Herramientas de desarrollo libres para aplicaciones de Realidad Aumentada con Android. Análisis comparativo entre ellas

Universidad Politécnica de Valencia

DSIC

Trabajo fin de Máster

Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Ana Serrano Mamolar

Supervisado por:

M. Carmen Juan Lizandra

M. José Vicent López

Septiembre 2012

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

DEFINICIONES Y ABREVIATURAS

A lo largo de este trabajo aparecen una serie de términos que son utilizados con un significado concreto. A continuación se muestra una lista con las definiciones y abreviaturas utilizadas interesante de aclarar.

Realidad Aumentada (RA): Tecnología que combina técnicas de reconocimiento de formas y visualización 3D para añadir virtualidad a una imagen real de forma coherente y en función de la localización de la escena real. Ver Sección 2.1.

Dispositivo móvil: aparatos electrónicos, generalmente de pequeño tamaño, con capacidades de procesamiento, conexión de red, memoria limitada y autonomía eléctrica, diseñados específicamente para una función y que puede ser manejado con dos manos. Los más comunes son los teléfonos móviles y tabletas.

Smartphone: Teléfono inteligente. Teléfono móvil con mayores prestaciones de procesamiento y conectividad que un teléfono normal.

Kit de desarrollo de software (SDK): Conjunto de herramientas de desarrollo software que permite al desarrollador crear aplicaciones para un sistema concreto.

Frames por segundo (fps): Número de imágenes por segundo o frecuencia a la cual se genera distintos fotogramas o imágenes.

Marker: marcador o imagen utilizada en algunas aplicaciones de realidad aumentada y cuya posición condiciona el posicionamiento del objeto.

Target: marcador natural o imagen que forma parte del entorno natural de una aplicación de RA y que sustituya a los clásicos marcadores.

Head Mounted Display (HMD): dispositivo de visualización similar a un casco, que permite reproducir imágenes creadas por ordenador sobre un "display" muy cercano a los ojos.

6DOF: Seis grados de libertad. Se refiere al movimiento de los objetos en el espacio tridimensional o capacidad de realizar los tres movimientos de traslación combinados con los tres de rotación.

Capítulo 1

INTRODUCCIÓN

1.1. Motivación

Se prevé que el número de usuarios de smartphones en el mundo ascienda a 1 billón en 2013 [IDC 09]. En España hoy el 55,6% de la población lo utiliza. No cabe duda de que los smartphones se han instaurado como una herramienta necesaria de nuestro día a día. De hecho, estos nuevos dispositivos han revolucionado las cifras de compras por internet, y sectores que están aprovechando estas nuevas tecnologías como el sector publicidad, turismo, entretenimiento e incluso educación.

Por otro lado, la realidad aumentada (RA) está cada vez más presente en la era digital, como nueva forma de conocer el mundo que nos rodea. En este sentido, los smartphones suponen una alternativa muy ventajosa para el desarrollo de este tipo de aplicaciones, porque hoy en día ya se puede decir que prácticamente cumplen con todos los requerimientos software y hardware que éstas exigen. Así pues, la RA ya no está ligada sólo a aplicaciones en PC o dispositivos especiales, sino que puede ser aplicada a dispositivos móviles corrientes, al alcance de todo el mundo, como son los smartphones.

Sin embargo, al mismo tiempo, pese a la proliferación de aplicaciones muy heterogéneas implementadas por desarrolladores independientes para estos dispositivos gracias a la sencillez de difusión y adquisición de ellas, las aplicaciones de RA están aún lejos de ello. La principal causa de este hecho es la escasez de herramientas libres que permitan generar de manera sencilla este tipo de aplicaciones. En algunos casos se ha adaptado, con gran esfuerzo, algún framework existentente para PC, pero sin lograr explotar todas las posibilidades que ofrece esta tecnología.

Cabe señalar, que la mayor parte de las aplicaciones de RA para aplicaciones móviles presentadas por la comunidad científica son de complejidad baja. En general consisten en ejemplos básicos diseñados con el propósito de demostrar el funcionamiento de la librería de RA subyacente, ocultando así la cantidad de posibilidades que podría ofrecer la RA en situaciones reales.

Es por esto que en el presente trabajo se realiza una investigación de las diferentes herramientas libres disponibles para desarrolladores de RA que quieran ver sus aplicaciones funcionando en un dispositivo móvil. Así pues, en este trabajo se realiza un análisis minucioso de estas herramientas, comparando los aspectos más importantes para un desarrollador que se inicie en esta disciplina.

Actualmente el mercado de los smartphones está liderado por dos plataformas, Android y iPhone [IDC 09], como puede verse en la *Figura 1-1* .

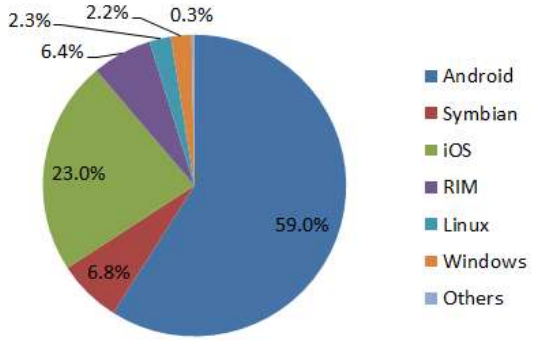

Figura 1-1. Uso de sistemas operativos para smartphones

Comparando ambas plataformas, Android proporciona más libertad al desarrollador [CHENG 10]. Al ser de código abierto, el desarrollador puede crear e instalar cualquier aplicación en cualquier dispositivo. Además Android está más extendida y disponible en más dispositivos, mientras que iPhone está sólo disponible para los productos de Apple, como el iPhone, iPod y iPad. Hecho que sitúa a Android en una posición más ventajosa, pues, al tener que adaptarse a los requerimientos de diferentes dispositivos su funcionalidad aumenta. Además, ello permite que los desarrolladores de Android hagan aplicaciones más útiles y potentes basadas en los requerimientos de los usuarios.

Nos hemos centrado en el desarrollo de aplicaciones para sistemas Android, además de por su mayor expansión, por ser una plataforma libre y gratuita con las ventajas que eso conlleva desde el punto de vista del desarrollador.

Se considera un aspecto importante el hecho de trabajar con software libre ya que proporciona una mayor libertad para los desarrolladores. El hecho de tener acceso al código fuente y la posibilidad de modificarlo, nos ha permitido realizar una comparativa más exhaustiva de cada una de las librerías a comparar.

Tras un estudio exhaustivo se ha seleccionado una de las librerías como mejor opción para el desarrollo de aplicaciones de RA en dispositivos móviles. Un buen SDK debería integrar fácilmente tecnologías gráficas de alta complejidad, así como

generación de las propias aplicaciones de RA, cumpliendo al mismo tiempo, al menos, con estas características básicas:

- Funcionalidad: El desarrollador cuenta con un amplio conjunto de herramientas y tecnologías para el desarrollo de sus aplicaciones, sin complicaciones. Obviamente todas las funcionalidades deben ser integrables con dispositivos móviles.
- Código abierto: para cualquier desarrollador esta característica resulta esencial a la hora de modificar o mejorar el código suministrado y poder así adaptarlo a sus requerimientos técnicos y funcionales.
- Escalable: fácilmente integrable con las nuevas tecnologías que vayan surgiendo (gráficos más complejos, nuevas formas de interacción de usuario con la aplicación, etc.).
- Eficiente: la eficiencia se mide tanto en la etapa de reconocimiento y seguimiento o tracking, como en la generación de gráficos. Las aplicaciones de RA actuales son cada vez más exigentes, y su migración a los dispositivos móviles no debería minar estas exigencias.
- Facilidad de uso: Aunque se presuponen conocimientos de programación, la no especialización en tecnología gráfica o de reconocimiento de formas no debería ser un impedimento para poder desarrollar aplicaciones básicas.

Este proyecto se enmarca dentro del proyecto “Desarrollo y validación de una librería de Realidad Aumentada con reconocimiento de características naturales”, con código 2004 de la convocatoria de 2011, financiado por la UPV.

1.2. Bases de partida de la investigación y objetivos científicos

Como ya se ha comentado en la sección anterior, el principal objetivo de este trabajo de investigación consiste en **estudiar las herramientas existentes para el desarrollo de aplicaciones de RA para dispositivos móviles con sistema operativo Android e identificar ventajas e inconvenientes entre ellas.**

El desarrollo de aplicaciones de RA implica la resolución de una serie de problemas de cierta complejidad que varía en función de las peculiaridades que las propias aplicaciones lleven consigo.

Uno de los problemas con los que se encuentra el desarrollador a la hora de crear su primera aplicación de RA lo ocasionan todas las tareas relacionadas con la propia

interacción usuario máquina. Concretamente, en las aplicaciones de RA basadas en visión como las que aquí se van a tratar, destacan como tareas complejas la obtención de la posición del modelo a aumentar en función de la escena real, el dibujado de las distintas capas gráficas y la interacción con los objetos virtuales.

Es por ello que ya existen distintos frameworks desarrollados que ayudan a resolver estas tareas, y son éstos el objeto de estudio del presente trabajo. Los principales aspectos a tener en cuenta a la hora de analizar un kit de desarrollo (SDK) de RA son los siguientes:

- **Tipo de marcadores**. Como se comentará más adelante se han hecho grandes avances en esta fase de reconocimiento, pudiendo llegar a utilizar marcadores 3D en lugar de los típicos 2D así como marcadores naturales.
- **Velocidad y eficiencia en el reconocimiento**. La rapidez con la que un marcador es detectado influye enormemente en el aumento del grado de satisfacción del usuario de la aplicación. Por otro lado la eficiencia de la aplicación también se mide en el modo en el que el marcador permanece siendo reconocido frente a cambios de orientación, iluminación, o ser parcialmente ocultados.
- **Renderizado de objetos virtuales**. Los objetos a “aumentar” pueden ser desde los más simples a los más complejos. Dependiendo de las características del SDK a utilizar podremos ampliar o reducir esta gama. Además hay que tener en cuenta el tipo de formato 3D admitido por el SDK. Todo ello influirá en la tasa de frames por segundo de la etapa de renderizado de la aplicación y por tanto del grado de satisfacción del usuario.

Por lo tanto, los objetivos secundarios del presente trabajo son:

- Llevar a cabo un estudio exhaustivo para identificar todas las herramientas existentes de código abierto para el desarrollo de aplicaciones de RA para Android.
- Hacer funcionar todas las herramientas en el dispositivo, estudiar la arquitectura básica de éstas y conocer sus características principales.
- Diseñar una batería de pruebas común que permita identificar la herramienta que ofrece mejores prestaciones.
- Analizar los resultados obtenidos y obtener conclusiones.

Para conseguir estos objetivos, el trabajo se divide en tres etapas:

1. **Estudio previo de cada framework.**

   En esta etapa se consigue familiarizarse con todos los SDK a estudiar, incluidos los necesarios para aplicaciones Android. En este apartado también se indicarán las dificultades encontradas a la hora de instalar cada uno de ellos para facilitar la tarea a futuros desarrolladores. También se incluyen los pasos a realizar para poder empezar a desarrollar aplicaciones Android.
   En esta etapa se estudiarán aspectos básicos de cada framework para familiarizarse con su estructura y funcionamiento, averiguar sus primeras características diferenciadoras del resto y sacar las primeras conclusiones.

2. **Homogenización de las aplicaciones a realizar**.

   Para poder comparar de manera justa las distintas librerías, deben estar sometidas a las mismas condiciones. La complejidad o no de éstas es algo relativa, pues depende de las capacidades ofrecidas por cada framework. En cualquier caso, para una correcta comparación se debe:

   - ✓ Conseguir que todos los frameworks funcionen con el mismo marcador.
   - ✓ Conseguir que todos los frameworks aumenten el mismo objeto, lo que significa unificar formatos o realizar transformaciones precisas. Los objetos a aumentar deben ser iguales para poder comparar la capacidad de renderizado de cada librería, esto es, al menos deben tener el mismo número de polígonos.

3. **Preparación de batería de pruebas en base a los objetivos planteados y a los resultados obtenidos en la etapa anterior.**

   Tras un estudio previo realizado de las características y capacidades de cada librería en base a su documentación y pruebas iniciales, se establece una batería de pruebas común para todos los SDKs de manera que puedan extraerse conclusiones interesantes de cara a futuros desarrolladores. Esta batería de pruebas mide:

   - ✓ La capacidad de reconocimiento de marcas naturales
   - ✓ La capacidad de reconocimiento del marcador cuando éste está ligeramente oculto por otro objeto que lo oculta.

- ✓ La capacidad de renderizado medida en frames por segundo en función del número de polígonos del objeto a renderizar y del número de objetos.
- ✓ La efectividad de la aplicación frente a la modificación de las condiciones del entrono, cambios de iluminación, cambio de posición del marcador, tamaño del marcador, etc.

4. **Ejecución de pruebas, análisis de resultados y extracción de conclusiones.**

   En esta etapa, la más importante, se realiza el testeo de cada framework en base a la batería de pruebas elaborada en la etapa anterior y las aplicaciones básicas desarrolladas en la primera etapa. En la actualidad no existe ningún estudio comparativo de este tipo, de librerías específicas para el desarrollo de aplicaciones Android de RA. Existen comparativas "comerciales" de otros frameworks, pero nunca desde el punto de vista del desarrollador.

   Tras la realización de todas las pruebas se elabora una tabla resumen donde se especifican los parámetros más diferenciadores, como posibilidad de utilizar el marcador parcialmente tapado, tipo de marcadores a utilizar, rendimiento en el reconocimiento del marcador o rendimiento de la fase de renderizado, entre otras.

   Una vez realizado el estudio se obtienen las conclusiones finales y se plantean trabajos futuros.

1.3. Estructura

El resto del documento está estructurado como sigue:

Capítulo 2: Se realiza una introducción a la historia de la RA, para lectores no familiarizados, hasta llegar al actual paradigma de la RA en dispositivos móviles. Revisión de la literatura más significativa, clasificación de alternativas e introducción inicial a las herramientas que se estudian.

Capítulo 3: Se realiza una presentación de las herramientas seleccionadas para el presente trabajo, así como API de Android y otros frameworks utilizados para la creación de objetos, marcadores, etc..

Capítulo 4: Se detallan las tareas realizadas para la homogenización de las aplicaciones desarrolladas con cada librería. Se detallan las pruebas ejecutadas para su estudio comparativo.

Capítulo 5: Se presentan las conclusiones y se introducen los trabajos futuros.

.

Capítulo 2

ESTADO DEL ARTE

2.1. Introducción a la realidad aumentada

Existen muchas definiciones de RA, pero hay dos que son las más comúnmente aceptadas [AZUMA 97] y [MILGRAM 94].

La definición de [AZUMA 97] dice que la RA:

- Combina elementos reales y virtuales.
- Es interactiva en tiempo real.
- La registración es en 3D.

Se dice que la RA es un híbrido entre el mundo real y el mundo virtual. Paul Milgram [MILGRAM 94] clasificó por primera vez los distintos espacios de realidad "mixta" desde el punto de vista de continuidad del contexto. En la Figura 2-1, Milgram definía un modelo "continuo virtual". Este concepto describe que existe una escala continua entre lo completamente real y lo completamente virtual. Entre ambos existe la virtualidad aumentada (está más próxima al entorno virtual) y la RA (más próxima al entorno real). La posición de la RA en este sencillo esquema indica que, en esta, la mayor parte de la información es real, incluyéndose complementos virtuales.

