UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UN NUEVO ENFOQUE DE REALIDAD AUMENTADA EN INTERIORES Y SUS APLICACIONES EN DISPOSITIVOS MÓVILES ANDROID

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

César Eduardo Aguirre Pineda

DIRECTOR DE TESIS:
ING. AMAURY H. PEREA MATSUMURA
2015

Para quienes creen, luchan y trabajan día a día por la educación pública en México y en la Universidad. Gracias.

Agradezco a mis padres, por su infinito amor y apoyo que siempre me han brindado.

Tabla de contenido

Índice de figuras

Introducción

Definición del problema

El uso de dispositivos móviles para presentar contenidos de Realidad Aumentada es una tendencia actual en la que existen distintos enfoques para llevar a cabo el tracking del usuario y presentar los contenidos virtuales pertinentes y en sincronía con el mundo real.

En este contexto, la localización en espacios interiores (como dentro de una casa o edificio) se ve obstaculizada por la ausencia de sensores, embebidos en los dispositivos móviles, con la capacidad de determinar las características del ambiente y la posición precisa de un usuario respecto a las paredes o los límites de un espacio cerrado.

Por esta razon, los enfoques actuales de Realidad Aumentada en interiores están orientados al reconocimiento de patrones de imágenes para sobreponer elementos virtuales en elementos reales. Sin embargo, estos patrones (Códigos QR, u otros patrones de imágenes) con frecuencia son invasivos ya que pocas veces forman parte del entorno real y suelen ser difíciles de manipular por el usuario estándar.

Objetivo

El presente trabajo de tesis tiene como objetivo:

- Poner en práctica los conocimientos adquiridos en la carrera para para proponer un nuevo enfoque de Realidad Aumentada que permita resolver los problemas relacionados con la ubicación y colocación de elementos virtuales en interiores mediante el uso de dispositivos móviles, excluyendo el uso de otros elementos tales como targets, patrones gráficos, u otros sensores no embebidos en el dispositivo.
- Verificar la viabilidad de dicho enfoque, utilizando los conocimientos generados, en una aplicación móvil de diseño de interiores, ayudando a su pronta adopción en nuestro país como una herramienta para potenciar la productividad y el entretenimiento de la población en general.

La investigación realizada repercutirá en el ámbito habitual de las aplicaciones de Realidad Aumentada: Publicidad, Educación, Artes, Ocio y Entretenimiento.

Resultados esperados

- Que el enfoque propuesto para realidad aumentada en interiores se ajuste y funcione correctamente en la mayoría de espacios interiores.
- Corroborar la viabilidad del enfoque propuesto mediante la elaboración de un prototipo de aplicación móvil de diseño de interiores

Capítulo 1
Industria móvil

Los sistemas de cómputo móviles, son aquellos que pueden desplazarse fácilmente y cuyas capacidades de cómputo pueden ser aprovechadas aún cuando se encuentran en movimiento (Reza, 2005, p.3).Es decir, que sin importar en donde se encuentre el usuario (sea en lo alto de un edificio, en movimiento, dentro de un automóvil o avión) podrá interactuar con dichas plataformas para consultar o procesar información.

Ciertamente el anhelo por conseguir todas esas bondades de un sistema de cómputo no es nuevo, ni tampoco los esfuerzos por alcanzar dichos objetivos. Sin embargo, es innegable que actualmente nos encontramos inmersos en una revolución tecnológica, en la que buena parte de los avances y cambios vertiginosos están ocurriendo en la industria móvil.

El uso de dispositivos móviles ha rebasado fronteras de costos y espacios[1], y hoy en día, es una realidad su uso cotidiano y profundamente arraigado en una vasta parte de la sociedad.
A continuación se repasa brevemente la evolución histórica del cómputo móvil, con el objetivo de obtener un mejor entendimiento de las condiciones actuales.

Historia

Inicios[2]: Equipos portables de comunicación

Una de las necesidades principales del ser humano, ha sido siempre la comunicación. Por ello, no es de extrañar que los primeros desarrollos de dispositivos móviles (y en su mayoría también los actuales) estuvieran ligados completamente a las tareas de telecomunicación

[1] Según la consultora the Competitive Intelligence Group, al cierre de 2014 en México existen 103.9

[2] El cómputo móvil se remonta hasta el 500 a.C, con el uso del ábaco. Un sistema de cómputo móvil por su reducido tamaño, portabilidad y facilitador del cálculo (Reza, 2004, p.4)

En la década de los 40s, surgen los Radio-Teléfono Móvil. Los primeros equipos de comunicación móvil, previos a la tecnología celular.

Eran dispositivos portátiles para los estándares de portabilidad de la época, sin embargo se encontraban muy lejos de ser piezas de uso personal (eran dispositivos de gran tamaño).
Solían montarse en automóviles, contaban con su propio número y tenían la desventaja de necesitar la intervención de un operador para su funcionamiento.
Las pocas líneas disponibles en el servicio, y su costo elevado contribuyeron a su limitada adopción.

Dispositivos móviles

Es hasta la década de los 80s con el surgimiento de la tecnología celular, que la telefonía se vuelve portátil, personal y accesible con relativa facilidad.
En esta misma época y gracias al aumento en las capacidades de procesamiento comienza el surgimiento masivo de dispositivos móviles de distintos propósitos (algunos ya alejados de las comunicaciones), como por ejemplo: videojuegos, calculadoras, relojes, PDAs.

Proliferación comercial

En los años 90s, el ecosistema de dispositivos móviles se hace aún más variado con la introducción al mercado de dispositivos como:

- Pagers (Beepers)
- Reproductores multimedia
- Cámaras y videocámaras digitales.
- PNA (Asistentes de navegación personal GPS)

Durante esta década resalta la proliferación comercial de computadoras portátiles de propósito general: Pocket PC, PDAs, Smartphones.

Revolución móvil

La evolución de los dispositivos móviles en la primera década de los años 2000, no fue en lo absoluto lenta.
Características como pantallas touch LCD a color, procesadores cada vez más potentes, proliferación de estándares de comunicación como Infrarrojo, Bluetooth, WiFi, un gran ecosistema de sistemas operativos móviles y la posibilidad de desarrollar software por terceros eran ya una realidad para esos años.

También lo eran la existencia en el mercado de una gran cantidad de diversos dispositivos móviles. (Cámaras, Celulares, Reproductores de música y video)

Grandes fabricantes de dispositivos como Palm, HP, Nokia y Microsoft se encontraban fuertemente posicionados en una industria prolífera que crecía a ritmo constante.
Sin embargo, en junio de 2007 el lanzamiento de un producto marcó un parteaguas dentro y fuera de la industria móvil, el iPhone.

Para entender, el tamaño del impacto en la figura 1.1 se analiza el EBIT (Earnings Before Interest and Taxes) en los años anteriores y posteriores al lanzamiento del iPhone 5.

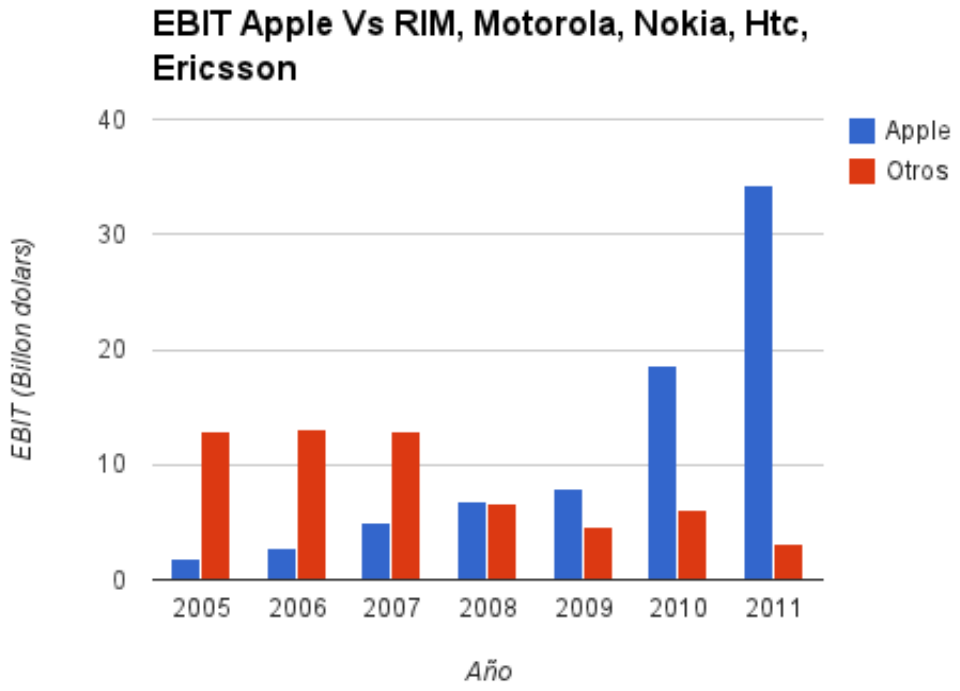


Figura 1.1
Gráfica del EBIT de Apple Vs Competidores, antes y después del lanzamiento del primer iPhone (Fuente: Wikinvest.com)

Se puede apreciar que Apple no solamente se posicionó como un actor importante en la industria, sino que alcanzó el liderazgo y en dos años por sí misma obtuvo un mejor posicionamiento que todos sus competidores juntos.

Pero, ¿cuáles fueron los factores tecnológicos que causaron la disrupción[3] en innovación del lanzamiento de iPhone?

- Pantalla Completa

La introducción de una pantalla que fuera del tamaño del mismo teléfono proporcionó el espacio necesario para un teléfono multitarea y capaz de mostrar mayor cantidad de contenidos. Esto contribuyó directamente a una nueva experiencia de exploración de la web desde un dispositivo móvil.

- Touch

Las pantallas sensibles al tacto no son un invento nuevo. Su origen data del año 1965, y su primera implementación en un teléfono celular en el año 1993 (IBM Simon[4]). Inclusive la inclusión de pantallas táctiles fueron una constante de las PDAs.
La contribución del iPhone en este rubro fue popularizar un teléfono en el que la interacción con la pantalla táctil fuera a través del dedo del usuario. Eliminando la necesidad de cualquier artefacto externo como un stylus e inclusive, haciendo prescindible la presencia de un teclado físico.

- Gestos

La posibilidad de utilizar el dedo del usuario, combinado a la capacidad de detectar múltiples puntos de touch al mismo tiempo, permitió el surgimiento de gestos para controlar un gran abanico de acciones de forma rápida e intuitiva utilizando únicamente los dedos. Esto supuso un gran cambio y mejora en la experiencia de usuario.

- Sensores

Desde la primera generación del iPhone es posible identificar la presencia de diversos sensores: acelerómetro de 3 ejes, sensor de proximidad, sensor de luz ambiental, GPS.

---

[3] Disrupción, palabra aún no reconocida por el Diccionario de la Lengua Española, pero utilizada frecuentemente en el argot tecnológico como traducción de la palabra en inglés disruption,
Diccionario Cambridge:
Disrupt: Evitar que algo, generalmente un sistema, proceso o evento continúe sucediendo de una forma usual o esperada.

[4] El IBM Simon es considerado por muchos el primer smartphone por sus características: 1MB de memoria, pantalla touchscreen en blanco y negro. Aplicaciones para manejo de email, fax, lista de contactos, calendario, notas a mano y capacidad de descargar o agregar nuevas aplicaciones vía tarjetas PCMCIA.

La inclusión de estos sensores, supuso un cambio radical en el potencial de crear innovadoras experiencia de uso.

- Aplicaciones desarrolladas por terceros

La intención inicial de Apple, era fomentar el uso de webapps, razón por la cual el desarrollo de aplicaciones por terceros no estuvo presente desde el lanzamiento del iPhone. Sin embargo en marzo de 2008, 9 meses despues del lanzamiento, fue liberado el primer SDK público para el desarrollo de apps iOS.

Es importante destacar, que ninguna de las características mencionadas fueron inventadas por Apple, sin embargo, su contribución fue la inclusión de todas ellas en un solo dispositivo y la sinergia resultante.

Plataformas Móviles

Panorama Actual

En la actualidad nos encontramos en un escenario con gran cantidad de participantes en la industria móvil y hablando particularmente de el ecosistema de los sistemas operativos, las opciones comerciales son diversas. Entre las principales se encuentran:

- iOS
- Android
- Windows Phone
- Blackberry

La evolución de cada una de estas plataformas ha sido por demás interesante, con altibajos, cambios abruptos y condenas al éxito y al fracaso en lo que parece ser apenas la gestación de un nuevo orden en el naciente ecosistema.

La siguiente gráfica, muestra la evolución de las principales plataformas móviles en los últimos años.
En ella se observa la decadencia de plataformas como el caso de Symbian y BlackBerry, la consolidación y competencia vigente entre Android y iOS y el intento de Windows Phone por ocupar un lugar del ecosistema.

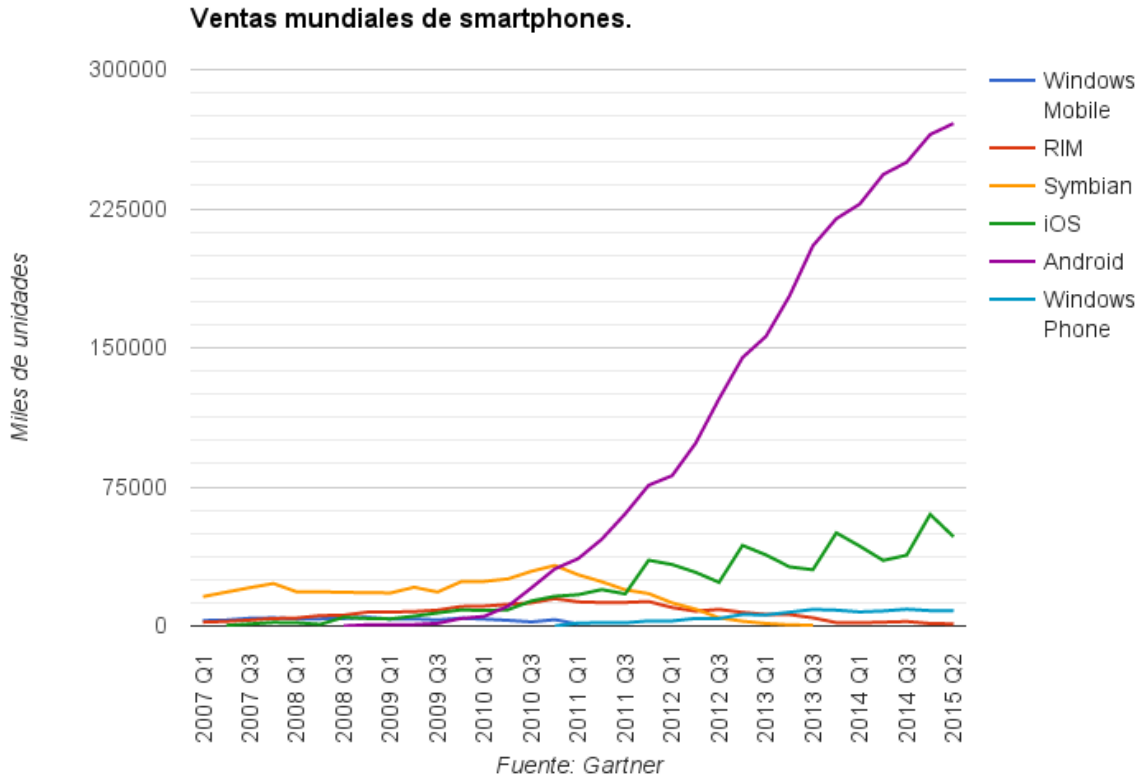


Figura 1.2

Comparación de sistemas operativos, según las ventas mundiales de smartphones, desde el lanzamiento del iPhone (Fuente:Gartner)

En esta otra gráfica se muestra la distribución actual de los diferentes OS móviles a nivel mundial.

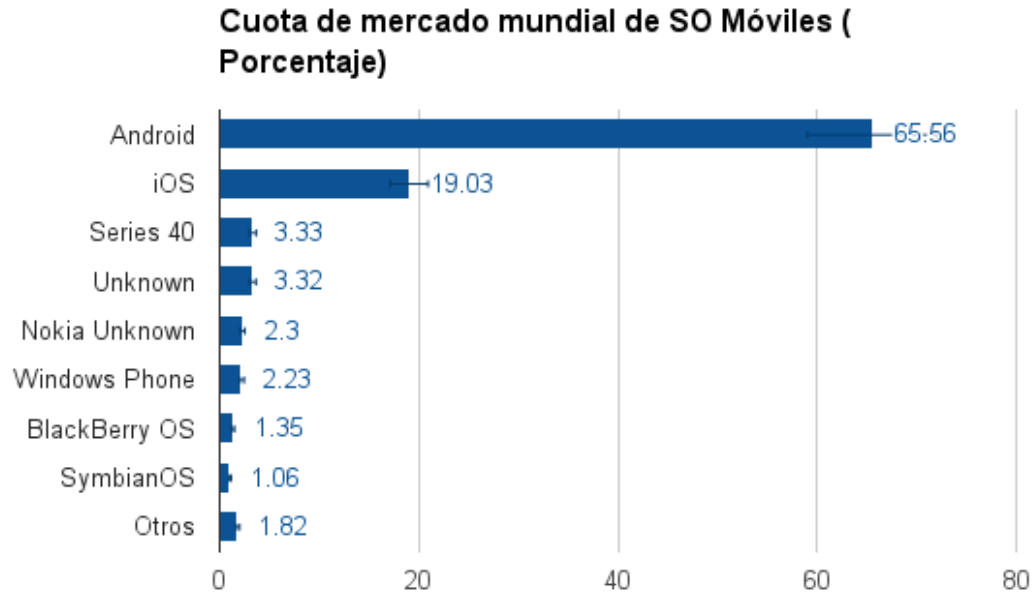


Figura 1.3

Cuota de mercado mundial de SO móviles (Fuente: Statscounter)

Cabe señalar que en el estudio[5] de los mercados seccionados por áreas geográficas podemos encontrar grandes diferencias en la preferencias de los usuarios. Por ejemplo, en el continente asiático nos encontramos con gran cantidad de feature phones[6] con SO Nokia S40 y Android, y muy poca presencia de iOS. Al contrario de EU, Francia, Japón, donde los usuarios iOS superan el 50% del mercado.

Estudio de las plataformas actuales

iOS

iOS es un sistema operativo móvil de apple originalmente desarrollado para el iPhone y posteriormente utilizado en otros dispositivos tales como el iPod touch, iPad y Apple TV.[7]

El SO fue llamado simplemente OS X desde el anuncio del lanzamiento del iPhone hasta el lanzamiento del iPhone SDK en marzo de 2008. A partir de entonces su nombre sería iPhone OS.

[5] Stat Counter Global Stats. Mobile Operating System Market Share, Agosto 2015

[6] Featurephone son aquellos teléfonos cuyos precios y prestaciones tecnológicas se sitúan en un rango medio, debajo de los smartphones.

[7] Apple unveils iPhone, MacWorld , 9 enero 2007

Este sistema operativo no permite su instalación en hardware de terceros. El desarrollo, mantenimiento y actualización depende exclusivamente de Apple. El desarrollo de aplicaciones por terceros estuvo disponible hasta un año después de su lanzamiento, en junio de 2008.
La interfaz de usuario se basa en el concepto de manipulación directa usando gestos multitáctiles como deslizamientos, toques, pellizcos etc.
iOS se deriva de Mac OS X que a su vez está basado en Darwin BSD por lo que es un sistema tipo UNIX.
Su arquitectura consta de 4 capas principales: el núcleo del SO, la capa de servicios principales, la capa de medios y la capa de Cocoa Touch .

Su versión actual es iOS 8.4 (Versión 9 en beta)

El desarrollo de aplicaciones para esta plataforma requiere el uso de un equipo Mac y el IDE de programación XCode. El desarrollo en la plataforma se basa en el patrón MVC y sus lenguajes nativos de desarrollo son Objective C y Swift.
Para poder realizar pruebas en dispositivos físicos y distribuir las aplicaciones desarrolladas es necesario suscribirse al iOS Developer Program cuyo costo anual es de $99 dólares americanos.

Android

Android es un SO móvil de código abierto actualmente desarrollado por Google basado en el núcleo de Linux.
El desarrollo inicial corrió a cargo de Android Inc. (fundada por Andy Rubin) y en un inicio el sistema operativo estaba diseñado para proveer aplicaciones que funcionaran en cámaras.
En 2005 Andorid Inc fue adquirida por Google y en noviembre de 2007 fue anunciado el lanzamiento del SO Android en conjunto con la OHA (Open Handset Alliance, un conjunto de 85 compañías de tecnología que incluyen operadores móviles, fabricantes de dispositivos y semiconductores y desarrolladoras de software).[8]

Aunque en su lanzamiento su uso fue enfocado en dispositivos móviles, en la actualidad el sistema operativo es ejecutado en gran variedad de dispositivos tales como electrodomésticos, televisores, relojes de mano, automóviles etc.

Consta de una interfaz de usuario basada en manipulación directa[9], por medio de gestos táctiles (deslizamientos, toques, pellizcos, etc.)

[8] Industry Leaders Announce Open Platform for Mobile Devices, Noviembre 5, 2007

[9] Manipulación directa es un estilo de interacción humano-computadora que envuelve la representación continua de los objetos de interés junto a acciones reversibles e incrementales [Kwon 2011 et al.]. A diferencia por ejemplo de

Es el sistema operativo más adoptado a nivel mundial y cuenta con 1.5 millónes de aplicaciones en su tienda oficial.

El desarrollo de aplicaciones para Android requiere el uso de Eclipse + plugin ADT o bien el IDE Android Studio ambos disponibles en Windows, OS X y GNU/Linux. El lenguaje utilizado es Java, y puede combinarse con C++.
El desarrollo en la plataforma es gratuito, sin embargo para la publicación de aplicaciones en la tienda es necesario un pago único de $25 USD.

Windows Phone

Es un SO móvil desarrollado por Microsoft sucesor del sistema Windows Mobile, lanzado en Octubre de 2010 bajo el nombre de Windows Phone 7.
A diferencia de su predecesor, Windows Phone (WP) está orientado al mercado de consumo y no al sector empresarial. Actualmente se cuenta con una cuota de mercado del 2.23%, debajo de iOS y Android).
Su interfaz de usuario está basada en el sistema Metro, un diseño basado en mosaicos que están ligados a aplicaciones, características o funciones.

Su versión actual es Windows Phone 8.1.

Las aplicaciones en WP son distribuidas desde la tienda Windows Phone Store (antes Windows Phone Market Place). Hasta marzo de 2015 la tienda contaba con más de 400 mil apps.

El desarrollo de aplicaciones requiere la instalación de Visual Studio y pueden utilizarse uno o varios lenguajes a escoger: JavaScript, C#, Visual Basic o C++.
El desarrollo de aplicaciones es gratuito. Para la publicación de aplicaciones en la tienda, es requerida una cuenta de desarrollador con un costo de 19 USD anuales (cuenta individual) y 99 USD anuales (cuenta empresarial)

una interacción a base de comandos, en manipulación directa los objetos virtuales se representan gráficamente y se manipulan en forma similar a sus contrapartes reales.

	android	iOS	Windows Phone	BlackBerry
Datos Generales				
Compañía	Google	Apple	Windows	Blackberry
Lanzamiento	Septiembre 2008	Junio 2007	Noviembre 2010	Versión smartphone 3.6, marzo 2002
Dispositivos vendidos a nivel mundial	226.1	51.0	8.8	1.7
Zonas geográficas con mayor uso actualmente.	Mayoría en casi todas las regiones.	US (51.7%), UK(44.8%), Francia (35.8%), Australia (59.3%), Japon(66.7%)	Brasil (4.48%), Rusia (3.4%), México(4.39%), Francia (4.1%)	UK(8.28%), US(0.34%), Indonesia (10.5%)
Última versión	5.1.1	8	8	10.2.1
Mercado				
Tienda de Aplicaciones	Google Play	App Store	Windows Phone Store	Blackberry World
Costo por desarrollo	Gratis	Gratis. Necesario registrarse en el developer program ($99USD/año) para realizar pruebas en dispositivo físico	Gratis	Gratis
Costo por publicación	Pago único de $25 USD.	Gratis. (Necesario estar suscrito al developer program)	Suscripción a AppHub $99 Dls/año	Gratis
Revisión de app	No	Si	Si	
Modelo de ventas	70-30%	70-30%	70-30%	70-30%
Total de aplicaciones	1.5 millones (Septiembre 2014)	1.4 millones (Enero 2015)	400+ mil (Marzo 2015)	223 mil (Octuble 2013)
Tecnología				
Kernel	Monolítico (Basado	Híbrido	Híbrido	

	en linux)	(XNU)		
Entorno y lenguajes nativos de programación	IDE: Eclipse, IntelliJ Java (Android SDK)	IDE: Xcode Objective-C	Visual Studio C#	IDE: Momentics (Apps nativas), WebWorks (Html5) Distintas combinaciones: C/C++/Qt, Javascript/CSS/HTML, ActionScript/AIR, Java(Android)
Soporte Multitasking	Si	Limitado	Si	Si
Reconocimiento de voz	Si (Online y offline)	Sí (online)	Sí (online)	Sí (online)

Figura 1.4

Tabla comparativa de los principales sistemas operativos móviles

Capítulo 2

Desarrollo de software para plataformas móviles

Las características que diferencian los sistemas cómputo móviles con los estacionarios, también tienen repercusiones en el campo del desarrollo de software. Esto ha provocado el surgimiento de técnicas y metodologías específicas de desarrollo de software orientadas a dispositivos móviles.

Es importante notar que no solamente las características y capacidades del hardware determinan los objetivos y procesos de desarrollo de software. Las plataformas móviles son un ejemplo perfecto de esta aseveración.
En ellas, juega un papel fundamental las condiciones del usuario móvil que por supuesto son muy distintas a las de un usuario estacionario. Reza (2005, p.22) identifica las siguientes diferencias entre el usuario estacionario y el usuario móvil:

- El usuario móvil está en movimiento, al menos ocasionalmente, entre lugares conocidos y desconocidos.
- El usuario móvil frecuentemente no está enfocado en la tarea de cómputo
- El usuario móvil, espera con frecuencia tiempos más bajos de respuesta
- El usuario móvil se encuentra menos enfocado en la tarea de cómputo. Suele ejecutar e intercambiar tareas de forma frecuente o abrupta
- El usuario móvil puede requerir acceso al sistema en cualquier lugar, en cualquier momento.

Características del software móvil

Una aplicación es aquella pieza de software que puede iniciarse y terminar individualmente, frecuentemente asociada a una interfaz gráfica y que realiza una cierta tarea (Mikkonen , 2007, p.61).

Como características generales de las aplicaciones, es posible identificar su independencia del resto del sistema, su capacidad de compartir datos con otras aplicaciones, y la gran cantidad de código que comparten unas con otras por el uso de APIs compartidas.

El software de aplicación[10] ha logrado establecer una relación de sinergia con los dispositivos móviles de los últimos años. Tal ha sido el éxito, que por una parte, las aplicaciones móviles (o también llamadas "apps") han contribuido a la revolución móvil, y el desarrollo de nuevas capacidades de hardware móvil han motivado el desarrollo acelerado y variado de aplicaciones.

Anteriormente hemos estudiado las diferencias del usuario móvil al usuario estacionario. Con el fin de planear y desarrollar aplicaciones móviles de calidad será necesario también tener en cuenta las diferencias hardware.

Un enfoque común para el desarrollo de aplicaciones móviles orientado en la usabilidad ha sido definido por Salmre (2005) :

- Ámbito

La importancia de considerar el propósito fundamental de la aplicación. Identificar tanto lo que hace, como lo que no hace. En dispositivos móviles, donde la atención del usuario es limitada, es particularmente necesario no saturar la cantidad de tareas que la aplicación pueda realizar.

- Desempeño

Como hemos mencionado, las necesidades del usuario móvil requieren tiempos de respuesta más alto. Es necesario incluir los mecanismos que faciliten este aspecto en el diseño de aplicaciones móviles.

- Diseño de interfáz de usuario

Los recursos para presentar información en móviles, son completamente distintos a las estaciones fijas.
Al presentar información en pantallas reducidas, frecuentemente es preferible optar por presentar información oportuna y reducida, que presentar grandes volúmenes de información en formatos innovadores o poco intuitivos.

- Modelo de datos y aspectos de memoria

[10] Según la típica clasificación de software, se identifican los siguientes: software de aplicación, software de sistema, software de programación.

Las restricciones en almacenamiento debe ser un factor a considerar en el desarrollo de aplicaciones móviles. Mayor memoria se relaciona con mayores costos y consumo de energía del equipo.
Las diferentes capas de abstracción (Modelo de Datos) también pueden ser restricciones en el desarrollo de aplicaciones. A mayor abstracción, es más fácil compartir código, la productividad aumenta y se facilita el acoplamiento entre piezas de software. Sin embargo, a mayores niveles de abstracción, suelen ocurrir impactos en transmisión, procesamiento y almacenamiento de datos, que se reflejan en alto consumo de energía, tiempos lentos de respuesta y costos elevados por conexión.

- Comunicaciones y entrada/salida de datos

La comunicación es una característica inherente a los sistemas móviles de cómputo. Debe entenderse por comunicaciones, todo lo que determina en qué forma el sistema obtiene información más allá de la que dispone. Sea aquella información interna (archivos y subsistemas) o externa (Bases de Datos remotas, sockets, Servicios Web, etc).

El correcto balance entre almacenar y/o procesar información con los recursos del dispositivo o bien, delegar dichas funciones a sistemas externos tendrá como objetivo maximizar la usabilidad.

Metodologías de desarrollo para dispositivos móviles

El ciclo de vida de una aplicación móvil es muy distinto al de los sistemas web tradicionales o aplicaciones para computadoras de escritorio.
En gran parte, como consecuencia de la aparición de las tiendas de aplicaciones y sus dinámicas de uso propuestas, nos encontramos con tiempos de uso sumamente reducidos e intermitentes así como dinámicas para obtener, instalar, y en su caso desechar aplicaciones a un ritmo acelerado. Las actualizaciones del software móvil ocurren también con mucha mayor frecuencia que en los sistemas tradicionales.

La gran competencia, también favorece el movimiento veloz de usuarios de una aplicación a otra de características similares, si es que la primera no llegara a satisfacer por completo sus necesidades o sus expectativas de calidad. Las tiendas de aplicaciones móviles brindan la posibilidad de testing en fases alpha y beta, y funcionan como un medio veloz y eficiente para recolectar el feedback de los usuarios finales de la aplicación.

Por estas características (de alto cambio y gran cantidad de retroalimentación) del entorno en el que coexisten las aplicaciones móviles, las metodologías que más se adaptan para el desarrollo de ellas son las llamadas *metodologías ágiles*.

Desarrollo Ágil

La ingeniería de software ágil, combina una filosofía con un conjunto de recomendaciones para el desarrollo. Dichas recomendaciones están orientadas, sobre todo, a la satisfacción del cliente, tolerancia al cambio, entregas prontas e incrementales del producto de software y a equipos de desarrollo reducidos y con mucha comunicación (Pressman , 2014, p. 66).

En febrero de 2001, de una reunión meramente recreativa, de un conjunto de 17 personas (consultores y desarrolladores de software representantes de Extreme Programming, SCRUM, DSDM entre otros) surgió el "Manifiesto por el Desarrollo Ágil de Software", como una alternativa a los pesados procesos de desarrollo, dirigidos a la documentación, más que al software en sí.

El Manifiesto propone:

"***Individuos e interacciones*** sobre procesos y herramientas
Software funcionando sobre documentación extensiva
Colaboración con el cliente sobre negociación contractual
Respuesta ante el cambio sobre seguir un plan
Esto es, aunque valoramos los elementos de la derecha, valoramos más los de la izquierda." [11]

Un proceso de desarrollo Ágil, se caracteriza según la manera en que aborde un número de suposiciones clave de la mayoría de los productos de software. (Pressman, 2014, p. 69).

