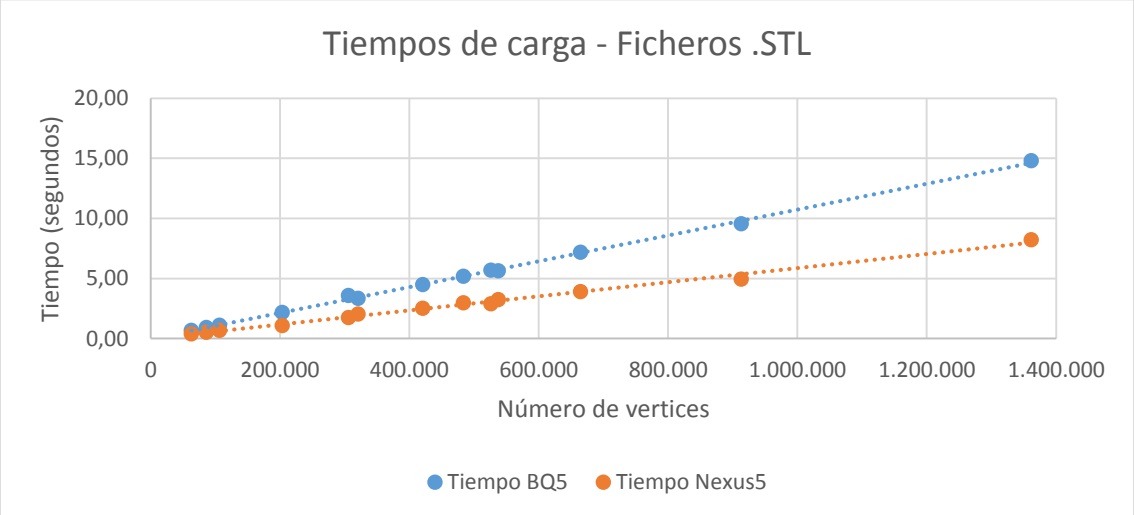

Figura 5.2.3 – Tiempos de carga STL

Lo primero a destacar antes de entrar en detalle con los resultados es que, debido a la potencia superior del terminal Nexus 5 frente al Aquaris 5, es de lógica asumir que los tiempos de carga serán menores en el primero frente al segundo.

Centrándonos en los datos, lo primero que llama la atención es el tiempo de carga de los modelos en formato OBJ frente al formato STL, en función del número de vértices. El formato STL se carga mucho más rápido que el formato OBJ, pese a que los modelos del formato STL tienen una cantidad muchísimo mayor de vértices que los modelos del formato OBJ. Se ha llegado a la conclusión de que esto es debido a la forma en que están codificados dichos formatos. La codificación en ASCII es mucho más lenta de procesar que la codificación directamente en binario. Además, al saber en todo momento qué datos esperar en la codificación binaria de STL hace que la lectura de estos ficheros sea mucho más rápida y eficiente.

Se puede apreciar que los tiempos de carga crecen de forma lineal con el número de vértices, lo cual se corresponde con lo que se esperaba que ocurriera antes de realizar las pruebas. Destacar que la ralentización que pueda sufrir la aplicación tanto a la hora de cargar como de visualizar los modelos se va a deber al número de polígonos totales que se estén mostrando por pantalla, y no al número de modelos en sí mismo. Centrándose en el formato STL, cualquier combinación de modelos cuya combinación de vértices no sobrepase los 200.000 vértices tendrá unos tiempos de carga lo suficientemente bajos como para no resultarle molestos a un usuario en lo que a rendimiento se refiere. Por el contrario, cualquier carga que supere este límite empezará a notarse en la fluidez de la visualización a través de la cámara, así como en el funcionamiento global del terminal. Sobrepasar los 600.000 vértices supone tiempos de carga superiores a los 4 segundos en el mejor de los casos, y no es recomendable cargar tal cantidad de polígonos con dispositivos móviles de estas características, tanto por motivos de memoria, como de rendimiento, como de batería.

Para finalizar, comentar que siempre que sea posible escoger el tipo de formato del modelo, se debería escoger STL sobre OBJ, ya que posee menores tiempos de carga, y permite cargar figuras más complejas en menos tiempo. Por supuesto, para figuras pequeñas o para casos en los que los tiempos de carga no sean de una importancia relevante, se puede usar libremente el formato que sea. Eso sí, recalcar una vez más, que el rendimiento de los terminales empieza a resentirse alrededor de los 200.000 vértices mostrados por pantalla.