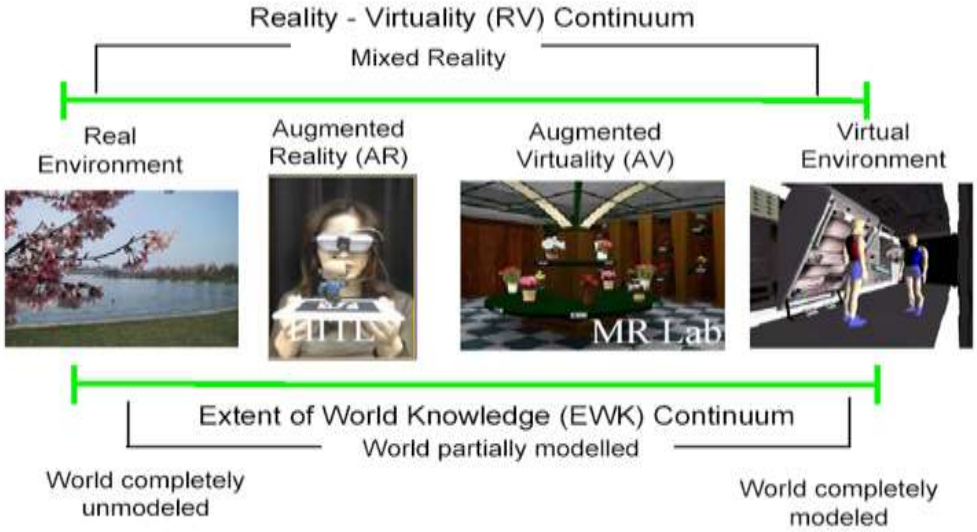

Figura 2-1. Reality-Virtuality continuum.

Estos complementos virtuales suelen ser modelos creados por ordenador, objetos 3D, o 2D aunque son menos frecuentes. Estos objetos podrían ser incluso texto, iconos o cualquier imagen que amplíe la información de la imagen real. En algunas aplicaciones específicas se ha utilizado vídeo, sonido e incluso olores, pero de aquí en adelante sólo haremos referencia a los objetos 3D como parte de la RA.

Actualmente, el término RA se ha extendido enormemente debido al creciente interés por esta tecnología del público en general. Sin embargo en términos generales podemos decir que la RA permite enriquecer la perspectiva del usuario mediante la superposición de objetos virtuales en el mundo real de manera que convenza al usuario de que estos objetos forman parte del entorno real o ampliar su información.

Se puede decir que la RA adquiere presencia en el mundo científico a principio de los años 1990, cuando el avance de la tecnología permitió combinar imágenes generadas por ordenador con la visión que tiene el usuario del mundo real, es decir, cuando aparecieron los primeros ordenadores de procesamiento rápido, las técnicas de renderizado de gráficos en tiempo real, y los sistemas de tracking portátiles de precisión.

Hoy en día la RA está siendo perfeccionada por diversos grupos e investigación de todo el mundo en las tecnologías involucradas, tales como el tracking de la posición del usuario, procesado de la señal, visualización de la información, visión artificial, generación de imágenes virtuales, renderizado de gráficos, estructuración de la información, y hasta computación distribuida.

La RA se ha aplicado con éxito a numerosas áreas. Entre ellas cabe citar: publicidad y marketing [MEDINA 10] [CONNOLLY 10], turismo [REYTMAYR 03], entretenimiento [SZALAVÀRI 98], cirugía [FUCHS 97] (*Figura 2-2*), industria [REGENBRECHT 05] (*Figura 2-2*), tratamiento de fobias [JUAN 05] [JUAN 06] (*Figura 2-3*) y aprendizaje [GONZÁLEZ 12] [JUAN 11] (*Figura 2-4*). Estos son sólo algunos ejemplos, pero la RA se va introduciendo cada vez más en las actividades cotidianas de cualquier persona.

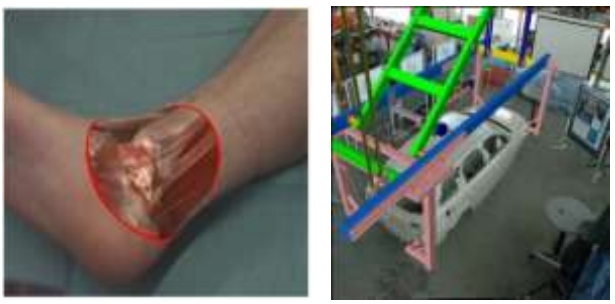

Figura 2-2. Ejemplos de RA aplicada a la cirugía y a la industria.

Figura 2-3. Ejemplo de RA aplicada al tratamiento de fobias (fobia a las alturas y fobia a las arañas)

Figura 2-4. Ejemplo de RA aplicada al aprendizaje

En todo sistema de RA son necesarias cuatro tareas principales para poder llevar a cabo el "aumento" de la realidad. Éstas son (Figura 2-5):

Figura 2-5. Etapas del proceso de Realidad Aumentada

Siguiendo el gráfico de la *Figura 2-5,* en primer lugar, se captura la escena real con la cámara. Después se procede a su procesamiento con el fin de solucionar uno de los principales problemas de la RA: el seguimiento del punto de vista (Viewpoint tracking). Éste es un problema clave dado que condiciona el posicionamiento de los objetos virtuales para su visualización por parte del usuario.

Posteriormente, se pasa a la fase de renderizado del objeto virtual, que debe estar colocado en la posición anteriormente calculada. Y finalmente se pasa a la etapa de

visualización, dónde se superponen las capas real y virtual. En algunas aplicaciones de RA puede existir una fase intermedia entre la tercera y la cuarta que es la que comprende las interacciones del usuario.

Una de las principales características que diferencia a unas aplicaciones de RA de otras es precisamente el método utilizado para el Viewpoint tracking. Para algunas aplicaciones que requieren mucha precisión, como es el caso de la cirugía, el correcto posicionamiento del objeto virtual es esencial.

Existen básicamente dos métodos totalmente diferentes para solucionar el problema del viewpoint tracking:

- **Localización espacial utilizando GPS**: En base a las coordenadas GPS, el software calcula la posición de los objetos virtuales a añadir a la escena. Aunque se dice que la tecnología GPS puede llegar a tener precisión de hasta centímetros, lo habitual son unos pocos metros de precisión, debido al error de triangulación de los satélites. Este método por tanto, no puede utilizarse en sistemas de RA donde la precisión sea crítica. Por el contrario, éste es el método más rápido debido a que sólo necesita conocer la posición del GPS, sin tener que pasar por fase de reconocimiento que suele tener un coste computacional más alto. Se puede aumentar la precisión de estos sistemas mediante brújulas digitales (para conocer la dirección hacia la que estamos observando) u acelerómetros (para conocer la orientación de la cámara). Generalmente, los nuevos smartphones ya integran los tres sensores.
- **Reconocimiento espacial utilizando visión artificial**: éste es un método bastante más complejo a nivel de software y puede, a su vez, dividirse en otros dos:
    - **Reconocimiento de marcadores físicos (o marker tracking)**: este método se basa en la detección de lo que se conoce como “markers”. Suele consistir en un cuadrado blanco y negro con un patrón asimétrico en el interior.

Figura 2-6. Ejemplo de marker de RA

    - **Reconocimiento espacial sin marcadores (Markerless Tracking)**: ésta es una técnica aún más compleja. En este caso el software de RA

debe ser capaz de reconocer diferentes objetos que componen la escena del mundo real. Este método integra muchas técnicas de visión artificial. En las aplicaciones actuales que utilizan este método se suele restringir el reconocimiento a ciertos objetos como caras o manos humanas, superficies, u objetos con una forma concreta. Estos objetos se denominan targets. [LEE 07] [FERRARI 01] [CHEN 08] [SIMON 00].

De aquí en adelante nos centraremos únicamente en los sistemas de RA pertenecientes al segundo grupo, es decir, sistemas con reconocimiento espacial basado en visión artificial. En la *Figura 2-7* puede verse un ejemplo de la ejecución completa del proceso de RA con markers.

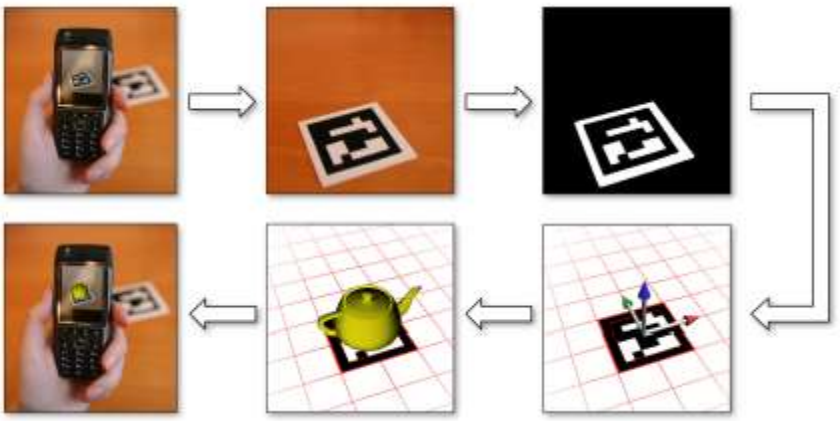

Figura 2-7 Diagrama de funcionamiento ARToolKitPlus. En [WAGNER 07]

2.2. Realidad aumentada en dispositivos móviles

Como ya se ha comentado, la investigación y desarrollo de software para el uso de la RA está en continuo auge. Sus múltiples aplicaciones y su atractivo aspecto visual está haciendo que cada vez más usuarios se interesen por ellos, y que los desarrolladores de software lo integren en sus proyectos.

A continuación se ofrece una visión general de la trayectoria histórica del desarrollo de aplicaciones de RA hasta la introducción en el mundo de los dispositivos móviles. Esta revisión es un resumen del estudio de [Wagner 09].

1968

Ivan Sutherland crea el primer sistema de RA, que también es considerado como el primera sistema de realidad virtual. Se utiliza un HDM seguido por dos mecanismos de rastreo 6DOF, un tracker mecánico y un dispositivo ultrasónico. Debido a la baja capacidad de proceso de los ordenadores del momento, sólo se podían mostrar gráficos alámbricos en tiempo real.

1982

Aparece el primer ordenador portátil con diseño plegable, el Grid Compass 1100. Tenía un procesador Intel 8086, 350 Kbytes de memoria y una pantalla con 320x240 píxeles de resolución; lo cuál era extraordinario para esa época. Sin embargo sus 5kg de peso lo hacía difícilmente portátil.

1992

Tom Caudell y David Mizell acuñan el término “realidad aumentada” para referirse a la superposición del mundo real con información generada por ordenador. En [CAUDELL 92] se discuten las ventajas de la RA frente a la realidad virtual, como la menor potencia de proceso requerida debido al menor peso de las imágenes a renderizar. También reconocen el aumento de los requerimientos de registro con el fin de alinear mundo real y virtual.

IBM desarrolla el primer smartphone que saldría al mercado en 1993. El teléfono tenía 1MB de memoria y una pantalla táctil en blanco y negro de 160x293 píxeles de resolución.

1994

Milgram y Kishino [MILGRAM94] describen en su “Taxonomía de la Realidad Mixta” el conocido término del continuo de Milgram (Reality-Virtuality Continuum).

1995

Rekimoto y Katashi [REKIMOTO95] crean NaviCam, una estación de trabajo con cámara montada que se utilizaba para el seguimiento óptico. El equipo detectaba los marcadores codificados en la imagen de la cámara en vivo y mostraban información directamente sobre la secuencia de vídeo.

1996

Rekimoto [REKIMOTO95] presenta uno de los primeros sistemas de marcadores para permitir el seguimiento de la cámara con seis grados de libertad, los marcadores de matriz 2D (cuadrados con forma de código de barras, ver *Figura 2-8*).

Figura 2-8. Primer marcador 2D que permite 6DOF

1997

Ronald Azuma [AZUMA97] presenta el primer estudio sobre RA. En él presenta su definición de RA, hoy en día ampliamente reconocida.

1998

Bruce Thomas [THOMAS 98] presenta “Map in the hat”, un ordenador montado en una mochila que incluía GPS, brújula electrónica y HMD. Ver *Figura 2-9.*

Figura 2-9. Map in the Hat the B.H. Thomas

1999

Kato y Billinghurst presentan ARToolKit [ART99], una librería de seguimiento con 6DOF, utilizando markers para el reconocimiento de patrones. ARToolKit está disponible como código abierto bajo la licencia GPL y es todavía muy popular en la comunidad RA.

Hollerer et al. [HOLLERER99] presentan el primer sistema de RA móvil basado en GPS y sensores inerciales.

Nace el teléfono Benefon Esc! NT2002, el primer teléfono móvil GSM con GPS integrado. Debido a la limitación de memoria el teléfono descargaba los mapas de navegación bajo demanda.

Se define el protocolo Wireless Network, comúnmente conocido como WIFI.

2000

Julier et al. [JULIER 00] presentan BARS (Battlefield Augmented Reality system). El sistema consiste en un sistema portátil con conexión wifi y HMD (Figura 2-10). El sistema muestra de forma virtual una escena de batalla con información adicional sobre la infraestructura del entorno y sobre posibles enemigos.

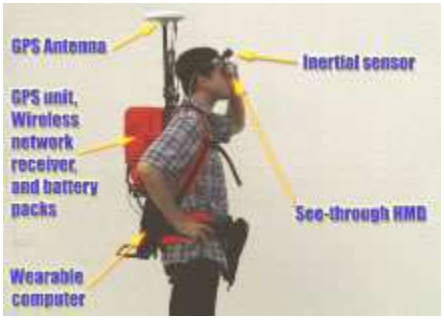

Figura 2-10. Sistema de RA portátil de [JULIER 00]

La empres Sharp lanza el primer teléfono móvil comercial con cámara integrada, el J-SH04- La resolución de la cámara era de 0.1 Megapixels.

2001

Fruend et al. [FRUEND01] presentan AR-PDA, un prototipo para construir sistemas de RA sobre PDA's. El diseño inicial incluye el aumento de imágenes reales con objetos

virtuales, para por ejemplo ilustrar el funcionamiento de algunos electrodomésticos y su interacción con ellos.

2002

Kalkusch et al. [KALKUSCKH02] presenta una aplicación de RA para guiar al usuario en el interior de un edificio hacia su destino. El sistema superpone un modelo de la estructura del edificio según se va avanzando por él, todo ello sobre un HDM. Utiliza marcadores de ARToolKit. Ver Figura 2-11.

Figura 2-11. Sistema de guiado basado en RA de [KALKUSCK 02]

2003

Wagner y Schmalsteig [SCHMALSTIEG03] crean un sistema de RA de guiado en interiores sobre una PDA. La aplicación provee al usuario mediante objetos aumentados de la información para llegar a su destino. Se trata del primer sistema autónomo e independiente. Utiliza Windows Mobile y está implementado con ARToolKit.

2004

Mohring et al. [MOHRING 04] presentan un sistema para el posicionamiento con marcadores 3D en teléfonos móviles.

Rohs y Gfeller [ROHS04] presentan Visual Codes, un sistema de marcadores 2D para teléfonos móviles. Estos marcadores pueden utilizarse sobre objetos físicos para superponer información virtual sobre dicho objeto.

2005

Henrysson [HENRYSSON 05] consigue portar ARToolKit para poder ejecutarlo en el sistema operativo Symbian. Basado en esta tecnología, presenta AR-Tennis, la primera aplicación de RA colaborativa para teléfonos móviles.

2006

Reitmayr et al. [REYTMAR 06] presentan un modelo híbrido de sistema de RA de seguimiento en entornos urbanos. El sistema captura en tiempo real la imagen del entorno con una PDA, utilizando la misma para la visualización de la escena. El sistema combina diferentes sistemas de posicionamiento para aumentar la precisión de la localización.

2008

Wagner et al. [WAGNER 08] presentan el primer sistema de RA 6DOF con tracking de marcadores naturales para dispositivos móviles, consiguiendo tasas de hasta 20 frames por segundo (fps). El sistema modifica los conocidos métodos SIFT y Ferns para mejorar la velocidad, optimizar y reducir la memoria necesitada.