- Es difícil predecir de antemano qué requerimientos del software persistirán y cuáles cambiarán. Es también difícil predecir cómo cambiarán las prioridades del cliente durante el curso del proyecto.
- Para muchos tipos de software el diseño y la construcción se traslapan. Para el beneficio de ambas actividades, es convenientes desarrollarlas en paralelo.
- El análisis, diseño y construcción no son tan predecibles como se desearía.

La principal característica de los procesos ágiles es su capacidad de manejar lo impredecible. La adaptabilidad, por tanto, es clave en dichos procesos.
Sin embargo, la adaptabilidad por sí misma no conduce a resultados, los cambios deben seguir un orden y estar siempre dirigidos a mejorar el producto de software de las entregas previas. Por ello los procesos ágiles deben ser adaptables en forma incremental.

[11] Manifiesto por el Desarrollo Ágil de Software
http://agilemanifesto.org/

Partiendo de estos principios, los procesos ágiles consisten de entregas de software mejorado en periodos cortos de tiempo. El cliente juega un papel crucial en estos procesos, ya que es el principal encargado de proveer la retroalimentación que hace de cada prototipo o pieza de software mejor de su versión anterior. Por esta razón la presencia del cliente es crucial no sólo durante la planeación, sino a lo largo de todo el desarrollo del proyecto.
La iteratividad de estas actividades, permite tanto al cliente como al desarrollador visualizar claramente si el producto de software avanza en la dirección correcta y realizar las correcciones necesarias prontas y a tiempo.

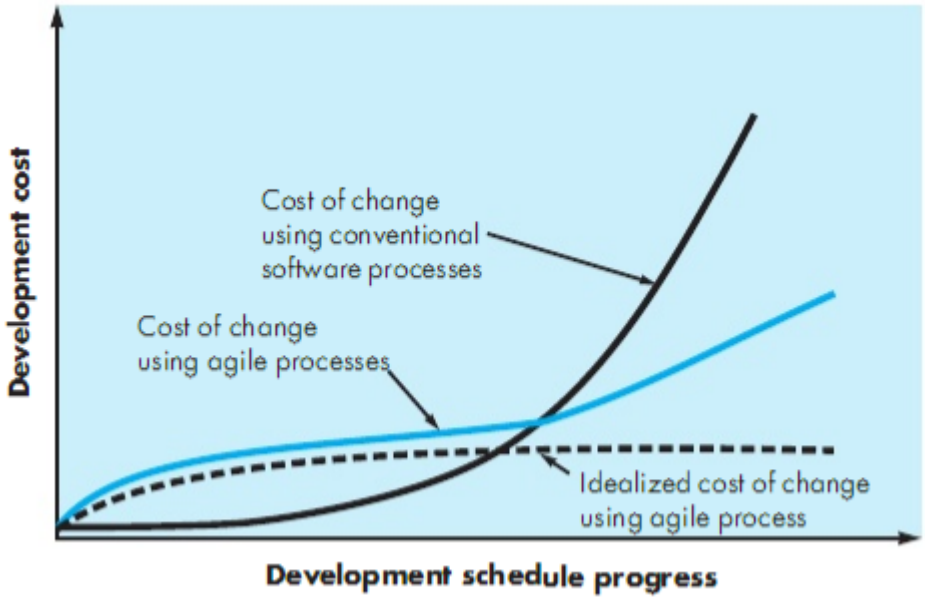


Figura 2.1
Gráfica de costo del cambio, en función del tiempo de desarrollo

A continuación se estudian las características de **Extreme Programming** y **SCRUM**, dos de las metodología ágiles más utilizadas en la actualidad.

XP (Extreme Programming)

Extreme Programming (Pressman p. 72) es una metodología de desarrollo de software creada por Kent Beck (Firmante del manifiesto ágil) y publicada en octubre de 1999 en un libro bajo el título *Extreme Programming Explained.*

XP, engloba un conjunto de reglas y prácticas que ocurren dentro del contexto de las 4 actividades del proceso de desarrollo de software: Planeación, Diseño, Desarrollo y Pruebas.

Durante la planeación, se escucha cuáles son las necesidades del cliente, y qué soluciones considera éste apropiadas. De las necesidades del cliente, se procede a encapsularlas en distintas historias de usuario, que se plasman por escrito en tarjetas, una tarjeta por cada historia de usuario.
Al inicio, las historias de usuario presentan altos niveles de abstracción, sin embargo, en el transcurso del proyecto es posible agregar nuevas, refinarlas o descartarlas.
Para cada historia de usuario, el cliente asigna una prioridad y el equipo de desarrollo realiza estimaciones del tiempo para su desarrollo. Si este tiempo sobrepasara las 3 semanas, se requiere que el cliente divida la historia de uso en 2 o más.

Una vez explorados los requerimientos y teniéndolos plasmados en historias de usuario (y con la conciencia que es prácticamente un hecho que cambiarán en el futuro), el cliente junto con los desarrolladores planifican qué historias de usuario se implementarán en la siguiente iteración, basado en su prioridad y costo.
Después de la primera iteración y para todas las siguientes, se calcula la velocidad del proyecto. En términos simples, esta es la medida que determina cuántas historias de usuario se implementan por ciclo. Este indicador ayudará a estimar fechas de entrega y a programar las tareas para las próximas iteraciones.

El diseño en XP sigue el principio de KIS (Keep it simple). XP plantea elegir un diseño sencillo sobre uno complejo. A diferencia de otras metodologías que proponen diseñar con la escalabilidad en mente, en XP se prefiere que el diseño se enfoque a resolver los requerimientos actuales, sin contemplar el diseño de funcionalidad extra a la especificada en las historias de usuario. Se prefiere un refactor del software cuando surja algún cambio en los requerimientos, que diseñar siempre tratando de solucionar o prevenir problemas que a menudo no se presentarán.

Las fases de pruebas y desarrollo en XP también presenta características especiales. Antes de iniciar el desarrollo de software, se elaboran los test unitarios basados en las historias de usuario que se implementarán en la iteración. Así el desarrollo se enfocará en lo que se debe ser implementado para aprobar los tests y acota el desarrollo únicamente a lo requerido e indispensable.

Otra particularidad en XP es el desarrollo en parejas. Se propone que dos desarrolladores trabajen en una misma estación de trabajo al mismo tiempo a fin de mejorar la calidad del software.

Los roles en XP son (Priolo, 2009):

- Cliente
- Programador
- Encargado de pruebas (Tester)

- Encargado de seguimiento (Tracker)
- Entrenador (Coach)
- Gestor(Big Boss)

Es importante notar que estos roles pueden o no son exclusivos de una sola persona. Así una misma persona puede jugar uno o más roles mientras su experiencia y disponibilidad así lo permita.

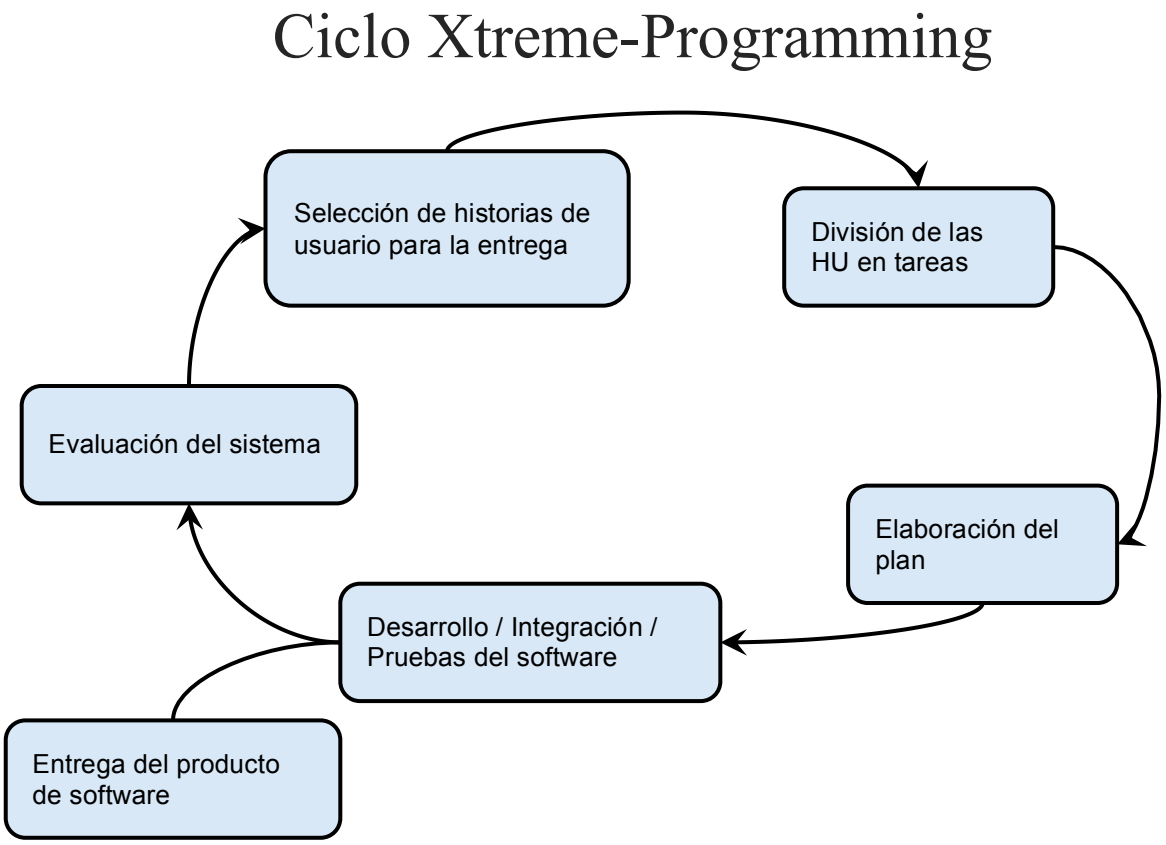


Figura 2.2
El ciclo de entrega XP (Sommerville, 2011)

SCRUM

SCRUM es un framework (marco de trabajo) de desarrollo ágil de software creado por Jeff Sutherland a principios de los 90s.

Más que un método que defina procesos técnicos del desarrollo de software es un método ágil para la administración de proyectos. Por ello puede convivir con otras metodologías tales como XP.

Hay 3 fases en SCRUM:
En la primera se definen los objetivos generales del proyecto y se diseña la arquitectura del software. La segunda etapa (y la parte innovadora de SCRUM) es el ciclo de sprints iterativos en donde se desarrolla progresivamente el producto y por último la fase de entrega en donde se elaboran los manuales de uso, la documentación del proyecto y se inspeccionan las lecciones aprendidas.

El Backlog:
Es una lista ordenada por prioridades de los requerimientos o características que brindan valor de negocio al cliente.

Sprints:
Unidades de trabajo en las que se evalúa las tareas pendientes por realizar, se seleccionan las funciones a desarrollar y se procede a su implementación. La duración de un sprint es fija y puede durar de 2 a 4 semanas.
Las tareas y funcionalidades a realizar durante un sprint corresponden con las del backlog y su selección está a cargo de todos los miembros del equipo en conjunto con el cliente.

Los equipos en SCRUM son autoorganizados, es decir que una vez que se ha acordado cuál será el trabajo a realizar en el sprint, el equipo decide libremente cómo lo hará.

SCRUM propone juntas diarias de corta duración (15 minutos) al inicio del día con todos los miembros del equipo. Tres preguntas clave son contestadas por cada miembro del equipo:
¿Qué has hecho desde la última junta grupal?
¿Con qué obstáculos te has encontrado?
¿Qué logros planeas alcanzar para la próxima junta?

Estas juntas ayudan a evaluar la evolución del trabajo y poder identificar la presencia o probable aparición de problemas en fases tempranas.

Durante el sprint, en la fase de desarrollo, el equipo es aislado del cliente y la organización con toda la comunicación encausada vía el Scrum Master. El rol de esta persona es proteger al equipo de distracciones externas, da seguimiento al backlog, cuantifica el progreso y realiza la comunicación con el cliente y la administración interna de la empresa.

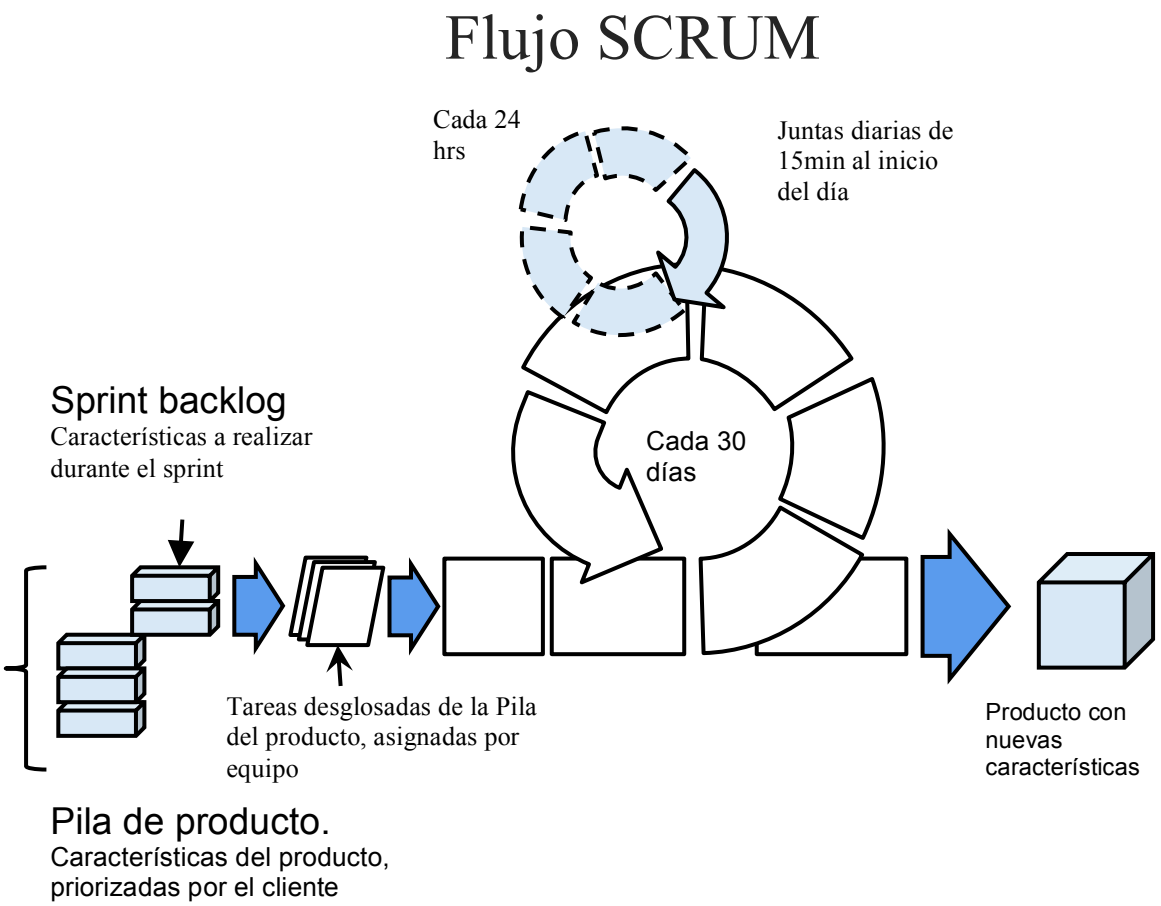


Figura 2.3
Scrum, flujo del proceso (Pressman, 2014)

Mobile-D

Es una metodología para el desarrollo ágil de software móvil desarrollada por el VTT (Centro de investigación técnica de Finlandia). Además de su aplicación en el ámbito de desarrollo de software, es capaz de ser implementada en diversos contextos tales como seguridad, finanzas, logísticas y simulación de producto.

La metodología se basa en otras soluciones probadas y consolidadas.
Extreme Programming (XP) para las prácticas de desarrollo, Crystal Methodologies para escalabilidad y RUP como base en el diseño del ciclo de vida.

La metodología consta de 5 fases:

1. Exploración.
   El propósito de la primera fase es la planeación y establecimiento del proyecto. En ella se sientan las bases para ejecutar una implementación controlada del software. Se define el alcance del proyecto, fechas de entrega, roles y el ambiente de desarrollo.

2. Iniciación (o iteración 0)
   Cuyo objetivo es asegurar el cumpliemiento de las siguientes fases. En esta fase, se asegura contar con los recursos tecnológicos, físicos y humanos que el proyecto requiere.

3. Producción
   En donde se lleva a cabo la implementación del producto de software.
   Se repiten iterativamente las subfases con días de planificación, días de trabajo y días de entrega.

   Los días de planificación se enfocan a mejorar el desarrollo. Priorizando y analizando los requerimientos, estableciendo el contenido de la iteración y creando los tests que habrá de cumplir el software desarrollado.

   Los días de trabajo en los que se implementan las funcionalidades usando Test Driven Development[12]

   Los días de entrega cuyos objetivos son asegurar la integración de los distintos componentes, verificar que el software cumple con los requerimientos previamente plasmados en pruebas (Acceptance Testing).

4. Estabilización
   Si la complejidad del proyecto ha requerido que el sistema sea subdividido en subsistemas generados por distintos equipos, en esta etapa se realiza la integración en un solo producto.

5. Prueba y ajustes del sistema
   Cuyo propósito es validar el sistema basado en la documentación producida a lo largo del proyecto y entregar un sistema con la menor cantidad de errores como sea posible.

[12] Test Driven Development (TDD) o Desarrollo dirigido a pruebas indica que antes de realizar una funcionalidad debe existir una prueba que verifique su funcionamiento.

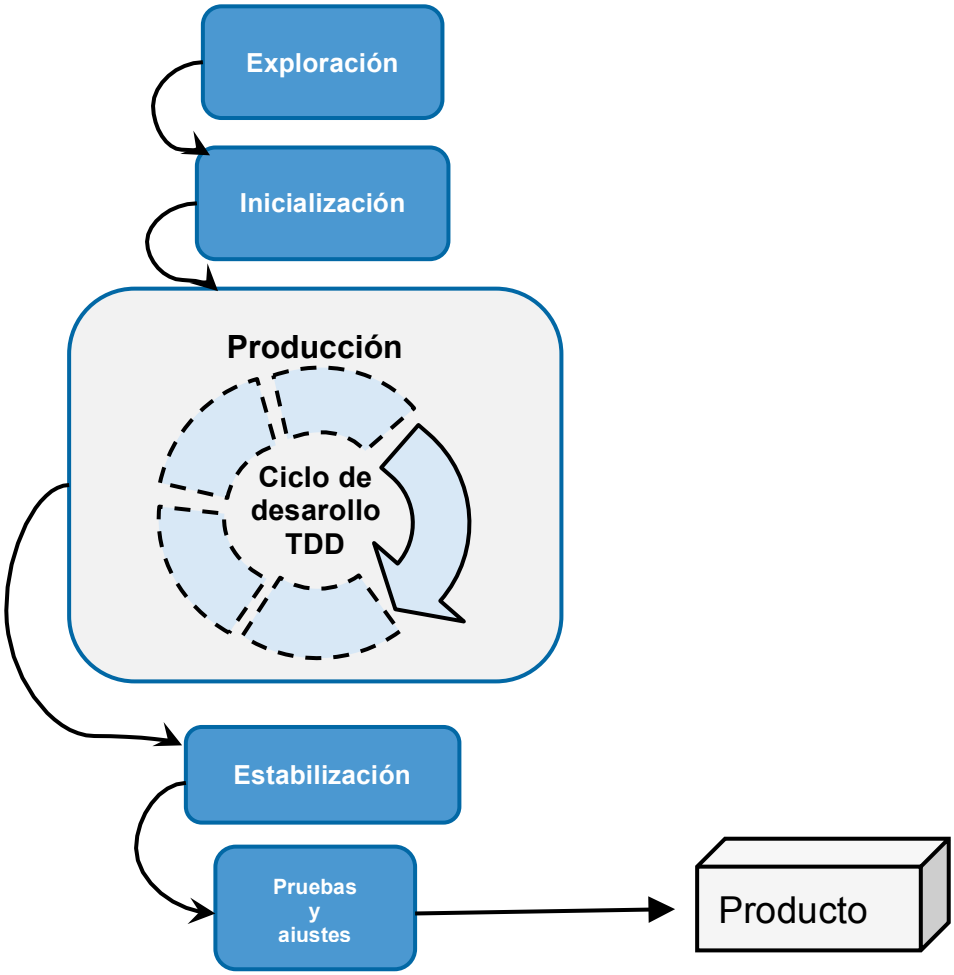


Figura 2.4
Mobile-D, flujo del proceso
(Software Technologies Research Programe, VTT,
http://agile.vtt.fi/mobiled.html)

Elección de una plataforma móvil

Aspectos a considerar en la elección de una plataforma

- Tecnología disponible

A menudo, las características tecnológicas son equivalentes y/o compatibles entre las diversas plataformas. Sin embargo, existen diferenciadores técnicos que pueden determinar la elección o el descarte de alguna(s) plataforma(s).

Por ejemplo, el lenguaje de programación y el volumen y claridad de la documentación existente varía entre plataformas. Es importante considerar estos 2 factores que impactan directamente en el tiempo de desarrollo y en la curva de aprendizaje.

Dentro de las condiciones tecnológicas de una plataforma es posible identificar aquellas que son dependientes de la plataforma en sí misma (como el lenguaje de programación, la arquitectura del SO) y aquellas que derivan indirectamente de la plataforma (por ejemplo, el software o herramientas de terceros disponibles en cierta plataforma).

- Target de Mercado y Tipo de consumidores por plataforma

Las características del mercado también son determinantes en la elección de plataforma, y a menudo es necesario investigar variables más allá de la cantidad de usuarios por plataforma. Los patrones de comportamiento y de compras del usuario, predominancia geográfica y tipo de teléfonos de cada una de las plataformas son ejemplos de dichas variables. El modelo de negocios de la aplicación o del producto asociado a la misma determina aquellas características que son relevantes. Es posible identificar distintas necesidades según el tipo de aplicación. (aplicación gratuita, de difusión/promoción, de servicios, de paga, in-app purchase, empresarial, etc.)

- Tiendas de aplicaciones

Debido a que los principales canales de difusión (y en ciertas plataformas los únicos) son las tiendas de aplicaciones, también es pertinente estudiar las características de las mismas.

  - Esquema de negocios (ganancias del desarrollador por cada venta y en compras in-app)
  - Competencia
  - Necesidades del negocio

- Cuota de mercado en el target.

Tecnología disponible para el desarrollo en la plataforma.
Esquema de negocio de la plataforma (revisión de apps, Integración con necesidades del negocio, porcentaje de ganancias, competencia en el app store), porcentaje para el desarrollador, comportamiento de los clientes de cierta plataforma, competencia de aplicaciones en el store, integración con las necesidades del negocio (apps internas).

Soluciones Multiplataforma[13]

En gran cantidad de casos luego del estudio de las plataformas móviles y de las características de la aplicación a desarrollar, se concluye que su target no es limitado a una sola plataforma, sino a más de una.

En esos casos, existe la posibilidad de optar el uso de tecnologías de desarrollo multiplataforma. La filosofía sobre la cual se basan es: "Desarrolla una vez, distribuye en varias plataformas".

A diferencia del enfoque tradicional (nativo) de desarrollo de apps, el desarrollo multiplataforma permite implementar una aplicación con un solo código base que puede ser ejecutado en más de una plataforma. Los lenguajes nativos varían entre plataformas, por ejemplo Objective-C para iOS, Java para android y C# para WP.
Usando una solución multiplataforma en el desarrollo de una app se utiliza el mismo lenguaje y entorno de programación para todas las plataformas objetivo.
Si bien, el objetivo de desarrollo multiplataforma en móviles es uno, los enfoques con los que se alcanza éste son variados. (Heitkötter et. al. 2013) identifica los siguientes:

Webapps

Este enfoque aprovecha la estandarización (no siempre perfecta) de tecnologías Web, para crear sitios diseñados para ser visualizados apropiadamente en dispositivos móviles.
Entre las ventajas de este enfoque, se encuentra desde luego la nula curva de aprendizaje para aquellos que poseen conocimientos previos web.
Mientras que en las desventajas se encuentra principalmente la inaccesibilidad o accesibilidad reducida a recursos del dispositivo tales como sensores, cámara, GPS, almacenamiento en archivos, contactos entre otros.

Híbridas

El enfoque híbrido surge de la combinación de las tecnologías web (HTML5, javascript, CSS), con la funcionalidad nativa. Las aplicaciones híbridas empaquetan todo su código fuente, junto con un engine que es el encargado de renderizar las páginas en un WebView normal. Dicho engine, además, provee las librerías y el puente necesario con el sistema operativo para agregar funcionalidades nativas, inaccesibles en el enfoque Web.
Cabe resaltar que en este enfoque, a pesar de resolverse el acceso a funciones específicas de cada plataforma, el look & feel sigue asemejándose al de una webapp.
El principal exponente de este enfoque es PhoneGap/Cordova.

[13] *Crossplatform / Multiplatform*: La presencia de un mismo software en distintas plataformas. El funcionamiento y "look and feel" del software puede no ser completamente idéntico entre las plataformas.

Entornos de ejecución autocontenidos (Self-contained Runtime Environments)

Este tipo de aplicaciones no utilizan WebViews para desplegar el contenido. Por el contrario, implementan su propio runtime que renderiza las interfaces gráficas. La propia implementación de su ambiente de ejecución permite que los elementos gráficos sean o bien nativos o customizados.

Encontramos herramientas como Titanium Appcelerator cuyo runtime es capáz de crear elementos gráficos nativos al vuelo, proporcionando el look & feel nativo para cada plataforma. También encontramos herramientas como Corona, para el desarrollo de videojuegos, cuyo runtime crea elementos gráficos customizados optimizados para alto desempeño.

Capítulo 3
Realidad aumentada

Realidad Aumentada es la tecnología que presenta un mundo virtual (generado por computadora) que enriquece, más que reemplazar, el mundo real (Feiner, Macintyre,Seligmann , 1993).
El objetivo de esta tecnología es presentar información pertinente en situaciones reales que sería difícil (o imposible) percibir con únicamente los sentidos del ser humano.

La Realidad Aumentada es frecuentemente ligada con Realidad Virtual. Sin embargo hay que aclarar que existen diferencias entre ellas.

La Realidad Virtual se refiere a "Ambientes inmersivos, interactivos, multisensoriales, centrados en vista, tridimensionales generados por computadora, y la combinación de tecnologías requeridas para la construcción de dichos ambientes." (Cruz-Neira, 1995)

Se distingue que la Realidad Virtual[14] pretende la inmersión completa del usuario y la interacción con elementos únicamente virtuales, mientras que la Realidad Aumentada proporciona una inmersión "a medias" que permita interactuar tanto con elementos reales como con elementos virtuales que se comportan en sincronía con el mundo real.

Azuma (1997) identifica tres características claves de sistemas de Realidad Aumentada[15]:

- Mezclar imágenes virtuales con el mundo real.
- Registro de información 3D.
- Interacción en tiempo real.

[14] Experiencia Sintética, Mundos Virtuales, Mundos Artificiales, Realidad Artificial, Ambientes Virtuales son términos que a menudo se usan de forma indistinta.
[15] Dichas características fueron cumplidas por primera vez en 1968.
Sutherland I., A head-mounted three-dimensional display. In Proc. of the Fall Joint Computer Conference. AFIPS Conference Proceedings, vol. 33. AFIPS, Arlington, VA., 757–764, 1968

Esas características corresponden a los componentes de los sistemas de RA mencionados por Silva (2003).
Un ***sistema de Tracking***, responsable crear la ilusión de que ambos mundos (virtual y real) coexisten en uno solo.
Los sensores son el hardware encargado de obtener información del mundo real. Por medio de sensores, el sistema es capaz de determinar la posición y orientación del usuario y características del ambiente como temperatura y luminosidad.
Obtener la mayor cantidad de información posible, tiene como fin poder lograr la fusión realista de los mundos real y virtual. A menudo la calidad de los sensores, determinarán la calidad de la experiencia de RA.

Los procesadores son componentes de hardware fundamentales de los sistemas de RA. Ellos, reciben la información de los sensores para implementar las "leyes naturales" y otras reglas del mundo virtual, y para generar las señales requeridas a ser mostradas en el display. (Craig, 2013 , p.40). Deficiencias en el procesamiento se traducen en altas latencias, desalineación entre el mundo real y el virtual y finalmente en experiencias de uso deficientes..

Un ***generador de escena***, capaz de generar objetos virtuales. Éstos pueden ser realistas o no, dependiendo de las necesidades de la aplicación de RA. Esta tarea es también responsabilidad de los procesadores (en muchos casos se cuenta con procesadores de gráficos especializados en esta tarea).

Un ***display*** que combine y despliegue la realidad aumentada resultante. En la actualidad es posible encontrar distintos tipos de displays: cascos, 'head-mounted' displays, monitores/pantallas, gafas, por mencionar algunos.

Evolución

Los inicios en el desarrollo de la tecnología se remontan a la década de los 60s. En 1962 Morton Heilig (considerado el padre de la realidad Virtual) patenta ***Sensorama*** una máquina que proyectaba 5 cortos cinematográficos que enriquecía la experiencia por medio de video estereoscópico, vibraciones y olores.

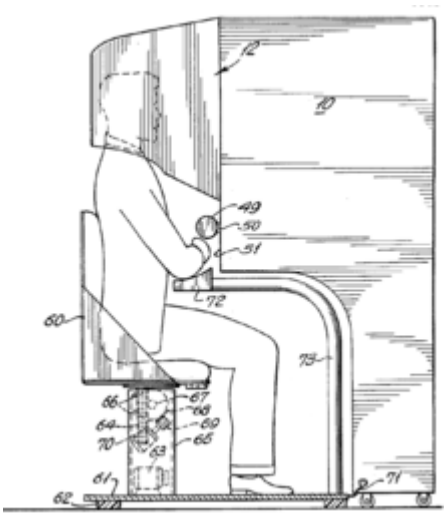


Figura 3.1
Sensorama

En 1968 Ivan Sutherland desarrolla el primer HMD (Head Mounted Display). Los HMD son dispositivos de visualización atados a la cabeza o bien, que forman parte de un casco, que pueden desplegar imágenes para uno u ambos ojos.

El HDM inventado por Sutherland constaba de un visor óptico transparente capaz de renderizar wireframes en tiempo real y que estaba suspendido y anclado al techo para realizar el tracking del usuario.

Éste es considerado el primer sistema de Realidad Aumentada (Schmalstieg, Langlotz, Billinghurst, 2008, p.13).

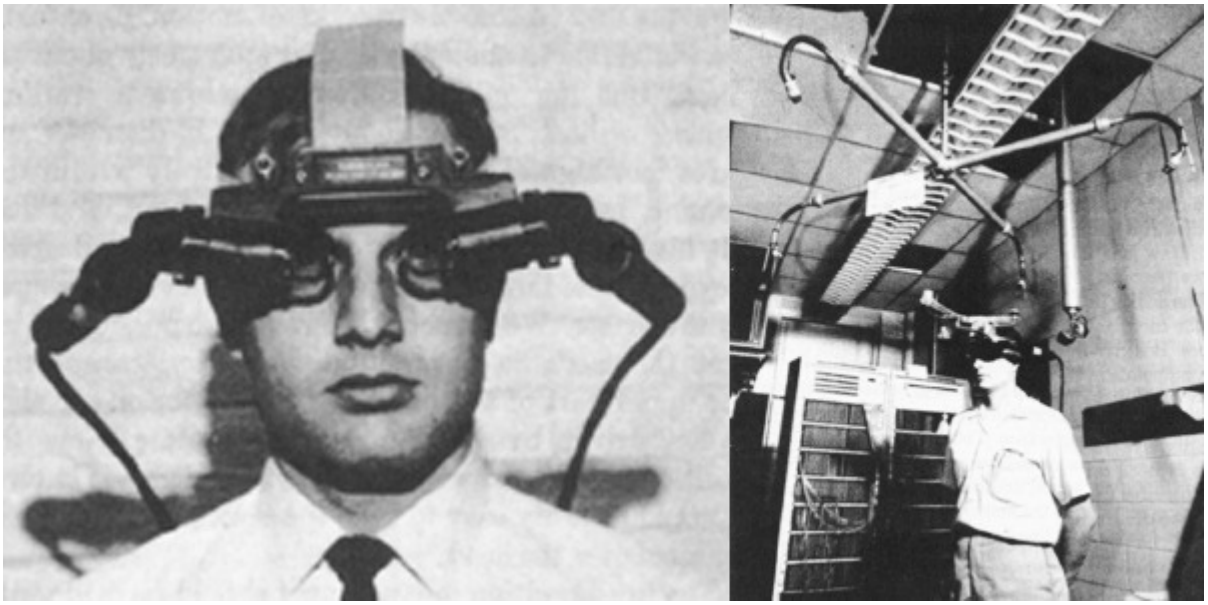


Figura 3.2
Primer head monted display (HMD) desarrollado por Ivan Sutherland

El término Realidad Aumentada fue adoptado en 1992 por Thomas Preston Caudell, de la empresa aeronáutica Boeing, para describir un sistema que ayudaría a los trabajadores en el ensamble e instalación de cables eléctricos en aeronaves. (Caudell & Mizell, 1992).