Fotogramas por segundo

Las mediciones se han realizado en función del número de vértices mostrados por pantalla y de los FPS de los intervalos de tiempo en los que se estaban visualizando los modelos (Figura 5.2.4).

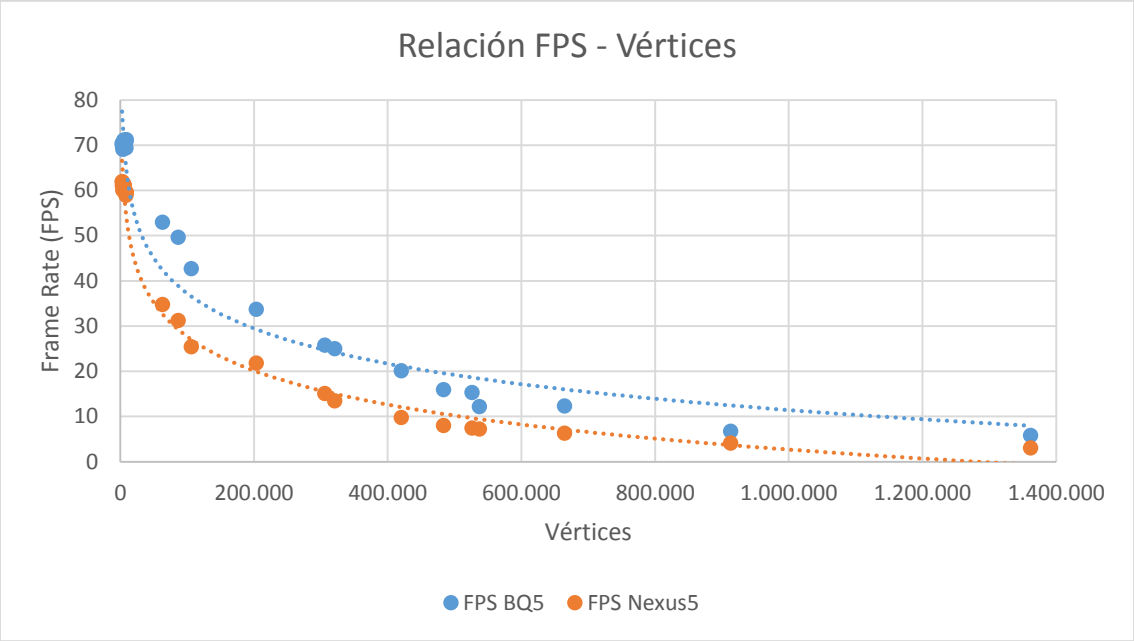

Figura 5.2.4 – Relación FPS – Vértices

Lo primero que se puede apreciar es que los FPS medidos para el Smartphone Aquaris 5 son más elevados que en el móvil Nexus 5. Esto se debe al tamaño de las resoluciones de pantalla de los dos dispositivos. La resolución en el caso del Nexus 5 es prácticamente el doble que en el Aquaris 5.

Se puede apreciar claramente como en visualizaciones que quedan por debajo de los 100.000 vértices se obtiene alrededor de unos 30FPS en el caso del Nexus 5 y 40FPS en el caso del Aquaris 5, mientras que con menor cantidad de vértices, estos modelos se pueden visualizar prácticamente a la velocidad máxima permitida por cada Smartphone. Por otra parte, es a partir de los 200.000 vértices que ambos dispositivos empiezan a resentirse de la cantidad de polígonos mostrados por pantalla y empiezan a sufrir ralentizaciones. Durante las pruebas también se pudo apreciar que a partir de este umbral ambos dispositivos empezaban a calentarse en exceso debido al requerimiento de cálculo por parte de los procesadores gráficos, además del alto consumo de batería asociado a esta necesidad. A partir de los 600.000 vértices se obtienen unas mediciones alrededor de los 10FPS llegando a bajar a unos 3FPS con más de 1.000.000 de vértices en pantalla.

A raíz de estos resultados se llega a la conclusión de que, cualquier combinación de modelos que no supere los 200.000 vértices se podrán visualizar sin ningún problema. No es nada recomendable superar dicha limitación, siendo incluso interesante bloquear a nivel interno cualquier intento de sobrepasar dicho límite, por motivos de seguridad para el dispositivo.