METAIO [METAIO 08] presenta en el ISMAR'08 un sistema comercial de RA para el guiado en un museo utilizando marcadores naturales.

Mobilizy lanza Wikitude [WIKITUDE09], una aplicación que combina el GPS y la brújula digital para mostrar datos de la wikipedia sobre lugares u objetos en sistemas Android.

2009

Kimberly Spreen et al. [ARHRR 10] desarrollan ARhrrr!, el primer videojuego de RA con una calidad gráfica al nivel de los juegos comerciales. Esta aplicación utiliza el kit de desarrollo Tegra de Nvidia, optimizado para las GPU's del momento. Todo el procesamiento se realiza en la GPU, salvo el referido al posicionamiento, haciendo que la aplicación funcione con un alto ratio de frames por segundo.

A día de hoy están al alcance de casi cualquier persona smartphones con las prestaciones necesarias para servir como soporte de estas aplicaciones y sacar el máximo provecho de ellas. La mayor parte de los dispositivos móviles adquiridos por los usuarios en el último año disponen de cámara de vídeo de alta resolución y alta capacidad de procesado. Este hecho, y los recientes estudios de mercado, hacen pensar que la integración de la RA en el creciente mercado de los dispositivos móviles forman un campo de investigación de interés. [CLEMENS 11].

Como se ha visto en el resumen de antecedentes, Wagner y Schalstieg [WAGNER 03] fueron los primeros en identificar el potencial de los dispositivos móviles para RA. En la comparativa que muestra la Tabla 2-1 se observa cómo han mejorado los dispositivos móviles en los aspectos más relevantes para RA: procesadores más rápidos, más memoria, mejores interfaces de entrada, pantallas más grandes y de mayor calidad gráfica, más sensores y mejora de las posibilidades de conexión. La comparativa de la tabla es una actualización de la presentada por Clements & Schmalstieg [CLEMENS 11].

	2003	2012
CPU	400 MHz Intel xScale	Doble núcleo ARM 1.2 GHz
RAM	64 MB	1GB
Soporte Hardware GFX	ninguno	OpenGL ES
Cámara	Cámara color 320x240 frontal y trasera integrada por CP jacket	Cámara frontal y trasera de 2 a 8 Megapixels
Display	3.8”, 16-bit 240x320	4,3”, 800x480
Interfaz	Huella dactilar	Pantalla táctil y teclado
Sensores	Wifi Bluetooth GPRS	GPS, magnetómetro, RFID, acelerómetro, giroscopio de 3 ejes, sensor de proximidad, sensor de luz ambiente
Conexión	900-1200 mAh	Wifi Bluethooth USB 3G, GPRS
Batería	900-1200 mAh	1650 mAh
Sistema Operativo	Windows Pocket PC Windows Mobile	Android iOs
Precio	900€	400-500€

Tabla 2-1. Comparación de hardware de dispositivos móviles para RA, de 2003 a hoy

Con la llegada de los sistemas operativos iPhone OS y Android, el término Smartphone se acuñó para indicar la capacidad de estos dispositivos para la implementación de aplicaciones informáticas complejas. La mejora de interfaces, y en particular la introducción de la pantalla táctil, y el fácil acceso a las aplicaciones,

"Apps", por parte de los usuarios, explica su gran éxito comercial. Este fuerte interés comercial también ha traído consigo la mejora de las herramientas de desarrollo de software dedicado a smartphones.

En definitiva, en conjunto, el nuevo paradigma de dispositivos móviles ofrece todos los ingredientes necesarios para convertir la RA, de un software de uso casi exclusivo a su uso por un público masivo.

Sin embargo, a pesar de todas las mejoras en aspectos logísticos y técnicos, no cabe duda de que todavía existen importantes obstáculos para una implementación a gran escala de las aplicaciones de RA.

2.3. Herramientas para la implementación de aplicaciones de Realidad Aumentada

A continuación se realiza una revisión de las herramientas para desarrollar aplicaciones de RA con dispositivos móviles más conocidas y gratuitas.

ARtoolKitPlus

ARToolKitPlus en una variante de ARToolKit [ART 99], optimizada para el desarrollo de aplicaciones móviles. Fue desarrollada por la Universidad de Graz en 2007 [WAGNER 07]. Inicialmente era de código cerrado por lo que no está muy documentada. La librería implementa módulos para el cálculo de la orientación y posición de la cámara relativa a los marcadores en tiempo real. Debido a su elevada demanda más tarde se liberó su código. Sin embargo su poca documentación la hace poco recomendable para el uso por parte de desarrolladores poco experimentados. Además el proyecto ha sido abandonado y la librería ha sido reemplazada por StudierStube Tracker y Studierstube ES desarrollados por la misma universidad.

Studierstube ES

StudierStube ES [WAGNER 09] es una extensión del framework de desarrollo Studierstube para dispositivos móviles. Se trata de una librería de visión artificial para la detección de la orientación y posición de marcadores 2D con respecto a la cámara. Fue desarrollada por la Universidad de Graz al igual que su predecesora ARToolKitPlus. Como mejoras de esta destaca el hecho de ser multiplataforma (Windows XP, WindowsCE & Windows Mobile, Symbian, Linux, MacOS, Iphone).

Acepta diferentes marcadores, diferentes a los clásicos utilizados por ARToolKit, pero no reconocer marcas naturales. Se trata de código cerrado (no se distribuye).

Este framework se caracteriza por su gran rendimiento en dispositivos con bajas prestaciones y buen aprovechamiento de la memoria. Permite el uso de hasta 4096 marcadores.

El framework ofrece soporte para la calibración de la cámara mediante el toolbox de Matlab e incluye un algoritmo de umbralización adaptativo para los casos de iluminación variable.

Layar

Layar es un navegador de RA, desarrollado para Android e iOS. Tiene licencia privativa por lo que no dispone de acceso al código fuente.

El funcionamiento del software se basa en el geoposicionamiento y no en el reconocimiento de marcas.

Está basado en un sistema de capas que funcionan sobre el navegador de realidad y que pueden ser mostradas o no a elección del usuario. El desarrollador implementa estas capas, en 2D o 3D, para añadir información aumentada a la imagen real (Figura 2-12). El sistema se compone de la aplicación cliente que se ejecuta en el dispositivo, un servidor central que provee los datos y un servidor privado para que el desarrollador gestione esos datos y los envíe al servidor central para finalmente visualizarlos en la aplicación. Las capas definidas por el usuario pueden ser puestas a disposición de la comunidad de manera centralizada. La etapa de renderizado de objetos 3D está optimizado para su uso en dispositivos móviles.

Figura 2-12. Ejemplo de aplicación realizada con Layar

Este framework está enfocado especialmente al desarrollo de aplicaciones de turismo y entretenimiento. Utilizando los sensores inerciales integrados en los dispositivos móviles analiza la posición del usuario y le ofrece información de los puntos de interés cercanos a él (museos, monumentos, restaurantes, etc).

Mixare

[MIXARE] (mix Augmented Reality Engine) es un framework de código abierto para RA, publicada bajo la licencia GPLv3 3. Mixare está disponible para sistemas Android y para iPhone.

Este framework permite construir aplicaciones completas y proporciona funciones para asociar coordenadas espaciales y texto. Es decir, su funcionalidad se resume a permitir asociar texto a localizaciones mediante posicionamiento GPS y acceso a datos por conexión de red. Las visualizaciones de Mixare están limitadas a cajas de texto e imágenes 2D.

AndAR

AndAR es un SDK de código abierto para el desarrollo de aplicaciones de RA para Android basadas en el reconocimiento de marcadores [ANDAR 12]. Utiliza marcadores del tipo ARToolKit. Permite la carga de objetos 3D con formato .obj. Esta librería se presenta con más detalle en el apartado 3.2 y posteriores.

NyARToolkit

NyARToolkit es un SDK de código abierto para el desarrollo de aplicaciones de RA basadas en el reconocimiento de marcadores [NYATK 12] . Se trata de un framework multiplataforma disponible para Android,Java, C#, AS3, C++ y Processing. Utiliza marcadores del tipo ARToolKit, y dispone de soporte para diferentes formatos 3D (.mqo, .md2, .obj) mediante el uso de la librería min3D. Esta librería se presenta con más detalle en el apartado 3.3 y posteriores.

Vuforia

[VUFORIA] es una plataforma de desarrollo de aplicaciones de RA para Android e iOS desarrollada por el departamento de I+D de la empresa Qualcomm en Austria. Esta

plataforma fue publicada en 2010 y por ejemplo en el último año se ha desarrollado más de 1.000 aplicaciones con ella. Grandes marcas comerciales han utilizado esta plataforma para las campañas publicitarias de sus productos. Una de las principales ventajas de esta plataforma es que se basa en el reconocimiento de marcas naturales, incluyendo objetos 3D, y que existe una extensión para Unity 3D que permite crear escena virtuales con animaciones y muy completas.

La plataforma se presenta en código abierto aunque su uso con Unity 3D requiere de la adquisición de la licencia de esta. Este SDK es presentado con más detalle en el apartado 3.4 y posteriores.

Metaio

[METAIO] Mobile SDK es una plataforma de desarrollo de aplicaciones de RA para dispositivos Android e iOS creada por la empresa Metaio en Alemania. LA empresa dispone de más de 10 años de experiencia en el desarrollo de esta tecnología y podee otras plataformas de desarrollo para PC y Web. Las aplicaciones se basan en el reconocimiento de marcas naturales, e integra la gravedad en los módulos de reconocimiento para añadir precisión.

El código del SDK para móviles ha sido liberado recientemente. Este incluye un motor de renderizado que soporta distintos formatos 3D (.md2 animado y .obj estático). También puede utilizarse con Unity 3D aunque requiere adquirir su licencia. Este SDK es presentado con más detalle en el apartado 3.5 y posteriores.

Capítulo 3

HERRAMIENTAS ESTUDIADAS

3.1. Introducción

Tras un estudio inicial de las plataformas de desarrollo disponibles para el desarrollo de aplicaciones de RA nos decidimos por centrarnos en aquellas, de código abierto, especialmente preparadas para su uso con Android y basadas en visión artificial para el posicionamiento de los objetos. Concretamente se han escogido dos librerías que utilizan marcadores de tipo ARToolKit y otras dos que reconocen marcadores naturales.

Como ya se ha comentado, en este trabajo se analizan cuatro librerías de RA:

- Vuforia, librería publicada por Qualcom a finales de 2010 y escrita en Java y C++.
- Metaio Mobile SDK, librería escrita en Java, implementada por Metaio y liberada a finales de 2011.
- NyARToolkit: Librería e clases derivada de ARtoolKit publicada por primera vez en 2008 y escrita íntegramente en Java.
- AndAR: proyecto que trata de adaptar ARToolKit a dispositivos Android mediante una API escrita en Java y publicada en 2010.

El dispositivo de referencia que se ha utilizado para las pruebas ha sido el Samsung Galaxy SII GT-I9100. Este dispositivo cumple las características mostradas en la Tabla 2-1.

La versión de Android utilizada es la que lleva el dispositivo por defecto, Android 2.3.4.

Una aplicación Android está compuesta por un conjunto de ficheros empaquetados en formato .apk y guardada en el directorio /data/app del sistema operativo Android. Este directorio necesita permisos de superusuario por razones de seguridad. Es necesario por tanto configurar el dispositivo previamente para poder realizar pruebas con él. Como mínimo hay que activar la opción “Depuración de USB” en el menú de definición de opciones para el desarrollo de aplicaciones.

Para el desarrollo de los ejemplos ha sido necesaria la utilización del siguiente software:

- Eclipse: entorno de desarrollo integrado de código abierto multiplataforma.

- JDK6: Kit de desarrollo Java.

- SDK Android: Kit de desarrollo para aplicaciones Android.

- Blender 2.48

Android

Android es un sistema operativo móvil basado en Linux desarollado por Google. Existe una amplia comunidad de desarrolladores que escriben aplicaciones con el fin de extender la funcionalidad de los dispositivos móviles. A día de hoy se han sobrepasado las 600.000 aplicaciones, de las cuales, dos tercios son gratuitas [LIN 12].

La estructura de Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientas a objetos sobre el núcleo de las librerías Java en una máquina virtual Dalvik con compilación en tiempo de ejecución.

Todas las aplicaciones en Android pueden descomponerse en cuatro tipos de componentes principales. Cada aplicación debe estar formada por uno o más de estos componentes, que son declarados de forma explícita en un archivo XML denominado *AndroidManifest.xml.*Este archivo también puede incluir valores globales, clases que implementa, datos que puede manejar, permisos, etc. Este archivo es por tanto básico en cualquier aplicación de Android.

Respecto a los cuatro tipos componentes en los que se divide la aplicación, éstos son: Activity, Service, content provider y broadcast provider.

Las clases de tipo Activity son el componente más habitual. Reflejan una determinada actividad llevada a cabo por una aplicación y va asociada a una ventana y su respectiva interfaz de usuario. En nuestras aplicaciones habrá al menos una de estas actividades definida (sólo puede haber una actividad activa en cada momento), en la que se invocará a los principales procesos de RA:

- Captación de la imagen a través de la cámara
- Reconocimiento y tracking del marcador
- Renderizado de objetos
- Visualización

Android SDK

El Software Development Kit (Kit de desarrollo de software) de Android, incluye un conjunto de herramientas de desarrollo como depurador de código, librerías, simulador de teléfonos basado en QEMU, documentación, ejemplos y tutoriales. La IDE oficialmente soportada por este SDK es Eclipse, junto con el complemento ADT.

En nuestro caso no se ha utilizado el emulador de teléfonos, pues el SDK permite controlar dispositivos Android conectados mediante USB. Este SDK también nos ha permitido realizar pruebas de compatibilidad con versiones anteriores de Android, descargando componentes de herramientas anteriores.

Native Development Kit.

Las librerías utilizadas incluyen librerías escritas en C además del código base en Java, por lo que es necesaria la instalación de este NDK. Así, los programas Java pueden llamar a clases nativas por medio de la función System.loadLibrary, que está dentro de las clases estándar Java en Android.

Se pueden compilar e instalar aplicaciones completas utilizando las herramientas de desarrollo habituales. El depurador ADB proporciona un shell root en el Simulador de Android que permite cargar y ejecutar código nativo ARM o x86. Este código puede compilarse con GCC en un ordenador normal. La ejecución de código nativo es difícil porque Android utiliza una biblioteca de C propia (libc, llamada Bionic). Se accede al dispositivo gráfico como un framebuffer disponible en /dev/graphics/fb0. La biblioteca gráfica que utiliza Android para controlar el acceso a este dispositivo se llama Skia Graphics Library (SGL), disponible con licencia de código abierto. Skia tiene implementaciones en win32 y Unix, permitiendo el desarrollo cruzado de aplicaciones, y es el motor de gráficos que soporta al navegador web Google Chrome.

Android Open Accessory Development Kit

La plataforma de Android 3.1 (portado también a Android 2.3.4) introduce soporte para Android Open Accesory, que permite interactuar a dipositivos USB externos (accesorios USB Android) interactuar con el dispositivo en un modo especial llamado "accessory". Cuando un dispositivo Android está en modo "accessory" el dispositivo externo actúa como hub usb (proporciona alimentación y ennumera los dispositivos) y el dispositivo Android actúa como dispositivo USB. Los accesorios Android USB están diseñados específicamente para conectarse a dispositivos Android y utilizan un

protocolo simple (Android accessory protocol) que les permite detectar dispositivos Android que soportan modo "accessory".

Blender

Blender es un software CAD multiplataforma dedicado al modelado, animación y creación de gráficos tridimensionales.

Se puede utilizar para modelar, aplicar texturas, animación, renderizado, partículas y otras simulaciones, edición no lineal, composición, y creación de aplicaciones 3D interactivas, incluyendo juegos.