En 2008, (Schmalstieg et. al. ,2008) distinguen una nueva era en el campo de la RA, cuya gestación se remonta a principios de la década de los 2000. La denominan Realidad Aumentada 2.0, y se caracteriza por:

- Nuevos equipos, portátiles, más accesibles y más ergonómicos que los primeros HMDs.
- Usos más allá del ensamblaje industrial, entrenamiento médico y militar.
- Experiencias de RA, con gran presencia en dispositivos móviles.
- Despliegue y disponibilidad de aplicaciones de RA a escala global y para cientos de miles de personas al mismo tiempo.
- Los usuarios visualizan, y también crean y actualizan contenidos de RA.
- Las aplicaciones de RA son autónomas y pueden descargar nuevos módulos para extender sus capacidades.

En forma análoga a la WEB 2.0, la Realidad Aumentada 2.0 provee experiencias que fomentan la creatividad, colaboración, comunicación, intercambio de información, y contenidos generados por el usuario. El presente trabajo de tesis pertenece completamente a esta nueva generación de Realidad Aumentada.

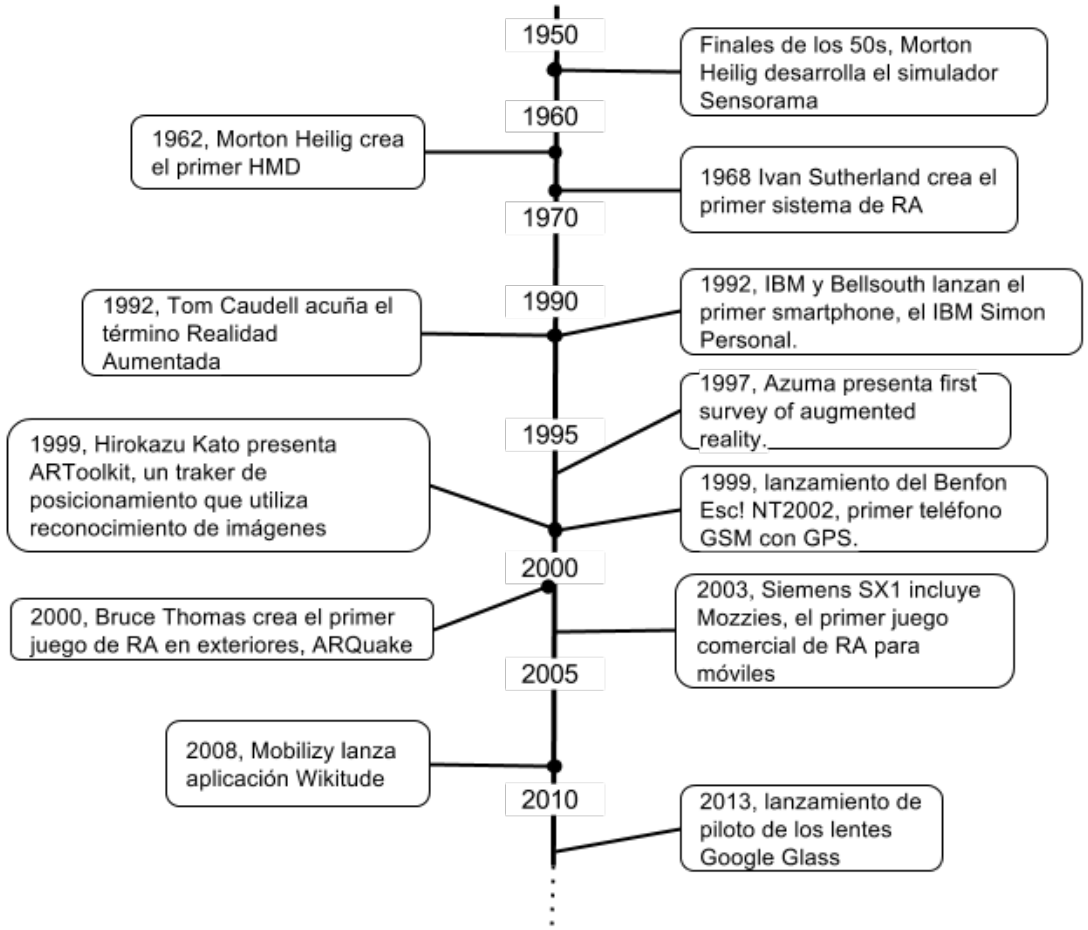


Figura 3.3
Historia de la RA. Breve línea de tiempo. (Yuen, 2011)

Aplicaciones

La Realidad Aumentada ha conseguido posicionarse en un gran número de ámbitos de diversas áreas de conocimiento más allá de la milicia o academia, comunes y limitados en sus inicios.

Dentro los principales se destacan:

Medicina

La imagenología es uno de los campos con mayor acción tecnológica dentro de la medicina. La Realidad Aumentada tiene presencia en esta campo en cirugías guiadas por imagen.

Los estudios preoperatorios como tomografías computarizadas (CT), o escaneados con resonancias magnéticas (MRI) brindan la información necesaria para conocer la anatomía interna y planear los procedimientos quirúrgicos.

Por medio de Realidad Aumentada, es posible empalmar la información obtenida de las CT o MRI en el proceso quirúrgico.

También, existen aplicaciones en el área de la psicología, para el tratamiento de fobias. Consisten principalmente de la inmersion del paciente en un ambiente virtual o semi virtual, en donde podrá encarar sus fobias. Este tipo de tratamientos ventajas como: el terapeuta estará presente en cada sesión, él mismo podra controlar las variables del ambiente y la experiencia virtual suele ser menos agresiva que la experiencia real.

Figura 3.4
Ejemplos de aplicaciones de RA en medicina

Entretenimiento

Las características de visualización 3d, e interactividad han contribuido a la gran aceptación tecnología de RA en la industria del entretenimiento. Dentro de esta industria existen aplicaciones en televisión[16], videojuegos , juguetes entre otros.

[16] PVI Virtual Media Services antes Princeton Electronic Billboard es la empresa propiedad de ESPN pionera de la inserción de contenidos virtuales en tv durante la década de los 90s.

Educación

Dado el desarrollo reciente, y la inherente característica de evolución de la interfaz de uso que supone el uso de RA, los investigadores suponen que existen un gran número de beneficios resultado de la 'aumentación' o enriquecimiento de los ambientes de enseñanza y aprendizaje. (Yuen, S.; Yaoyuneyong, G.;& Johnson, E. ,2011, p.126)

La realidad Aumentada tiene el potencial de atraer, estimular y motivar a estudiantes a explorar materiales de clase desde distintos ángulos, ayuda en la enseñanza de temas en los que los estudiantes difícilmente podrían tener experiencias de primera mano (como en astronomía, paleontología, geografía, etc.), fomenta la creatividad y la imaginación, permite a los estudiantes aprender a su propio ritmo y en sus propias rutas, entre muchos otros beneficios (Yuen, S. et al. ,2011, p.127).

Publicidad

Las marcas siempre están en constante búsqueda por llamar la atención de sus clientes potenciales. Por ello deben estar al día en cuanto a los nuevos y novedosos canales de comunicación que han surgido en los últimos años. La RA aumentada no ha pasado desapercibida, y quizás la publicidad y mercadotecnia sean las áreas que más la han explotado comercialmente. Grandes empresas como Coca-Cola, Starbucks, Pepsi, Lego, IKEA, Henz, entre muchas otras han realizado campañas publicitarias basadas en Realidad Aumentada.

Entrenamiento

El uso de RA para entrenamiento presenta muchas ventajas. Puede brindar experiencias de capacitación en las que el usuario no se encontrará aislado en una cabina, sino en un ambiente semi-real. Las ventajas de capacitarse en un ambiente semi-real, son que el usuario podrá comprender la relación de los componentes virtuales con componentes reales, resultando en una capacitación más completa, además de una transición más 'suave' al ambiente real donde se aplicarán los conocimientos adquiridos.

Manufactura

El uso de RA en procesos de manufactura, mantenimiento y reparación fue uno de los primeros enfoques en ser explorado. Empresas como Boeing y Mitsubishi han aplicado tecnología de RA para capacitar a sus técnicos en el ensamble de aviones y de sistemas HVAC, respectivamente.

En este campo, la RA probado ser una solución eficiente y de bajo costo (Caudell & Mizell, 1992).

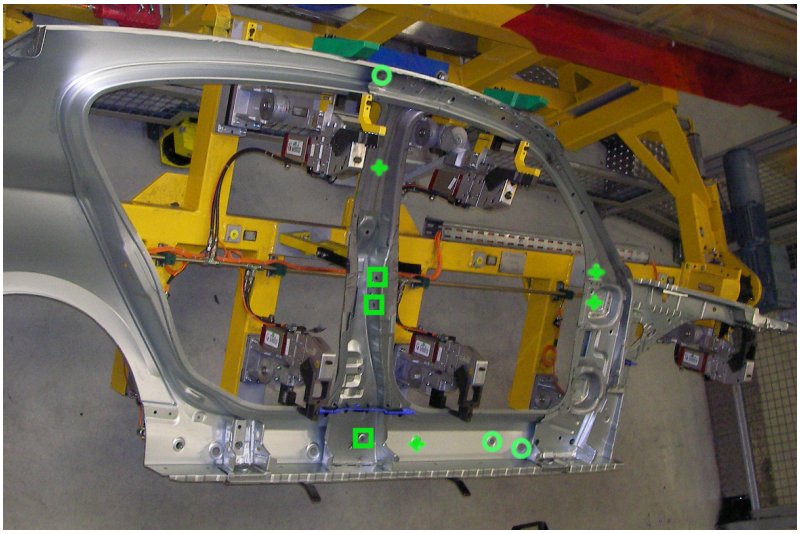


Figura 3.5
Ejemplo de aplicación de RA en manufactura.
Asistencia en línea de ensamblaje

Clasificación de sistemas de RA

De acuerdo con la tecnología utilizada para desplegar contenidos, es posible clasificar los sistemas de RA en 4 tipos (Silva et al., 2003): ópticos transparentes, sistemas retinales virtuales, video transparentes, basados en monitor (o proyector).

Los dispositivos ópticos transparentes son dispositivos montados sobre la cabeza (HMD head mounted display) que posicionan combinadores ópticos frente a los ojos del usuario. Estos combinadores son parcialmente transparentes y parcialmente reflectivos, de forma que el usuario puede ver tanto el mundo real como imágenes virtuales en ellos.

Compañías como Google, Sony, Olympus han desarrollado productos comerciales de este tipo de sistemas. Los lentes Google Glass de lanzamiento reciente son ejemplos de estos sistemas.

Los sistemas retinales virtuales, en lugar de proyectar sus contenidos en una superficie, lo hacen con rayo de luz modulado directo hacia la retina. El resultado, es la percepción de imágenes inexistentes como si estuvieran frente al usuario.

Los sistemas de video transparentes, combinan la información capturada por medio con cámaras con imágenes virtuales para elaborar una composición que se muestra al usuario. En este tipo de sistemas opacos las imágenes del entorno no son vistas directamente, sino como resultado de las

imágenes capturadas por una cámara. La proyección de las imágenes combinadas puede realizarse tanto en lienzos opacos como directamente a la retina.

Los sistemas de RA basados en monitor al igual que los sistemas de video transparentes despliegan la combinación de video con imágenes virtuales en pantallas. Estas pueden ser televisores, computadoras o dispositivos móviles como smartphones y tablets.
Debido al relativo bajo costo y proliferación reciente de dispositivos móviles y pantallas, además de la eliminación de las cuestiones relacionadas a los HMD este enfoque se ha posicionado como el preferido para brindar tecnología comercial y a gran escala de RA.

La clasificación de los sistemas de RA puede ser no solamente desde el punto de vista del despliegue de la información. Es posible clasificar estos sistemas desde otros puntos de vista tales como el tracking y el entorno de uso.

De acuerdo al punto de vista del tracking, es posible distinguir sistemas de RA de tracking natural (aquellos que no necesitan de elementos externos para realizar el tracking) y de tracking artificial (aquellos que necesitan la presencia de elementos adicionales al ambiente para realizar el tracking, tales como patrones de imágenes o marcadores).

Según el entorno de uso de los sistemas de RA, es posible clasificarlos también en aquellos de uso para interiores, para exteriores y para ambos.

Realidad aumentada y dispositivos móviles

Como se ha mencionado antes, la RA ha tenido un gran auge debido a la proliferación y mejora de dispositivos móviles durante la última década. Además, se espera un crecimiento aún mayor con el lanzamiento de productos de tecnología móvil más allá de smartphones y tablets como las Google Glass.

Para entender el gran impulso que los dispositivos móviles han brindado al desarrollo de la RA recordemos los componentes básicos de hardware de un sistema de RA: procesadores, sensores, sistema de tracking y display.
Todos ellos están integrados en piezas únicas de hardware y mejorando constantemente en los nuevos dispositivos móviles sean tablets o smartphones. La presencia de dichos componentes embebidos, hacen de la tarea de desarrollar un sistema de RA, una tarea meramente de software con las facilidades que ello implica.

Enfoques de Realidad Aumentada en Dispositivos Móviles

Es posible clasificar las aplicaciones de RA móviles dependiendo la tecnología que utilizan.

Geolocalización

Este tipo de aplicaciones realizan el tracking de la posición del usuario vía GPS y despliegan información pertinente según los sitios reales que se encuentran cercanos al usuario. Este enfoque es muy utilizado para RA en exteriores, es posible determinar hacia qué dirección está enfocando el usuario (por medio de magnetómetros) y sobreponer elementos virtuales encima de imágenes capturadas con la cámara.

Podemos mencionar como ejemplos Wikitude, Layar, Unam 360.

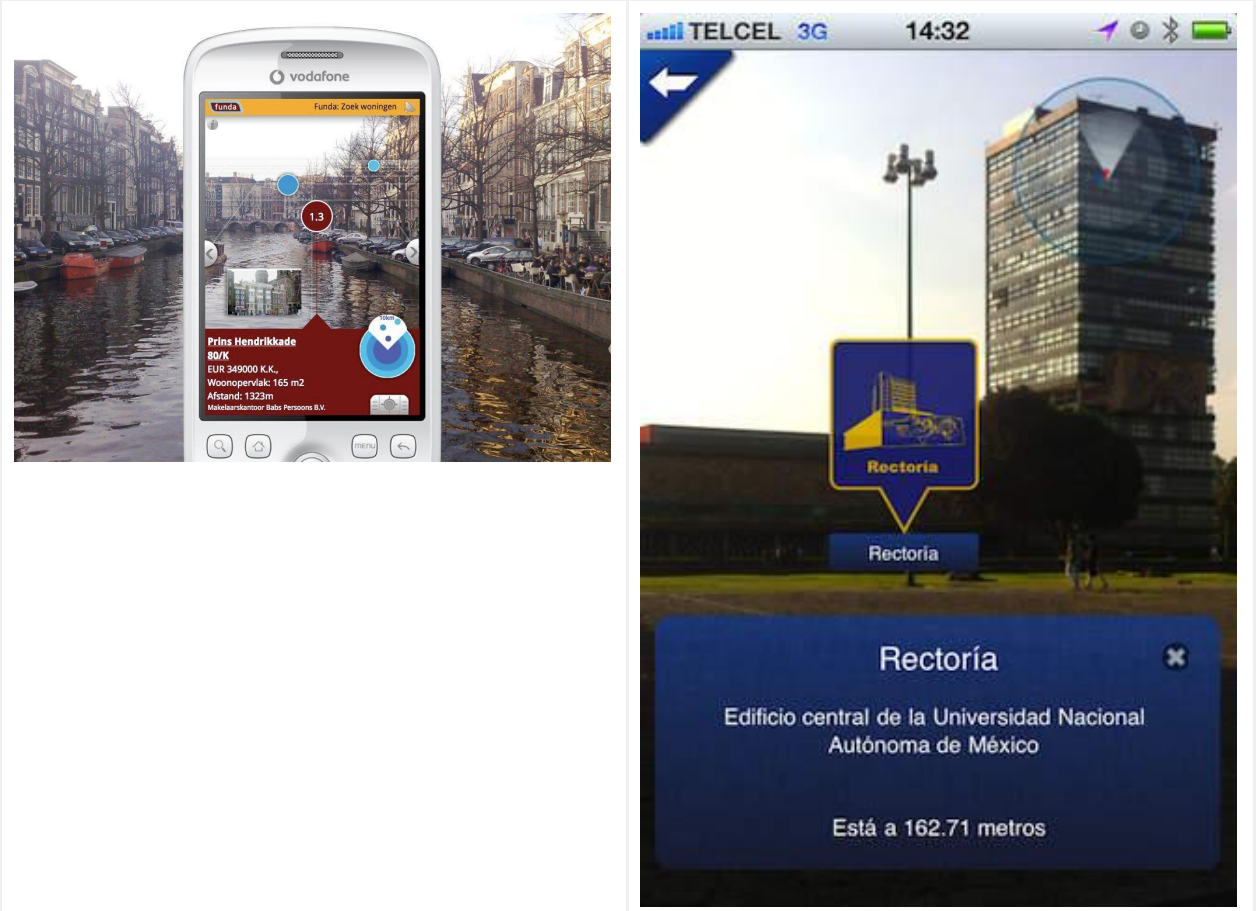


Figura 3.6

Ejemplos de aplicaciones móviles de RA usando geolocalización

Reconocimiento de patrones (Imágenes)

Este tipo de aplicaciones se enfoca en realizar el tracking de la posición del usuario (estrictamente hablando es tracking del dispositivo) por medio del reconocimiento de patrones de imágenes. Usando esta técnica, se calcula la posición relativa de la cámara respecto a la imagen que se está reconociendo.

Conociendo esa posición, es posible colocar contenidos virtuales y desplegarlos sobre las imágenes capturadas por la cámara. Si se realiza este procedimiento por cada imagen, dará la impresión de que los objetos virtuales se encuentran en el mundo real.

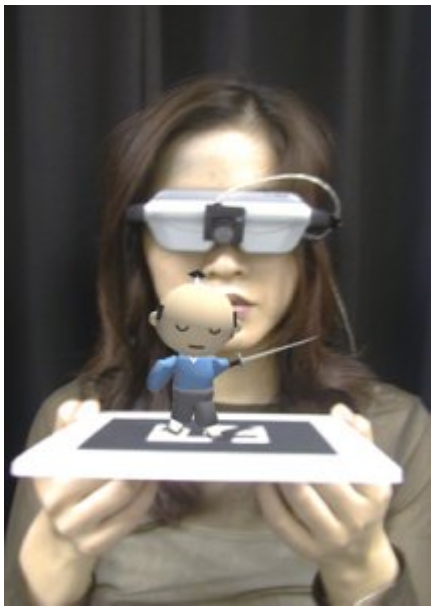


Figura 3.7

Ejemplos de aplicaciones de RA usando reconocimiento de patrones

Los patrones a detectar son imágenes, que habitualmente suelen ser códigos QR, rostros, paisajes, pinturas o cualquier gráfico con patrones extraíbles.

Los contenidos virtuales a desplegar bien pueden ser Imágenes, Objetos 3D o videos.

Este enfoque sólo requiere de una cámara y el procesamiento para la detección de patrones, por lo que no es exclusivo de los dispositivos móviles y puede ser aplicado en secuencias de video o con webcams. Fue propuesto por primera vez por Kato (Kato & Billinghurst, 1999) quien también creó ARToolkit, una librería libre para el tracking de imágenes.

Diversas compañías han encontrado aplicaciones comerciales para esta tecnología. (Vuforia de Qualcomm, Metaio, Layar, Aurasma)

Orientación

Un tercer tipo de tecnología, es el uso de sensores para determinar la orientación del usuario respecto a la superficie terrestre. (Giroscopios, Acelerómetros, Detectores de Campo Magnético), de tal forma que la posición de la información virtual depende de la dirección a la que se encuentre viendo el usuario (o apuntando con el dispositivo).
Diversos HMDs contaban con esta capacidad de orientación, por lo que tampoco es un enfoque exclusivo de los dispositivos móviles.

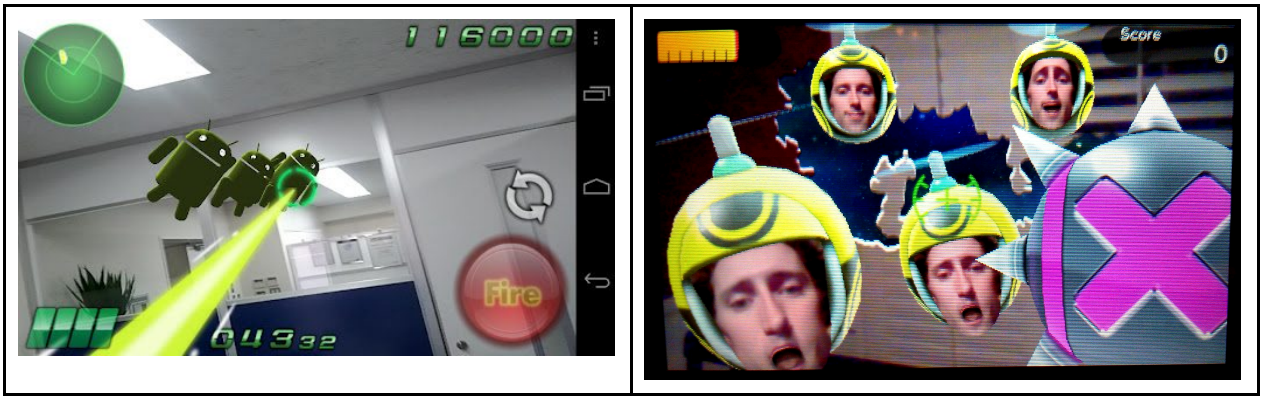


Figura 3.8

Ejemplos de aplicaciones de RA usando sensores de orientación

Es importante aclarar que estas 3 tecnologías no son excluyentes, y la combinación de ellas es una posibilidad para el enriquecimiento de la experiencia de uso.

Enfoque	Sensores requeridos
Geolocalización	GPS
Reconocimiento de patrones	Cámara
Orientación	Cámara, Giroscopio, Acelerómetro, Magnetómetro (brújula)

Figura 3.9

Tabla de sensores requeridos en los diversos enfoques de RA

Herramientas para aplicaciones de Realidad Aumentada

En la actualidad hay un gran abanico de herramientas para el desarrollo de aplicaciones de Realidad Aumentada en móviles. La variedad es tal, que inclusive existen herramientas que no requieren conocimientos y/o habilidades previas de desarrollo.

Exploradores de RA

De forma análoga a la que un usuario navega en internet por medio de un explorador, interpretando archivos en textos, páginas web o contenidos multimedia, en la actualidad existen herramientas que permiten explorar el mundo real en busca de contenidos virtuales para ser desplegados con tecnología de Realidad Aumentada.

Layar es un ejemplo de este tipo de herramientas. Una solución end-to-end para empresas interesadas en elaborar campañas utilizando la tecnología de RA. Provee tanto un editor de contenidos como una aplicación móvil multiplataforma para explorar contenidos de campañas de RA.

Las empresas mediante la aplicación Web de Layar, suben sus contenidos virtuales como imágenes, videos, botones, etc. a la nube, y los asocian con marcadores o imágenes que detonan el despliegue de dichos contenidos. Los usuarios de la aplicación móvil de Layar, saben que una imagen es explorable cuando incluye el ícono distintivo de Layar.

Este, no es el único esquema de funcionamiento de Layar.
Como un explorador de contenidos de RA, inclusive va más allá del enfoque de reconocimiento de patrones pues también es capaz de reconocer la posición del usuario y mostrar etiquetas virtuales acordes a su posición y orientación (enfoque de RA con Geolocalización).

Junaio (de Metaio), Aurasma, Wikitude son otros ejemplos de Exploradores de Realidad Aumentada.

Colaborativas

La exploración de ambientes reales en busca de contenidos de RA involucra al menos 2 actores: Un usuario generador de contenidos y un usuario explorador o consumidor de contenidos.

Si bien este es el esquema habitual, no es el único posible. Si a dicho esquema, se brinda la posibilidad de que el consumidor de contenidos sea también capaz de generarlos, da lugar una experiencia de uso nueva y probablemente enriquecida con mayor potencial de interacciones sociales.

MixAR, creada por Hololabs es un ejemplo de herramienta colaborativa, que facilita la creación de experiencias de RA sin necesidad de conocimientos en programación. Su enfoque es una sola aplicación, que además de servir para visualizar contenidos de RA, permite al usuario crear sus propios contenidos, como imágenes o modelos 3D a partir de fotografías tomadas con el smartphone.

Herramientas para desarrolladores

Evidentemente la flexibilidad en el diseño y capacidades de una aplicación aumentan cuando el desarrollo corre por cuenta propia. A este nivel, existen diversas herramientas para desarrolladores, que aprovechan las habilidades y conocimientos del programador móvil para crear apps de RA, sin que éste tenga necesariamente conocimientos o experiencia previa en el desarrollo de software de Realidad Aumentada.

ARToolkit (Kato & Billinghurst, 1999) es una librería open source para el desarrollo de aplicaciones de RA. Utiliza el reconocimiento de patrones para calcular la posición y orientación de la cámara, relativas a un marcador físico, en tiempo real. Con esa información, es posible desplegar modelos 3D virtuales que aparentan coexistir con elementos del mundo real. La primera versión de esta librería fue desarrollada en el año 1999. Las características de los dispositivos móviles de aquella época distaban mucho de las mínimas requeridas para realizar el procesamiento de imágenes. La configuración utilizada en aquel experimento fueron unas gafas HMD i-O modificadas con una cámara a color añadida, conectadas a una estación Unix SGI O2 que realizaba el procesamiento, conectada nuevamente hacia el display HMD para mostrar las imágenes procesadas, a una velocidad de 7 a 10 fps.
Hoy en día, los elementos de esa configuración, se encuentran embebidos en la mayoría de los smartphones comerciales y existen ports (adaptaciones) móviles de la librería.

Para desarrolladores móviles, existen diversas herramientas, con distintos enfoques y grados técnicos de profundidad que suelen ayudar en tareas como:

- Extracción de patrones de imágenes (para definir marcadores propios probablemente más discretos o acordes a la identidad gráfica y necesidades de la aplicación)
- Almacenamiento y recuperación de patrones de marcadores en la nube.
- Integración multiplataforma.
- Reconocimiento de marcadores y ubicación relativa de la cámara.
- Render de imágenes, videos y/o objetos 3D sobre marcadores.

Wikitude SDK y Vuforia SDK de Qualcomm son ejemplos de este tipo de herramientas.

Herramientas para no programadores

DAQRI, es un ejemplo de dichas herramientas. Mediante DAQRI 4D Studio,es posible crear aplicaciones de Realidad Aumentada sin necesidad de programación. Sólo se indican los marcadores que se reconocerán con la cámara y se asignan contenidos multimedia (imágenes, videos, modelos 3D) que se sobreponen al detectar los marcadores. Todo a través de una interfaz gráfica intuitiva.

Metaio Creator (de Metaio) es otra herramienta para no programadores con características similares.

Ejemplos de aplicaciones en Dispositivos Móviles

Las posibilidades de aplicación de la tecnología de Realidad Aumentada son muy extensas, y a pesar de su corto tiempo de inclusión en mercado de los dispositivos móviles, ya es posible constatar la gran variedad de nichos en los que ha influido:

Arquitectura

Las aplicaciones de la RA en Arquitectura son variadas.
Existen aplicaciones como *ARki* o *VisuAR*, que son capaces de proyectar en la pantalla de la tablet o smartphone un modelo arquitectónico 3D cuando se detecta un target. Ese target puede ser el mismo plano de la construcción y así aumentar la información previa. (plano 2D a modelo 3D).
Otro enfoque de uso de la tecnología de RA en arquitectura, es el presentado por la aplicación MagicPlan[17]. En esta aplicación, se generan planos de una habitación sin más herramientas que una tablet o smartphone.

Educación

El área de la educación, quizás sea de las más beneficiadas en cuanto al desarrollo de productos de Realidad Aumentada. Destacan aplicaciones como Libros con RA (o comúnmente llamados 'Libros Mágicos') con contenidos 3D accesibles utilizando una webcam o la cámara de un dispositivo móvil.
Una práctica común llevada a cabo por los educadores, es el uso de juegos didácticos para facilitar la asimilación de conceptos. En este ámbito, la RA brinda la oportunidad de utilizar nuevas formas de aprendizaje altamente visuales e interactivas.
El aprendizaje basado en descubrimientos es otro ámbito en donde distintas aplicaciones de RA ya están haciendo aportaciones. Utilizando el GPS, las aplicaciones fomentan la exploración al

[17] El enfoque de operación para esta aplicación, comparte muchos principios con el propuesto para el presente trabajo de tesis.

mostrar información relevante de un sitio de interés, como monumentos o sitios históricos, cuando el usuario se encuentre cerca de ellos.

Publicidad

Las marcas están en todas partes, e interactuamos con ellas cotidianamente. Las reconocemos con facilidad gracias a la identidad visual que las caracteriza, y gracias a ella, también penetran y permanecen más fácilmente en nuestra mente.

El uso de realidad aumentada, ha demostrado que es capaz de proyectar la funcionalidad de la identidad gráfica de una marca más allá de sus métodos habituales. Con ella se ha logrado escalar en la atención del usuario, agregando un nuevo componente: la interactividad.

Hoy en día es posible, reconocer la identidad gráfica de una marca y agregar contenido interactivo tales como gráficos, juegos, videos, promociones entre otras muchas posibilidades.

Figura 3.10
Ejemplo de aplicación móvil de publicidad con RA

Arte

En el campo del arte, la RA ofrece nuevas experiencias al espectador. En donde la vista, o lo visible (paisajes, murales, pinturas, instalaciones), es solo una parte de la historia que se cuenta. Con ayuda de aplicaciones de RA, se enriquecen las obras de arte, agregando contenido virtual y en muchos casos, también la posibilidad de interacción.

Los museos, no sólo de arte, también han comenzado a adoptar la tecnología de RA como un medio de presentar contenidos y recorridos guiados multimedia y narrativa interactiva.

Algunos ejemplos:
Natural History Museum, London (en alianza con Samsung), desarrollaron el juego "A Gift for Athena" diseñado para atraer a niños de 7 a 11 años a la galería del Partenón.

Royal Ontario Museum, con la app ROM Ultimate Dinosaurs, que hace que los dinosaurios extintos hace más de 65 millones de años, "cobren vida".

American Museum of Natural History, que cuentan con un catálogo de 10 aplicaciones móviles, entre ellas una de RA para la exhibición de Beyond The Planet Earth: The future of Space Exploration.

Videojuegos

Los mecanismos de interacción del usuario con un videojuego son un factor de suma importancia y un componente esencial de la experiencia de usuario. En este sentido, la RA ha creado o mejorado la forma en que el usuario envía o recibe información a un videojuego.

La agregación de elementos virtuales en un ambiente real ofrece el potencial de crear escenarios de videojuegos ligados con la realidad.