Capítulo 6

CONCLUSIONES

Mediante la realización de este proyecto, a nivel personal, se ha conseguido reforzar los conocimientos relacionados con el desarrollo de aplicaciones que puedan conectarse a una red de área local y compartir información. Por otra parte, se ha aprendido a utilizar la popular plataforma de desarrollo Unity para la creación de aplicaciones gráficas para dispositivos móviles basados en Android. Además, mediante la utilización de Vuforia, se ha conseguido apreciar la utilidad y potencia de las herramientas de Realidad Aumentada a la hora de interactuar con los usuarios y de generar contenido.

Respecto al proyecto, se han conseguido llevar a cabo los objetivos planteados. Mediante la creación de un sistema cliente-servidor, se ha conseguido separar el almacenamiento de los ficheros del visor, permitiendo así liberar la memoria de los dispositivos móviles, aprovechando la capacidad de almacenamiento superior de los ordenadores de sobremesa. Además, los ficheros con los datos se pueden modificar en cualquier momento, haciendo que no sea necesario tener que volver a compilar o reinstalar ninguna de las aplicaciones en ningún momento.

Mediante la investigación de las tecnologías basadas en RA, se ha observado que a día de hoy existen multitud de estudios y herramientas que giran en torno a esta tecnología, además de la existencia de empresas de desarrollo de software importantes en el sector del entretenimiento y del desarrollo de videojuegos. Queda patente que la investigación y explotación de la realidad aumentada es un campo abierto y que aún queda una gran vía abierta para la investigación y desarrollo de nuevos dispositivos, simuladores y métodos que usen la realidad aumentada dentro de múltiples campos y diversas formas. Además la utilización de la RA puede desembocar en la creación de potentes herramientas de simulación y aprendizaje, como puede ser en el entorno de la medicina o en la industria de desarrollo.

Otro de los aspectos tratados en el proyecto ha consistido en la creación de un sistema de importación de ficheros. Al haberse desarrollado de forma modular y por separado del resto del proyecto, queda abierta la opción de poder ampliarlo a medida que aparezcan nuevos formatos de fichero de código abierto con su especificación disponible. Además, al estar desarrollado en el lenguaje C#, éste puede ser usado en cualquier otra aplicación basada en .NET mediante la creación de una librería externa. La habilidad de poder revisar y procesar estos ficheros en tiempo de ejecución ha sido vital para llevar a buen término el desarrollo de las aplicaciones presentadas.

Para finalizar, y a modo de opinión personal, destacar la utilidad y potencia de la realidad aumentada, y que es cuestión de tiempo que empiece a integrarse en la vida cotidiana, con herramientas de apoyo y guía. En concreto, pienso que esta tecnología podría llegar a tener una utilidad incalculable a la hora de desarrollar aplicaciones que sirvan de ayuda a personas con discapacidades, tanto físicas como mentales.

6.1.- Futuras expansiones

Aunque las aplicaciones están completas y son totalmente funcionales, en una aplicación de estas características siempre hay espacio para la mejora y ampliación. A continuación se dejan planteadas una serie de posibles mejoras, que por no formar parte del objetivo principal del desarrollo y de la problemática estudiada en este proyecto, se plantean para un futuro:

- Aunque la conexión entre las aplicaciones sobre una red de área local funciona sin problemas, se considera que la solución óptima sería preparar un servidor web. Debido a que la aplicación está dirigida a un público que no tiene por qué tener ningún tipo de conocimiento relacionado con el mantenimiento de servidores, esta idea fue descartada. Además, se deseaba tener acceso rápido a los ficheros que se podían cargar en el cliente. Como ampliación o modificación, sería interesante buscar una forma lo más económica posible de preparar un servidor para alojar los ficheros con los modelos, con la dificultad añadida de que dicho servidor sea lo más fácil de configurar y mantener por una persona que no posea unos conocimientos avanzados en este campo.
- A nivel de interfaz gráfica, se han utilizado los aspectos y configuraciones básicas que ofrece Unity. Crear una interfaz exclusiva y con aspectos propios le daría mayor personalidad a la herramienta, haciéndola más agradable de utilizar para el usuario.
- A medida que surjan nuevos formatos de almacenamiento de modelos, se pueden agregar extensiones al importador de ficheros para que los cubran, ampliando las capacidades de la aplicación cliente.