Desde el inicio este software ha sido distribuido de forma gratuita pero sin el código fuente, con un manual disponible para la venta, aunque posteriormente pasó a ser software libre. Actualmente es compatible con todas las versiones de Windows, Mac OS X, Linux, Solaris, FreeBSD e IRIX.

Las características más destacadas de este software frente al resto son las siguientes:

- Multiplataforma, libre, gratuito. El espacio en disco que ocupa este software es realmente pequeño comparado con otros paquetes de 3D, dependiendo del sistema operativo en el que se ejecuta.
- Capacidad para una gran variedad de primitivas geométricas, incluyendo curvas, mallas poligonales, vacíos, NURBS, curvas Bezier, etc.
- Junto a las herramientas de animación se incluyen cinemática inversa, deformaciones por armadura o cuadrícula, vértices de carga y partículas estáticas y dinámicas.
- Edición de audio y sincronización de vídeo.
- Características interactivas para juegos como detección de colisiones, recreaciones dinámicas y lógica.
- Posibilidades de renderizado interno versátil e integración externa con potentes trazadores de rayos libres como kerkythea, YafRay o Yafrid.
- Lenguaje Python para automatizar o controlar varias tareas. Es posible la creación de scritps para tratar los objetos poligonales que hay en la escena.
- Acepta formatos gráficos como TGA, JPG, Iris, SGI, o TIFF. También puede leer ficheros Inventor.

- Motor de juegos 3D integrado, con un sistema de bloques lógicos Para más control se usa programación en lenguaje Python.
- Simulaciones dinámicas para partículas y fluidos.
- Modificadores apilables, para la aplicación de transformación no destructiva sobre mallas.
- Sistema de partículas estáticas para simular cabellos y pelajes, al que se han agregado nuevas propiedades entre las opciones de *shaders* para lograr texturas realistas.

Para nuestro proyecto, se ha utilizado Blender para la creación de los objetos 3D de los ejemplos. Como se explicará más adelante, cada librería soporta unos tipos de formato de objetos 3D diferentes. Los scripts en python que incluye la versión 2.48 de Blender, permiten la importación y exportación de formatos prácticamente en desuso como metasequoia (mqo). Aunque existen algunas dificultades para la importación y exportación correcta de los objetos, especialmente con las texturas, esta herramienta ha facilitado bastante las tareas de creación de objetos.

OpenGL ES

OpenGL (Open Graphics Library) es una aplicación estándar que define una API multilenguaje y multiplataforma para aplicaciones con gráficos 2D y 3D.

OpenGL ES fue diseñado por el grupo Khronos y es una API ligera para gráficos 3D utilizada en dispositivos integrados como teléfonos móviles, PDAs y consolas de videojuegos. Esta API ha sido seleccionada como la API para gráficos oficial en la plataforma Android. Se trata de una potente API de bajo nivel que es muy similar a la de OpenGL y que proporciona la mayor parte de la funcionalidad de OpenGL.

Una de las reducciones de esta API respecto a OpenGL es que no hay compatibilidad con primitivas que no sean puntos, líneas o triángulos, por lo que no es compatible con quads o polígonos de n lados. Éste es un aspecto muy importante a la hora de diseñar nuestros objetos virtuales.

Sin embargo, estas reducciones en la API de OpenGL producen una API mucho más compacta y manejable para dispositivos móviles, a ocupar unos poco cientos de KB.

Partiendo de unos conocimientos previos básicos de OpenGL, para el desarrollo de nuestras aplicaciones de RA con OpenGL ES es importante conocer a priori las clases más importantes:

GLSurfaceView:

Es la clase que facilita el enlace entre OpenGL y las Views y Activity de Android. Además es la encargada de la inicialización de la parte gráfica de nuestra aplicación.

En Android, todo el renderizado debe realizarlo la interfaz *GLSurface.Renderer* Esta interfaz define los tres métodos esenciales para la creación de nuestras escenas:

- *onSurfaceCReated()*: Inicialización de la superficie dónde se dibujará. Hay que señalar que OpenGL ES no dispone de las funciones glVertexf() que estábamos acostumbrados a utilizar con OpenGL. En su lugar, las coordenadas de cada punto se pasan en matrices de floats y un ushort que especifica el orden en que son dibujados. Esta técnica se conoce como Vertex Buffer Object.
- *onDrawFrame()*: dibujado de la escena.
- *onSurfaceChanged():* se invoca cuando el dispositivo cambia de posición, para ajustar la escena a este cambio.

GL10:

Es el interfaz que sirve de capa de abstracción entre Java y OpenGL. Para utilizar funciones OpenGL se utilizan sus métodos estáticos. Las constantes que utilizan nuestros métodos son definidas en G10.

Se debe pasar una instancia de este objeto a todos los métodos de GLSurfaceView antes mencionados.

3.2. AndAR

AndAR es una librería creada en 2010, por Tobias Domhan [DOMHAN 10], especialmente para dispositivos Android mediante una API escrita en Java. Está basada en el proyecto ARtoolKit [KATO 99]. Hereda la licencia dual de ARToolKit, por lo que, aunque está licenciado bajo GNU GPL v3, existe la posibilidad de utilizarla en aplicaciones de código cerrado mediante pago de una licencia a ARToolworks.

Esta librería funciona con marcadores básicos. La librería maneja una plantilla de texto para gestionar los marcadores. Estas plantillas pueden generarse mediante el software mk_patt de ARToolKit, que sirve para convertir una imagen patrón en una plantilla que pueda ser reconocida por la librería. En la documentación de ARToolKit se puede encontrar la plantilla para crear nuestro propio marcador, puesto que éste debe tener unas proporciones exactas para poder ser reconocido por la librería Figura 3-1.

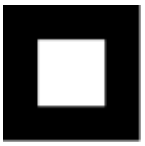

Figura 3-1. Plantilla marker Artoolkit

La Figura 3-2 muestra el diagrama de clases de una aplicación simple que utilice AndAR. A continuación se explica brevemente esta estructura.

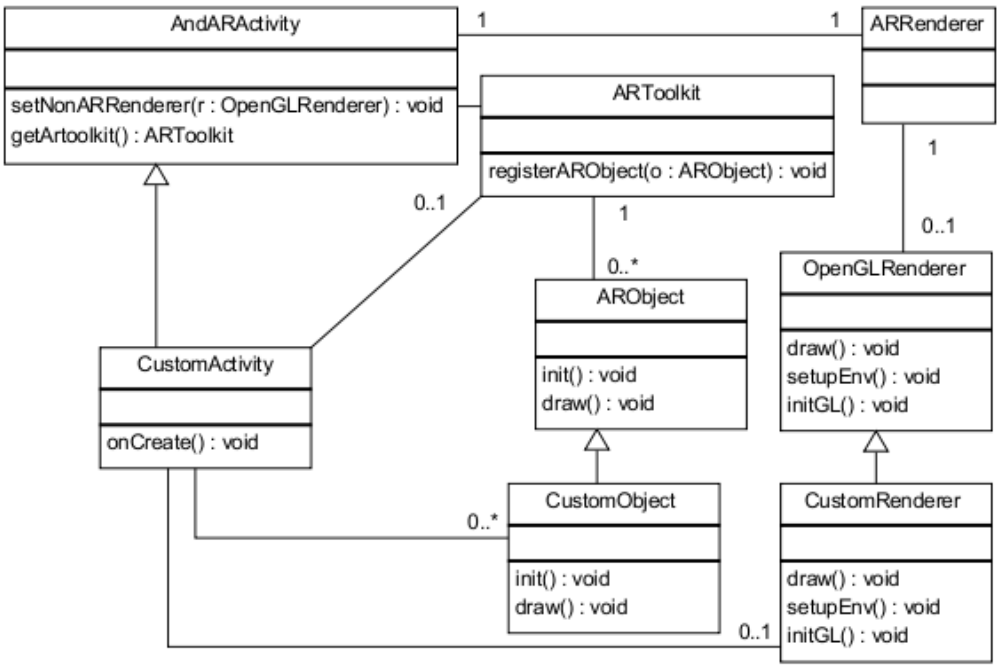

Figura 3-2: Diagrama de Entidad-Relación de AndAR

En primer lugar vemos la “actividad prinicipal” AndARActivity. Siempre se creará al menos una actividad extra CustomActivity, que hereda de esta clase abstracta. Esta clase implementa todas las tareas relativas al manejo de la cámara, detección de marcadores y visualización de vídeo. En nuestra clase podremos gestionar las tareas relativas a inserción de objetos, desde onCreate.

Por otro lado, la clase ARRenderer es la responsable de todo lo relacionado con OpenGL. Esta clase se utiliza para implementar el renderizado, y es la que permite mezclar la realidad con los objetos aumentados. Desde esta clase podemos gestionar parámetros de iluminación, posición y cualquier tarea relacionada con la inicialización de OpenGL. En nuestro caso, como se explicará más adelante, se ha extendido la clase ARObjects creando una clase capaz de mostrar modelos en formato .obj.

ARObject es la clase responsable de proporcionar constantemente la información sobre el marcador. Como ya hemos comentado que esta librería está basada en ARToolKit, utiliza los mismos patrones de datos generados para ARToolKit básico, y que pueden crearse mediante la herramienta mk_patt. Ésta es la primera limitación identificada. Esta limitación se basa en que el patrón debe ser almacenado en un archivo dentro de la carpeta assets del proyecto. Esto es imprescindible para poder encontrarlo cuando se crea el objeto.

3.3. NyARToolkit

NyARToolkit es una librería también basada en ARToolKit creada por Ryo Iizuka en 2008 [NYATK 08]. En este caso no fue desarrollada específicamente para desarrollo aplicaciones Android, sino que el proyecto NyARToolkit es compatible con plataformas Java, C#, ActionScript 3, Silverlight 4, C++, Processing y obviamente Android, aunque no todos al mismo nivel en términos de estabilidad. De esta misma librería surgió otra, FLARToolKit, aún no disponible para Android.

NyARToolkit para Android ha sido desarrollada por un grupo de usuarios de Android en Japón y no existe apenas documentación online, y la que hay está escrita en japonés. Por eso, para entender esta librería hay que sumergirse directamente en el código fuente de Java para Android. Incluso la documentación del código está hecha en japonés, por lo que, y para que sirva de aviso al lector, a la hora de configurar el entorno de trabajo con esta librería es necesario editar las preferencias de codificación de todos los archivos a UTF-8, o no desaparecerán los errores.

Esta librería se encuentra aún en fase de desarrollo, pero no tiene una comunidad grande de desarrolladores detrás, por lo que los avances son muy lentos.

Hasta hace poco, NyarToolkit para Android dependía del formato MQO (metasequoia) para modelos 3D. Se trata de una aplicación gratuita (de código cerrado) y shareware para modelado y texturas 3D muy popular en Japón, pero en ningún sitio más. Al menos existe un exportador para Blender a formato MQO. Sin embargo, la última versión de NyARToolkit puede importar formatos MD2 y OBJ, ambos también compatibles con Blender.

La estructura de una aplicación básica es igual a la de AndAR, aunque este SDK utiliza un motor de renderizado externo [MIN3D], en lugar de cargar los objetos directamente. Min3D es una librería ligera 3D para Android que utiliza Java con OpenGL ES. Ofrece compatibilidad con la versión 1.5 de Android y OpenGL ES 1.0 y posteriores. Su documentación está integrada en bloques dentro del propio código.

3.4. Vuforia

Vuforia es un SDK desarrollado por Qualcomm, una empresa productora de chipsets para tecnología móvil. En 2010 la empresa lanzó algunas aplicaciones propias que hacían uso de tecnologías de RA, y finalmente ese mismo año anunció que ponía a disposición de los desarrolladores sus frameworks de desarrollo al que denominaron Vuforia. Está disponible para Android e iOS y se basa en el reconocimiento de imágenes basado en características especiales, por lo que también soporta marcadores naturales (targets) o RA sin marcadores. Además dispone de un plugin para interactuar con Unity3D y ofrece la posibilidad de crear botones virtuales para ampliar las vías de interacción con el usuario.

Una aplicación de RA basada en Vuforia estará integrada por los siguientes componentes fundamentales (*Figura.* 3-3):

Camera:

Este módulo se asegura de que cada frame capturado pase al tracker. En este módulo se debe indicar cuándo la aplicación inicia la captura y cuando termina. El tamaño y formato de cada frame dependerá del dispositivo móvil utilizado.

Image Converter:

Este módulo convierte el formato de la cámara a un formato interoperable con OpenGL y para el tracking de los marcadores. Esta conversión incluye reducción de la tasa de muestreo con el fin de disponer de la imagen de la cámara en diferentes resoluciones.

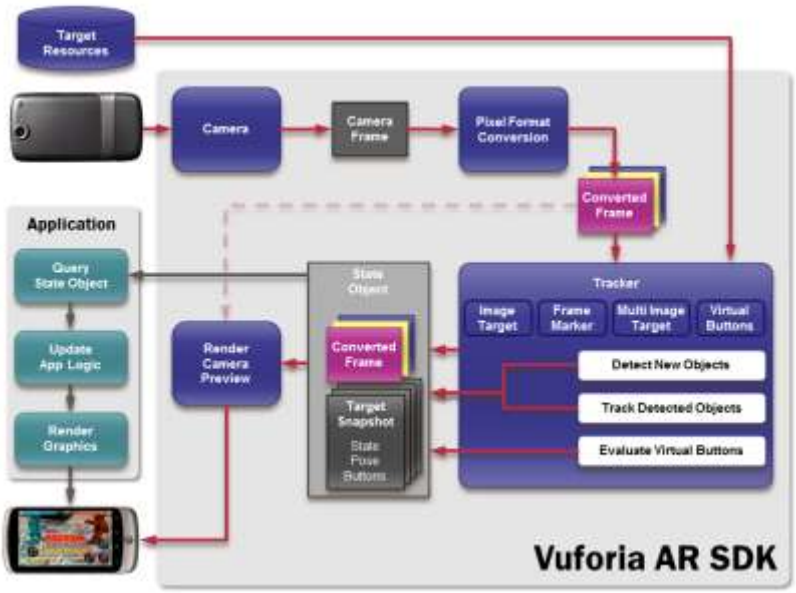

Figura. 3-3. Diagrama de flujo del SDK de Vuforia

Tracker:

Este módulo contiene los algoritmos de visión artificial que se encargan de la detección y rastreo de los objetos de cada frame. Diferentes algoritmos se encargan de la detección de nuevos "targets" o "markers" y de evaluar los botones virtuales. Los resultados son almacenados en un objeto de estado. Este módulo puede cargar múltiples conjuntos de objetos, pero nunca puede haber más de uno activo al mismo tiempo.

Video Background Renderer:

Este módulo procesa la imagen almacenada en el objeto de estado. El rendimiento de la representación de vídeo de fondo está optimizado para dispositivos específicos.

Todos estos componentes deben ser inicializados en nuestra aplicación. En cada frame se actualiza el objeto de estado y se llama a las funciones de renderizado. Así pues, nuestra aplicación siempre debe realizar estos tres paso:

1. Consultar el objeto de estado para comprobar nuevos targets o markers detectados.
2. Actualizar la lógica de la aplicación con los nuevos datos de entrada

3. Renderizar los elementos virtuales.

Los targets o marcadores son creados mediante un sistema online (Target Management System). Una vez creada la imagen que servirá como target o marcador, se accede a este sistema. Se crea un nuevo proyecto, y se sube la imagen. El sistema analiza la imagen y le asigna una calificación que indica la efectividad del marcador en función del número de características especiales detectadas por el sistema. El siguiente paso es convertir la imagen a formatos entendidos por la librería. El sistema nos devuelve dos archivos: un .xml con la configuración del target o marcador y un archivo binario que contiene los datos rastreables.