Más allá de la visualización, las características de hardware de la mayoría de los dispositivos actuales (procesadores multinúcleo, acelerómetros, giroscopios, sensores de luz, etc.) han fomentado la innovación en los mecanismos de interacción del usuario con el videojuego.

Decoración de interiores

La utilidad de sobreponer elementos virtuales sobre entornos reales no está limitada a objetos ficticios. De hecho, la RA es de gran utilidad cuando se quiere simular la presencia de objetos potencialmente reales.

Como ejemplos, podemos mencionar:
IKEA: El catálogo de IKEA, para posicionar todo tipo de muebles utilizando marcadores. (desarrollado por Metaio).

Mitsubishi Electric encontró que el 70% de sus compradores potenciales optaban por la instalación de sistemas HVAC (heating, ventilation, air conditioning) de ductos por encima de

sus unidades eléctricas más eficientes para pared. Cuando los contratistas preguntaban a sus clientes la razón, la respuesta común era que no sabían cómo se verían ya colocados en sus casas. Como solución, crearon una aplicación de RA que permitía simular por medio de marcadores, como se verían los sistemas HVAC una vez colocados.

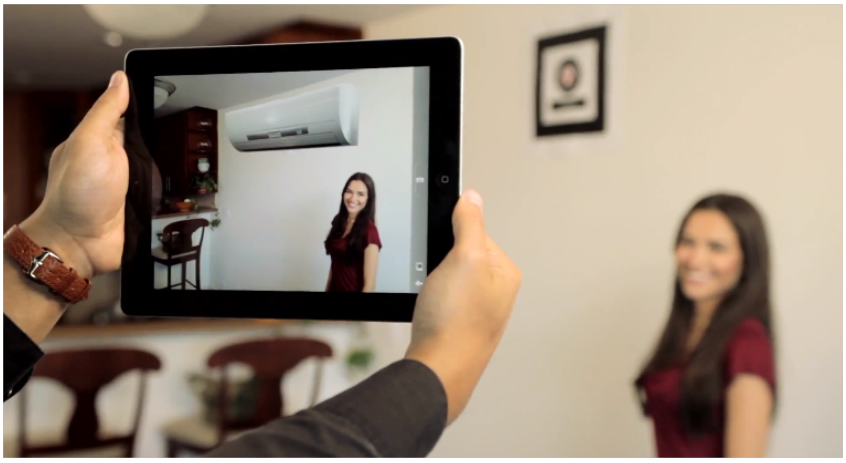


Figura 3.11
Ejemplo de aplicación móvil de publicidad con RA, Mitsubishi MeView.

Estudio de los enfoques actuales de RA en interiores mediante el uso de dispositivos móviles

La localización del usuario para mostrar contenido pertinente de Realidad Aumentada ha sido una de las principales técnicas para la elaboración de aplicaciones móviles de RA.

La funcionalidad de dichas aplicaciones recae en el análisis de la información recibida por el GPS adicionada con la información obtenida de sensores de orientación. Usando estos datos es posible por ejemplo, indicar lugares de interés como museos, restaurantes, lugares turísticos cercanos al usuario y datos de utilidad como la distancia, horarios de servicio, etc.

Sin embargo, estas aplicaciones son orientadas a su uso en exteriores en donde no afectan las limitaciones del GPS que se mencionan a continuación.

Precisión
La precisión de un GPS depende de diversos factores tales como:
Campo despejado sin obstrucciones (edificios, árboles, cables, etc.), condiciones atmosféricas y la calidad del receptor GPS.

Si bien, la mayoría de los fabricantes ubican precisión en un rango de 3 a 15 metros el 95% del tiempo, en la actualidad los dispositivos comerciales de alta precisión son capaces de ofrecer lecturas con error horizontal menor a 2.199 m el 95% del tiempo[18].

Alcance/Penetración

Las señales de los satélites GPS se ubican en las frecuencias de 1.57542 GHz (L1) y 1.2276 GHz (L2) y requieren "visión" directa con los receptores. No son capaces de penetrar agua, aceite o paredes de concreto. Estos factores, dificultan la obtención de información en entornos rodeados de árboles o edificios.

Estudio de las soluciones existentes

Ante las limitantes de localización en interiores descritas han surgido diversas tecnologías que buscan resolver el problema.

WiFi

Como se ha mencionado, la tecnología GPS obtiene la posición del usuario utilizando el método de trilateración conociendo la posición relativa del mismo respecto a 4 satélites.

En el caso de interiores, suele ser común la necesidad de ubicar al usuario, dentro de ambientes que cuentan con una gran infraestructura de red, tales como centros comerciales y aeropuertos.

Dentro de estos ambientes, es común la presencia de redes WiFi, con múltiples puntos de acceso (hotspots) colocados de forma distribuida. La idea central, radica en ubicar al usuario, según la intensidad, ángulo y tiempo de las las señales recibidas de los hotspots del espacio interior. Es decir, en lugar de utilizar satélites, se utilizan puntos de acceso para trilaterar y obtener su posición.

Dentro de los inconvenientes de esta tecnología destacan la necesidad de la existencia de planos arquitectónicos (para ubicar previamente los hotspots) y la presencia de una red WiFi configurada correctamente según las necesidades de éste procedimiento.

Bajo el mismo enfoque, también es posible utilizar otras tecnologías inalámbricas como bluetooth.

[18] Información obtenida de la Federal Aviation Administration (FAA)

Beacons

Utilizando piezas de hardware autónomas que implementan comunicación Bluetooth 4.0 (Bluetooth Smart o Bluetooth Low Energy, BLE),es posible detectar cuando un usuario (con receptor BLE) se encuentra cerca de alguna con rangos de precisión en cm.
Las aplicaciones de esta tecnología son variadas, sobre todo en centros comerciales. Desde detectar cuando un cliente entra y sale de establecimientos hasta notificar promociones o brindar información adicional de productos de manera oportuna cuando se esté cerca de ellos.

Aunque el desarrollo de esta tecnología se remonta al año 2006, por la empresa NOKIA bajo el nombre de Wibree, aún se espera su auge en el corto plazo. Distintos factores pueden ser decisivos en el éxito de esta tecnología, destacando 2 principalmente: Las mejoras tecnológicas de Bluetooth Smart respecto a versiones anteriores y otras tecnologías similares y la inclusión de emisores y receptores BLE en los dispositivos de última generación[19].

Las principales desventajas de utilizar beacons como método de localización en interiores son:

- La limitada información proporcionada. Es posible detectar la distancia del usuario al beacon, pero no su posición relativa al mismo, ni las características del ambiente.
- La necesidad de adquirir y colocar beacons en espacios interiores. Generalmente son objetos ajenos y tal vez contrarios a las características naturales del entorno.

Reconocimiento de patrones

Uno de los enfoques más populares en el desarrollo de RA, indistintamente en interiores y exteriores, es el de reconocimiento de patrones de imágenes 2D. En éste, no se conoce la posición u orientación absoluta del usuario, sino la relativa a un objeto real llamado marcador.

Por medio de la cámara del dispositivo móvil, se detecta un patrón de imagen en el marcador. Analizando la perspectiva desde la cual es visto por la cámara, es posible estimar la posición relativa del usuario al marcador aplicando transformaciones gráficas en orden inverso. Conocer la posición relativa del usuario al marcador, permite dibujar objetos virtuales en la posición y perspectiva correctas (Siempre en relación con el marcador).

Este enfoque es particularmente ha sido particularmente exitoso, ya que hoy en día practicamente cualquier dispositivo móvil cuenta con una cámara y capacidad de procesamiento suficiente requeridos para su ejecución.

[19] El iPhone 4S fue el primer dispositivo comercial en incorporar Bluetooth 4.0
En la actualidad los principales sistemas operativos móviles (iOS7, Android 4.3+, WP y Blackberry), implementan la especificación de Bluetooth 4.0

Entre las desventajas de esta técnica se encuentran:

- Es necesario un proceso previo que extraiga y procese los patrones de cada marcador.
- Habitualmente se utilizan tantos marcadores como objetos virtuales se necesiten.
- La variación en las condiciones ambientales como la iluminación o presencia de múltiples marcadores en una escena, pueden mermar la capacidad de detección de patrones.
- Cuando la cámara pierde de vista el patrón reconocible, también se pierde por completo la ubicación del usuario relativa a éste.

SLAM (Simultaneous localization and mapping)

SLAM es un problema enfrentado comunmente en el campo de la robótica que consiste en la capacidad de un robot de construir o actualizar el mapa de un ambiente desconocido, al mismo tiempo que se tiene conciencia de la localización del robot dentro de el. Todo esto mientras el robot recorre el ambiente.

Si bien SLAM, surgio como un problema de navegación a ser enfrentado por robots, es completamente aplicable en el campo de RA y dispositivos móviles. En lugar de tratarse de un robot, el agente bien puede ser un dispositivo móvil que se encuentre constantemente reconociendo el ambiente y realizando estimaciones de la ubicación del dispositivo dentro de él , con el objetivo de desplegar objetos virtuales en el ambiente reconocido.

La diferencia entre robots y dispositivos móviles radica en sus capacidades para sensar el ambiente. Mientras que para robots existen sensores, como odómetros o sensores laser para medir distancias, en dispositivos móviles comerciales, en el mejor de los casos se cuenta con sensores de acelerómetro, giroscopio y cámara que no son capaces de medir directamente distancias de desplazamientos[20].
Por esta razon, en los últimos años se desarrollado la tecnología de Visual SLAM (vSLAM) que utilizan principalmente los sensores de cámara.

El proyecto Tango de google es un ejemplo de aplicación. Consite en un dispositivo (en prototipo) con una cámara común y una cámara con sensor de profundidad, que utilizando visión por computadora y SLAM , es capaz de llevar a cabo el tracking de desplazamiento y modelar el área en el que se encuentra el usuario, conforme éste la explora.

[20] La empresa x-io ha desarrollado un algoritmo para detección de movimiento utilizando sus propios sensores de aceleración. Estos sensores aun no se encuentran embebidos en dispositivos móviles pero han demostrado que es posible medir desplazamientos solo con dicho sensor.
https://www.youtube.com/watch?v=6ijArKE8vKU

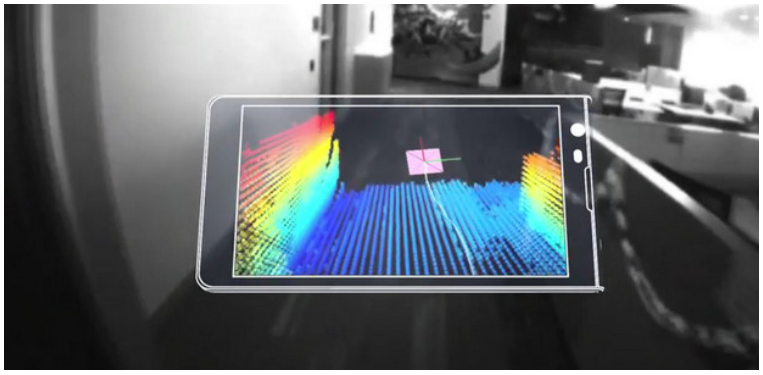


Figura 3.12
Dispositivo del proyecto tango. Consta de sensores de profundidad

Por otro lado actualmente se desarrollan enfoques puramente visuales (sin sensores de profundidad) con resultados muy prometedores. Schmalsteg D. Et al (2014) recientemente han desarrollado un software que utiliza la cámara del dispositivo y un algoritmo vSLAM. El procesamiento es híbrido: el mapeo del ambiente lo realiza el cliente quien envía los fotogramas a un servidor que estima la localización. (Figura 3.13)

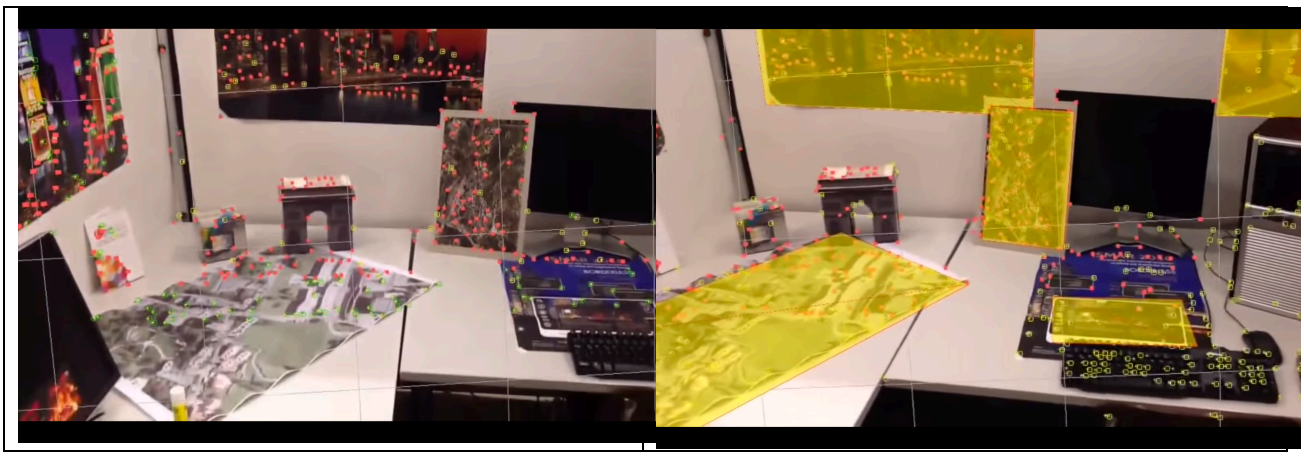


Figura 3.13
Ejemplo de localización utilizando visión monocular y un algoritmo vSLAM.

Wikitude, plataforma de RA, está por lanzar comercialmente (en septiembre de 2015) el soporte para vSLAM que permite agregar contenidos virtuales sin necesidad de agregar códigos QR u otros patrones de imágenes. La plataforma es capáz de extraer las características visuales del ambiente 3D y efectuar el tracking del usuario a partir de las mismas.

Por la naturaleza de este enfoque a mayor cantidad de características extraibles, mayor la presición del mapeo del espacio. Es decir es más fácil elaborar el mapa de una habitación con muchos muebles u otros objetos que se traduzcan en detalles visuales, que el de una habitación completamente vacía. Ese es quisas el único inconveniente de la tecnología.

vSLAM es una de las líneas de desarrollo más prometedoras en el campo de la realidad aumentada, y aunque sus aplicaciones habían sido hasta el momento únicamente en proyectos de investigación, en la actualidad nos encontramos su transición de tecnología en desarrollo a tecnología de uso comercial.

Capítulo 4

Planteamiento de un nuevo enfoque de RA para interiores

Como se ha explicado, la realidad aumentada es la unión sinérgica[21] de 2 mundos: el mundo real y el mundo virtual. Ellos se complementan mutuamente para formar una realidad "enriquecida" en la que el usuario podrá percibir e interpretar información que no sería visible a simple vista sin ayuda de la tecnología de RA.

Todo sistema de RA debe ser capaz de:

- Reconocer características del ambiente real, para presentar contenido virtual apropiado.
- Presentar en tiempo real los contenidos virtuales apropiados.

El enfoque propuesto no utiliza el reconocimiento de patrones para realizar el tracking del mundo real. En lugar de ello, propone un proceso de pre-reconocimiento del ambiente, sin más herramientas que el dispositivo móvil. El enfoque utiliza los conceptos básicos de trigonometría para obtener una representación matemática de un espacio interior.

Si bien, para su demostración de uso, se asumen habitaciones con formas de prismas rectangulares, el enfoque puede ser extendido para su uso en otras formas geométricas.

Enfoque trigonométrico de medición de distancias

La medición de distancias es crucial para el enfoque propuesto. Con ellas se obtendrá la información necesaria para modelar la forma y las dimensiones y la orientación de una habitación.

[21] Sinergia. Acción de dos o más causas cuyo efecto es superior a la suma de los efectos individuales. Sinergia. (2012) Def 1e. Diccionario de la lengua española (DRAE) http://lema.rae.es/drae/?val=sinergia.

En el caso de mediciones de distancias largas el método indicado es usar el sensor GPS, en conjunto con los métodos de la API que calculan la distancia entre dos posiciones basándose en el modelo WGS84 de elipsoide terrestre.
Sin embargo como se ha mencionado, las características del sensor GPS (resolución y alcance) no permiten su uso en la medición de distancias cortas y en interiores, que son el tipo de escenario a abordar.

Por ello, se propone utilizar será el enfoque trigonométrico, basado en la aplicación Smart Measure[22] desarrollada por Smart Tools Co.
Las características de este enfoque son ideales para el proyecto.

- Es soportado por gran cantidad de dispositivos, sólo requiere la presencia de un giroscopio y acelerómetro, presentes en la mayoría de los smartphones de gama media-alta.
- No es invasivo, el teléfono es la única herramienta necesaria.
- Permite calcular distancias y facilita también el cálculo de la altura de un objeto.
- Puede ser calibrado, para ajustarse a distintos dispositivos

Descripción

Este enfoque consiste en la medición de distancias a partir de métodos trigonométricos basados en la rotación del dispositivo y la altura del usuario.
Por medio de los sensores de acelerómetro, detector de campo magnético y giroscopio es posible detectar la rotación de un dispositivo sobre sus 3 ejes.

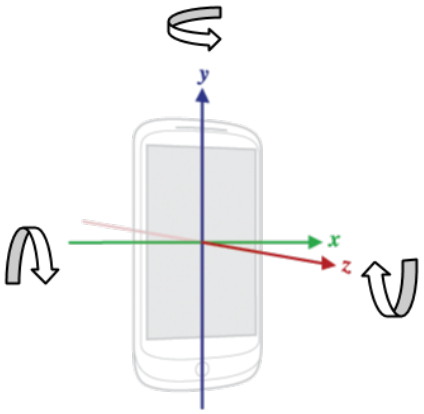


Figura 4.1
Rotación de un dispositivo móvil sobre sus 3 ejes

[22] https://play.google.com/store/apps/details?id=kr.sira.measure

Si al mismo tiempo, activamos la cámara, y apuntamos con ella a diversos objetos, la rotación del dispositivo cambiará y será única para cada objeto que sea enfocado por la cámara.

A continuación, se describe cómo es posible calcular la distancia existente entre el usuario y un un objeto utilizando la cámara y conociendo la rotación del dispositivo y la altura del usuario.

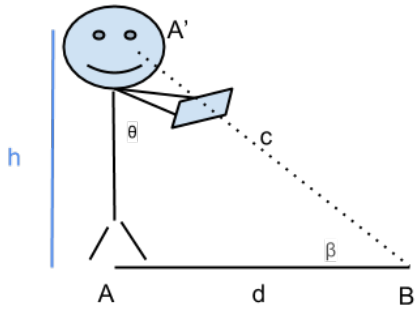


Figura 4.2
Ilustración. Cálculo de la distancia de un objeto, usando el método trigonométrico y la orientación del dispositivo.
Utilizando el dispositivo móvil y apuntando a la base del objeto

En la figura 4.1 se puede observar la medición de la distancia AB. Observar que se forma un triángulo rectángulo A'AB. Conociendo la altura del usuario y el angulo θ, la distancia AB se calcula:

Por ley de senos:

$$\frac{d}{sen(\theta)} = \frac{h}{sen(\beta)}$$

$$\Rightarrow d = \frac{h \cdot sen(\theta)}{sen(\beta)}$$

El único parámetro requerido en esta solución es la altura del usuario.

La distancia del dispositivo a los ojos del usuario no afecta en los resultados de la medición, siempre y cuando no se deje de ver hacia el punto de medición, y la inclinación del mismo no sea distinta a la 'natural'.

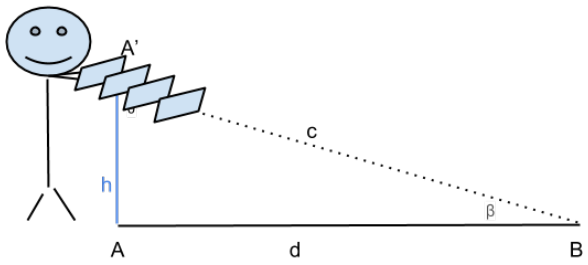


Figura 4.3
Medicion con el dispositivo a distintas distancias.
La distancia del dispositivo a los ojos del usuario no afecta la medición, los ángulos se conservan.

Efectividad

Para analizar la efectividad del método trigonométrico se utilizaron 3 dispositivos. Cuyas características se indican en la figura 4.4.
El experimento consistió en realizar la medición de la distancia del usuario a distintas marcas ubicadas en el suelo, ubicadas cada 0.5 [m]. El piso es parejo sin ningun tipo de pendiente.
Se utilizaron los sensores de acelerómetro y magnetómetro, con un filtro pasa bajos y un promedio de las últimas 50 lecturas, para obtener la matriz de rotación y posteriormente la rotación específica sobre el eje de interés.

	Dispositivo 1 (D1) Xperia ION	Dispositivo 2 (D2) Samsung Galaxy S3 Mini	Dispositivo 3 (D3) BLU Studio 5.0 LTE
Dimensiones	121.6 x 63 x 9.9 mm (4.79	133 x 68 x 10.8 mm	145.5 x 72.5 x 9 mm

	x 2.48 x 0.39 in)	(5.24 x 2.68 x 0.43 in)	(5.73 x 2.85 x 0.35 in)
Pantalla	4.0 inches (~59.4% screen-to-body ratio)	4.55 inches (~63.1% screen-to-body ratio)	5.0 inches (~65.3% screen-to-body ratio)
Resolución	480 x 800 pixels (~233 ppi pixel density)	720 x 1280 pixels (~323 ppi pixel density)	540 x 960 pixels (~220 ppi pixel density)
SO	Android OS, v4.1 (Jelly Bean)	Android OS, v4.1 (Jelly Bean)	Android OS, v4.2 (Jelly Bean)
Chipset	NovaThor U8420	Qualcomm MSM8260 Snapdragon	Qualcomm MSM8926 Snapdragon 400
CPU	1 GHz dual-core Cortex-A9	Dual-core 1.5 GHz	Quad-core 1.2 GHz Cortex-A7
GPU	Mali-400	Adreno 220	Adreno 305
Almacenamiento interno	8/16 GB, 1 GB RAM	16 GB, 1 GB RAM	4 GB, 1 GB RAM
Cámara	5 MP, 2592 x 1944 pixels, autofocus, LED flash	12 MP, 4000 x 3000 pixels, autofocus, LED flash	8 MP, 3264 x 2448 pixels, autofocus, LED flash
Sensores	Accelerometer, gyro, proximity, compass	Accelerometer, gyro, proximity, compass	Accelerometer, gyro, proximity

Medición de distancias, utilzando sensores Acelerómetro/Magnetómetro

	Distancia medida D1	Error D1	Distancia medida D2	Error D2	Distancia medida D3	Error D3
0.5 m	0,3	40,00%	0,35	30,00%	0,45	10,00%
1.0 m	0,85	15,00%	0,85	15,00%	1	0,00%
1.5 m	1,4	6,67%	1,4	6,67%	1,6	-6,67%
2.0 m	2	0,00%	1,8	10,00%	2,15	-7,50%
2.5 m	2,5	0,00%	2,2	12,00%	2,75	-10,00%
3.0 m	2,95	1,67%	2,65	11,67%	3,3	-10,00%
3.5 m	3,45	1,43%	3,05	12,86%	3,85	-10,00%
4.0 m	4	0,00%	3,4	15,00%	4,4	-10,00%
4.5 m	4,5	0,00%	3,85	14,44%	5	-11,11%
5.0 m	5,05	-1,00%	4,2	16,00%	5,6	-12,00%
5.5 m	5,6	-1,82%	4,55	17,27%	6,25	-13,64%
6.0 m	6,1	-1,67%	4,85	19,17%	6,85	-14,17%
6.5 m	6,6	-1,54%	5,1	21,54%	7,5	-15,38%
7.0 m	7,1	-1,43%	5,45	22,14%	8,15	-16,43%
7.5 m	7,7	-2,67%	5,7	24,00%	8,7	-16,00%
8.0 m	8,3	-3,75%	6,05	24,38%	9,3	-16,25%

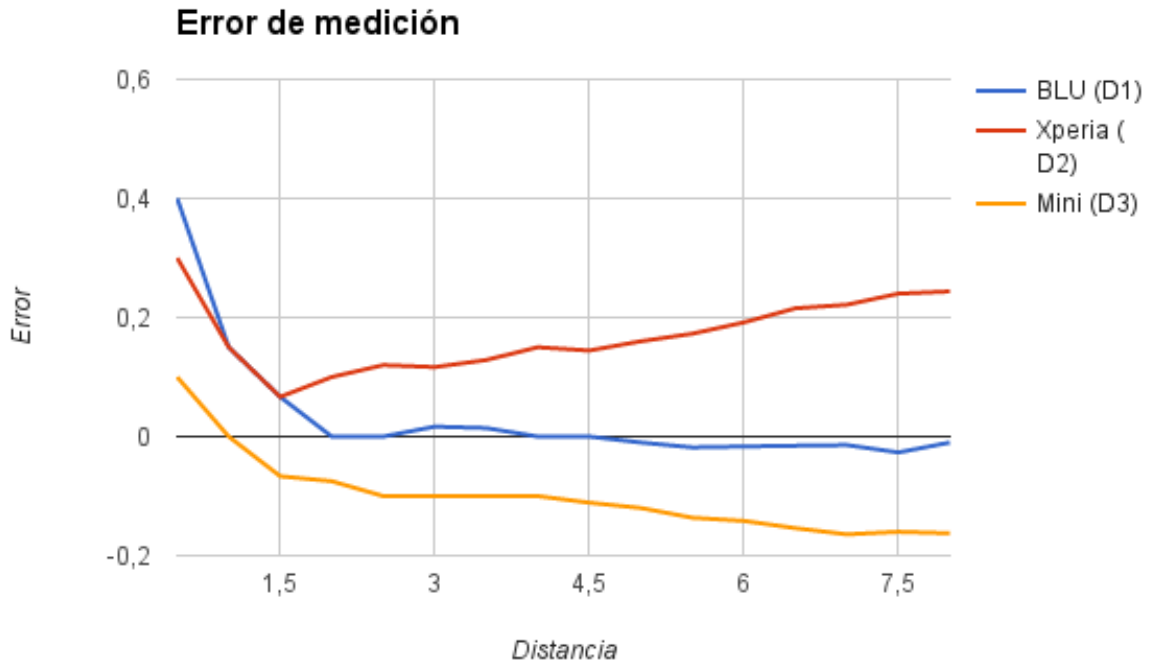


Figura 4.4
Gráfica del error de medición de distintos dispositivos.

Luego del experimento es posible identificar los siguientes hechos:
La fragmentación de sensores es un problema serio a considerar. La misma aplicación arroja resultados distintos cuando es ejecutada en distintos dispositivos.

Las distancias en las que se obtienen las mejores mediciones, son aquellas entre 1.5m y los 6m. Más allá de ese límite el pulso o la capacidad del usuario de enfocar a un punto afectan considerablemente la medición.

La medición de distancias menores a 1.5 m, independientemente del dispositivo utilizado son menores a las distancias reales. Podemos inferir este error como una característica del método trigonométrico en sí, más que a las diferencias de hardware entre los dispositivos.

Corrección del error específico del dispositivo

Si bien la medición, arroja errores para los dispositivos D2 y D3, estos no son del todo valores arbitráreos. Fue posible modelar la función del error para cada dispositivo (aplicando regresión lineal) y compensar los valores leídos para arrojar valores más cercanos a los verdaderos.

Para esta modelación, se realizó el experimento en orden inverso. Se enfocó con el dispositivo a distintos puntos en el suelo y se aplico un modificador en la altura del usuario para que la medición obtenida utilizando el método trigonométrico coincidiera con la verdadera.

$$h = h_{usuario} - \delta$$

Así se encontró que para D2 y D3, el cambio en los valores de δ conforme aumentaban las distancias tuvo un comportamiento lineal.

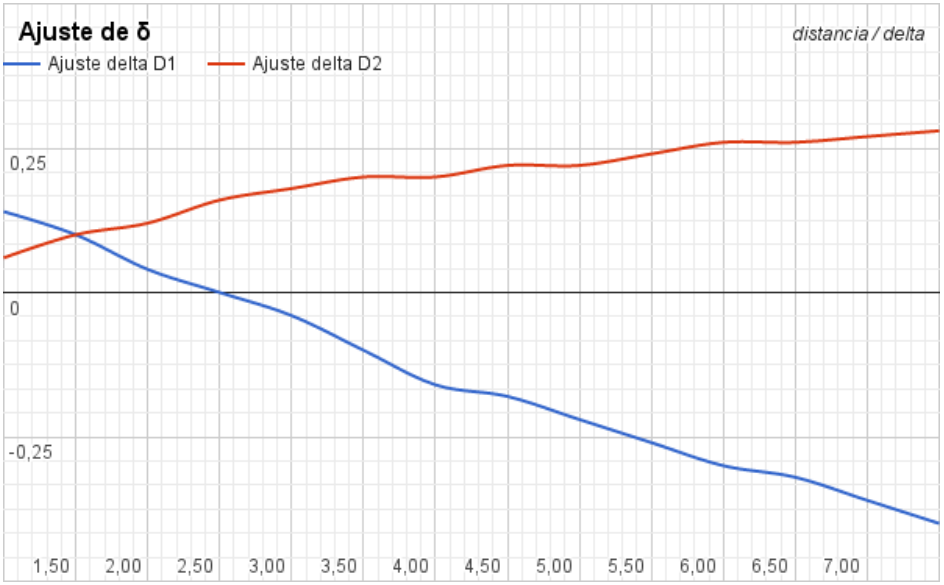


Figura 4.5

Determinación de una función para corrección del error, específica para cada dispositivo

Obtención de las características del espacio interior

El enfoque propuesto requiere de un proceso de pre-reconocimiento en dos pasos, para determinar las dimensiones y la orientación de la habitación o espacio interior.

1. Obtención de la altura de la habitación

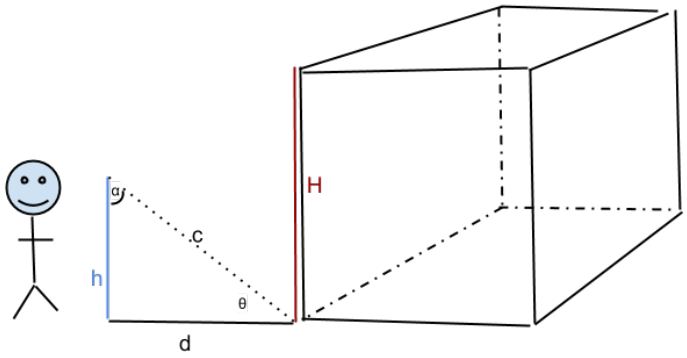


Figura 4.6
Obtención de la altura de la habitación. Primer paso.

La altura del usuario es el único valor requerido en la solución y es especificado por él mismo al comenzar a utilizar la aplicación.

Para calcular la altura de la habitación es requerido que al menos una de las aristas verticales que se forman de las 4 paredes tenga visibles sus vértices inferior y superior (sin que la obstruyan muebles u otros objetos).

En un primer paso, cuando el usuario apunta con la cámara del dispositivo hacia el vértice inferior de una arista, los sensores detectan la inclinación del teléfono (ángulo α). Conociendo además la altura del usuario (h) es posible determinar el ángulo θ , y el largo de las rectas d y c'

$\theta + \alpha + 90° = 180°$

$\theta = 180° - 90° - \alpha$

Por ley de senos:

$$\frac{h}{sen(\theta)} = \frac{d}{sen(\alpha)} \Rightarrow d = \frac{h\, sen(\alpha)}{sen(\theta)}$$

$$c = \sqrt{d^2 + h^2}$$

En un segundo paso, cuando el usuario apunta con la cámara del dispositivo hacia el vértice superior de la misma arista, se cuenta con la información necesaria para calcular la altura de la habitación.