Capítulo 7

BIBLIOGRAFÍA

[ABQ5 16] Especificación técnica de Smartphone Bq Aquaris 5 - http://www.bqreaders.com/productos/aquaris-5.html, fecha de consulta: 08/09/2014.

[CLR 01] Common Language Runtime (CLR) Documentation - http://msdn.microsoft.com/es-es/library/8bs2ecf4(v=vs.110).aspx, fecha de consulta: 02/09/2014.

[DISE 13] Di Serio, Á., Ibáñez, M. B., & Kloos, C. D. (2013). Impact of an augmented reality system on students' motivation for a visual art course. Computers & Education, vol. 68, 586-596.

[FURIO 01] Furió, D., González-Gancedo, S., Juan, M.C., Seguí, I., Rando, N. (2013), Evaluation of learning outcomes using an educational iPhone game vs. traditional game, Computers & Education, vol. 64, pp. 1-23.

[FURIO 02] Furió, D., González-Gancedo, S., Juan, M.C., Seguí, I., Costa, M. (2013). The effects of the size and weight of a mobile device on an educational game, Computers & Education, vol. 64, pp 24-41.

[HELIX 14] Web de desarrollo 3D HelixToolkit for .NET - https://github.com/helix-toolkit, fecha de consulta 01/09/2014.

[HEND 11] Henderson, S., Feiner, S. (2011), "Exploring the Benefits of Augmented Reality Documentation for Maintenance and Repair," Visualization and Computer Graphics, Transactions on, vol.17, no.10, pp.1355, 1368.

[JUAN 05] Juan, M.C., Alcañiz, M., Monserrat, Botella, C., Baños, R.M. and Guerrero, B. (2005). Using augmented reality to treat phobias, IEEE Computer Graphics and Applications, vol. 25, no. 6, pp. 31-37.

[JUAN 10] Juan, C. M., Toffetti, G., Abad, F., & Cano, J. (2010). Tangible cubes used as the user interface in an augmented reality game for edutainment. In IEEE 10th international Conference on Advanced Learning Technologies (ICALT), pp. 599-603.

[MEHM 12] Mehmet, K., Yasin, O. (2012), Augmented Reality in Education: Current Technologies and the Potential for Education, Procedia - Social and Behavioral Sciences, vol. 47, pp. 297-302.

[MONO 14] Web oficial de Monodevelop – http://www.monodevelop.com/, fecha de consulta: 02/09/2014.

[NET 13] Web oficial de Microsoft .NET Framework – http://www.microsoft.com/net/, fecha de consulta: 02/09/2014.

[NEX5 16] Especificación técnica de Smartphone LG Nexus 5 - http://www.movilcelular.es/movil/lg-nexus-5-d821-16gb/1245, fecha de consulta: 08/09/2014.

[NOVA 14] Web oficial de los estudios Novarama - http://www.novarama.com/, fecha de consulta: 07/09/2014.

[OBJ 14] Información sobre la especificación del formato de fichero Wavefront OBJ - http://www.fileformat.info/format/wavefrontobj/egff.htm, fecha de consulta: 03/09/2014.

[STL 99] Información sobre la especificación del formato de fichero STereoLithography STL - http://www.ennex.com/~fabbers/StL.asp, fecha de consulta: 03/09/2014.

[UNIT 14] Web oficial de Unity – http://www.unity3d.com/, fecha de consulta: 04/09/2014

[VAN 10] Van Krevelen, D. W. F., & Poelman, R. (2010). A survey of augmented reality technologies, applications and limitations. In *International Journal of Virtual Reality*, vol. 9, no. 2, p. 1.

[VUFO 01] Web oficial de desarrollo de Vuforia – http://developer.vuforia.com/, fecha de consulta: 02/09/2014.

[VUFO 02] Web de Vuforia Qualcomm para la creación y gestión de marcadores - https://developer.vuforia.com/target-manager, fecha de consulta: 04/09/2014.

[WPF 45] Windows Presentation Foundation Documentation - http://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx, fecha de consulta: 02/09/2014.

[ZHOU 08] Zhou, F., Duh, H. B.-L., Billinghurst, H. B.-L. (2008) Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In 7th Int'l Symp. On Mixed and Augmented Reality (ISMAR'08), pp. 193–202.