3.5. Metaio Mobile SDK

Metaio Mobile SDK es un SDK desarrollado por la empresa alemana [METAIO]. Hasta este año todas sus herramientas de desarrollo eran de pago, pero han liberado su SDK para móviles debido al éxito que estaba teniendo. La versión completa, de pago, incluye reconocimiento de caras y reconocedor de QR. Aunque la versión gratuita no incluye estas funcionalidades y que las aplicaciones desarrolladas deben incluir una marca de agua de la empresa, este SDK incluye un potente reconocedor de marcadores naturales, es decir, se pueden desarrollar aplicaciones “markerless”.

Esta empresa ganó la “tracking competition” del ISMAR 2011.

Otro aspecto muy llamativo a señalar de este framework, aunque no forma parte del estudio comparativo, es que incluye el concepto de gravedad en las tareas de reconocimiento así como de renderizado [KURZ 11]. Esto resulta un gran avance, pues facilita el reconocimiento en situaciones particulares como puede ser el reconocimiento de una ventana en concreto de un edificio de 100 plantas (Figura. 3-4).

Figura. 3-4. Ejemplo de aplicación de la gravedad en el reconocimiento. [KURZ 11]

Por otro lado el hecho de añadir gravedad a los objetos aumentados mejora enormemente la experiencia del usuario. Imaginemos una aplicación para que el usuario se pruebe pendientes, el hecho de que éstos se muevan como lo haría un

objeto real con gravedad, aporta mucho realismo a la escena. Esto es sólo un simple ejemplo de lo que supone esta mejora. En la Figura. 3-5 se muestra otro ejemplo. Póngase atención a la orientación de la llama en función de la orientación del marcador:

Figura. 3-5. Ejemplo de la aplicación de la gravedad en los objetos virtuales. [KURZ 11]

Además de éste, la empresa dispone de otros dos SDK, de pago. El Metaio Web SDK, pensado para aplicaciones web, y enfocado principalmente al comercio electrónico. Y el Metaio PC SDK, pensado para aplicaciones de RA más robustas. Este SDK incluye el reconocimiento facial.

Existe una última aplicación desarrollada por esta empresa, que también ha sido utilizada en este trabajo. Es el Metaio creator, una herramienta pensada para generar espacios de RA rápidamente y sin necesidad de tener conocimientos de programación, todo funciona con una interfaz gráfica en la que se añaden marcadores y objetos 3D que la herramienta se encarga de transformar en una aplicación de RA. Este software es de pago, pero está disponible una versión Demo, de la que se hablará más delante.

El metaio Mobile SDK está implementado de forma modular, al igual que los anteriores, de forma que divide la actividad de RA en tres componentes: Tracking, Captura y Renderizado (Figura. 3-6)

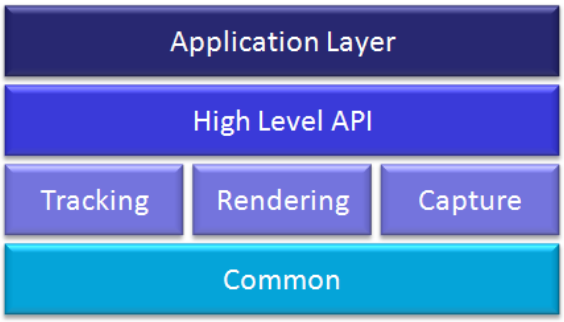

Figura. 3-6. Estructura del Metaio Mobile SDK

En la versión gratuita la etapa de tracking está oculta bajo una aplicación estándar configurable mediante un archivo de configuración .XML. Este archivo, que puede

obtenerse a partir de nuestro target con la versión Demo de Metaio Creator de una manera parecida a lo que ocurría con Vuforia, contiene los parámetros de configuración del target, tamaño y nombre de la imagen .png utilizada entre otras.

Una vez presentadas todas las SDK estudiadas en el trabajo, en la Tabla 3-1, se ofrece un resumen de las características mencionadas previo al estudio de las características funcionales.

	ANDAR	NYARTOOLKIT-ANDROID	VUFORIA	METAIO SDK MOBILE
Licencia	Libre	Libre	Libre	Libre con restricciones
Marcadores	✓	✓	✓	✓
Marcas naturales	X	X	✓	✓
Tracking 3D	X	X	✓	Pendiente de incluir
Formatos 3D	OpenGL, .obj	.mqo, .md2(animación), .obj	OpenGL	OpenGL, .md2(animación), .obj
Multiplataforma	No (solo Android)	SI: Android, Java, C#, C++, AS3, Processing	Android / iOS	Android / iOS
Documentación	Limitado	Sólo en Japonés	Completa	Completa
Soporte a desarrolladores	X	X	Sí mediante API	Sí mediante API
En desarrollo	✓	✓	✓	✓
Comunidad de desarrolladores	✓	✓	✓	✓

Tabla 3-1. Resumen de características de los SDK analizados.

Capítulo 4

ESTUDIO DE LAS HERRAMIENTAS

4.1. Introducción

Antes de comenzar a probar las SDK ha sido necesario sacar unas conclusiones previas y analizar minuciosamente los aspectos diferenciadores de las SDK con el fin de hacer un estudio efectivo y representativo.

Como se ha comentado en apartados anteriores cualquier aplicación de RA se puede estructurar en cuatro etapas:

1. Captura del entorno real.
2. Reconocimiento y tracking del marcador o target.
3. Renderizado de los objetos virtuales
4. Visualización o superposición del mundo real y virtual

Pues bien, las etapas 1 y 4, más que del SDK utilizado, dependerán en su mayoría de las características (especialmente hardware) del dispositivo utilizado. Por esto, son consideradas de interés para nuestro estudio únicamente las etapas 2 y 3, que darán lugar a las características más diferenciadoras de los SDK objeto de estudio. En la Sección 4 se explicarán con más detalle los parámetros escogidos para comparar los SDK y el por qué.

Como ya se ha comentado anteriormente, no existe hasta el momento ninguna publicación dedicada a la comparación completa, en todas sus facetas, de diferentes SDKs de RA a nivel funcional. Existen algunas publicaciones en las que se compara alguna SDK con desarrollos propios de los autores pero sólo de aspectos concretos, como puede ser el reconocimiento de marcadores [ZHANG 02], sin embargo ninguno específico para aplicaciones de RA para dispositivos móviles.

Es por ello que se ha tenido que realizar un estudio previo para tener una visión global de los aspectos comunes y diferenciadores de los SDKs y realizar pruebas de funcionamiento hasta identificar los parámetros a estudiar. Tras este estudio previo se ha definido una dinámica de trabajo con el fin de obtener conclusiones de las más específicas a las más globales. Cada SDK tiene sus particularidades por lo que resulta esencial partir de unas bases comunes para poder llegar a estas conclusiones. Con

estas bases comunes nos referimos a las condiciones de las pruebas a realizar. Por ejemplo, si queremos comparar la eficiencia del algoritmo de reconocimiento, no podemos utilizar diferentes marcadores para cada librería. Éstas suelen venir con unos marcadores definidos por defecto, y puede dar lugar a pensar que su eficiencia está condicionada a ese marcador.

Por esto la dinámica de trabajo definida parte de elaborar un marco de pruebas común, que es el siguiente:

- Definir un marcador común
- Definir objetos virtuales comunes

4.2. Batería de pruebas utilizada

En este apartado se define con más detalle el conjunto de test elaborado para el posterior estudio. Como ya se ha adelantado anteriormente, el estudio comparativo de las librerías se hace en base a dos factores globales principalmente:

1. El reconocimiento de marker o target en cada caso
2. El renderizado de los objetos virtuales

1.- Reconocimiento de los marcadores

Respecto al tipo de marcador, en primer lugar hay que decir que no todos los SDK ofrecen las mismas características, por lo que una comparación de igual a igual no sería justa. La diferencia más llamativa de los SDK y que hace que se haya dividido el estudio en dos grupos, es el hecho de que unas contemplen el reconocimiento de marcadores naturales y otras no. Mientras que AndAR y NyARToolkit no incluyen esta opción, las otras dos, Vuforia y Metaio sí las incluyen. La dificultad y las técnicas utilizadas para reconocer un marker y un target no son las mismas, por lo que no sería justo tratarlos por igual a la hora de estudiar las prestaciones funcionales de cada una.

Así pues, se han utilizado dos tipos de marcadores. O como se vienen llamando a lo largo del documento, marker y target.

1.1) AndAR y NyARToolkit utilizan el mismo tipo de marcador. Esto es debido a que ambas heredan esta característica de su mentora ARToolKit. Los marcadores ATK, que es como se denominan a los marcadores utilizados en ARToolKit, son los más utilizados en las aplicaciones de RA con marcadores, debido a la expansión de esta librería y a su sencillez. Probablemente una de las razones por las que son tan utilizados es porque el hecho de su forma cuadrada

proporciona al menos 4 puntos co-planares que facilitan enormemente la calibración de la cámara.

Para la elaboración del marcador se han seguido las especificaciones de dimensionado de la documentación de ARToolKit. Una vez configurada la imagen se ha hecho uso del software "mk_patt" de ARToolKit. Este software, haciendo uso de la captura de la imagen impresa con una webcam, proporciona el archivo de texto reconocido por las librerías con la configuración del patrón creado. Los marcadores utilizados para las pruebas realizadas con AndAR y NyARToolkit se muestran en la Figura. 4-1.

Figura. 4-1. Markers utilizado con AndAR y NyARToolkit: "music.png", "hiro.png", "android.png"

Señalar que se ha intentado utilizar otro tipo de marcadores, como en [ZHANG 02], que no cumplen con la plantilla de ARToolKit y se obtiene un rendimiento mucho más bajo. Los marcadores de tipo HOM creados por Siemens, directamente no pueden utilizarse con ARToolKit, y los implementados por IGD y SCR obtienen muy bajo rendimiento con esta librería.

Figura. 4-2. Ejemplos de marcadores HOM, IGD y SCR

1.2) En el caso de Vuforia y Metaio, como ya se adelantó en apartados anteriores, nos centramos en el uso de targets, o marcadores naturales.

Ambos ofrecen un servicio similar a la hora de elaborar nuestros propios targets.

Por un lado Vuforia, dispone de un servicio web, en el que, previo registro, podemos crear nuestros proyectos para crear nuevos targets. El procedimiento es el siguiente:

- Acceder a la pestaña “My trackables” de https://ar.qualcomm.at/projects
- Registro o Login.
- Crear nuevo proyecto o continuar con uno ya creado.
- Entrar en “Create Trackable” y subir el fichero que utilizaremos como target.
- La herramienta evalúa la calidad de nuestra imagen, lo que nos indica el grado de efectividad a la hora de ser reconocido en base al número de características especiales extraídas. Se puede mejorar la calificación del target utilizando máscaras de desenfoque con alguna herramienta de edición de imágenes tipo Gimp, que aumentan el contraste de la imagen (Figura 4-3).

Figura 4-3. Imagen original y después de aplicar filtro para optimizar el rendimiento del target (“cabañal.png”)

Para nuestro trabajo se ha utilizado un target con la máxima puntuación (*Figura 4-4*).

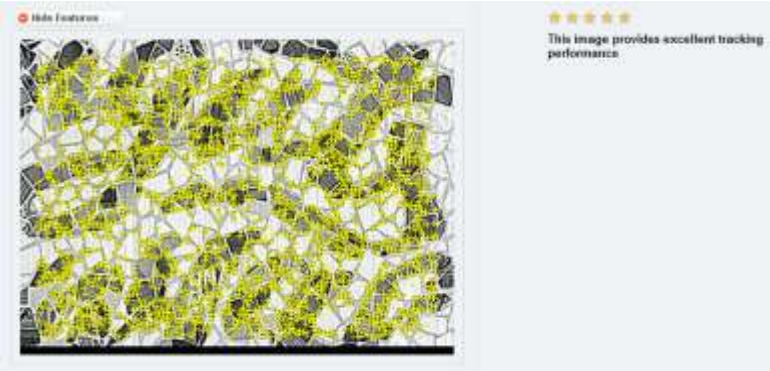

Figura 4-4. Ejemplo de extracción de características y evaluación del target con Vuforia

- Una vez obtenida la imagen con una evaluación conforme a nuestras necesidades, volvemos al menú principal y descargamos los ficheros que utilizará la librería en la etapa de reconocimiento y tracking con “Download Trackable Data”. La herramienta genera un fichero .xml y un fichero .dat que reflejan las características especiales de la imagen y el mapa 3D que deberemos incluir en nuestro proyecto.

En el caso de Metaio el proceso es similar, aunque en lugar de ser un servicio on-line se puede descargar la versión demo de la herramienta "Metaio Creator" como ya se ha adelantado en apartados anteriores.

El proceso es el siguiente:

- Crear nuevo proyecto.
- Arrastrar nuestra imagen hasta la pestaña "References". La herramienta se tomará un tiempo para evaluarla.
- Una vez hecho esto en el menú "Extras" seleccionamos la opción "Export Tracking Configuration File".
  Así la herramienta genera el fichero .xml que deberemos incluir en nuestro proyecto de RA.

Realmente, para utilizar un target u otro bastaría con editar el .xml de los ejemplos de la SDK, para acoplarlo a nuestra aplicación. Basta con ajustar el tamaño (recomendado de 100x100mm) de la imagen y el nombre del fichero .png utilizado. El interés de utilizar esta herramienta es obtener la evaluación de la imagen propuesta, que influirá en el rendimiento de nuestra aplicación. Además cuando se trata de una imagen no cuadrada la herramienta recorta la imagen, pero no arbitrariamente, sino seleccionando la región con mayor número de características especiales. Ello ha permitido que se pueda utilizar el mismo target en las pruebas realizadas con Vuforia (target rectangular) y con Metaio.

2. Objetos virtuales a renderizar

Esta es una característica que sí puede compararse por igual para todas las librerías. Una vez que cada aplicación haya reconocido su marker o target, pasa a la etapa de renderizado o aumento del objeto virtual. Por ejemplo, como se explica más adelante, una de las características a estudiar es la tasa de frames por segundo que ofrece cada librería frente a diversas situaciones. Esta comparación sí será hecha para todas por igual.

No obstante como se mostraba en la Tabla 3-1, cada librería acepta unos formatos de modelos 3D. En ella podemos ver que no existe ningún formato aceptado por todas al mismo tiempo, por lo que esto podría suponer un problema a la hora de compararlas.

El formato más común entre ellas es el formato wavefront obj, soportado por todas las librerías menos por Vuforia.

Pues bien, para poder ser utilizado por Vuforia se ha empleado un script escrito en perl que convierto archivos .obj al formato .h que maneja OpenGL ES. También podría haberse utilizado la librería GLUT para la carga de este tipo de objetos, pero dado que el fin de nuestro estudio no es incluir objetos complejos en cuanto a texturas y otras características, se ha optado por la primera solución.

Un modelo 3D en formato wavefront obj está compuesto por dos archivos ASCII, uno con extensión .obj en el que se define la geometría del objeto (coordenadas de los vértices, normales y texturas) y un archivo con extensión .mtl en el que se definen características de texturas, materiales e iluminación.

Los modelos definidos directamente para OpenGL ES consisten también en un archivo ASCII donde se especifica la geometría del objeto. Por ello es posible la conversión de un formato a otro. Aunque es muy importante tener en cuenta que en OpenGL ES las caras de los modelos se definen mediante triángulos y nunca como objetos, por lo que si nuestro objeto está definido a base de “quads” es importarte convertir todas las caras previamente, o el modelo no podrá ser utilizado.

Si se utiliza Blender para crear los objetos, como es este caso, bastaría con marcar la opción “Triangulate” a la hora de exportar el modelo .obj. Para obtener una correcta conversión del modelo a .h que asegure que los objetos utilizados en todas las librerías son exactamente iguales, es decir, que utilizan el mismo modelo en la etapa de renderizado, también es necesario marcar las opciones “Edges”, “Normals”, “High Quality Normals”, “UVS”, “Materials”, y “Apply Modifiers”.