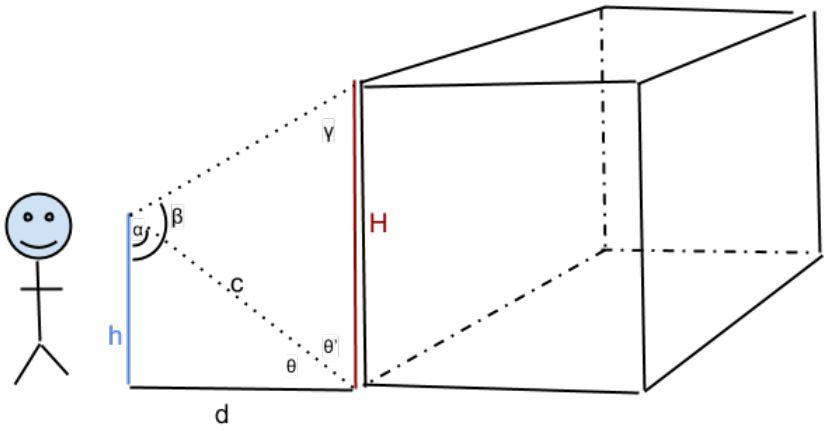


Figura 4.7

Obtención de la altura de la habitación. Segundo paso.

$$\theta' = 90° - \theta$$

$$\gamma = 180° - \theta' - (\beta - \alpha)$$

$$\frac{sen(\beta - \alpha)}{H} = \frac{sen(\gamma)}{c} \Rightarrow H = \frac{c\, sen(\beta - \alpha)}{sen(\gamma)}$$

2. Obtención de los vértices de la habitación

Cuando se conoce la altura de la habitación, y asumiendo que esta es uniforme, será posible determinar la forma, medidas y orientación de la habitación detectando la ubicación de sus vértices superiores o inferiores, relativas a la posición del usuario.

El proceso requerido es muy sencillo: el usuario tendrá que permanecer fijo en un punto dentro de la habitación y apuntar con la cámara de su dispositivo a los cuatro vértices superiores. El sensor de campo magnético, en conjunto con el acelerómetro devuelven la rotación del dispositivo, y por ende la orientación en grados de cada uno de los vértices respecto al norte magnético.

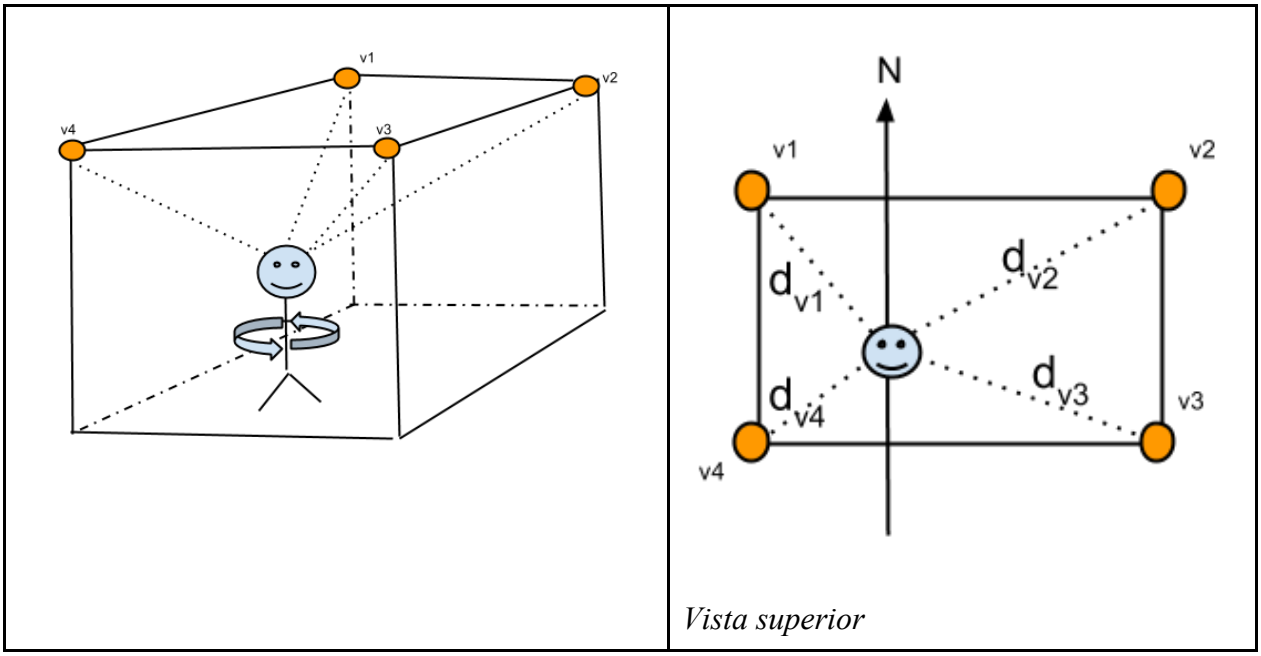


Figura 4.8
Obtención de la orientación de los vértices de la habitación.

Para completar el proceso de escaneo de la habitación, es necesario saber a qué distancia del usuario se encuentra cada vértice (d_{v1}, d_{v2}, d_{v3} ,d_{v4}). De nuevo, esto es sencillo haciendo cálculos trigonométricos:

Cada vez que el usuario apunta a un vértice superior, se forma un triángulo rectángulo como se ilustra en la figura.

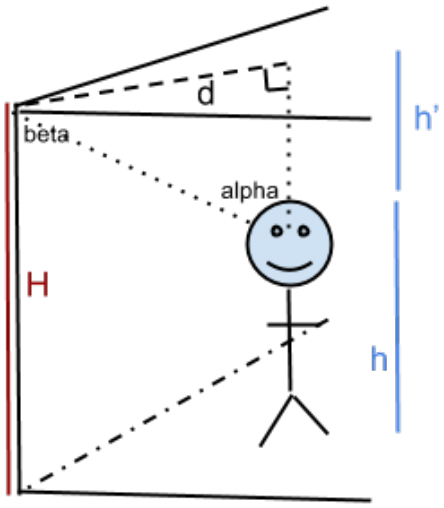


Figura 4.9
Obtención de la distancia a los vértices de la habitación.

El cateto d del triángulo, es la distancia del usuario al vértice. Y su valor por ley de senos, está dado por la fórmula:

$$\beta = 180° - 90° - \alpha$$

$$d = \frac{h' \, sen\,(\alpha)}{sen(\beta)}$$

Este cálculo es realizado cada vez que el usuario registra un vértice.

Cuando el usuario ha efectuado el proceso descrito hasta ahora, se conoce la altura de la habitación, y la ubicación de sus vértices relativa al dispositivo (y por ende al usuario). Esto es todo lo necesario para elaborar una representación matemática de la habitación: un prisma rectangular cuyas dimensiones y orientación en el espacio son conocidas. (Fig 4.8)

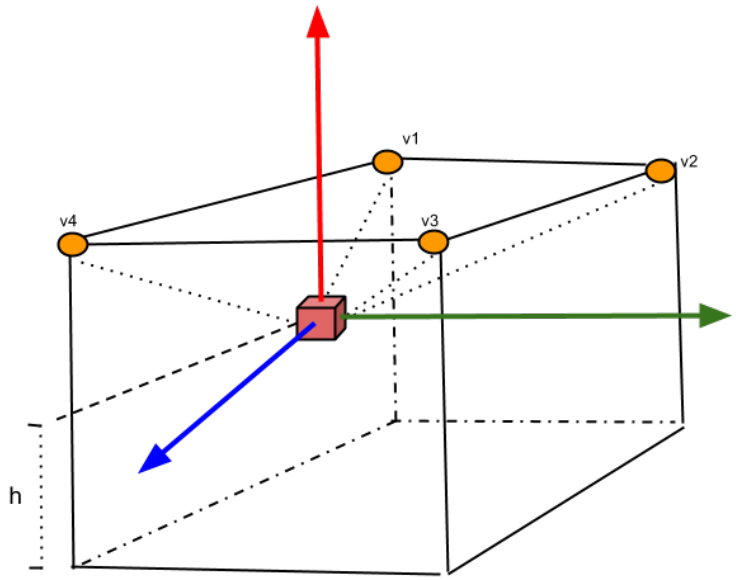


Figura 4.10
Modelo de la habitación (medidas, orientación y posición del usuario/teléfono luego del proceso de escaneo)
También se muestran los ejes del sistema de coordenadas terrestres o globales.

En este punto se conocen los vértices de la habitación, y por ende la las paredes que la conforman. En adelante, sería posible presentar contenidos de RA basados en las características del espacio interior (medidas y orientación).

Por ejemplo, podrían agregarse objetos virtuales tales como muebles, pinturas, puertas, etc. y anclarlos a cualquiera de las caras del cubo (que son las paredes) que se construye de las medidas recopiladas por los sensores.

Tracking del usuario

Como se ha mencionado, el objetivo de la realidad aumentada es presentar contenidos virtuales en sincronía, y acoplados coherentemente con objetos del mundo real.
En el ámbito de las aplicaciones móviles, suele usarse la cámara para obtener las imágenes del mundo real, sobre las cuales se agregan contenidos virtuales (modelos 3d, fotos, videos, etc)

En el enfoque propuesto, el usuario se encuentra dentro de una habitación que puede modelarse como un prisma rectangular, del cual como se discutió en la sección anterior, es posible conocer sus medidas y orientación.

En un segundo proceso, para determinar cuáles son los objetos virtuales que se dibujarán en pantalla junto con las imágenes capturadas desde la cámara, es necesario saber cuál es la posición y la orientación del usuario relativa al prisma que lo rodea. (Fig 4.8)
Monitorear los cambios en la posición y orientación del usuario es una tarea de los sistemas de tracking, presentes en todas las aplicaciones de RA.

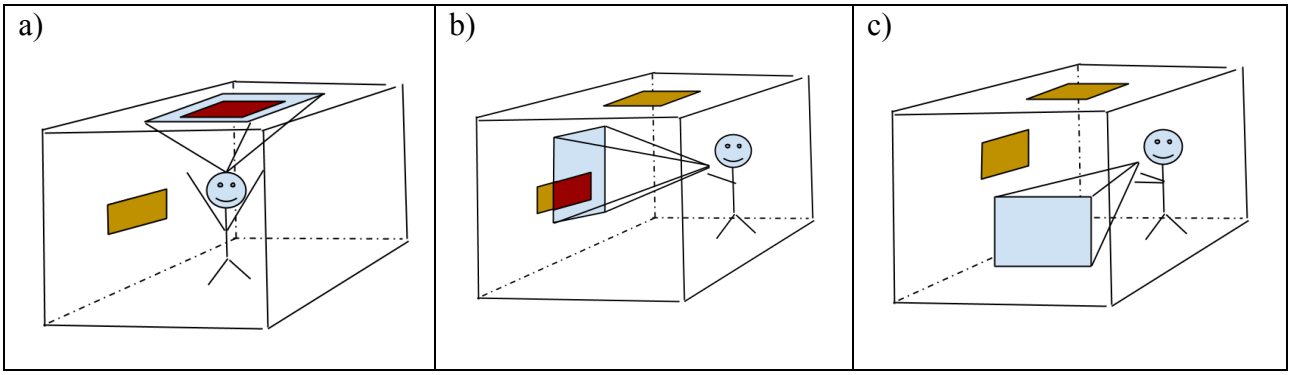


Figura 4.11
Render de objetos virtuales
a) El usuario apunta hacia arriba, se hace render del objeto virtual que se ubica en el techo. El objeto virtual, que se situa en la pared no es dibujado en lo absoluto
b) El usuario apunta hacia la pared. Solo se dibuja la porción del objeto virtual que corresponde con el campo de visión de la cámara
c) *El usuario apunta en una dirección en la que no hay situado ningún objeto virtual. El dispositivo mostrará solamente las imágenes del preview de la cámara.*

Detección de la rotación del usuario

Cuando el usuario explora el ambiente con su dispositivo sin cambiar su ubicación, es decir sin desplazarse, sus movimientos se traducen en distintas orientaciones del dispositivo. Por ello, para determinar la rotación del usuario, será necesario obtener la rotación del dispositivo.

En matemáticas, cualquier orientación de un objeto en el espacio 3D, puede ser expresada como una composición de 3 rotaciones elementales.

Dichas rotaciones ocurren sucesivamente sobre al menos 2 de los ejes de un sistema de coordenadas.
Las rotaciones son intrínsecas, si el sistema rota al mismo tiempo que el objeto o bien extrínsecas si el sistema permanece fijo conforme la aplicación sucesiva de las tres rotaciones.
Cuando el sistema es estático, se denomina sistema fijo o de coordenadas globales.

Sin considerar la posibilidad de rotaciones extrínsecas o intrínsecas, existen 12 secuencias de rotaciones divididas en 2 grupos:

- **Ángulos euler clásicos** (*z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y*)
- **Ángulos Tait–Bryan** (*x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z*).

Notar que en el segundo grupo se aplican las rotaciones sobre cada eje.

En álgebra lineal, la aplicación de una transformación de rotación se traduce a multiplicar el vector a rotar por una matriz de rotación. Utilizando matrices de rotación, la rotación en sentido antihorario sobre el eje z se define por la matriz:

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

La rotación antihorario sobre el eje y:

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}.$$

Y la rotación antihorario sobre el eje x:

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}.$$

Cada una de estas rotaciones, en realidad son extensiones de matrices de rotación 2D en los planos XY, YZ, y XZ.

De la definición de ángulos **Tait–Bryan**, podemos alcanzar cualquier orientación aplicando rotaciones sobre cada eje. Y ya que la multiplicación de matrices cumple la propiedad asociativa, podemos obtener una matriz de rotación 3D, multiplicando las 3 matrices de rotación de cada eje, que podrá posteriormente multiplicarse por el o los vectores que se deseen rotar.

Es importante señalar que la multiplicación de matrices no es conmutativa, por lo que el orden en que multipliquemos las matrices determinará el orden en que serán aplicadas las rotaciones y por ende la orientación final.

Por ejemplo, si se aplica la rotación de un vector utilizando la matriz R, resultado de:

$$R(\alpha, \beta, \gamma) = R_x(\alpha) \cdot R_z(\gamma) \cdot R_y(\beta)$$

Será equivalente a rotar el objeto primero sobre el eje *y*, luego sobre el eje *z* y al último sobre el eje *x*.
Y si aplicamos la rotación de un vector utilizando la matriz R, resultado de:

$$R(\alpha, \beta, \gamma) = R_z(\gamma) \cdot R_y(\beta) \cdot R_x(\alpha)$$

Será equivalente a rotar el objeto primero sobre el eje *x*, luego sobre el eje *y* y al último sobre el eje *z*.

Detección del cambio de la posición del usuario

Existen diversos antecedentes de calcular el movimiento de un usuario en interiores, por ejemplo, mediante la detección de pasos (podómetros) o bien mediante beacons. Para el caso de los podómetros, su poca precisión y eventual necesidad de re-calibración no son una opción viable. En el caso de los beacons, se trata de piezas de hardware adicionales, que van en contra del objetivo inicial del uso exclusivo del dispositivo móvil.

Si bien, los dispositivos tienen acelerómetros, cuya integración arroja velocidad, y segunda integración arroja distancias, el error de los sensores, agraviado por las integraciones también resultan en un enfoque no viable[23] [24].

La localización en exteriores utilizando GPS, antenas de telefonía o bien por beacons comparten una característica: se conoce de antemano la ubicación de los emisores. (satélites, beacons o antenas). El dispositivo es capaz de saber a qué distancia se encuentra de cada uno de los emisores, y triangulando, se puede obtener la posición relativa del dispositivo a los emisores.

---

[23] Google Tech Talk, Agosto 2, 2010
Presented by David Sachs.
https://www.youtube.com/watch?v=C7JQ7Rpwn2k

[24] Demostración de algoritmo para obtener la posición 3D
https://www.youtube.com/watch?v=6ijArKE8vKU

A continuación, se explican 2 métodos de localización en interiores, inspirados en la localización en exteriores por triangulación.

Método 1

En el enfoque planteado, podemos realizar algo semejante a la triangulación de satélites.
En el proceso para determinar las medidas del espacio interior, se determinó la ubicación y distancia de los 4 vértices superiores de la habitación al usuario. Dichos vértices podemos semejarlos con satélites, beacons o antenas. Conocemos su ubicación y distancia a la posición inicial del usuario (La posición desde la cual se hizo el escaneo de la habitación)

Si el usuario se mueve dentro de la habitación de una posición p1 a una posición p2, el sistema quedará descalibrado. La cámara mostrará imágenes desde la nueva perspectiva p2 mientras que la aplicación de RA dibujará elementos virtuales como si el usuario aún permaneciera en p1.

Es posible calibrar la nueva posición del usuario usando la información de la ubicación de los vértices antes del movimiento, recopilada en el proceso de escaneo.

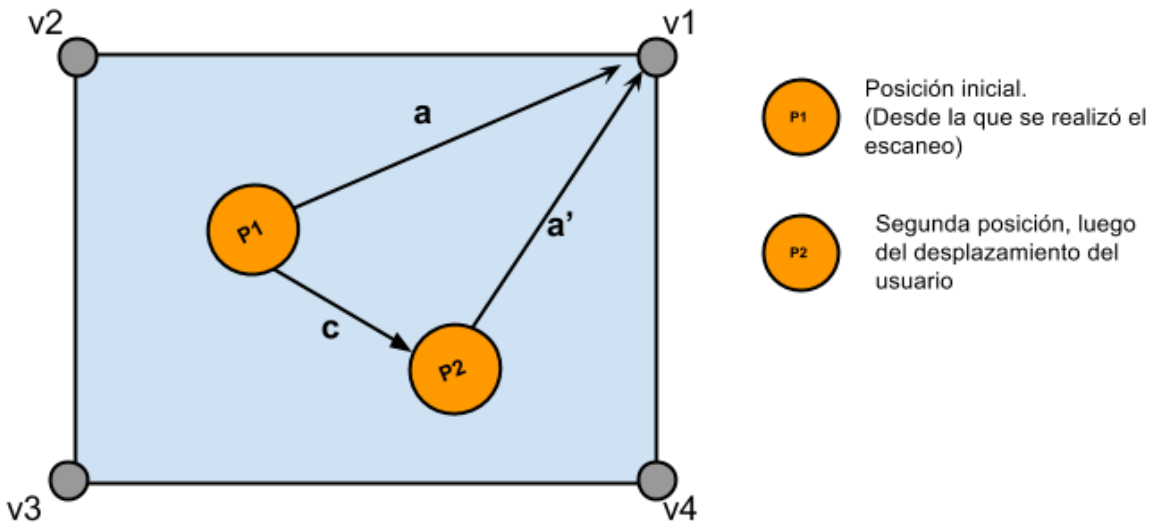


Figura 4.12
Detección del cambio de posición del usuario. Método 1.

El vector **a** indica la dirección y la distancia del usuario al vértice v1. (De hecho cuando se realiza el proceso de reconocimiento de la habitación, se calcula un vector de la posición del usuario a cada uno de los cuatro vértices)

Cuando el usuario se desplaza al interior de la habitación de p1 a p2 y vuelve a apuntar hacia el vértice v1, habrá un vector **a'** de su nueva posición al vértice v1.

a - a' = c

La nueva posición del usuario estará dada por:
p2 = p1 + **c**

Si bien, la detección del movimiento no ocurre en tiempo real, este método para calibrar la posición presenta la ventaja de no desfasarse o perder precisión en el tiempo. Todas las nuevas posiciones son relativas a la posición inicial, que es la posición en la que se escaneó la habitación.

Método 2

El método antes descrito, funciona cuando se conoce el vector **a** , que va de la posición inicial a v1.

Sin embargo, cuando el usuario introduce manualmente las medidas exactas de la habitación (largo, ancho y alto), está definida la dimensión de la habitación pero no su orientación. Por tanto se conoce la magnitud de **a** pero no su dirección.

Para solucionar este problema es necesario proporcionar más información al sistema, para ello se plantea un segundo método de detección de desplazamiento.

Consideraciones previas.
Cuando el usuario ha escaneado la orientación y distancias de los vértices, se conocen los vectores v1,v2,v3,v4. (Figura 4.11)

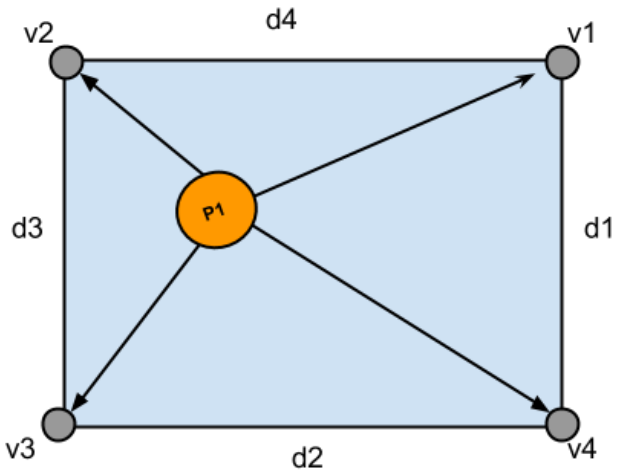
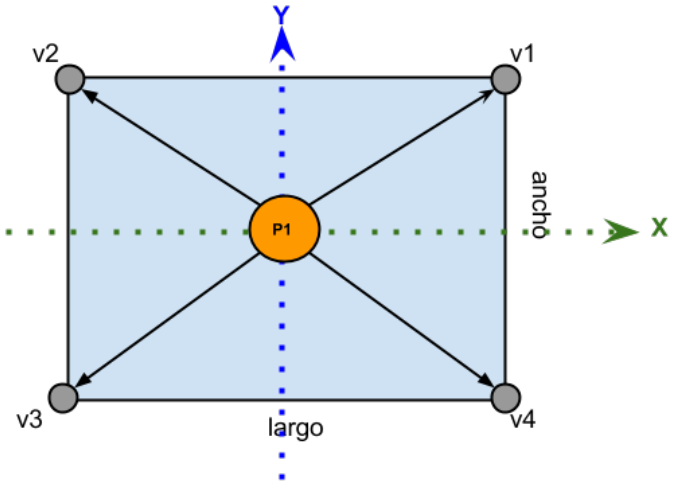


Figura 4.13
Vectores de posición de los vértices, luego del proceso de escaneado.

Si el usuario, introduce las medidas exactas, no se conoce la orientación de los vértices pero se asume que:

- el ancho es paralelo al eje *y*
- el largo es paralelo al eje *x*
- La posición inicial del usuario es el origen del sistema de coordenadas, que a su vez es el centro del paralelogramo



El método es:
Desde la nueva posición P_2, el usuario apunta a v1, luego apunta a v2. (Fig 4.12)

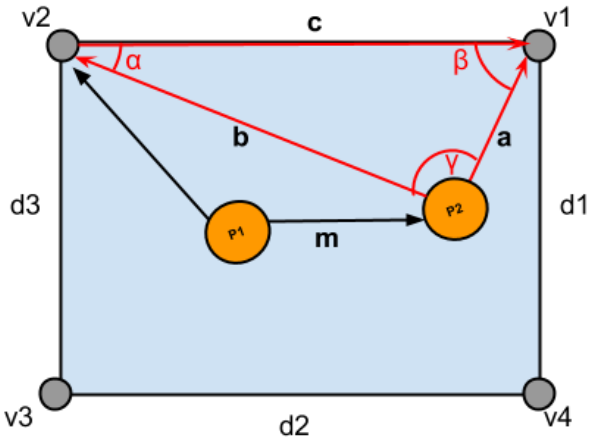


Figura 4.14
Detección del cambio de posición del usuario. Método 2.

Al realizar este procedimiento, se conocen las distancias a P_2v1 y a P_2v2, que son las magnitudes $|\mathbf{a}|$ y $|\mathbf{b}|$ respectivamente. Además se conoce también el ángulo $\boldsymbol{\gamma}$.

Cuando el usuario realizó el proceso de escaneo de la habitación, se obtuvieron posiciones x,y,z de los cuatro vértices v1,v2,v3 y v4.

Es decir los vectores

$$\overline{P_1V_1}$$
$$\overline{P_1V_2}$$
$$\overline{P_1V_3}$$
$$\overline{P_1V_4}$$

son conocidos.

Recordemos que la posición P1 es (0,0,h) donde h es la altura del usuario.

El vector **c** puede calcularse fácilmente:

$$\bar{c} = \overline{P1v1} - \overline{P2v2}$$

Se deberá elegir cuál de las dos magnitudes $|\mathbf{a}|$ y $|\mathbf{b}|$ es más confiable, según el modelo de medidas por método trigonométrico. En este caso elegimos arbitrariamente $|\mathbf{a}|$.

Por ley de senos:

$$\frac{sen(\gamma)}{|c|} = \frac{sen(\alpha)}{|a|}$$

$$\Rightarrow \alpha = angsen(\frac{|a| \cdot sen(\gamma)}{|c|})$$

$$\Rightarrow \beta = 180º - \alpha - \gamma$$

Hasta este punto se conocen las dimenciones y los ángulos del triángulo $P_2V_1V_2$

Cuando el usuario se desplaza de P_1 hacia P_2, los contenidos virtuales de RA ya no están en sincronía con el mundo real. Para recalibrar el sistema, deberá indicarse que el usuario ha realizado una traslación definida por el vector **m**.
Un enfoque común en computación gráfica, es representar el movimiento como un desplazamiento en sentido inverso de todos los demás objetos mientras éste permanece quieto. Usando dicho enfoque, cuando el usuario se traslada a una segunda posición, será posible alinear el mundo virtual con el mundo real, aplicando una matriz de traslación con las componentes del vector **(-)m** a todos los objetos virtuales.

$$\bar{m} = \overline{P_1V_2} - \bar{b}$$

Para obtener $\bar{b}$, calculamos el vector unitario $\hat{c}$, lo rotamos $-\alpha$ y lo escalamos según la magnitud de $\bar{b}$

$$\bar{b} = |\bar{b}|\, Rot(\alpha, \hat{c})$$

Los vectores a y b , son paralelos al plano XY, su componente en Z es constante (la altura usuario). Teniendo presente esta consideración, podemos operarlos en R2.

Para simplificar el cómputo, es posible calcular una sola matriz que combine la rotación:

$$Rot(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

con la operación de escalado.

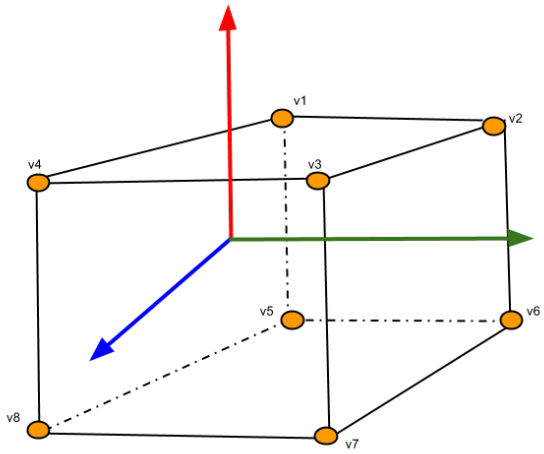
Así, las componentes de **a** quedan:

$a_x = |\mathbf{a}|\, b_x \cos\theta - |\mathbf{a}|\, b_y \mathrm{sen}\theta$

$a_y = |\mathbf{a}|\, b_x \operatorname{sen}\theta + |\mathbf{a}|\, b_y \cos\theta$

Adición de objetos virtuales

Luego del proceso de escaneo o bien de la introducción de las medidas de una habitación, el modelo de esta es un prisma rectangular delimitada por 8 vértices, como se ilustra en la siguiente imagen.



Los vértices principales y calculados en primera instancia son v_1, v_2, v_3 y v_4. Los vértices v_5, v_6, v_7 y v_8 corresponden con los anteriores y sus componentes *x*,*y* son las mismas.
La componente en z es la altura de la habitación para los vértices v_1, v_2, v_3 y v_4 , y para los vértices inferiores (v_5, v_6, v_7 y v_8) es 0.

Con estas consideraciones, si se quieren agregar objetos virtuales, al igual de como ocurre en el mundo real, estos deberán estar anclados a una de las caras del prisma. Por ejemplo, camas, sillas , mesas, tapetes, etc, se encuentran anclados al suelo por acción de la fuerza de gravedad. Podrán desplazarse pero siempre sin despegarse del suelo.
En el caso de pinturas, puertas y ventanas, se encuentran por lo general anclados a una pared.
De igual manera, objetos como lámparas se encuentran fijos al techo

En el modelo virtual, el suelo y el techo son planos paralelos a XY.
Y cada plano que representa una parede se determina por cuatro vértices.

v_2-v_6-v_7-v_3
v_3-v_7-v_6-v_4
v_1-v_4-v_6-v_5
v_1-v_5-v_6-v_2

Según la naturaleza del objeto virtual, este podrá ser anclado al piso, techo o paredes. Y ello determinará cómo se efectuarán las operaciones de traslación y rotación del mismo.

El siguiente enfoque para la colocar objetos virtuales es genérico y funciona para cualquier plano (pared) del prisma que modela la habitación:

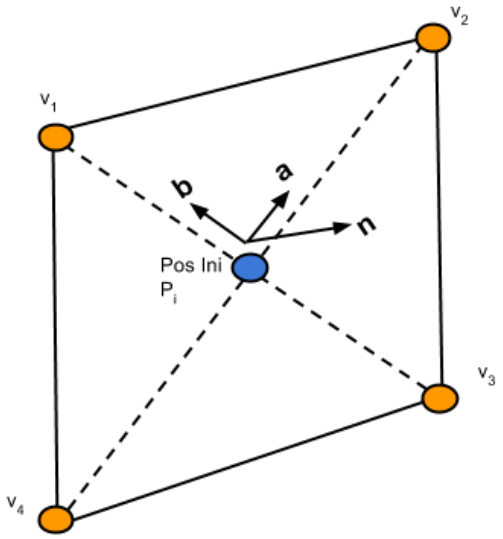


Figura 4.15
Anclaje de objetos virtuales a las paredes.

Cuando se decide colocar un objeto virtual sobre un plano, este será colocado en el centro del plano (Fig 4.13). El centro es la intersección de los segmentos v_1v_3 y v_4v_2

a y **b** son vectores unitarios contenidos en el plano.
a = *normaliza*($\mathbf{v_2}$ - $\mathbf{v_4}$)
b = *normaliza*($\mathbf{v_1}$ - $\mathbf{v_3}$)

Traslación

El movimiento del objeto, una vez anclado al plano estará restringido a 2 direcciones.
Cualquier punto **p** dentro del plano es alcanzado sumando a $\boldsymbol{P_i}$ vectores paralelos a **a** y **b.**

$$p = p_i + v\boldsymbol{a} + k\boldsymbol{b}$$

Donde *v* y *k* son constantes.

Centro de objetos virtuales

Al modelar un objeto virtual, se trabaja en coordenadas locales (para cada objeto). Y la localización del origen del centro de coordenadas o la ubicación del centro del objeto dependerá de la naturaleza del mismo. En las siguientes figuras se muestra el centro de coordenadas de distintos objetos 3D, teniendo en cuenta que ese será el punto en donde se anclará con el suelo, pared o techo.

En la figura 4.14 se muestra un mismo objeto 3D pero con el origen de su sistema de coordenadas en diferente posición.
La solución es mover el origen del sistema de coordenadas

a) b)

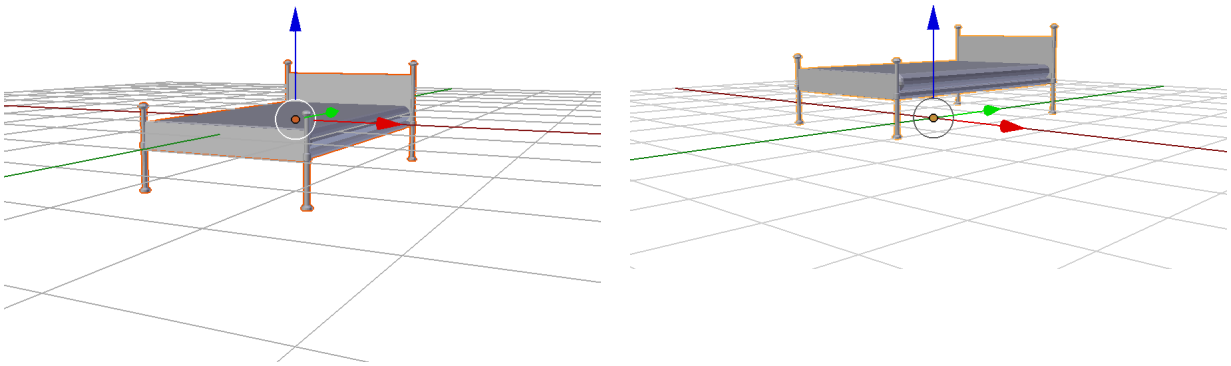


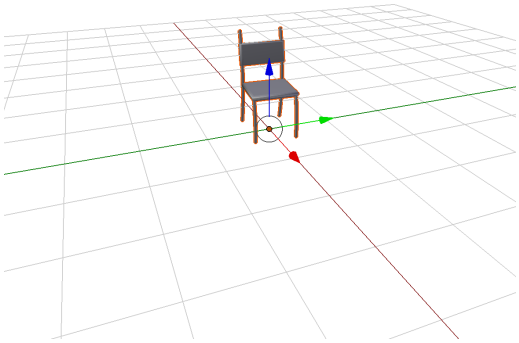
Figura 4.16

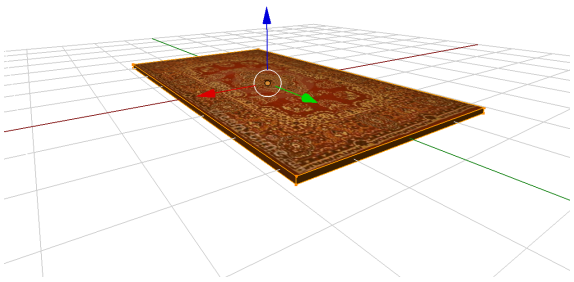
Mismo objeto con el origen en distintas posiciones.

a) El origen del modelo de la cama se ubica en medio del colchón y el resultado al anclar el modelo al piso sería que las patas "atravesarían" el piso.

b) El origen se posicione debajo del colchón, a la misma altura de las bases de las patas de la cama. Al anclar el objeto al piso, se verá correctamente.

Las siguientes figuras muestran la posición correcta del sistema de coordenadas locales en una silla o en un tapete.





Rotación

Los objetos anclados al techo, suelo o pared rotarán sobre un solo eje perpendicular al plano.
$\hat{n}$ es el vector unitario de dicho eje y es el producto vectorial:

$$\hat{n} = \bar{a}\ x\ \bar{b}$$

Expectativas de uso

Las características del enfoque propuesto, perfilan su aplicación en cualquier campo en el que se vislumbre agregar contenidos virtuales basados en la forma y dimensión de una habitación o espacio interior.

De sus posibles aplicaciones, la decoración en interiores parece una opción prometedora y divertida, para ser implementada en un primer prototipo.

Capítulo 5
Uso del enfoque propuesto en un caso práctico

Descripción de la aplicación

Objetivo

Desarrollar una aplicación móvil de RA que utilice el enfoque de RA descrito, que sea capaz de simular la presencia de objetos virtuales en una habitación. Se utilizará el enfoque propuesto para validar su factibilidad de uso en otras aplicaciones de RA.

El prototipo a desarrollar es una aplicación de una tienda de muebles ficticia. Dicha tienda pone a disposición de sus clientes a través de la aplicación, un catálogo de muebles virtuales con el objetivo de que el cliente pueda saber como se verá un mueble en su casa y así poder tomar una mejor decisión de compra.
El cliente podrá simular la presencia de objetos virtuales tales como sillas, mesas, pinturas, tapetes, entre otros.

Usuarios de la aplicación

La aplicación es de uso personal. Se contempla un solo usuario (El cliente).

Requerimientos de la aplicación.

- El usuario podrá escanear un espacio cerrado y agregar contenidos virtuales para crear un ambiente de RA.
- Los contenidos virtuales a agregar podrán ser cualquiera de los siguientes: cuadros/pinturas , muebles , tapetes.
- Con ayuda de la cámara del dispositivo el usuario podrá contemplar la habitación al tiempo que los contenidos virtuales agregados previamente, parecerán realmente existir.

- El usuario deberá registrar su altura en la aplicación y podrá cambiarla en la sección de configuración.
- Las unidades a utilizar en la aplicación podrán ser en Sistema Internacional, o Sistema Anglosajón. El sistema seleccionado será consistente para todos los valores requeridos.
- Antes de poder realizar la inclusión de contenidos de RA el usuario deberá escanear el espacio cerrado en el que se encuentre.
- El usuario podrá crear más de un ambiente virtual para un mismo espacio interior.
- Los contenidos virtuales estarán accesibles desde un catálogo.
- El usuario podrá modificar y borrar un ambiente virtual, creado previamente.
- El usuario podrá crear / editar / borrar más de un solo ambiente virtual.
- El usuario podrá clonar las medidas de una habitación escaneada previamente.
- La aplicación contará con una vista para agregar contenidos virtuales y con una vista para visualizar los contenidos virtuales agregados.
- Al ser una aplciación de RA la vista para realizar el proceso de escaneo de la habitación, la de agregar contenidos virtuales y la de visualización de contenidos virtuales funcionarán desplegando en el fondo las imágenes capturadas desde la cámara. La orientacion de dichas vistas será en modo horizontal (landscape).
- Este primer prototipo está orientado a teléfonos, no a tablets.

Elección de la plataforma

En el proceso de elección de la plataforma se consideraron Android, iOS y Windows Phone por ser las de mayor presencia en el mercado actual a nivel mundial.

Para el presente trabajo de tesis se ha optado por la elección de la plataforma Android por las siguientes razones:

- Costo de Desarrollo. El costo de desarrollo del prototipo se eleva con la necesidad de adquisición de hardware y licencias de desarrollo en iOS y WP.
- El producto es un prototipo no comercial con la intención de ser distribuido a través de una tienda de aplicaciones. En el caso de Android la aplicación puede ser publicada sin costo.
- Experiencia previa en el desarrollo de aplicaciones para Android.

La elección de una plataforma no descarta la factibilidad de implementación del prototipo en las otras. Ello depende más bien de las características del hardware en los dispositivos específicos.

Diseño

Flujo de la aplicación

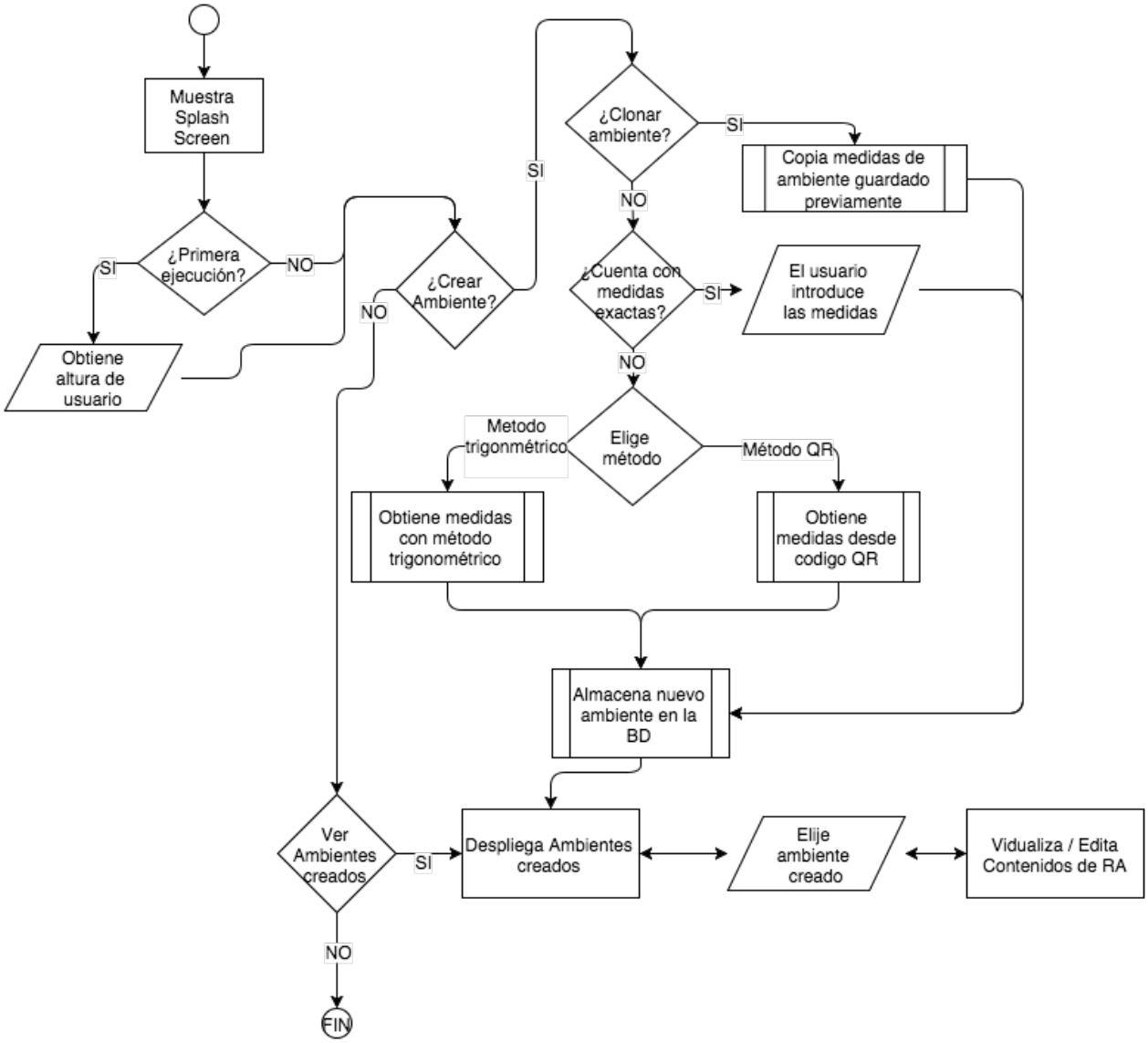


Figura 5.1
Diagrama de flujo de la aplicación

Diagrama Entidad-Relación

El modelo Entidad-Relación de la aplicación es sumamente sencillo y se ilustra en el siguiente diagrama

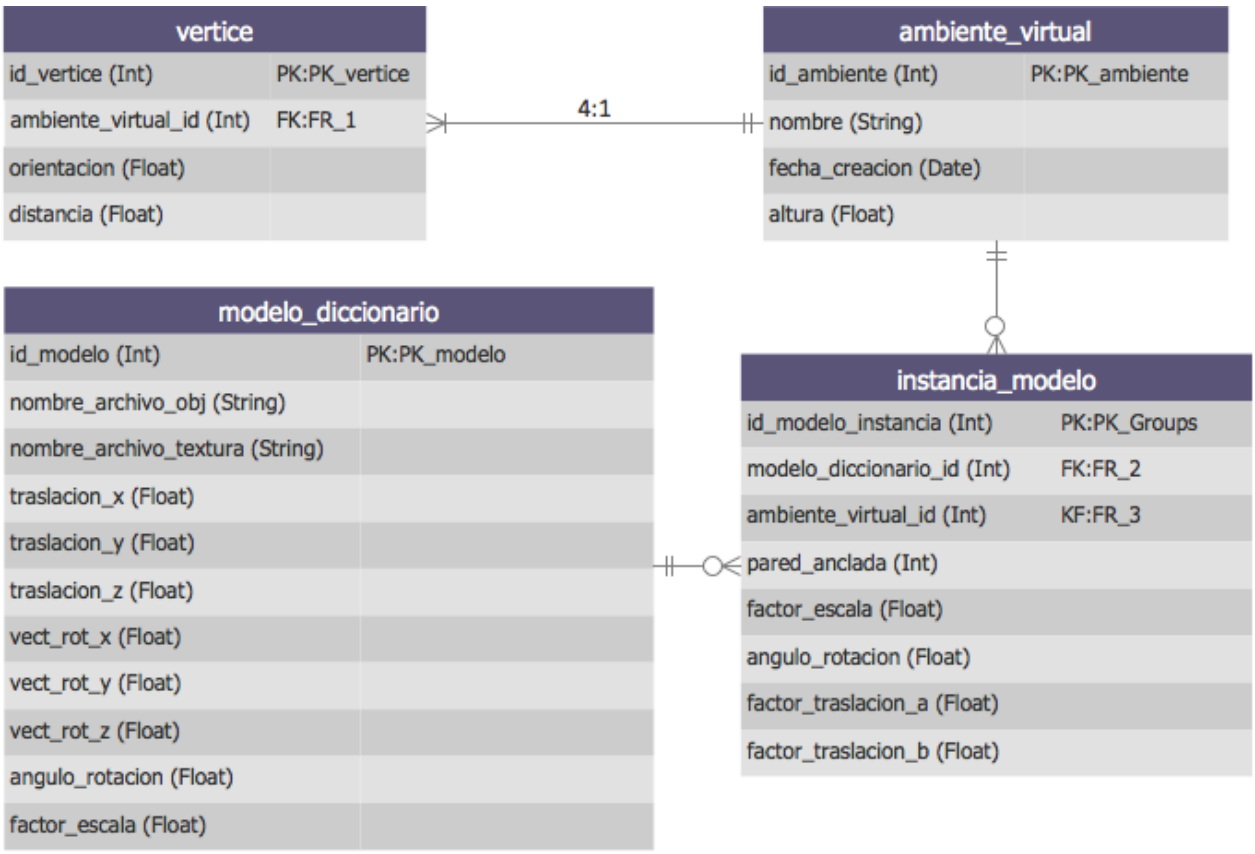


Figura 5.2
Diagrama ER de la aplicación

Diccionario de datos

Vértice

Describe un vértice de una habitación. Cada vértice se compone de una orientación (que es el ángulo que arrojaría una brújula cuando se está viendo hacia el) y una distancia del usuario al mismo.

Ambiente Virtual

Es el modelo de una habitación. Se compone de un conjunto de vértices y la altura de la habitación.
Si bien pudiera incluirse la información de los 4 vértices que componen la habitación en esta misma tabla, se dejan en una tabla separada a fin de poder extender la aplicación para que pueda funcionar en habitaciones con formas distintas a las de un prisma rectangular.

Modelo Diccionario

En esta tabla se almacena la información de cada modelo 3D que está disponible en el catálogo. Si bien en esta clase se definen los moldes de objetos 3D para sus posteriores instancias, se agregaron transformaciones para poder modificar la ubicación de su centro, rotación y tamaño sin tener que modificar el archivo original en algun programa de diseño.

Campo	Descripción
nombre_archivo_obj	El nombre del archivo obj con la definición del objeto 3d
nombre_archivo_textura	El nombre del archivo jpeg con la textura del objeto 3d
traslación (x,y,z)	Traslación del molde del objeto (para modificar la posición de su centro)
vect_rot (x,y,z)	Componentes del vector de rotación
angulo_rotación	Ángulo que se rota el molde del objeto. (En conjunto con el vector de rotación)
factor_escala	Factor de escala del molde del objeto

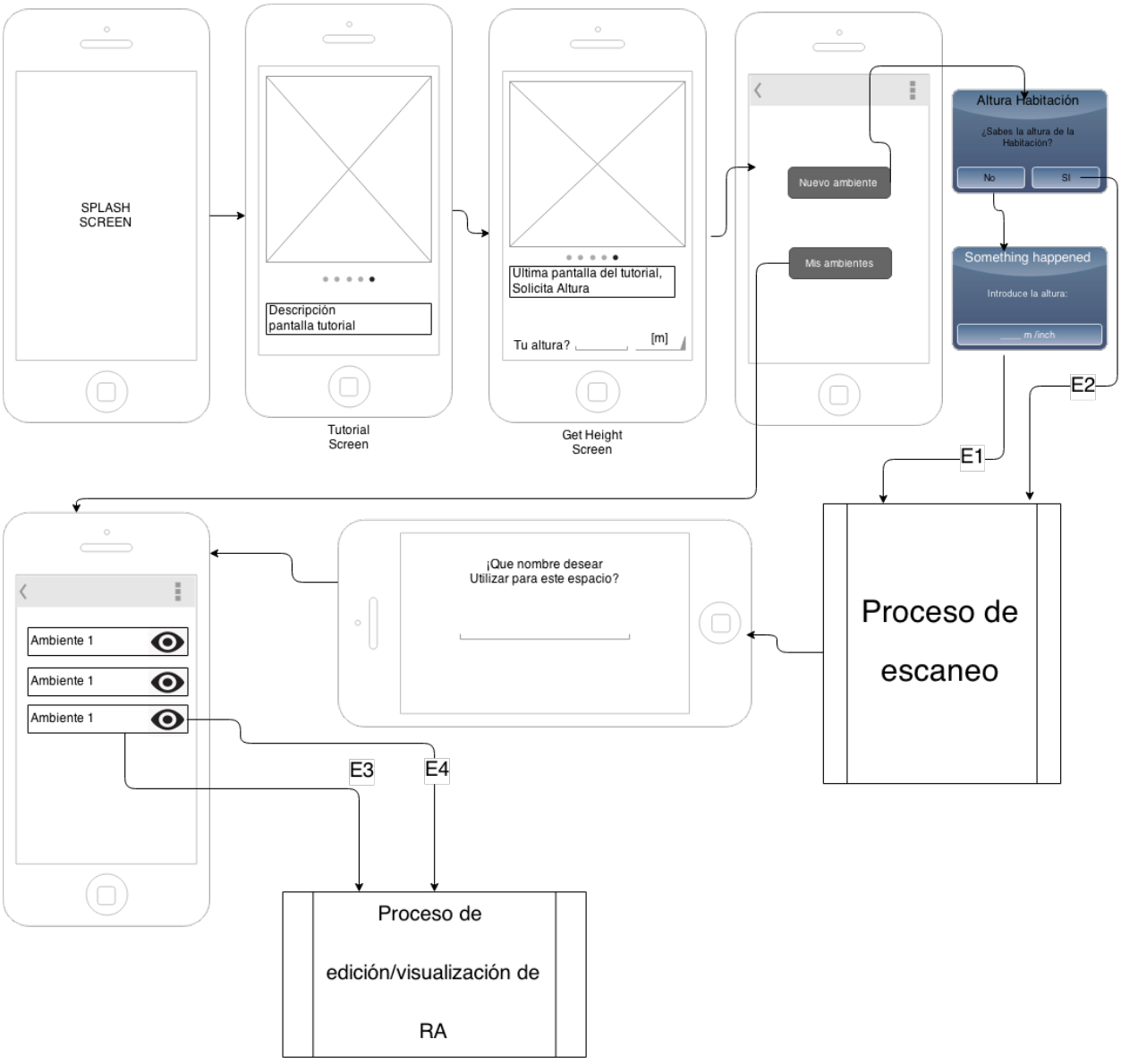
Instancia Modelo

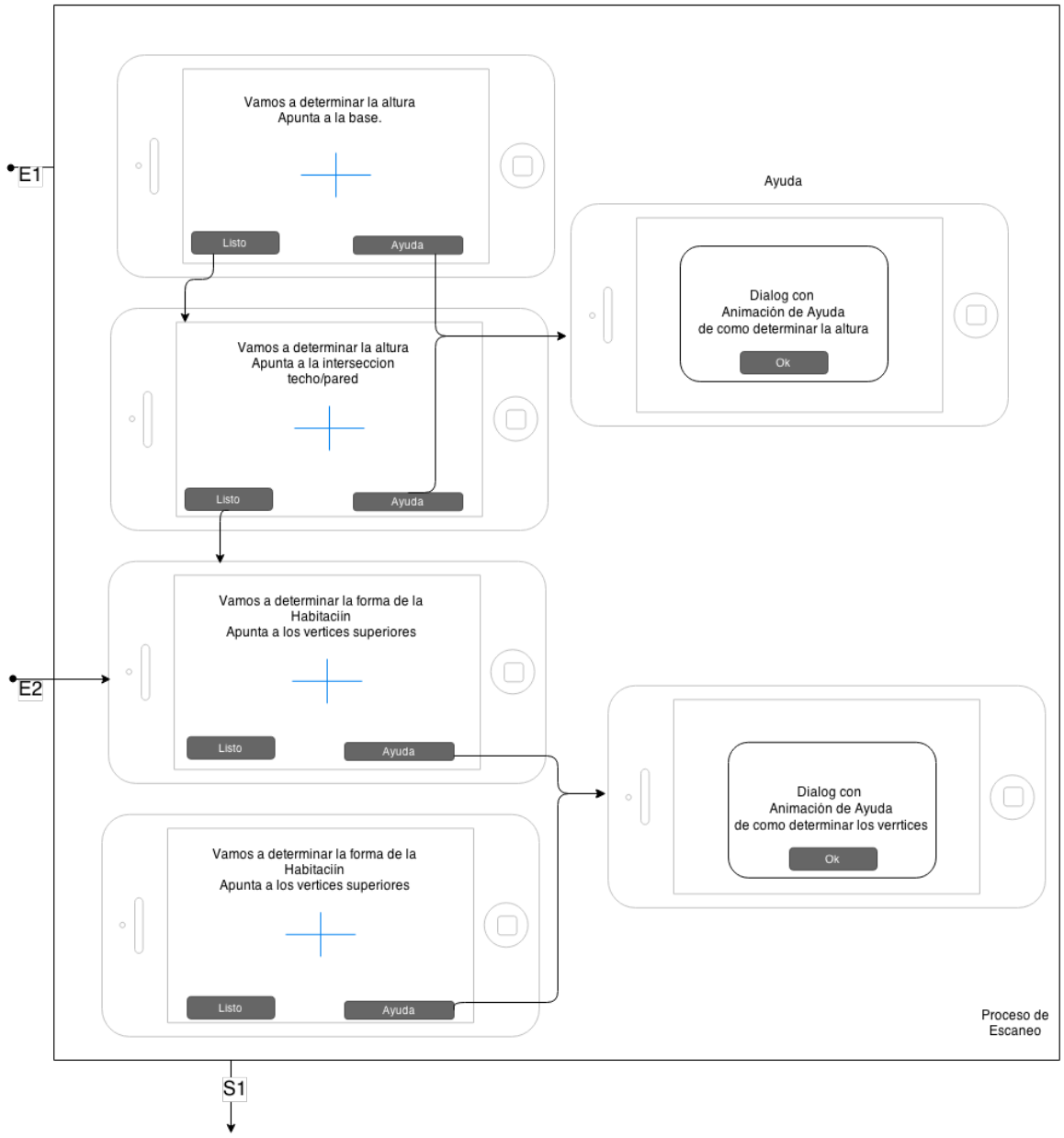
Representa un objeto 3D agregado por el usuario.
Dicho objeto está relacionado con un molde de objeto 3d y con el ambiente virtual en el que será colocado.
Debido a que estos objetos se encuentran “empotrados” en las paredes del ambiente sus transformaciones de rotación y traslación son restringidas.

Campo	Descripción
modelo_diccionario_id	El molde 3D de la instancia

ambiente_virtual_id	El ambiente en el que se coloca el objeto virtual
pared_anclada	La pared en la que se coloca el objeto virtual
factor_escala	El factor de escala del objeto virtual
angulo_rotacion	Ya que el objeto se encuentra colocado en una pared, la rotación está restringida en un solo plano y por ende solo se índica el ángulo de rotación sobre el mismo.
factor_traslacion (a y b)	Ya que el objeto se encuentra colocado en una pared, la traslación está restringida en un solo plano. Los vectores a y b son paralelos al plano y forman un ángulo de 90 grados entre sí. Cualquier punto de la pared se puede obtener de la suma de los vectores multiplicados por un escalar respectivamente (factor de traslación)

Mocks de la aplicación




E1
Vamos a determinar la altura
Apunta a la base.
Listo
Ayuda
Ayuda
Dialog con
Animación de Ayuda
de como determinar la altura
Ok
Vamos a determinar la altura
Apunta a la interseccion
techo/pared
Listo
Ayuda
E2
Vamos a determinar la forma de la
Habitaciin
Apunta a los vertices superiores
Listo
Ayuda
Dialog con
Animación de Ayuda
de como determinar los verrtices
Ok
Vamos a determinar la forma de la
Habitaciin
Apunta a los vertices superiores
Listo
Ayuda
Proceso de
Escaneo
S1

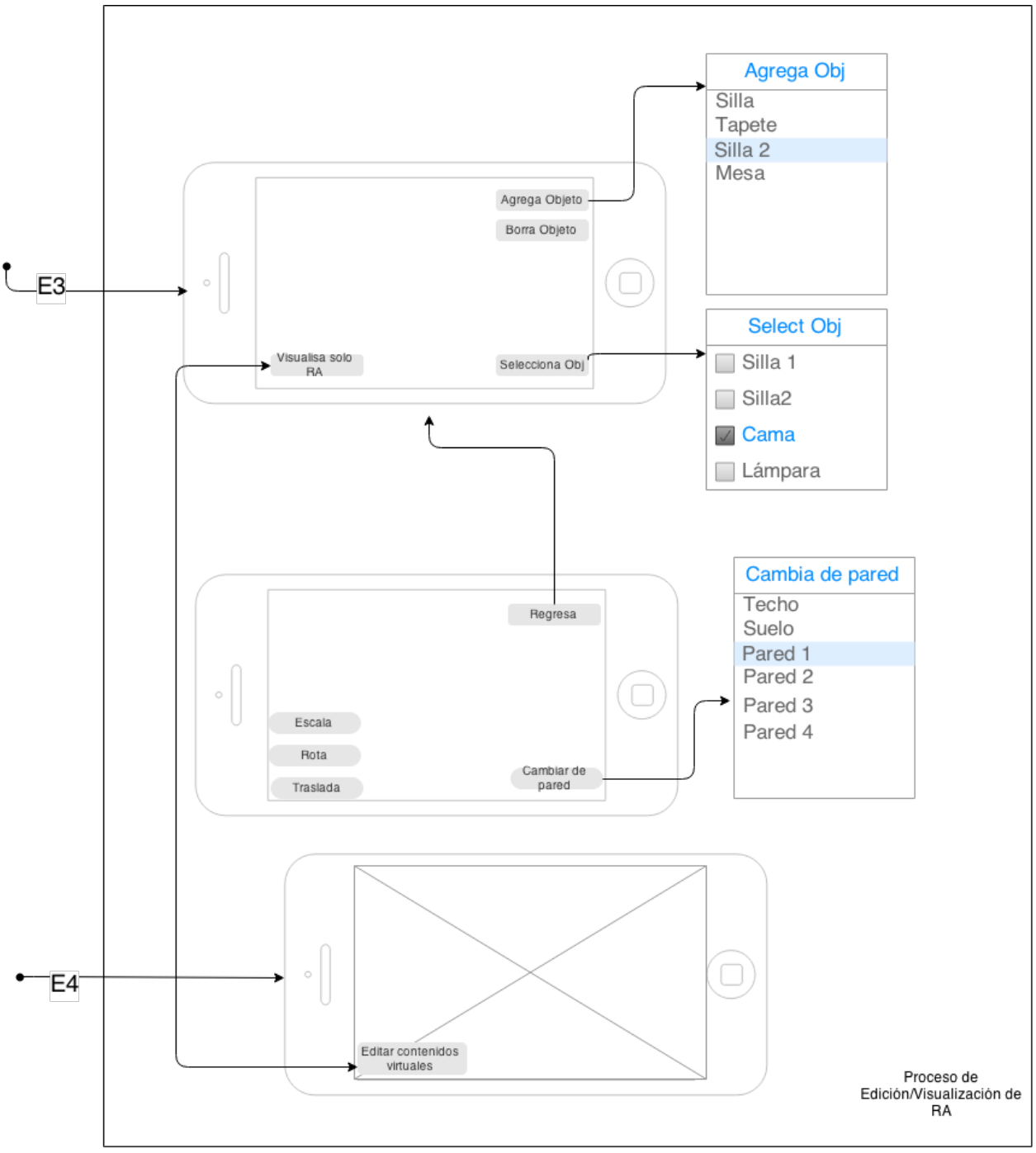


Figura 5.3
Mocks de la aplicación

Arquitectura de la aplicación

Los componentes de la aplicación se agrupan en módulos como se ilustra en el siguiente diagrama:

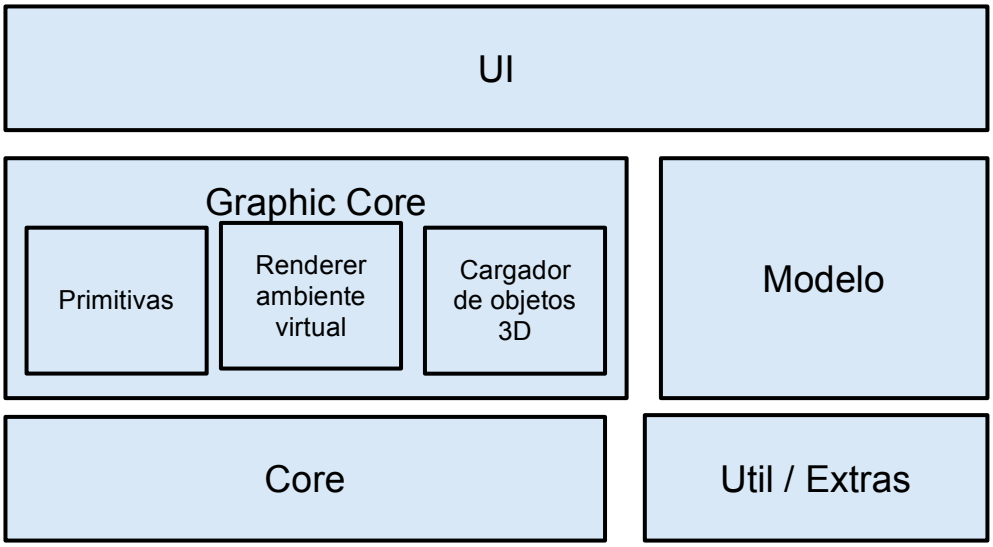


Figura 5.4
Arquitectura de componentes de la aplicación

Core

Con las clases que contienen las definiciones de constantes y configuraciones que afectan a todo el proyecto.

Util / Extras

Este módulo contiene herramientas útiles para el proyecto. APIs, Clases y librerías propias y de terceros. (Api para leer QR, Librería de matemáticas, Librería de ORM y otras utilidades)

Graphic Core

Contiene las clases relacionadas al uso de gráficos OpenGL. En este módulo se encuentran las definiciones de clases java para dibujar primitivas (triángulos, cubos, esferas, etc), el cargador de modelos 3D y el renderer del ambiente virtual.

Modelo

Módulo con las clases que definen las entidades del Modelo.
En el proyecto se utiliza ORMLite para el almacenamiento estructurado de la información.

UI

Vistas, fragments y demás clases relacionadas con la interfaz gráfica de la aplicación.

Desarrollo de la aplicación

Manejo de Sensores en la plataforma Android

El enfoque trigonométrico para la medición de distancias se basa en conocer la orientación del dispositivo, y por ende los ángulos que se forman con éste y otros marcos de referencia. En la actualidad, los sensores presentes en la mayoría de los dispositivos móviles permiten conocer la orientación del dispositivo (Además de otras variables tales como la aceleración, campo magnético, presión, temperatura, entre otras).

Sistemas de coordenadas

Para entender los valores arrojados por los sensores de movimiento y posición, es necesario conocer primero los sistemas de coordenadas que utiliza Android: el sistema de coordenadas global (x_E,y_E,z_E) y el sistema de coordenadas del dispositivo (x , y, z)

Figura 5.5
Sistemas de coordenadas locales en la plataforma Android

Las coordenadas locales, tienen como marco de referencia el dispositivo. Y su disposición es la siguiente[25]:
El eje *z*, perpendicular a la pantalla y en dirección de la misma. El eje *y* apuntando hacia arriba y el eje *x* hacia la derecha. Este sistema es fijo, y nunca cambia aunque cambie la posición u orientación del dispositivo.