A pesar de todo, una vez unificado el formato, la tarea no ha acabado, pues aunque aparentemente todas las librerías ya pueden trabajar con nuestro modelo .obj, cada una utiliza una configuración .obj diferente. Aunque no es el objetivo de este trabajo profundizar en este tema, se recomienda analizar los archivos o modelos que utiliza la librería como ejemplo antes de empezar a utilizar el nuestro propio, y adaptarlo para que pueda ser leído. Aún así, la exportación de materiales y texturas es una de las tareas que más tiempo ocupan en este tipo de trabajos.

Ahora sí, pasamos a definir los objetos utilizados en nuestro trabajo. Los objetos a renderizar generalmente dependen de la aplicación para la que esté diseñado nuestro proyecto y pueden ser desde los más sencillos hasta los más complicados. En algunos casos las aplicaciones requieren animación de los objetos, pero esta característica no se contempla en el estudio dado que sólo dos de las librerías lo soportan. Recordar que NyARToolkit y Metaio admiten formato .md2. Por su parte Vuforia dispone de una

extensión para ser utilizada con Unity 3D, pero no es libre. Pues bien, en nuestro caso no estamos desarrollando aplicaciones para ningún entorno concreto, sino la capacidad de las librerías para renderizarlos, por lo que el objeto en sí (sus colores, su realismo, tamaño, etc.) no es crucial. Así pues se han utilizado objetos a partir de primitivas a los que se le ha ido añadiendo complejidad. Por un lado se han utilizado esferas, a las que se les ha ido añadiendo resolución (número de caras). En total se han utilizado 6 modelos de esferas diferentes. También se ha empleado un modelo con agujeros para comprobar su comportamiento. Además se han realizado pruebas con un modelo no simétrico, concretamente una ficha de ajedrez para comprobar el comportamiento de ésta frente al giro de los marcadores. Ver *Figura 4-5* y *Figura 4-6*.

Figura 4-5. Muestra de algunas esferas utilizadas con diferente número de caras: 112, 8.604 y 130.560.

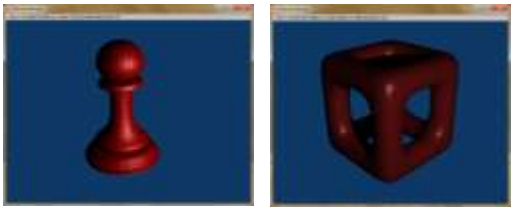

Figura 4-6. Muestra de objetos utilizados. Ficha de ajedrez:19.968 caras. Cubo agujereado: 46.080 caras

Para que las distintas librerías soporten los objetos y posteriormente integrarlos en la escena es muy importante que éstos sean exportados de forma correcta desde la herramienta utilizada para su creación, como puede ser Blender. Se deben cumplir los dos siguientes requisitos para una correcta visualización:

- Cada cara del objeto debe tener las normales especificadas.
- El objeto debe ser triangularizado al completo, es decir, todas las caras de éste deben de tener únicamente 3 vértices.

Además hay que tener en cuenta que Blender y OpenGL no utilizan el mismo sistema de coordenadas. Si este aspecto se tiene en cuenta a la hora de crear el objeto evitamos tener que hacer más transformaciones al objeto en la fase de renderizado.

Para cada frame procesado en el que se detecta un marcador, el sistema de tracking de cada SDK necesita resolver la matriz de transformación entre la cámara y el marker

o target, para posicionar el objeto 3D correctamente. Una vez calculada esta matriz se pasa al proceso de renderizado del objeto.

Generalmente la capacidad de renderizado de una aplicación se mide con la tasa de **frames por segundo**. A medida que aumenta la complejidad de la escena a renderizar esta tasa suele bajar. Una de las características básicas de una aplicación de RA es que sea en tiempo real [AZUMA 97], para ello la tasa de frames por segundo tiene que estar entre los 15 fps y 30 fps. Los dispositivos móviles de hoy en día ofrecen unas capacidades que cumplen con las expectativas de las aplicaciones de RA [FERNÁNDEZ 12]. Por ejemplo, el dispositivo utilizado para la realización de pruebas, Samsung Galaxy SII, ofrece hasta 60 fps, cuando se está renderizando solo la imagen captada por la cámara. Este dispositivo posee una tarjeta gráfica o (GPU) Mali-400MP, siendo una tarjeta de gama alta, probablemente la más potente del mercado, y llegando a "reproducir" 80 millones de triángulos por segundo. Es decir, que podemos estar seguros de que el dispositivo no ofrece ninguna limitación a la hora de probar el rendimiento de renderizado de los SDK.

Una vez hecho esto se han implementado aplicaciones similares, en la medida de lo posible, para cada uno de los SDK. Estas aplicaciones simplemente cumplen con las 4 etapas definidas anteriormente de cualquier aplicación de RA.

1.- Captura: Se utilizan las propias funciones ya implementadas por todas las librerías.

2.- Reconocimiento y tracking: Se modifican ligeramente algunas partes del código para adaptarlas al estudio. A través del LogCat de Android se puede extraer información en tiempo de ejecuciónde las distintas tareas realizadas en esta fase.

3.- Renderizado: Se modifican ligeramente algunas partes del código para adaptarlas al estudio, como la modificación de objetos a renderizar. A través del LogCat de Android se puede extraer información en tiempo de ejecución de las distintas tareas realizadas en esta fase, como la tasa de frames por segundo.

4.- Visualización: Se utilizan las funciones ya implementadas por todas las librerías.

4.3.Pruebas realizadas

1. Estudio de la eficiencia de la fase de reconocimiento y tracking

En este apartado se han realizado diversas pruebas para comprobar la eficiencia de los reconocedores frente a diferentes situaciones. Las pruebas realizadas son iguales para todos los SDK, aunque se ha utilizado el marcador correspondiente en cada caso, como ya se ha adelantado. Para Vuforia y Metaio se ha utilizado el marcador natural "Cabanyal.png" y para NyARToolkit y AndAR la marca fiducial "marker_music.png" (Figura. 4-1). Aunque para alguna prueba específica se han utilizado otros marcadores, como se comentará. Concretamente se ha comprobado el reconocimiento de marker o target en los siguientes casos.

1.1- Reconocimiento del marcador cuando éste es ocultado parcialmente

Como se puede observar en las capturas de pantalla realizadas para cada aplicación, justamente los SDK que utilizan marker no son capaces de reconocerlo cuando éste está parcialmente tapado (*Figura 4-7*). Sin embargo los SDK que utilizan marcadores naturales, sí. No obstante como puede verse en la *Figura 4-8*, Vuforia tiene un mejor comportamiento en este sentido que Metaio (*Figura 4-9*).

Como curiosidad decir que mientras Metaio y Vuforia sí que implementan la función ScreenShot de Android, andar y NyArtoolkit no lo hacen. Por ello las pruebas realizadas para estos SDKs han sido capturadas con una cámara de fotos externa. Esto es una desventaja, pues la acción ScreeShot añade funcionalidades a las aplicaciones de RA.

Figura 4-7. Ejemplo de ejecución de AndAR y NyARToolkit con una esquina del marcado oculta

Figura 4-8. Ejemplo de ejecución de Vuforia ocultando parcialmente el target

Figura 4-9. Ejemplo de ejecución de Metaio ocultando parcialmente el target

El hecho de que AndAR y NyARToolkit no funcionen al tapar el marcador era de esperar, pues el reconocimiento del marker empieza por reconocer el marco negro exterior, para después pasar a reconocer la imagen del centro. En la documentación de la librería ARToolKit [Kat99][Kat00], se resume el proceso de registro. En general, el sistema binariza en tiempo real la información capturada por la cámara para permitir mediante segmentación el reconocimiento de regiones de alto contraste (regiones blancas y negras); debido a que el sistema conoce la forma del marcador, busca regiones con formas similares, las cuales indican la presencia de un marcador. Una vez se ha identificado, se realiza un procesamiento digital a la imagen para extraer su contorno y a partir de ella los cuatro puntos donde se interceptan cada dos semirectas, dichos vértices son utilizados en el proceso de seguimiento para identificar la posición del marcador. Finalmente, la imagen interior del marcador se extrae y se compara con las imágenes almacenadas en el sistema y se le asocia el objeto virtual correspondiente. Es por ello que si alguna región del borde es tapada, no se llega a reconocer la forma del marcador, y por lo tanto tampoco éste. En [JARAMILLO 11] se realiza una revisión de diferentes técnicas utilizadas para salvar esta situación.

Sin embargo los SDK basados en marcadores naturales poseen una gran número de características especiales, que pueden basarse en colores, geometrías, etc. (ver

Figura 4-4). En nuestro caso además se ha utilizado el target óptimo según el propio SDK.

1.2- Reconocimiento del marcador según su tamaño

Teniendo en cuenta que la efectividad del reconocimiento depende de la información capturada por la cámara, el tamaño y el diseño del marcador influye mucho. Así pues en este apartado se ha tratado la influencia del tamaño del marcador en el reconocimiento.

Se ha utilizado dos diseños de marcadores diferentes para cada grupo de SDKs, para descartar los errores de detección debidos al diseño.

Para NyARToolkit y AndAR se han utilizado (1) “android.png” y (2) “music.png”, y para Metaio y Vuforia (3) “cabanyal.png” (*Figura 4-3*) y (4) “piedras.png” (*Figura 4-10*).

Figura 4-10. Target “piedras.png”

Se han utilizado 5 tamaños diferentes para cada prueba. Los tamaños recomendados para cada SDK son diferentes. De hecho unos están diseñados para utilizar marcadores cuadrados y otros rectangulares. Por eso, las pruebas se han realizado partiendo del tamaño recomendado por cada SDK, y valores proporcionales a éste.

En andar y NyARToolkit, el tamaño recomendado de marcador es de 80x80 mm.

En Vuforia en tamaño recomendado es de 247x175mm.

En Metaio recomiendan utilizar una imagen de 222x111mm para la creación del target, pero finalmente el fichero de configuración del tracking sólo contempla una imagen de 100x100mm, que de hecho es también proporcionada por la herramienta Metaio creator.

Los tamaños diferentes al original utilizados en las pruebas se muestran en la Tabla 4-1 como porcentajes de reducción sobre éste. Para cada SDK se especifica si el marcador es reconocido o no.

	AndAR		NyARToolkit		Vuforia		Metaio	
	(1)	(2)	(1)	(2)	(3)	(4)	(3)	(4)
Recomendado	✓	✓	✓	✓	✓	✓	✓	✓
70%	✓	✓	✓	✓	✓	✓	✓	✓
50%	✓	✓	✓	✓	✓	✓	✓	✓
33.3%	✓	✓	✓	✓	✓	✓	X	X
25%	X*	X*	X*	X*	✓	✓	X	X
12.5%	X*	X*	X*	X*	X	X	X	X

Tabla 4-1. Resultados de las pruebas realizadas para diferentes tamaños de marker o target

A pesar de que los marcadores deberían ser más sencillos de reconocer, a vista de los resultados obtenidos, NyARTolKit y AndAR no son las librerías que mejor se comportan en este apartado del estudio. Las casillas marcadas con asterisco merecen una explicación.

En primer lugar, hay que decir que las librería basadas en ARToolKit aparentemente reconocen el marcador aunque este se reduzca hasta el 12.5%. Sin embargo esto no son más que **falsos positivos**. Esto es, el sistema está reconociendo una imagen como marcador cuando en realidad no lo es [FIALA 05]. Con un marco cuadrado negro, pero el pequeño tamaño de las figuras que se encuentran en el centro de esta hace que sean muy fáciles de confundir. Se ha comprobado que a partir de tamaños del 25% se producen falsos positivos, y para tamaños del 12.5% reconocer cualquier figura que tenga un marco negro como marcador.

Hay que explicar que estas librerías, basadas en ARToolKit, en la etapa de reconocimiento asocian a cada marcador un coeficiente que mide el grado de confianza. Si este supera el umbral definido, el marcador se considera reconocido.

El valor de este coeficiente se puede imprimir a través del LogCat del SDK de Android. Pues bien, sólo las dos primeras pruebas registran un resultado de este coeficiente mayor de 0.8 (más del 80% de probabilidad de que el reconocimiento sea cierto). Cuando el tamaño del marcador está entre el 50% y el 30% del tamaño original el valor del coeficiente de confianza está entre 0.6 y 0.8. Por debajo de este tamaño, hasta el 12.5%, el cf supera el umbral establecido, pero nunca supera el 0.6. Esto explica los falsos positivos de estos dos casos.

Cabe señalar también, que el coeficiente cf obtenido en ambos casos con el marcador "Android.png" es siempre más alto que el obtenido con "music.png". esto era de espera dada a su mayor simplicidad. Sin embargo, a la vista de los resultados obtenidos, de cara al funcionamiento de la aplicación podemos decir que el diseño del marcador no ha influido en el reconocimiento.

En el caso de los otros dos SDK, se comprueba que Vuforia tiene un mejor comportamiento frente al tamaño del marcador que Metaio. En ambos casos es difícil que se produzcan falsos positivos, porque recordemos que el reconocimiento está basado en características específicas y no en la comparación pixel a pixel.

Hay que señalar que incluso realizando las pruebas con el target propuesto en los ejemplos del SDK de Metaio, la aplicación tiene problemas para reconocerlo cuando este reduce su tamaño a un tercio del original.

1.3- Reconocimiento del marcador según la distorsión de la perspectiva

En este caso quiere analizarse el comportamiento de cada SDK frente a distorsiones de la perspectiva. La distorsión de perspectiva se ha representado mediante ángulos de visión de 90º a 15º.

Para estas pruebas se han utilizado las mismas 4 imágenes que en el caso anterior. En la *Tabla 4-2* pueden verse los resultados obtenidos.

	AndAR		NyARToolkit		Vuforia		Metaio	
	(1)	(2)	(1)	(2)	(3)	(4)	(3)	(4)
90º	✓	✓	✓	✓	✓	✓	✓	✓
75º	✓	✓	✓	✓	✓	✓	✓	✓
60º	✓	✓	✓	✓	✓	✓	✓	✓
45º	✓	✓	✓	✓	✓	✓	✓	✓
30º	X	X	✓	✓	✓	✓	✓	✓
15º	X	X	X	X	✓	✓	X	X

Tabla 4-2. Resultados de las pruebas realizadas con diferentes ángulos de inclinación del marker o target

En este caso, hay que señalar la diferencia de funcionamiento entre NyARToolkit y AndAR. NyARToolkit define dos umbrales. Un umbral de 0.3 para descartar la existencia o no de marcador, y otro umbral de 0.5 para reconocer un marcador concreto. Cuando se supera el umbral de 0.3 el sistema realiza toma un vértice del

marco negro como referencia y realiza ajustes en base a la distorsión, para facilitar el reconocimiento del marcador y evitar los falsos positivos. Sin embargo AndAR únicamente utiliza un valor umbral de 0.5. Es decir para valores de confianza entre 0.3 y 0.5 NyARToolkit ofrece mejores prestaciones.

Observamos que el reconocimiento de marcadores naturales es mucho más eficiente cuando nos enfrentamos a la inclinación de éstas. De entre los dos SDKs el más eficaz es Vuforia, que no deja de reconocer el target en ninguna de las pruebas realizadas.

2. Estudio de la eficiencia del renderizado

En apartados anteriores ya se ha adelantado que la complejidad de la escena a aumentar influye en la eficiencia de renderizado de las aplicaciones. Esta eficiencia está reflejada en la tasa de frames por segundo.

Calcular este parámetro es relativamente sencillo. Basta con contar el número de frames que se renderizan en un segundo y calcular su inverso. Para ello se incluye este código en la función *onDrawFrame* (o su equivalente) del interfaz *GLSurface.Renderer* de Android sea cual que sea el modo en el que se implemente en cada uno de los SDK. El resultado de la tasa de fps puede verse a través del Logcat del SDK de Android.