[25] En dispositivos con orientación landscape la disposición del sistema de coordenadas locales cambia. http://android-developers.blogspot.mx/2010/09/one-screen-turn-deserves-another.html

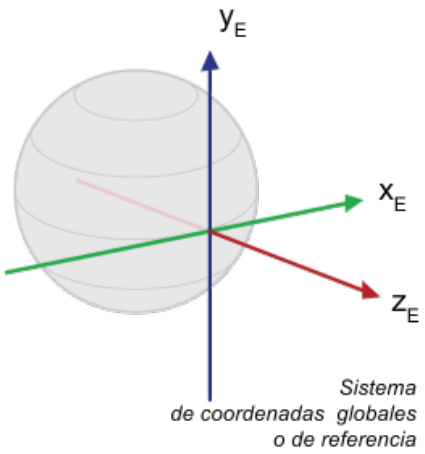


Figura 5.6
Sistemas de coordenadas globales en la plataforma Android

Las coordenadas globales, tienen como marco de referencia el globo terrestre y su disposición es la siguiente:
El eje Z_E apunta hacia el centro de la tierra y es perpendicular a la superficie terrestre.
Y_E es tangencial a la tierra en la posición del usuario, y apunta hacia el norte magnético.
X_E Es el producto *YZ*, y es tangencial a la tierra en la posición del usuario. Apunta hacia el oeste.

Sensores soportados por la plataforma Android

La plataforma Android agrupa los sensores soportados según su tipo:

Sensor	Tipo	Descripción	Usos Comunes
Sensores de Movimiento			
Acelerómetro	Hardware	Mide la fuerza de aceleración en m/s^2 que se aplica al dispositivo en los 3 ejes físicos (x,y,z), incluyendo la fuerza de gravedad.	Detección de movimiento. (shake, titlt, etc)
Gravedad	Software or Hardware	Mide la fuerza de gravedad en m/s^2 que está siendo aplicada al dispositivo en los 3 ejes físicos	Detección de movimiento. (shake, titlt, etc)

		(x,y,z)	
Girsocopio	Hardware	Mide la tasa de rotación del dispositivo en rad/s al rededor de cada uno de los 3 ejes físicos (x,y,z)	Detección de rotación. (spin, turn, etc.).
Aceleración lineal	Software or Hardware	Mide la fuerza de aceleración en m/s^2 que se aplica al dispositivo en los 3 ejes físicos (x,y,z), excluyendo la fuerza de gravedad.	Monitoreo de la aceleración en un solo eje.
Sensores de Posición			
Campo magnético (brújula)	Hardware	Mide la intensidad del campo magnético ambiental para cada uno de los 3 ejes físicos (x,y,z) en μT.	Crear una brújula.
Orientación (Obsoleto)	Software	Mide el grado de rotación que el dispositivo forma con los 3 ejes físicos.	Determinar la posición del dispositivo
Proximidad	Hardware	Mide la proximidad de un objeto (en cm) relativa a la pantalla del dispositivo.	Determinar si el dispositivo está en contacto con la oreja del usuario.
Vector de rotación	Software or Hardware	Mide la orientación del dispositivo proveyendo los tres elementos del vector de rotación del dispositivo.	Detección de movimiento y rotación.
Sensores Ambientales			
Temperatura ambiental	Hardware	Mide la temperatura ambiental en grados Celsius	Monitoreo de temperatura
Luz	Hardware	Mide los niveles de luz ambiental (iluminación) en lx (lux)	Control de brillo de pantalla.
Presión	Hardware	Mide la presión ambiental del	Monitorear

		aire en hPa o mbar.	cambios en la presión del aire.
Humedad relativa	Hardware	Mide la humedad ambiental relativa en porcentaje (%)	Monitorea el rocío, humedad absoluta y relativa.

Figura 5.7

Tabla. Sensores soportados en la plataforma Android

Para el prototipo de aplicación, los sensores sobre los que nos interesamos son aquellos que nos permitan determinar la orientación del dispositivo en el espacio. (Orientación, Acelerómetro, Campo Magnético, Giroscopio, Vector de rotación)

- Sensor de Aceleración

El sensor de aceleración `Sensor.TYPE_ACCELEROMETER` mide la aceleración aplicada al dispositivo, incluyendo la gravedad. Eso significa, que cuando el dispositivo, se encuentra en reposo en una superficie plana con la pantalla hacia el cielo, la lectura del sensor será (0 , 0 , -9.81[m/s^2])

El acelerómetro suele ocuparse para determinar si se muestra la pantalla en posición portrait o landscape, dependiendo si el dispositivo se encuentra en posición vertical u horizontal. Usando solamente este sensor no es posible determinar por completo la orientación del dispositivo (Pedley M, 2013 p.9) , para ello es necesario fusionar sus datos con los de otros sensores como un detector de campo magnético (Ozyagcilar T. , 2012) como se verá más adelante.

- Sensor de Giroscopio

Este sensor mide la velocidad de la rotación del dispositivo en cada uno de los ejes X_E,Y_E y Z_E en radianes por segundo (rad/s)

El uso habitual de este sensor es medir el ángulo de rotación del dispositivo, integrando los valores censados en el tiempo. En la práctica ningún sensor es totalmente exacto, y en el caso del giroscopio las mediciones del ángulo del dispositivo se desfasan al poco tiempo como consecuencia del ruido del sensor, agravado aún más luego de la integración. Es por ello que el uso del giroscopio suele ser junto a otros sensores que calibran y revierten el constante desfase.

- Sensor de Campo Magnético

El sensor de campo magnético detecta la intensidad del campo magnético terrestre en cada uno de los 3 ejes. La magnitud del campo magnético varía según la posición del usuario en el globo terrestre.

Estos sensores suelen ser impresos al medir la intensidad absoluta del campo. Los valores censados varían dependiendo de factores externos tales como la presencia de objetos metálicos, o bien el historial de mediciones del dispositivo.

Determinación de la orientación del dispositivo

Cuando las coordenadas del dispositivo se encuentran alineadas a las coordenadas globales, la rotación en cada eje es 0. Es decir $\varphi=\gamma=\psi=0$. (Ver tracking del usuario)
Esto ocurre cuando el equipo se encuentra sobre una superficie plana, paralela al suelo, con la pantalla hacia arriba y alineado de norte a sur. En cualquier otra orientación, se dice que el dispositivo se encuentra rotado.

La orientación del dispositivo en el espacio, está dada por la secuencia de rotaciones sobre los ejes X_E *(azimutal)*, Y_E *(roll)* , Z_E *(pitch)*. Estas rotaciones pueden estar expresadas en una sola matriz de rotación o bien pueden obtenerse como un arreglo que contenga los 3 ángulos.

Hasta antes de la versión 2.2, el método para obtener la rotación del dispositivo era utilizando el sensor de orientación (`TYPE_ORIENTATION`). Los valores retornados por el sensor son los ángulos azimutal, pitch y roll.

Actualmente el sensor ha quedado descontinuado y los métodos para conocer la orientación del dispositivo, se enfocan en obtener primero una matriz de rotación y a partir de ella se pueden obtener los ángulos de rotación sobre cada eje utilizando el método:

```
SensorManager.getOrientation (float[] R , float[] values)
```

Donde R es una matriz de rotación, y values es un arreglo de 3 dimensiones donde se guarda el resultado.

Asumiendo que la posición inicial de un dispositivo es cuando se encuentra sobre una mesa, con la pantalla hacia arriba y con la parte superior del dispositivo apuntando hacia el norte, el significado de los 3 ángulos es el siguiente:

`values[0]` = Azimuthal = rotación sobre el eje Z_E
El dispositivo reporta 0 radianes en la posición inicial (La parte superior apuntando hacia el norte).
Si la parte superior apunta hacia el este, `values[0]` = $\pi/2$, si apunta hacia el oeste `values[0]` = $-\pi/2$ y si apunta hacia el sur `values[0]` = π.

`values[1]` = Pitch = rotación sobre el eje X_E

El dispositivo reporta 0 radianes en la posición inicial. Cuando la parte superior del dispositivo se levanta, de la mesa quedando parado con la pantalla hacia el frente, `values[1]` = -π/2. Si la pantalla esta de espaldas, `values [1]` = π/2. Y si el dispositivo está reposando sobre la mesa, con la pantalla boca abajo, `values[1]` = π

`values[2]` = Roll = Rotación sobre el eje Y_E
El dispositivo reporta 0 radianes en la posición inicial. Cuando se gira el dispositivo, de forma que la pantalla apunta al oeste, `values[2]` = -π/2.
Cuando se gira el dispositivo, y la pantalla apunta hacia el este, `values[2]` = π/2. Girando el dispositivo, dejando la pantalla boca abajo `values[2]` = π

Aunque a primera vista, pareciera que es más complicado e innecesario obtener una matriz de rotación para luego obtener la rotación del dispositivo en lugar de utilizar directamente el sensor de orientación (descontinuado), existen ventajas en este nuevo enfoque.

El sensor de orientación `SENSOR_TYPE_ORIENTATION`, funciona mezclando la información "cruda" de los sensores de acelerómetro y campo magnético. Este proceso es transparente para el usuario, y elimina la posibilidad de utilizar filtros en dichos sensores. Otra desventaja derivada es que el sensor de orientación funciona únicamente con los dos sensores antes mencionados, eliminando la posibilidad de incluir información de de otros sensores (como el giroscopio) para mejorar las estimaciones.

Si bien, el formato de la información sensada por el sensor de orientación es fácilmente entendible (los valores de rotación en cada uno de los 3 ángulos), el cómputo de la información no es del todo eficiente como lo es el uso de matrices de rotación.

Usando una matriz de rotación es posible cambiar la disposición default del sistema de coordenadas locales (portrait / landscape). Por default la disposición es con el dispositivo en portrait.

Matrices de Rotación

Las matrices de rotación en Android tienen la siguiente forma:

$$R = \begin{pmatrix} cos\phi\ cos\psi - sin\phi \sin\psi\ sin\theta & sin\phi\ cos\theta & cos\phi\ sin\psi + sin\phi\ cos\psi \sin\theta \\ -sin\phi\ cos\psi - cos\phi\ sin\psi\ sin\theta & cos\phi cos\theta & cos\phi\ cos\psi\ sin\theta - sin\phi sin\psi \\ -sin\psi cos\theta & -sin\theta & cos\psi cos\theta \end{pmatrix}$$

Y es el resultado de la multiplicación de matrices:

$$\begin{pmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} \cos(\psi) & 0 & \sin(\psi) \\ 0 & 1 & 0 \\ -\sin(\psi) & 0 & \cos(\psi) \end{pmatrix}$$

Cada una de estas matrices es la rotación 2D sobre los planos XY , XZ , YZ

Utilizar matrices de rotación, en lugar de ángulos euler es conveniente en una aplicación de realidad aumentada por múltiples factores.

- La obtención de dicha matriz es directa a partir de los valores de los sensores de Acelerómetro y Campo Magnético.
- La matriz transforma cualquier vector del sistema de coordenadas del dispositivo al sistema de coordenadas globales.
- Aplicar una matriz de rotación, que representa la orientación del dispositivo en la tierra, en una escena virtual, dará la sensación de que los objetos se encuentran fijos en el espacio y genera la sensación de existencia de ellos en el mundo real.
- El cómputo de la orientación del dispositivo es más eficiente cuando ésta está representada en una sola matriz de rotación y no en 3 rotaciones (una por cada eje). Las matrices de rotación se aplican directamente en OpenGL

Dependiendo de los sensores que queramos utilizar, la plataforma Android dispone distintos métodos para obtener la matriz de rotación con la orientación del dispositivo.

Obtención de matrices de Rotación:

Usando sensores Acelerómetro y de Campo magnético:
Es posible determinar la orientación del dispositivo, a partir de los valores censados por los sensores de acelerómetro y de campo magnético usando el método de la plataforma Android:

```
SensorManager.getRotationMatrix( float[] R , float[] I , float[]
gravity , float[] geomagnetic )
```

El método recibe como parámetros:
R , un arreglo donde se guardará la matriz de rotación computada
I , un arreglo donde se guardará la matriz de inclinación computada.
gravity , el arreglo de valores con la información recopilada por el acelerómetro

geomagnetic , el arreglo de valores con la información recopilada por el sensor de campo magnético.

La matriz de rotación será inválida si el dispositivo se encuentra acelerando o en caída libre. Tampoco lo será cuando el dispositivo se encuentre cerca del norte magnético o bien cerca de otro campo magnético.

Usando Vector de rotación:

Los diversos sensores incluidos en los dispositivos móviles tienen cualidades y desventajas. En la práctica, las lecturas de distintos sensores pueden complementarse para obtener mejores resultados.
Por ejemplo, el acelerómetro responde rápidamente a cambios, sin embargo es ruidoso. Mientras que los ángulos medidos por el giroscopio (luego de integrar la velocidad angular) tienen poco ruido, sin embargo se desfasan rápidamente. Por tanto, un enfoque para determinar la orientación del dispositivo podría ser utilizar principalmente los ángulos medidos por el giroscopio y prevenir su desfase, calibrando constantemente usando el vector de aceleración que apunta hacia el centro de la tierra y no se desfasa.

El sensor de Vector de Rotación funciona precisamente fusionando las lecturas de los sensores de acelerómetro, campo magnético y giroscopio. El sensor se encuentra presente en dispositivos limitados y su implementación es propietaria, desarrollada por la empresa Invensense.

La salida de este sensor es en una forma similar a un cuaternión, que es una representación alternativa de una rotación con ventajas sobre ángulos euler, como la eliminación del gimbal-lock.
Al representar rotaciones, un vector de rotación puede ser convertido en una matriz de rotación. Para ello, Android nos provee del método:

```
SensorManager.getRotationMatrixFromVector(float[] R, float[] rotationVector)
```

El método recibe como parámetros:
R , un arreglo donde se guardará la matriz de rotación computada
rotationVector , el vector arrojado por el sensor de Vector de Rotación

Remapeo de coordenadas

Como se mencionó anteriormente, las matrices de rotación obtenidas con los sensores son utilizadas directamente en OpenGL para efectuar rotaciones en el mundo virtual. Esto es posible cuando la posición de los gráficos en la pantalla corresponden con la posición portrait vertical. En el caso del prototipo a desarrollar, la posición de la pantalla es landscape.

Será necesario cambiar el sistema de coordenadas locales y expresar las matrices de rotación en el nuevo sistema. (Fig 5.8)

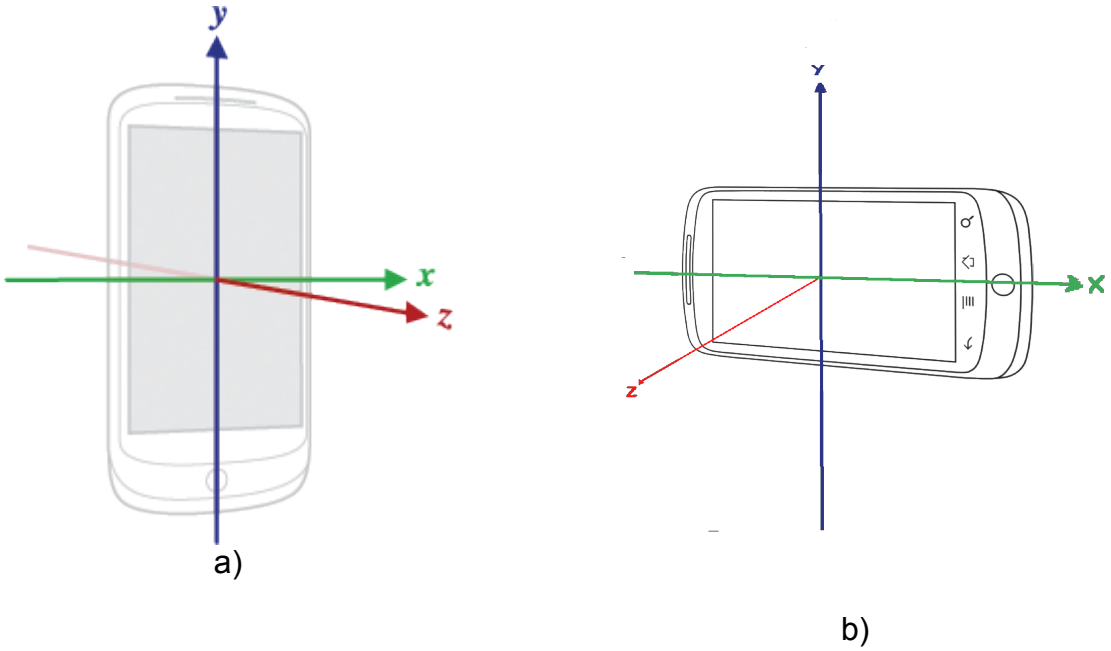


Figura 5.8

Disposición del sistema de coordenadas locales, luego de un remapeo de coordenadas

a) Disposición inicial

b) Disposición del sistema de coordenadas locales luego de la reasignación

Para ello, Android nos provee el método:

```
SensorManager.remapCoordinateSystem(float[] inR, int X , in Y,
float[] outR)
```

El método recibe como parámetros:

R , un arreglo donde se guardará la matriz de rotación computada

X , con las coordenadas locales alineadas a las coordenadas globales, indica en qué eje global y que dirección el eje x del dispositivo será mapeado
Y, con las coordenadas locales alineadas a las coordenadas globales, indica en qué eje global y que dirección el eje y del dispositivo será mapeado
outR , la nueva matriz de rotación

Análisis de los métodos para obtener la orientación

Como se mencionó, en la sección anterior, contamos principalmente con 3 métodos para la obtención de la orientación del dispositivo:

- Utilizando el sensor de orientación (descontinuado).
- Utilizando la fusión de la información de los sensores de acelerómetro y campo magnético.
- Utilizando el sensor de vector de rotación que fusiona acelerómetro, campo magnético y giroscopio.

La primera opción puede ser descartada por ser un método descontinuado y por las desventajas del mismo antes mencionadas.

Estudiando el par de métodos restantes, se identificó que cada uno cuenta con puntos a favor y en contra, que se describen a continuación.

Ruido

El sensor de vector de rotación, es un sensor por software en la mayoría de los dispositivos. Fusiona los valores del acelerómetro, campo magnético, giroscopio y los valores que arroja han sido filtrados de ruido previamente.

Sin embargo, la obtención de la rotación del dispositivo a partir de los datos del sensor de orientación y de campo magnético no utiliza ningún tipo de filtro, y arroja los resultados crudos.

Las figura 5.9, muestra las gráficas de rotación del dispositivo (en ángulos euler) obtenidos por ambos métodos, cuando el dispositivo se encuentra en reposo sobre una superficie plana y luego gira repetidas veces sobre un mismo eje.
Se observa claramente que los valores utilizando sensor de rotación han sido procesados.

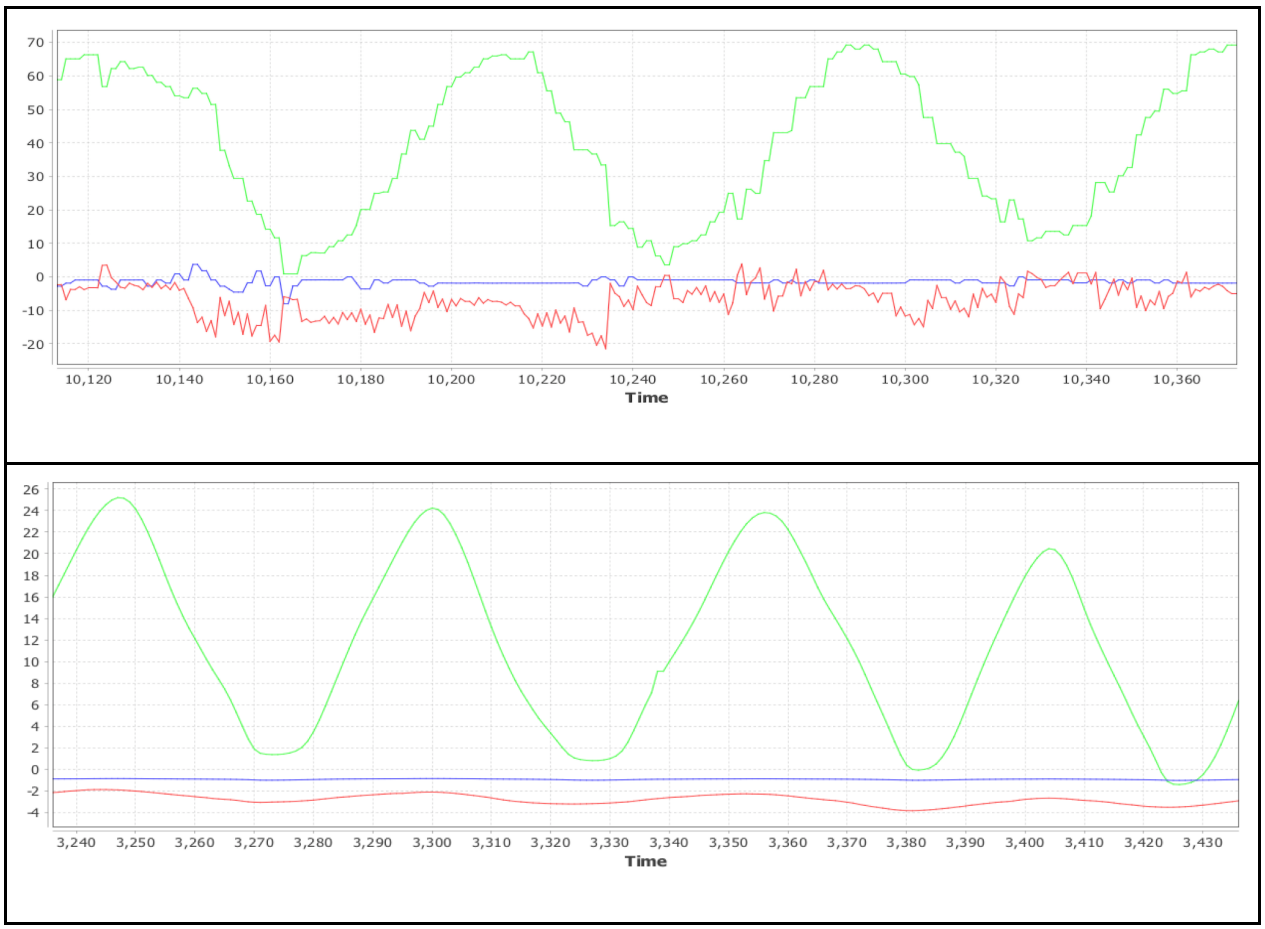


Figura 5.9

Comparación de la orientación del dispositivo obtenida a partir de distintos sensores.
a) Orientación a partir de los sensores de aceleración y campo magnético
b) Orientación a partir del sensor de Vector de Rotación

Es posible utilizar un filtro pasa-bajos para mitigar los efectos del ruido. Para ello se implementó un filtro por suavizado por peso. (Wrox, p108)

Donde:
(Nuevo valor) = (Último Valor) + x_i * a - (Last Value) * a
Donde ***a*** es el parámetro de suavizado. Entre más cerca se acerque el valor de ***a*** a 1, menor será el suavizado. Mientras que entre más se acerque el valor de ***a*** a 0, mayor será el suavizado, y la respuesta del sensor será más lenta.

Las siguientes gráficas (fig 5.10), muestran la rotación del dispositivo (en ángulos euler) utilizando la fusión de los sensores de acelerómetro y campo magnético, cuando el dispositivo se encuentra en reposo sobre una superficie plana y luego gira repetidas veces sobre un mismo eje.
A los valores obtenidos se les aplica un filtro pasa-bajos, con el método de suavizado por peso, y distintos valores en el parámetro **a**.

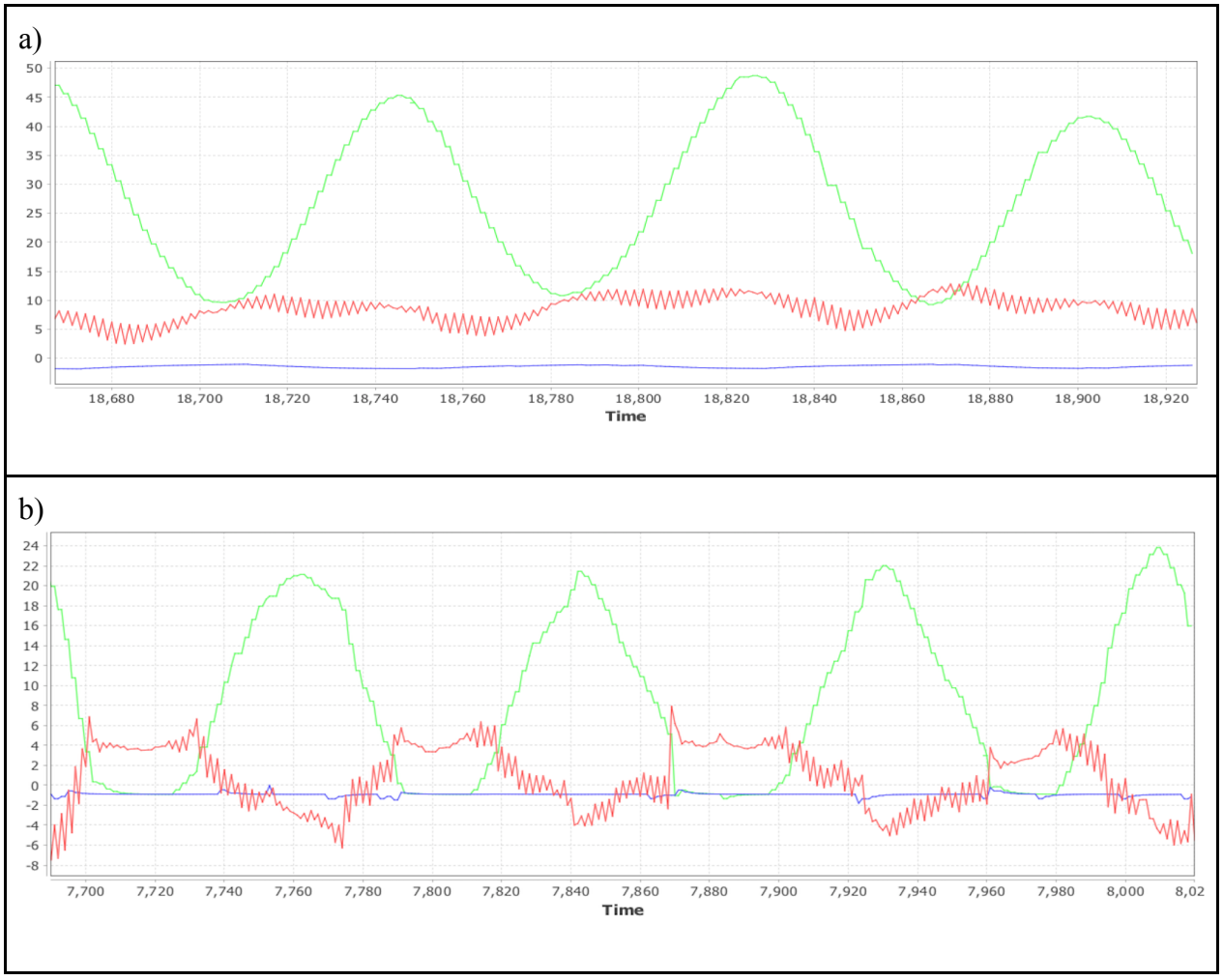


Figura 5.10

Orientación a partir de los sensores de aceleración y campo magnético aplicando un filtro pasabajos.
a) a=0.1 , b) a=0.5

Información primaria

Si bien es posible obtener la rotación del dispositivo a partir de los valores arrojados por los sensores de acelerómetro y campo magnético, se puede mejorar la estimación si se fusiona

también los valores sensados por el giroscopio. Dicha fusión la implementa el sensor de vector de rotación.

Fragmentación en la Implementación

En el caso del sensor de rotación, la implementación de la fusión de los sensores es particular de cada fabricante, y por tanto existe poco control sobre la información arrojada. Durante la experimentación se encontró el caso del Galaxy S3 mini (D2), cuya implementación del sensor de rotación tiene un retraso en los valores leídos.
En el caso del caso práctico este retraso ocasiona errores, ya que el usuario tiene que permanecer apuntando a algún vértice hasta que los valores se estabilicen, para evitar introducir datos falsos.

Corrección del error Humano

La corrección de la fluctuación aleatoria de los valores leídos por los sensores es corregida por medio de filtros.
Otro tipo de errores en la medición son aquellos que ocurren causados por equivocaciones humanas.
El uso de la aplicación contempla que el usuario mire fijamente hacia los distintos vértices de la aplicación y los seleccione dando un toque en la pantalla.
Ya sea por los movimientos inherentes al pulso del usuario, o por el movimiento involuntario del dispositivo al recibir el touch, puede existir un error en la lectura de los sensores.
Para abordar este problema, se implementa el uso de promedios de las últimas X mediciones. De esta forma si el usuario se encuentra mirando fijamente a un vértice, y al seleccionarlo se desvía del punto que había estado enfocando, la afectación en la lectura del sensor será mínima.

Las figura 5.11, muestra las gráficas de rotación del dispositivo (en ángulos euler, utilizando el sensor de vector de rotación), siendo sostenido a mano en una posición fija (en la medida de lo posible) y recibiendo touches constantes, con y sin aplicación de promedios.

a)

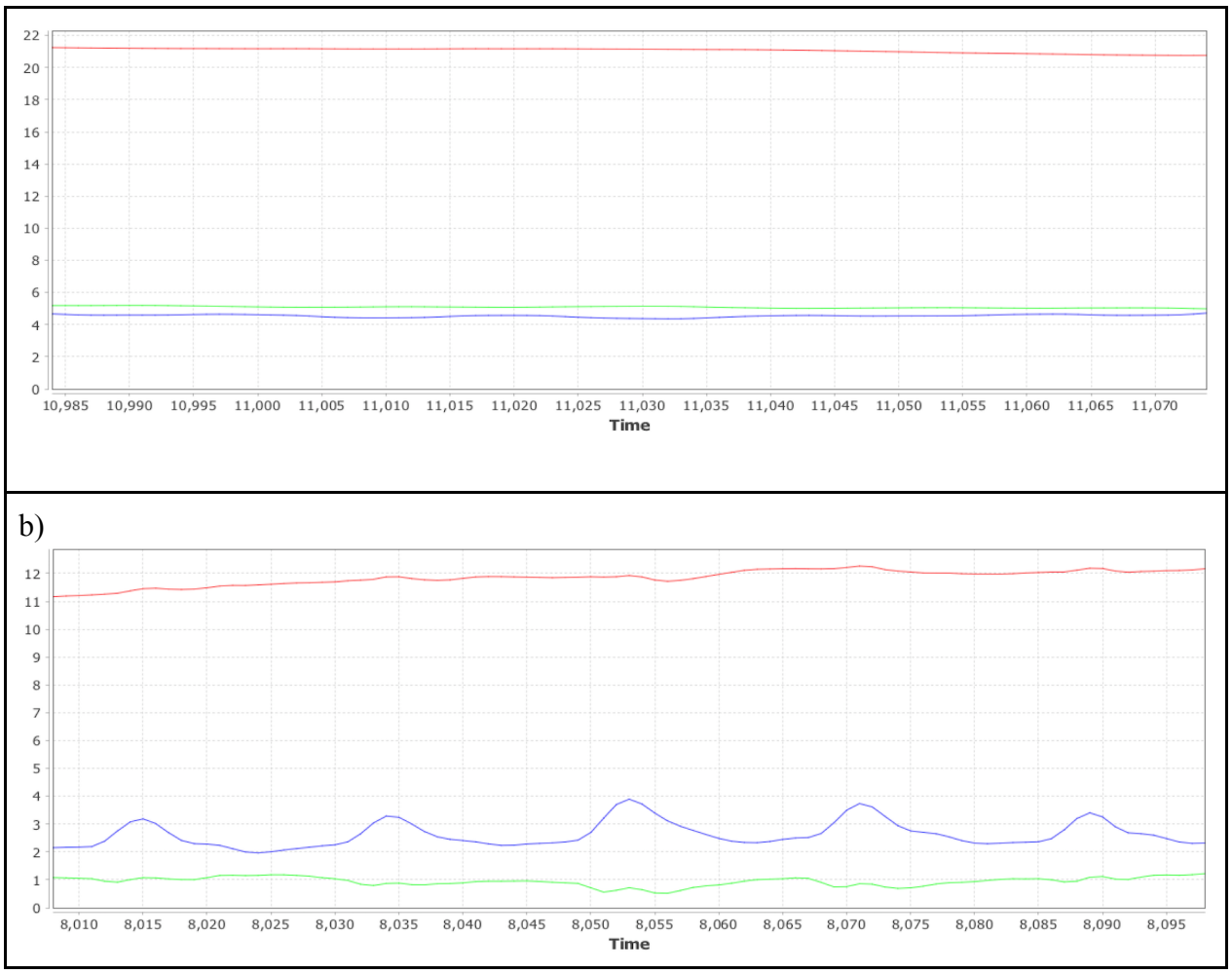


Figura 5.11

Corrección del error humano, aplicando promedio de los últimos n valores. Rotación del dispositivo mientras la pantalla es tocada en repetidas ocaciones.

a) Aplicando promedios de los últimos 50 valores (Las variaciones en la orientación por el touch son eliminadas, e inclusive las variaciones causadas por el pulso del usuario)

b) Sin aplicar promedios

Consideraciones de OpenGL

Ajuste del View Frustum

Para lograr el objetivo de elaborar una aplicación de Realidad Aumentada, es imprescindible que tanto el mundo virtual como el real se acoplen el uno con el otro. Para el caso de la aplicación a desarrollar, debe ocurrir que la cámara virtual de OpenGL tenga los mismos parámetros de

configuración que la cámara del dispositivo. De esta forma los objetos virtuales (OpenGL) serán dibujados en pantalla como si en realidad fueran vistos desde la cámara real.