Señalar que en Vuforia la función de renderizado se realiza con las funciones nativas en C, por lo que tras cualquier transformación de este código debe volver a compilarse la librería de forma externa a eclipse, con el comando *ndk-build* y actualizar el proyecto una vez compilado con éxito.

En este apartado se estudia el rendimiento de cada SDK en términos de frames por segundo modificando la escena aumentada. Por un lado se analiza el comportamiento de la tasa de renderizado en función del número de caras del objeto, recordemos que una cara de nuestros modelos es cada uno de los triángulos que lo foma. Por otro lado se estudia este comportamiento en función del número de objetos renderizados.

2.1- Tasa de frames por segundo en función del número de caras

En este apartado se estudia la eficiencia del sistema de renderizado del SDK. Partiendo de que ya se ha resuelto el hecho de que todas utilicen el mismo objeto se realiza la misma prueba para todos los SDK. Se comienza por un objeto muy simple de tan sólo 112 caras, y se va aumentando hasta llegar a los objetos más complejos.

En la *Tabla 4-3* se recogen los valores recogidos de fps durante las ejecuciones con los diferentes objetos renderizados. En la última prueba realizada AndAR no ha sido capaz de renderizar el objeto, la aplicación reporta un error y se suspende la ejecución.

	AndAR	NyARToolkit	Vuforia	Metaio
Esfera de 112 caras	30 fps	27 fps	30 fps	30 fps
Esfera de 480 caras	30 fps	27 fps	30 fps	30 fps
Esfera de 1.984 caras	24 fps	23 fps	30 fps	27 fps
Esfera de 8.064 caras	22 fps	22 fps	30 fps	27 fps
Esfera de 28.560 caras	15 fps	15 fps	23 fps	22 fps
Esfera de 130.560 caras	-	7 fps	15 fps	12 fps
Ficha ajedrez 19.968 caras	20 fps	20 fps	30 fps	25 fps
Cubo agujereado 46.080 caras	12 fps	10 fps	15 fps	11 fps

Tabla 4-3. Resultados de las pruebas realizadas con objetos de diferente número de caras.

A la vista de los resultados lo más llamativo es el bajo rendimiento de las librerías NyARToolkit y AndAR frente a las otras dos. Entre ellas NyARToolkit es la que peor resultado ofrece, incluso con el objeto más simple. Ello se debe a que NyARToolkit utiliza un motor de renderizado externo, min3D, lo cual ralentiza esta tarea enormemente. Esto es más llamativo cuando los objetos son más complejos.

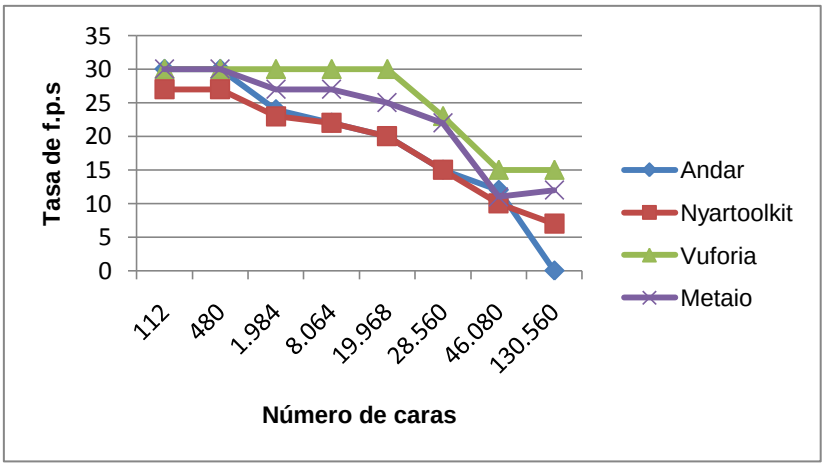

Figura 4-11. Gráfico de resultados. FPS frente a número de caras de los objetos

El SDK que ofrece mejores resultados en todos los casos es Vuforia (*Figura 4-11*). Una explicación es que Vuforia utiliza OpenGL directamente, que sólo contiene funciones de bajo nivel para mostrar y representar gráficos. El hecho de definir el

objeto directamente desde primitivas, resulta mucho más eficiente computacionalmente que tener el hecho de leer y cargar los archivos .obj y .mtl. Es por ello que la tarea de dibujar objetos como los que manejamos, con texturas simples, se realizan de manera rápida mediante comandos simples. Hablamos de texturas simples, porque son imágenes de un solo color de 256x256 pixels.

Por otro lado, señalar que Qualcomm, la empresa que está detrás de este SDK, es la desarrolladora de la familia de System on Chip (SoC) Snapdragon para la mayoría de dispositivos móviles, el que se ha utilizado está entre ellos. Es por ello que el SDK de Cualqcomm, Vuforia, genera los gráficos de manera óptima para sacar el máximo rendimiento a las prestaciones de Snapdragon.

Una tasa de menos de 15 fps no tiene sentido para este tipo de aplicaciones, por lo que se puede decir que para objetos a partir de 40.000 caras, Vuforia es la única que podría utilizarse de manera efectiva.

2.2- Tasa de frames por segundo en función del número de objetos

En este apartado del estudio se analiza la influencia del número de objetos renderizados en la tasa de fps Para la realización de estas pruebas se han utilizados los objetos simples, esferas, concretamente el modelo de 480 caras (*Figura 4-12* y *Figura 4-13*).

Estas pruebas no han podido ser realizadas con Metaio dado que la función de renderizado está oculta en la versión gratuita. Sólo es posible cargar un modelo y modificar su matriz de transformación.

	AndAR	NyARToolkit	Vuforia	Metaio
1 objeto	30 fps	27 fps	30 fps	30 fps
5 objetos	27 fps	27 fps	30 fps	-
25 objetos	20 fps	17 fps	30 fps	-
50 objetos	14 fps	12 fps	30 fps	-
75 objetos	-	-	23 fps	-
100 objetos	-	-	11 fps	-

Tabla 4-4. Resultados de las pruebas realizadas con diferente número de objetos

A partir de 75 objetos, AndAR y NyARToolkit no son capaces de realizar el renderizado y se suspende la ejecución de la aplicación.

Figura 4-12. Ejemplo de ejecución de AndAR con 33 objetos

Figura 4-13. Ejemplo de ejecución de Vuforia con 100 objetos

Capítulo 5

CONCLUSIONES

5.1. Conclusiones del estudio comparativo

En este trabajo se planteaban una serie de objetivos que se han conseguido. Tras una búsqueda exhaustiva, se han seleccionado cuatro herramientas que cumplían con las especificaciones definidas. Tras superar las dificultades encontradas para hacerlas funcionar en el dispositivo móvil, se han realizado pruebas básicas para familiarizarnos con la arquitectura de cada una de ellas. Tras este paso, se ha definido la metodología del estudio a realizar unificando formatos para todas las herramientas. Por un lado, se han estudiado los markers y targets comunes a utilizar, hasta escoger los diseños óptimos para la realización de pruebas y se han traducido a los formatos utilizados en cada herramienta. Por otro lado, se han creado escenas virtuales comunes para todas las herramientas, escogiendo el formato .obj como estándar para todas (y una transformación de éste para Vuforia). El tratamiento de los objetos presenta particularidades en cada herramienta, por lo que su uso en cada una de ellas no es trivial. A lo largo del documento se exponen algunas claves para la exportación de estos objetos desde frameworks de diseño 3D para hacerlos compatibles con cada herramienta. Señalar al respecto que la principal dificultad se encuentra en el manejo de texturas si el desarrollador no tiene conocimientos de gráficos por computador.

Tras el estudio previo, se ha elaborado una batería de pruebas para analizar el rendimiento de cada herramienta respecto a diferentes parámetros. Cada módulo de la batería de pruebas ha sido ejecutado en todas las herramientas y se ha ido sacando conclusiones de los resultados obtenidos.

En este apartado, para finalizar, se realiza un análisis global de resultados (ver Tabla 5-1. Resumen de los resultados obtenidos). Siguiendo la dinámica de todo el trabajo, estas conclusiones se presentan en tres grupos.

	AndAR	NyARToolkit	Vuforia	Metaio
Comportamiento frente a oclusiones del marker/target	No reconoce el marker	No reconoce el marker	Rendimiento alto	Rendimiento bajo
Comportamiento frente al tamaño del marker/target	Rendimiento medio	Rendimiento medio	Rendimiento alto	Rendimiento medio
Comportamiento frente a la distorsión de la perspectiva del marker/target	Rendimiento bajo	Rendimiento medio	Rendimiento alto	Rendimiento medio
Comportamiento frente el número de caras del objeto aumentado	• Rendimiento alto hasta 1.000 caras • Rendimiento medio de 1.000 a 40.000 caras • Rendimiento bajo a partir de 40.000 caras	• Rendimiento alto hasta 1.000 caras • Rendimiento medio de 1.000 a 40.000 caras • Rendimiento bajo a partir de 40.000 caras	• Rendimiento alto hasta 20.000 caras • Rendimiento medio de 20.000 a 130.000 caras	• Rendimiento alto hasta 20.000 caras • Rendimiento medio de 20.000 a 40.000 caras • Rendimiento bajo a partir de 40.000 caras
Comportamiento frente al número de objetos aumentados	• Rendimiento alto hasta 10 objetos • Rendimiento medio de 10 a 50 objetos • Rendimiento bajo a partir de 50 objetos	• Rendimiento alto hasta 10 objetos • Rendimiento medio de 10 a 30 objetos • Rendimiento bajo a partir de 30 objetos	• Rendimiento alto hasta 50 objetos • Rendimiento medio de 50 a 90 objetos • Rendimiento bajo a partir de 90 objetos	-

Tabla 5-1. Resumen de los resultados obtenidos

NyARToolkit VS AndAR

Con respecto al reconocimiento de marcadores, NyARToolkit ofrece un mejor comportamiento. Aunque ambas tienen el problema de los falsos positivos, en los peores casos de iluminación y orientación del marcador hemos comprobado que este SDK se comporta mejor.

Sin embargo respecto al renderizado de los objetos utilizados, AndAR tiene un mejor comportamiento en general. El hecho de utilizar un motor de renderizado externo como min3D, mejora la calidad de los gráficos para aplicaciones especiales, pero esto ralentiza el tiempo de renderizado.

No obstante hay que señalar la versatilidad de NyARToolkit en cuanto a los formatos soportados en comparación con AndAR. Esta última, en su versión inicial sólo contemplaba renderizado con OpenGL ES, aunque ya se ha implementado una extensión de la clase ARObject que posibilita la utilización de modelos .obj sin afectar prácticamente al rendimiento de fase de renderizado. Por otro lado, NyARToolkit admite distintos formatos, metasequoia (.mqo), wavefront .obj, VRML y md2, este último incluye animaciones.

Existen trabajos que mejoran las capacidades gráficas de estas librerías, pero el objeto de este estudio era compararlas en su versión disponible para el desarrollador.

A pesar del peor rendimiento de NyARToolkit en la fase de renderizado, las diferencias no son exageradas. El hecho de ser multiplataforma también es una ventaja añadida ya que lo hace independiente de la arquitectura de cara a los desarrolladores. Sin embargo, si tuviéramos que recomendar una librería a un desarrollador no experimentado, sin duda recomendaríamos AndAR por su estructura más intuitiva y su mayor documentación.

Vuforia VS Metaio SDK Mobile

Respecto al reconocimiento del target se puede decir que, en general, Vuforia ofrece un comportamiento mejor que Metaio.

Por un lado, Vuforia se comporta mejor ante oclusiones del marcador. Se han probado diferentes marcadores para descartar la posibilidad de la concentración de características en la región ocultada en Metaio. Al tratarse de una actividad que se realiza fuera del tiempo de ejecución de las aplicaciones, Vuforia hace un estudio exhaustivo de las características especiales, por la que cada imagen, concretamente

sus ficheros de tracking, dispone de tal número de características registradas que no es un problema ocultar algunas de ellas. Aunque no se puede asegurar, porque se encuentra en una capa abstraída de la versión gratuita, es posible que Metaio emplee menos características en el reconocimiento y tracking de los targets. Sin embargo una buena mejora para Vuforia sería disponer de una versión off-line para la extracción de características de los targets.

En las pruebas realizadas para diferentes tamaños del target e inclinación se obtienen resultados similares. Pero Vuforia es la que mejor comportamiento refleja. Se deduce que la razón es la misma que la expuesta en el párrafo anterior.

A vista de los resultados obtenidos en las pruebas de renderizado de objetos con diferente número de caras Vuforia, ofrece un comportamiento mejor. Como se ha explicado anteriormente, era de esperar por su uso de OpenGL y optimización de los algoritmos para sacer el máximo partido al dispositivo móvil. Además, el módulo de renderizado de Vuforia se encuentra en las funciones nativas en C++, a diferencia de Metaio que está escrito íntegramente en Java.

Por otro lado, el motor de renderizado utilizado por Metaio para el dibujado de objetos virtuales (.obj) está oculto en la versión libre. Ello ha impedido realizar la prueba de inserción de distinto número de objetos con este SDK, y supone una gran desventaja a la hora de implementar nuestras propias aplicaciones.

Por otro lado hay que señalar que, para el mismo target y el mismo objeto a renderizar, Metaio se comporta peor en algunas ocasiones produciéndose momentos de jitter y parpadeo. Esto se aprecia especialmente cuando el target se mueve rápidamente.

Comparación global

El uso de markers o targets es la principal característica que diferencia a los SDK y las divide en dos grupos. NyARToolkit y AndAR utilizan los clásicos marcadores utilizados en ARToolKit consistente en un cuadrado negro con un símbolo no simétrico en el centro. El uso de estos marcadores está muy extendido por la baja complejidad de los algoritmos de tracking. Sin embargo, las prestaciones hardware y software de los actuales dispositivos móviles permiten la utilización de algoritmos más complejos como los algoritmos de tracking basados en marcadores naturales.

Como se exponía en la sección anterior, los markers ofrecen peores prestaciones frente a cambios de iluminación y posición de estos que los marcadores naturales. Además una de las principales debilidades de los markers es la necesidad de que el éste esté siempre visible, lo que imposibilita la interactuación del usuario tocando el marcador durante la ejecución de la aplicación, pues ello haría desaparecer el objeto virtual. Hecho que supone un gran hándicap en algunas aplicaciones.

Por otro lado, es cierto que la carga computacional del reconocimiento de markers es algo menor que el reconocimiento de marcas naturales. Sin embargo, dado las altas prestaciones de los dispositivos móviles de hoy este aspecto no se considera un problema.

Después de haber mostrado algunas de las principales diferencias entre ellos, se puede considerar que los sistemas de RA basados en marcadores naturales son una opción mejor para las aplicaciones finales. Además, debido a que utilizan imágenes normales u objetos del entorno no resultan intrusivos como los markers.

Así pues, ha quedado demostrado que el uso de marcadores naturales resulta la mejor opción a la hora de elegir un framework de desarrollo de aplicaciones de RA.

En este trabajo se han utilizado dos SDKs de este tipo. Entre ellos, se ha visto que Vuforia ofrece un mejor comportamiento en la fase de reconocimiento. Se ha comprobado que esta herramienta es mucho más eficaz frente a oclusiones del marker. Además, a diferencia de Metaio, realiza un mejor tracking del target cuando éste se mueve rápidamente. Aunque esta funcionalidad no se ha abordado en el estudio, Vuforia ofrece tracking de objetos 3D. De momento sólo está preparado para reconocer paralelepípedos, utilizando cada cara como target. Se han realizado pruebas y uno de los aspectos a destacar es que respeta perfectamente la geometría y tamaño del paralelepípedo sea cual sea la perspectiva, no produciéndose oclusiones entre éste y el objeto aumentado.

Concluimos por tanto que la mejor elección desde el punto de vista de localización del viewpoint es Vuforia.

En la parte gráfica hemos visto que el estándar OpenGL ES es el más eficiente. Pero, también limita el grado de complejidad de los gráficos. OpenGL sólo dispone de funciones de bajo nivel para representar gráficos. Para representar cualquier modelo basta con especificar las coordenadas de sus vértices, de las texturas y de las normales, así como ejecutar operaciones básicas de transformación. Ello podría resultar tedioso, pero hemos visto que existe la posibilidad de convertir un archivo

creado en Blender (o cualquier otra herramienta) en un archivo .h que puede ser directamente incluido en nuestra función de render, y dibujar los vértices mediante *glDrawArrays(GL_TRIANGLES, 0, num_vertices).*

Sin embargo, cuando se trata de mallas muy complejas, animaciones, iluminación y texturas, es preferible cargar los modelos en módulos aparte. Incluso es posible utilizar motores de renderizado, como min3D escrito en Java. Aunque ya hemos probado esta opción para nuestros modelos con NyARToolkit y hemos visto que, este motor en concreto, no ofrece un buen rendimiento en términos de frames por segundo.

En la gráfica de la *Figura 4-11* puede verse que Metaio y Vuforia ofrece un comportamiento bastante mejor que los otros dos SDKs en términos de frames por segundo. El comportamiento de ambas es bastante similar hasta que se empieza a dibujar objetos de elevado número de caras, que es cuando empieza a funcionar mejor Vuforia.

Aunque Metaio ofrece mayor versatilidad en cuanto a formatos de modelos 3D admitidos, esta etapa de renderizado está oculta en la versión gratuita, por lo que no se le puede sacar mucho partido.

Por tanto, la mejor opción desde el punto de vista del renderizado es también Vuforia.

Para concluir, se ha elegido el SDK de Vuforia como mejor opción para el desarrollo de aplicaciones móviles de RA. Las pruebas y posterior análisis de resultados a lo largo del documento justifican nuestra elección. Por último, es la herramienta que mejor se adapta a las necesidades del proyecto en el que se enmarca esta tesina.

5.2. Trabajos futuros

Seguidamente, se plantean una serie de líneas de trabajo futuras como continuación del presente trabajo.

- **Utilización de múltiples targets**: puesto que Vuforia presenta esta posibilidad, se podría realizar un estudio de rendimiento al utilizar múltiples targets siguiendo la línea del presente trabajo.
- **Inserción de botones virtuales**: Esta es otra de las opciones que ofrece Vuforia. Los botones virtuales se podrían crear sobre el propio marcador puesto que, como hemos visto, la herramienta sigue reconociendo el target cuando éste está parcialmente ocluido. Esto supone una gran ventaja a la hora de crear aplicaciones en las que la interacción del usuario sea importante,

como podrían ser las enfocadas al aprendizaje. Se plantea por tanto el estudio de esta funcionalidad más profundamente, así como un posible estudio de la mejora de la experiencia del usuario al introducir esta opción**.**

- **Reconocimiento de objetos tridimensionales:** Esta opción sí que ha llegado a probarse en el presente trabajo con Vuforia. Sin embargo, sólo para objetos paralelepípedos. Se podría realizar un análisis de las posibilidades que ofrece la herramienta para el reconocimiento de objetos de distintas formas.
- **Inclusión de animaciones en la escena aumentada:** Como se ha comentado, Vuforia dispone de una extensión para Unity3D, herramienta para el desarrollo de videojuegos para diversas plataformas incluída Android. Sin embargo, también se ha hablado de que la extensión de Unity3D que permite interactuar con Vuforia no es gratuita. Sin embargo, sería muy interesante poder incluir animaciones en nuestras escena aumentadas, por lo que se plantea un estudio de alternativas para integrar motores de renderizado o desarrollar el nuestro propio, de manera que se puedan incluir objetos 3D con formatos que soporten animaciones, como puede ser .md2. Entre estas posibilidades, cabe mencionar a OSG.

BIBLIOGRAFÍA

[ANDAR 12] Proyecto AndAR - Android Augmented Reality. http://code.google.com/p/andar/, fecha de consulta: 7-9-2012.

[ART 99] Kato, H., Billinghurst, M.(1999) Marker tracking and HMD calibration for a video-based augmented reality conferencing system, Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR 99), pp: 85-94.

[AZUMA 97] Azuma, R. (1997). A survey of augmented reality. Presence: Teleoperators and Virtual Environments, 6(4): 355–385.

[CAUDELL 92] Caudell, T. P., Mizell, D. W. (1992) Augmented Reality: An Application of Heads-Up Display Technology to Manual Manufacturing Processes, Twenty-fifth Hawaii International Conference on System Sciences, vol. 2, pp:659-669.

[CLEMENS 11] Clemens, A., Schmalstieg, D. (2011) Challenges of Large-Scale Augmented Reality on Smartphones, ISMAR 2011 Workshop: Enabling Large-Scale Outdoor Mixed Reality and Augmented Reality.

[CONNOLLY 10] Connolly, P., Chambers, C., Eagleson, E., Matthews, D., Rogers, T. (2010) Purdue University . Augmented Reality Effectiveness in Advertising

[CHEN 08] Chen, I. Y-H, Macdonald, B., Wünsche, B. (2008) Markerless Augmented Reality for Robots in Unprepared Environments. Australian Conference on Robotics and Automation, disponible para descarga en: http://www.cs.auckland.ac.nz/~burkhard/Publications/ACRA2008ChenMacDonaldWuensche.pdf, fecha de consulta: 7-9-2012.

[CHENG 10] Cheng, L. (2010) University of Central Florida. Analysis and Comparison with Android and iPhone Operating Systems, disponible para descarga en: http://www.eecs.ucf.edu/~dcm/Teaching/COP5611Spring2010/Project/AmberChang-Project.pdf, fecha de consulta: 7-9-2012.

[CLEMENS 11] Clemens, A., Schmalstieg, D. (2011) Challenges of Large-Scale Augmented Reality on Smartphones, Workshop: Enabling Large-Scale Outdoor Mixed Reality and Augmented Reality, International Symposium on Mixed and Augmented Reality (ISMAR'11), disponible para descarga en: http://www.navteq.com/outdoor_mar2011/challenges.pdf

[CONNOLLY 10] Connolly, P., Chambers, C., Eagleson, E., Matthews, D., Rogers, T. (2010) Augmented Reality Effectiveness in Advertising. 65th Midyear Conference Engineering Design Graphics Division of ASEE, disponible para descarga en: http://edgd.asee.org/conferences/proceedings/65th%20Midyear/Connolly_Cha

mbers_Augmented_Reality_%20Effectiveness%20in%20Advert.pdf, fecha de consulta: 7-9-2012.

[DOMHAN 10] Domhan, T. (2010) Augmented Reality on Android Smartphones. Universidad de Stuttgart, disponible para descarga en: http://www2.bwcon.de/fileadmin/_softwareforschung/downloads/WISTA/Tobias_Domhan_Studienarbeit.pdf, fecha de consulta: 7-9-2012.

[FERNÁDEZ 12] Fernández, V., Orduña, J. M., Morillo, P. (2012) Performance Characterization of Mobile Phones in Augmented Reality Marker Tracking. 12th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE'12), pp. 537-549.

[FIALA 05] Fiala, M. (2005) Artag, a fiducial marker system using digital techniques. In CVPR'05, pp 590 –596.

[FUCHS 97] Fuchs, H., Livingston, M. A., Raskar, R., Colucci, D., Keller, K., State, A., Crawford, J. R., Rademacher, P., Drake, S.H., Meyer, A. (1997) Augmented reality visualization for laparoscopic surgery. In Medical Image Computing and Computer-Assisted Intervention, pp 934-943.

[FERRARI 01] Ferrari, V., Tuytelaars, T., Van Gool, L. (2001) Markerless Augmented Reality with a Real-time Affine Region Tracker, IEEE and ACM Intl. Symposium on Augmented Reality, pp. 87-96.

[GONZÁLEZ 12] González-Gancedo, S., Juan, M. C., Seguí, I., Rando, N., Cano, J., (2012) Towards a mixed reality learning environment in the classroom, GRAPP, pp. 434- 439.

[HENRYSSON05] Henrysson, A., Billinghurst, M., Ollila, M. (2005) Face to Face Collaborative AR on Mobile Phones, Proceedings of the 4th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 05), pp. 80-89.

[HOLLERER 99] Höllerer, T., Feiner, S., Terauchi, T., Rashid, G. , Hallaway, D. (1999) Exploring MARS: Developing indoor and outdoor user interfaces to a mobile augmented reality system. Computers and Graphics, 1999, Vol 23.,pp. 779–785.

[IDC 09] IDC (2009) Worldwide Internet Usage and Commerce 2009–2013. Forecast: Digital Marketplace Model and Forecast Version 2.1, Informe, Código: IDC08601, compra en: https://www.reportbuyer.com/checkout/basket.html?stage=1&add=ae18e57fb535e48a4f2cd40fab8e5c18&option=price_pdf, fecha de consulta: 7-9-2012.

[JARAMILLO 11] Jaramillo, G. E. (2011) Corrección del Error en el Proceso de Registro en los Sistemas de Realidad Aumentada Utilizando Técnicas Heurísticas. Tesis doctoral. Universidad Nacional de Colombia.

[JUAN 05] Juan, M. C., Botella, C., Baños, R., Alcañiz, M., Guerrero, B., Monserrat, C. (2005) Augmented Reality for the treatment of spider and cockroach phobias.

First prototype and first treatments, IEEE Computer Graphics & applications, N. Nov-Dic, pp. 31-37.

[JUAN 06] Juan, M. C., Baños, R., Botella, C., Pérez, D., Alcañiz, M., Monserrat, C. (2006) An Augmented Reality system for acrophobia. The sense of presence using immersive photography, Presence: Teleoperators and virtual environments, 15(4): 393-402.

[JUAN 11] Juan, M. C., Furió, D., Alem, L., Ashworth, P., Cano, J. (2011) ARGreenet and BasicGreenet. Two mobile games for learning how to recycle, WSCG'11, pp. 25-32, disponible para descarga en: http://wscg.zcu.cz/WSCG2011/!_2011_WSCG_Full_papers.pdf, fecha de consulta: 7-9-2012.

[KALKUSCKH 02] Kalkusch, M, Lidy, T, Knapp, M., Reitmayr, G., Kaufmann, H., Schmalstieg, D. (2002) Structured Visual Markers for Indoor Pathfinding, Proceedings of the First IEEE International Workshop on ARToolKit (ART02), pp. 8.

[KATO 00] Kato, H., Billinghurst, M., Poupyrev, I., Imamoto, K., Tachibana, K. (2000) Virtual Object Manipulation on a Table-Top AR Environment. Proceedings of the International Symposium on Augmented Reality, pp.111-119.

[KATO 99] Kato, H., Billinghurst, M. (1999) Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality '99, pp 85–94.

[KURZ 11] Kurz, D., Benhimane, S. (2011) Gravity-Aware Handheld Augmented Reality. IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR2011), pp. 111-120.

[LAYAR] Framework Layar: http://www.layar.com/what-is-layar/, fecha de consulta: 7-9-2012.

[LEE 07] Lee, T., Höllerer, T. (2007) Handy AR: Markerless Inspection of Augmented Reality Objects Using Fingertip Tracking. IEEE International Symposium on Wearable Computers (ISWC '07), pp. 83-90.

[LIN 12] Lin, J. (2012) Understanding and capturing people's mobile app privacy preferences. Tesis doctoral. Carnegie Mellon University (Pittsburgh).

[MEDINA 10] Medina, M., García-Cervigón, M., Carazo, O., Costas, H. (2010) Augmented reality for a new social network marketing. Mobile Augmented Reality Summit, disponible para descarga en http://www.perey.com/MobileARSummit/ReadyPeople-AR_for_social_marketing.pdf, fecha de consulta: 7-9-12.

[METAIO 08] Miyashita, T., Meier, P., Tachikawa, T., Orlic, S., Eble, T., Scholz, V., Gapel, A., Gerl, O., Arnaudov, S., Lieberknecht, S. (2008) An Augmented Reality Museum Guide, Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, (ISMAR 2008), pp. 103-106.

[METAIO] SDK de Metaio para dispositivos móviles. http://www.metaio.com/software/mobile-sdk/, fecha de consulta: 7-9-12.

[MILGRAM 94] Milgram, P., Takemura, H., Utsumi, A., Kishino, F. (1994) Augmented Reality: A class of displays on the reality-virtuality continuum. Proceedings of Telemanipulator and Telepresence Technologies. pp. 2351–34.

[MIN3D] Min3D. http://code.google.com/p/min3d/, fecha de consulta: 7-9-12.

[MIXARE] Mixare. http://www.mixare.org/, fecha de consulta: 7-9-12.

[MOHRING04] Möhring, M., Lessig, C., Bimber, O. (2004) Video See-Through AR on Consumer Cell Phones, Proceedings of the 3th IEEE/ACM international Symposium on Mixed and Augmented Reality (ISMAR 04), pp. 252-253.

[NYATK 12] NyARToolkit. http://nyatla.jp/nyartoolkit/wp/?page_id=198, fecha de consulta: 7-9-12.

[REKIMOTO95] Rekimoto, J., Nagao, K. (1995) The World through the Computer: Computer Augmented Interaction with Real World Environments, Proceedings of the 8th annual ACM symposium on User interface and software technology (UIST '95), pp. 29-36.

[REYTMAYR 03] Reitmayr, G., Schmalstieg, D. (2003) Collaborative Augmented Reality for Outdoor Navigation and Information Browsing, Symposium Location Based Services and TeleCartography, pp. 53-62.

[REYTMAYR 06] Reitmayr, G., Drummond, T. (2006) Going Out: Robust Model- based Tracking for Outdoor Augmented Reality, Proceedings of 5th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2006), pp. 109-118.

[REGENBRECHT 05] Regenbrecht, H., Baratoff, G., Wilke W. (2005) Augmented Reality Projects in Automotive and Aerospace Industry, IEEE Computer graphics and applications, 25(6): 48-56.

[SIMON 00] Simon, G., Fitzgibbon, A. W., Zisserman, A. (2010) Markerless tracking using planar structures in the scene, Augmented Reality, IEEE and ACM International Symposium on Augmented Reality (ISAR 2000). pp. 120-128.

[SZALAVÁRI 98] Szalavári, Z., Eckstein, E., Gervautz, M. (1998) Collaborative Gaming in Augmented Reality, Proceedings of VRST'98, pp.195-204.

[THOMAS 98] Thomas, B. H., Demczuk, V., Piekarski, W., Hepworth, D., Gunther, B. (1998) A wearable computer system with augmented reality to support terrestrial navigation, Proceedings of Second IEEE International Symposium on Wearable Computers (ISWC '98), pp. 168-171.

[VUFORIA] SDK de Vuforia para Android, disponible para descarga en: https://ar.qualcomm.at/qdevnet/sdk/android, fecha de consulta: 7-9-12.

[WAGNER 03] Wagner, D., Schmalstieg, D. (2003) First steps towards handheld augmented reality. IEEE International Symposium on Wearable Computers, pp 127–135.

[WAGNER 07] Wagner, D. (2007) Handheld Augmented Reality, Tesis doctoral, Graz University of Technology.

[WAGNER 08] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., Schmalstieg, D. (2008) Pose tracking from natural features on mobile phones, 7th IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR 2008), pp. 125-134.

[WAGNER 09] Wagner, D., Schmalstieg, D. (2009) History and future of tracking for mobile phone augmented reality. International Symposium on Ubiquitous Virtual Reality. pp. 7-10.

[ZHANG 02] Zhang ,X., Fronz, S., Navab, N. (2002) Visual Marker Detection and Decoding in AR Systems: A comparative Study. IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR'02), pp. 97-106.