La cámara virtual se configura con el método
`GL.gluPerspective(gl , fovy, aspect, zNear , zFar)`

Que define el volumen de visión. Todo lo que se encuentre dentro de dicho volumen será mostrado en pantalla.
Los parámetros que lo definen son:

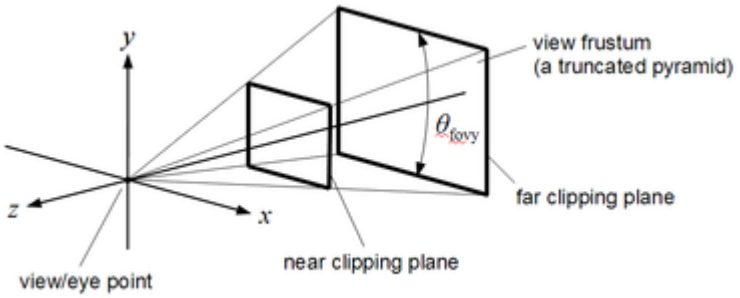


Figura 5.12
Componentes del view frustrum, OpenGL.

fovy (field of view) Especifica el campo de visión en grados, en la dirección del eje *y*
aspect Especifica el radio de aspecto que determina el campo de visión en la dirección de x. El radio de aspecto es el ancho/alto del viewport
zNear , zFar Son los valores de distancia en los que se ubican los planos zNear y zFar. Solo los objetos que se encuentren en el espacio acotado por estos dos planos serán dibujados.

Como se ha mencionado, es primordial configurar este view frustum para que coincida con el de la cámara real. Los campos de visión, tanto en *x* como en *y* de la cámara virtual deben de ser los mismos a los de la cámara real del dispositivo. Desde la plataforma Android se proveen los mecanismos para obtener la relación de aspecto de las imágenes que captura la cámara. Bastará con calcular el fov en la dirección de *x* o en la dirección de *y*. El otro se puede calcular usando la relación de aspecto.

En cuanto a los parámetros zNear/zFar, se trata de parámetros que se encuentran presentes únicamente en cámaras virtuales. Se utilizarán los valores 0.3 y 30, que proporcionan una profundidad de visión adecuada para los propósitos de la aplicación.

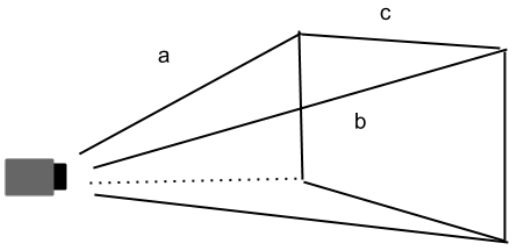
La plataforma Android provee de los métodos: `Camera.Parameters.getVerticalViewAngle()`, y `Camera.Parameters.getHorizontalViewAngle()` que retornan los ángulos de visión para x y y respectivamente. Sin embargo dichos métodos no están 100% soportados por todos los dispositivos. En el caso de los dispositivos de prueba con los que se contó, arrojan el valor constante de 360[26], para ello fue necesario obtener manualmente los ángulos de visión a partir del siguiente método.

1. Se coloca el dispositivo fijo, enfocando hacia alguna pared.
2. Luego se marcan en la pared los límites superiores en los que la cámara alcanza a ver.
3. Se miden las distancias de los segmentos a , b , c.

El fov horizontal se determina por ley de cosenos:

$$c^2 = a^2 + b^2 - 2ab\cos\gamma$$

$$\gamma = angcos\ (\frac{c^2 - a^2 - b^2}{-2ab})$$



El fov vertical se calcula exáctamente con el mismo método.

En las siguientes imágenes se muestra la vista previa de la cámara y encima, una vista openGL con un prisma de las dimensiones de la habitación.
Cuando los parámetros de la cámara virtual (OpenGL) son los mismos que los de la cámara real, las aristas del prisma deben coincidir con las aristas reales de la habitación.

[26] Bad Programming even at great leagues.
En muchas implementaciones de Android, los métodos *getVerticalViewAngle / getHorizontalViewAngle* arrojan un valor constante, con el objetivo de pasar los tests automatizados.

Figura 5.14

Comparación de la superposición de objetos virtuales, modificando el parámetro de zoom

a) View frustrum ajustado a la cámara real, el fov es 52.5

b) View Frustrum con mayor zoom al de la cámara real, el fov es 40.5

c) View frustrum con menor zoom al de la cámara real, el fov es 74.5

Carga de objetos 3D

Existen diversos frameworks para poder cargar objetos 3D en Android. Algunos son game engines completos mientras que otros se enfocan únicamente a la carga de objetos virtuales. Algunos ejemplos son: Min3D, AndEngine, LibGDX, GPCT, Unity 3D.
Sin embargo después de estudiarlos, se encontró que presentan diversos problemas. Algunos son proyectos no consolidados, proyectos con poca o nula documentación y años sin soporte o nuevas actualizaciones.

En el caso de los Game Engines disponibles en el mercado, sin duda se trata de plataformas consolidadas, estables, con mucha documentación y ayuda en foros de desarrollo. Sus desventajas son más bien la curva de aprendizaje, ya que en su mayoría se trata de herramientas que funcionan con un IDE y lenguaje de programación distintos al nativo de la plataforma Android. Otra desventaja es el costo de dichas herramientas. (Por ejemplo Unity3D $75 USD al mes)

Al momento de enfrentar el problema de la carga de objetos, el flujo de la aplicación y el proceso de reconocimiento de medidas de espacios interiores ya estaba implementado. Volver a desarrollar la aplicación desde cero, solo para poder cargar objetos virtuales no era una opción viable.

Para la implementación de este prototipo, se decidió elaborar un cargador de modelos simple, basado en el cargador de Zechner (Zechner, 2012, p. 569)
El código completo se adjunta en el apéndice A.

El cargador implementado soporta objetos Wavefont (*.obj) con características particulares.

- Las caras de los objetos deben ser triángulos

- Opcionalmente pueden contener coordenadas de texturas (una sola textura por archivo obj)
- No soporta materiales
- El nombre del archivo obj debe ser igual al de su textura correspondiente

Los archivos obj, son archivos con texto plano en un formato sumamente sencillo y fácil de entender para una persona.
El formato obj consiste en una instrucción por línea. Cada línea comienza con una instrucción, seguida de sus parámetros.

v x y z	**v (vertex position)** indica la definición de un vértice cuya posición es x , y , z
vn i j k	**vn (vertex normal)** indica la definición de una normal de un vértice. i,j,k son las componentes en x,y,z de la normal.
vt u v	**vt (vertext texture)** indica que la línea define una coordenada de textura. Las texturas son imágenes 2D y por ende, las coordenadas solo tienen 2 componentes
f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3	**f (face)** indica que la línea define una cara. Cada vértice de la cara tiene una posición en el espacio (**v**), puede tener una normal (**vt**), y puede estar mapeado a una textura (**vn**). Los índices vt y vn son opcionales.

En el siguiente ejemplo, desplegamos el contenido de un archivo obj que dibuja un triángulo:

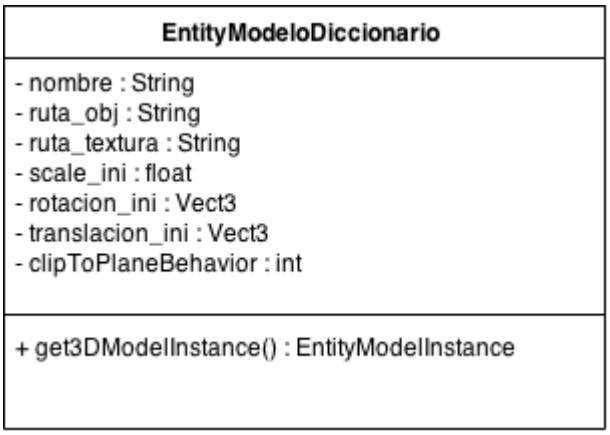
```
#Comentarios con el caracter '#'
#Lista de vértices
v -0.5 -0.5 0
v 0.5 -0.5 0
v 0 0.5 0
#Lista de normales
vn 0 0 1
vn 0 0 1
vn 0 0 1
#UV mapping
vt 0 1
```

vt 1 1
vt 0.5 0

#La única cara del objeto, utilizando toda la información anterior
f 1/1/1 2/2/2 3/3/3

Moldes e instancias 3D

La definición de un modelo 3D, usando archivos *.obj permite crear y posicionar en escena no solo una instancia del mismo.
Si bien, cada archivo obj es un molde con el cual se pueden crear tantas instancias como sean necesarias, dichos archivos únicamente contienen especificaciones gráficas (vértices, normales, materiales, texturas, etc.)
Para el prototipo en desarrollo, se requieren parámetros adicionales a la información contenida en un archivo obj, si queremos especificar un estándar de un "molde" para la creación de objetos virtuales.
La clase ***EntityModeloDiccionario*** es el molde, de la cual se crearán instancias de modelos 3D virtuales.



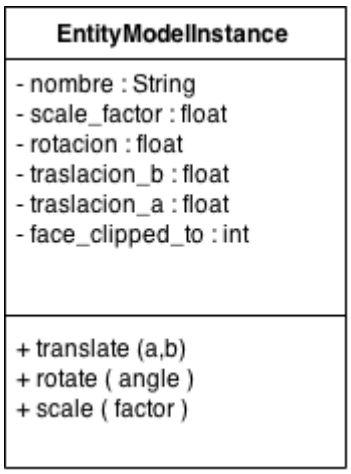
EntityModeloDiccionario contiene la ruta al archivo obj, y al archivo de textura.

Contiene también los atributos, scale_ini , rotacion_ini y translacion_ini, que permiten modificar el tamaño del objeto, su orientación y la ubicación del origen del sistema de coordenadas locales.
Estos parámetros son útiles para realizar cambios que afecten a todas las instancias de un modelo sin necesidad de modificar el archivo obj.

Además, la bandera clipToPlaneBehavior define el comportamiento de las instancias, especificando si es un objeto que se ancla al piso, al techo o las paredes.

Así, el catálogo de objetos virtuales disponibles en el prototipo de aplicación es una lista de objetos EntityModeloDiccionario. El usuario podrá interactuar directamente con éste catálogo para posicionar tantos objetos virtuales desee, sean de uno o varios tipos.

Cuando se desea obtener una instancia de un objeto virtual, se solicita a través del factory method `get3DModelInstance()` de **EntityModeloDiccionario** que retorna un objeto del tipo **EntityModelInstance**



Cada vez que se agrega un objeto virtual en la escena, se crea una instancia de EntityModelInstance. Y es posible crear muchas instancias de un mismo modelo (por ejemplo muchas sillas iguales para un comedor).

Recordar que un objeto virtual está forzosamente anclado a un plano (pared). Es decir, la posición del objeto virtual está delimitada a un plano (y como tal puede expresarse con 2 componentes a y b). Y la rotación ocurre sobre un solo eje (perpendicular al plano en el que se encuentra anclado el objeto virtual). Ver Adición de objetos virtuales.

Renderizado

El funcionamiento del render se describe a continuación

Inicalización:
Se realizan procesos de inicialización y se obtiene la información y requerida para el Render.
Como:

- Obtención del dato de la altura del usuario.
- Obtención del objeto AmbienteVirtual (Con información de los vértices de la habitación)
- Carga de las texturas de los objetos virtuales
- Obtención, de ser el caso, de los objetos virtuales previamente agregados por el usuario.

- Inicialización de los parámetros OpenGL. (Configuración del depth-buffer, configuración del clear color, configuración del viewPort y del viewFrustrum)

Ciclo de renderizado:

```
While true
     Limpia pantalla
     Carga matriz ModelView
     Carga matriz identidad en matriz ModelView
     If (vista desde afuera)
          Trazlacion en dirección del eje z diez unidades
     End
     PushMatriz
          Traslada el ambiente en sentido opuesto al
desplazamiento del usuario
          Dibuja aristas del ambiente
          For(i=0 ; i<numObjetosVirutales ; i++)
               Dibuja objetoVirtual(i)
          End
     PopMatriz
End
```

El código completo del render se adjunta en el apéndice B.

Resultados

La fragmentación de dispositivos, particularmente de los sensores, en la plataforma Android hace poco viable una implementación que funcione correctamente en todos los dispositivos.
Los inconvenientes principales de esta fragmentación son:

- Las lecturas de los sensores de acelerómetro, giroscopio y campo magnético varían entre dispositivos. Esto afecta directamente a la calidad de la medición de las distancias.
- En ocaciones, los dispositivos no reportan correctamente los parámetros de sus cámaras. Esto dificulta el proceso de dibujar objetos virtuales en sincronía con la vista previa obtenida desde la cámara, restando calidad a la experiencia.

Sin embargo es posible personalizar la aplicación para un dispositivo en particular, cualquier dispositivo. En dicha situación, el enfoque propuesto permite, elaborar un modelo del ambiente interior en el que se encuentra el usuario y desplegar contenidos virtuales acordes a la ubicación del usuario.
El enfoque permite también llevar tracking del usuario en tiempo real cuando realiza cambios en su orientación, y tracking (no en timpo real) cuando realiza cambios en su ubicación.

Los escenarios en los que funciona mejor el enfoque es en aquellos espacios cerrados en los que el usuario se pueda situar en el centro, y cullas paredes y techo no se encuentren más allá de 6m de distancia.

Por la naturaleza de los sensores empleados, el usuario deberá permanecer lejos de interferencias electromagnéticas, y ubicado tambien, lejos del norte magnético

Conclusiones y trabajo futuro

El presente trabajo de tesis, corrobora la factibilidad parcial del enfoque propuesto, dificultada principalmente por la fragmentación de los dispositivos Android.
Como se ha mencionado, el enfoque propuesto no es exclusivo para dicha plataforma, y por ende su aplicación en una plataforma menos abierta (como iOS) puede facilitar su implementación en un ambiente de producción.

Se identifican múltiples factores en los que el trabajo se puede expandir y también distintos ámbitos en los que se espera que el trabajo pueda contribuir.

Como áreas de mejora, se identifican:

- Utilizar reconocimiento de patrones para detectar bordes de paredes. Esto contribuiría a mejorar el tracking de los cambios de orientación y posición del usuario.
- Desarrollar un método para extraer las características de los sensores involucrados en la orientación, que permitan modelar funciones de correción del error a la hora de medir distancias con el dispositivo. Esto resolvería el problema de la fragmentación.
- Mejorar la interacción con objetos virtuales: touch, pick, drag to move, etc.

Se espera, que el enfoque propuesto pueda combinarse con dispositivos como el proyecto tango Android, o el 3D scanner para iOS, que permiten escanear entornos 3D con precisión. Aunque estos aún son dispositivos en fase beta y lejos aun de ser comercializados masivamente.

Otra área particularmente interesante por las caracterísiticas del proyecto, en la que se espera que el trabajo pueda contribuir, es en la apuesta por la masificación de la Realidad Virtual mediante el uso de dispositivos móviles, de la mano del Google Cardboard (el primer HMD de fácil adquisición)

Bibliografía

Ron Jeffries (Marzo 2011), *What is Extreme Programming,*
http://ronjeffries.com/xprog/what-is-extreme-programming/

StatCounter, Global Stats
http://gs.statcounter.com/

Azuma, R.T., (Agosto 1997). A Survey of Augmented Reality. *In Presence: Teleoperators and Virtual Environments*. 6 (4), pp.355-385

Caudell T.P. & Mizell D.W. (1992). *Augmented reality: an application of heads-up display technology to manual manufacturing processes* In System Sciences, 2:659-69.

Craig A.B. (2013) *Understanding Augmented Reality: Concepts and Applications*. 1st ed. : Morgan Kaufmann.

Cruz-Neira C., Virtual Reality Overview, *ACM SIGGRAPH 1993 Notes: Applied Virtual Reality*, ACM SIGGRAPH 1993 Conference, Anaheim, California, Agosto 16, 1993

B'Far, Reza. (2005). *Mobile Computing Principles*. Cambridge University Press.

B'Far, Reza. (2005). *Mobile Computing Principles: Designing and Developing Mobile Applications with UML and XML*. Cambridge University Press.

Feiner, S., Macintyre, B., Seligmann, D., (1993). Knowledge-based augmented reality. *Communications of the ACM - Special issue on computer augmented environments: back to the real world*. 36 (7), pp.53-62

Heitkötter H, Hanschke S, Majchrzak T.A. (2013). *Evaluating Cross-Platform Development Approaches for Mobile Applications*. In: José Cordeiro, Karl-Heinz Krempels (ed), Web Information Systems and Technologies 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers. 1st ed. : Springer Berlin Heidelberg. pp.120-138.

Heitkötter H, Hanschke S & Majchrzak T.A. (2013). *Comparing cross-platform approaches for mobile applications* In: José Cordeiro, Karl-Heinz Krempels (ed), Web Information Systems and Technologies 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers. 1st ed. : Springer-Verlag Berlin Heidelberg. pp.299-311.

Johnson E.A. (1965) Touch Display – A novel input/output device for computers. Electronic

Letters 1 (8): 219-220

Kato H & Billinghurst M. (1999). *Marker Tracking and HMD Calibration for a Video-based Augmented Reality Conferencing System. Proceedings.* 2nd IEEE and ACM International Workshop on Augmented Reality, pp.85-94

Kwon B.C., Javed W, Elmqvist N. & Yi J.S., (2011). Direct manipulation through surrogate objects. *CHI '11: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp.627-636

Mikkonen T. (2007). *Programming Mobile Devices: An Introduction for Practitioners*. 1st ed. England: Wiley.

Milette G & Stroud A. (2012) *Professional Android Sensor Programming*. 1st ed. Indianapolis, Indiana: John Wiley & Sons, Inc.

Pressman R. & Maxim B, (2014). Software Engineering: A practitioners aproach. 8th ed. Newy York, NY: McGraw-Hill Education.

Priolo S. (2009). *Métodos ágiles*. 1st ed. Banfield-Lomas de Zamora, Argentina: Manual Users.

Salmre I, (2005). *Writing Mobile Code: Essential Software Engineering for Building Mobile Applications: Essential Software Engineering for Building Mobile Applications*. 1st ed. Addison-Wesley Professional.

Schmalstieg D, Langlotz T, and Billinghurst M, (2008). 'Augmented Reality 2.0'. In: *Coquillart, Sabine, Brunnett, Guido, Welch, Greg (ed), Virtual Realities. Dagstuhl Seminar 2008*. 1st ed. : Springer Vienna. pp.13-37.

Schmalstieg D, Ventura J, Arth C, Reitmayr G, *Global Localization from Monocular SLAM on a Mobile Phone*, (2014), en IEEE VR 2014.

Silva R, Oliveira J.C. & Giraldi G.A. (2003). *Introduction to Augmented Reality, LNCC Research Report #25/2003*, National Laboratory for Scientific Computation, ISSN: 0101 6113.

Sommerville I. (2005). *Ingeniería del software* . 7th ed. Madrid.: Parson Educación. S.A.

Sommerville I. (2011). *Software Engineering.* 9th ed. Boston, Massachusetts: Pearson Education, Inc.

Van Krevelen D.W.F. & Poelman R. (2010). *A Survey of Augmented Reality Technologies, Applications and Limitations*. The International Journal of Virtual Reality, 9, 1-19

Yuen S, Yaoyuneyong G & Johnson E. (2011). *Augmented reality: An overview and five*

directions for AR in education. Journal of Educational Technology Development and Exchange, 4(1), 119-140.

Zechner M. & Green R., (2012). *Beginning Android Games*. 2nd ed. CA, USA: Apress.

Apéndice A

Código Fuente del Cargador de Objetos

```
public class ObjLoader {
	public static Vertices3 load(
		GLGraphics glGraphs, String file , int textureIndex , Context ctx){
		InputStream in = null;

		//Arreglos LINEALES para almacenar vertices, normales y uv's
		float[] vertices = null;
		float[] normales = null;
		float[] uv = null;
		List<String> lines = null;

		try {
			in = ctx.getResources().getAssets().open(
				file , Context.MODE_WORLD_READABLE);
			lines = readLines( in );

			vertices = new float[ lines.size() * 3];
			normales = new float[ lines.size() * 3];
			uv = new float[ lines.size() * 2 ];

		} catch (IOException e) {
			e.printStackTrace();
		}

		//Contadores de vertices, normales, uvs y faces
		int numVertices = 0;
		int numNormals = 0;
		int numUV = 0;
		int numFaces = 0;

		//Se guardan los indices de vertices, normales y uvs
		//asociados a las caras
		int[] facesVerts = new int[lines.size() * 3];
		int[] facesNormals = new int[lines.size() * 3];
		int[] facesUV = new int[lines.size() * 3];

		int vertexIndex = 0;
		int normalIndex = 0;
		int uvIndex = 0;
		int faceIndex = 0;

		for(int i = 0 ; i< lines.size() ; i++){
			String line = lines.get(i);

			//Encuentra una coordenada de un vértice
			if(line.startsWith("v ")){
				String[] tokens = line.split("[ ]+");
				vertices[vertexIndex] = Float.parseFloat( tokens[1] );
				vertices[vertexIndex+1] = Float.parseFloat( tokens[2] );
```

```
                        vertices[vertexIndex+2] = Float.parseFloat( tokens[3] );
                        vertexIndex += 3;
                        numVertices++;
                        continue;
                    }

                    //Encuentra una normal de un vértice
                    if (line.startsWith("vn ")) {
                        String[] tokens = line.split("[ ]+");
                        normales[normalIndex] = Float.parseFloat(tokens[1]);
                        normales[normalIndex + 1] = Float.parseFloat(tokens[2]);
                        normales[normalIndex + 2] = Float.parseFloat(tokens[3]);
                        normalIndex += 3;
                        numNormals++;
                        continue;
                    }

                    //Encuentra una coordenada de textura
                    if (line.startsWith("vt")) {
                        String[] tokens = line.split("[ ]+");
                        uv[uvIndex] = Float.parseFloat(tokens[1]);
                        uv[uvIndex + 1] = Float.parseFloat(tokens[2]);
                        uvIndex += 2;
                        numUV++;
                        continue;
                    }


                    //Encuentra una definicion de una cara (face).
                    //Asumimos que todas las caras serán triángulos
                    if( line.startsWith("f ")){
                        String[] tokens = line.split("[ ]+");

                        for(int j=1 ; j<=3 ; j++){
                            String[] parts = tokens[j].split("/");

                            facesVerts[faceIndex] = getIndex(
                                        parts[0], numVertices);
                            if (parts.length > 1){
                                facesUV[faceIndex] = getIndex(parts[1], numUV);
                            }
                            if (parts.length > 2){
                                facesNormals[faceIndex] =
                                        getIndex(parts[2], numNormals);
                            }
                            faceIndex++;
                        }
                        numFaces++;
                        continue;
                    }

                }

                float[] verts =
new float[(numFaces * 3) * (3 + (numNormals > 0 ? 3 : 0) + (numUV > 0 ? 2 : 0))];

                for(int i=0 , vi=0 ; i < numFaces*3 ; i++ ){
                    int vertexIdx = facesVerts[i] * 3;
                    verts[ vi++ ] = vertices[vertexIdx];
                    verts[ vi++ ] = vertices[ vertexIdx+1 ];
                    verts[ vi++ ] = vertices[ vertexIdx+2 ];
```

```
                if(numUV >0 ){
                        int uvIdx = facesUV[i] * 2;
                        verts[vi++] = uv[uvIdx];
                        verts[vi++] = 1 - uv[uvIdx + 1];
                }

                if (numNormals > 0) {
                        int normalIdx = facesNormals[i] * 3;
                        verts[vi++] = normales[normalIdx];
                        verts[vi++] = normales[normalIdx + 1];
                        verts[vi++] = normales[normalIdx + 2];
                }
        }

        Vertices3 model = new Vertices3(
                glGraphs , numFaces * 3, 0, false, numUV > 0,
                textureIndex, numNormals > 0);
        model.setVertices(verts, 0, verts.length);
        return model;
    }

    private static List<String> readLines(InputStream in) throws IOException {
        List<String> lines = new ArrayList<String>();

        BufferedReader reader = new BufferedReader(new InputStreamReader(in));
        String line = null;
        while ((line = reader.readLine()) != null)
                lines.add(line);
        return lines;
    }


    private static int getIndex(String index , int size){
        int idx = Integer.parseInt(index);
        if(idx < 0){
                return size + idx;
        }else{
                return idx - 1;
        }
    }
}
```

Apéndice B

Código Fuente del Render

```
public class Renderer implements GLSurfaceView.Renderer {
	Context context;
	GLGraphics graficos;
	EntityAmbiente ambiente;
	Ambiente ambienteEscaneado;
	Ejes ejes = new Ejes();

	//Vector de desplazamiento del usuario, a partir de la posicion inicial
	//El valor de Z permanece constante, y es la altura del usuario
	//(Al nivel de sus ojos)
	float[] desplazamientoUsuario = new float[]{0.0f , 0.0f , 0.0f};

	HashMap<String,VirtualObject> objetos3D;
	Collection<EntityModelInstance> instanciasObjetos3D;

	//Referencia a un objeto seleccionado para su edición
	VirtualObject objSelected = null;
	String objSelectedKey = null;

	//Arreglo que almacena los ids de las texturas.
	//Cada objeto puede tener una textura.
	int[] textureIDs;
	private String[] textureNames;

	float fovYAngle = 42.5f;//= 33.0f; //25.0f;
	float zNear = 0.3f;
	float environmentZRot = 0.0f;

	//Indica si el ambiente se ve desde afuera, o desde dentro
	public boolean outterView = false;
	//Indica si se dibujan los eje y el poliedro
	public boolean renderViewMode = true;
	private DBHelper mDBHelper;

	// Constructor with global application context
	public Renderer(Context ctx ,
			GLSurfaceView mGLSurfaceView,
			HashMap<String,EntityModeloDiccionario> diccionarioModelos,
			EntityAmbiente ambiente){
		context = ctx;
		graficos = new GLGraphics( mGLSurfaceView );
		this.ambiente = ambiente;
		ambienteEscaneado = new Ambiente(ambiente);

		ambienteEscaneado.enableMarkerAtVertex(4);
		ambienteEscaneado.enableMarkerAtVertex(5);

		desplazamientoUsuario[2] = Util.getUserHeight(ctx , true);

		objetos3D = new HashMap<String, VirtualObject>();
```

```
        //Obtenemos todas las texturas posibles de la aplicacion
        textureNames = new String[diccionarioModelos.size()];
        int i=0;
        for(EntityModeloDiccionario modelo : diccionarioModelos.values()){
              textureNames[i] = modelo.getRutaTextura();
              i++;
        }
}

// Call back when the surface is first created or re-created
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig arg1) {

        graficos.setGL( gl );
        loadTextures( objetos3D, gl );
        reloadPreviousObjects();

        // Set color's clear-value to black
        gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        // Set depth's clear-value to farthest
        gl.glClearDepthf(1.0f);
        // Enables depth-buffer for hidden surface removal
        gl.glEnable(GL10.GL_DEPTH_TEST);
        // The type of depth testing to do
        gl.glDepthFunc(GL10.GL_LEQUAL);
        // nice perspective view
        gl.glHint(GL10.GL_PERSPECTIVE_CORRECTION_HINT, GL10.GL_NICEST);
        // Enable smooth shading of color
        gl.glShadeModel(GL10.GL_SMOOTH);
        // Disable dithering for better performance
        gl.glDisable(GL10.GL_DITHER);
        // Setup Texture, each time the surface is created (NEW)
        gl.glEnable(GL10.GL_TEXTURE_2D);  // Enable texture (NEW)

}

//Metodo para cargar Todas las texturas
private void loadTextures(
                    HashMap<String, ?  extends Vertices3> objetos3d ,
                    GL10 gl) {
        textureIDs = new int[ textureNames.length ];
        gl.glGenTextures( textureNames.length , textureIDs, 0);

        AssetManager mAssetManager = context.getAssets();
        if(mAssetManager == null){
              Log.e("", "Error. Textura No encontrada");
        }
        for(int i=0 ; i<textureNames.length ; i++){
              gl.glBindTexture( GL10.GL_TEXTURE_2D, textureIDs[i]);
              // Set up texture filters
              gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                                  GL10.GL_TEXTURE_MIN_FILTER,
                                  GL10.GL_NEAREST);
              gl.glTexParameterf(
                    GL10.GL_TEXTURE_2D,
                    GL10.GL_TEXTURE_MAG_FILTER,
                    GL10.GL_LINEAR);

              try {
                    InputStream istream = null;
                    Bitmap bitmap = null;
```

```
                    istream = mAssetManager.open( textureNames[i] );
                    // Read and decode input as bitmap
                    bitmap = BitmapFactory.decodeStream(istream);
                    istream.close();
                    // Build Texture from loaded bitmap for
                    //the currently-bind texture ID
                    GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
                    bitmap.recycle();

             } catch (IOException e) {
                    e.printStackTrace();
             } finally {
             }
      }
}

private void reloadPreviousObjects(){
      //Agregamos los objetos virtuales previamente guardados
      instanciasObjetos3D = ambiente.getInstanciasModelos();
      for( EntityModelInstance instanciaModelo : instanciasObjetos3D ){
             DBHelper h = getHelper();
             try {
                    Dao<EntityModeloDiccionario, Integer> dao
                           = h.getModeloDiccionarioDao();
                    dao.refresh( instanciaModelo.getModeloDiccionario() );
             } catch (SQLException e) {
                    e.printStackTrace();
             }
             addObj( instanciaModelo );
      }
}

//Método para remover un objeto3D de la escena
public synchronized void removeObject( String id ){
      objetos3D.remove( id );
      for(EntityModelInstance mI : instanciasObjetos3D){
             if(mI.getNombre().equals( id )){
                    Log.d("", "Quita instancia " + id);
                    instanciasObjetos3D.remove(mI);
                    break;
             }
      }
}

//Metodo para agregar un Objeto3d a la escena
public synchronized String addObj( EntityModeloDiccionario modelo ){
      int modelCounter = 1;
      while(
             objetos3D.containsKey( modelo.getNombreObjeto()+" "+modelCounter)
      ){
             modelCounter++;
      }
      String tag =  modelo.getNombreObjeto()+" "+modelCounter;
      addObj(modelo, tag , true);

      return tag;
}

public synchronized void addObj( EntityModelInstance instancia ){
      EntityModeloDiccionario modelo = instancia.getModeloDiccionario();
      String nameInstancia = instancia.getNombre();
```

```
        //Determinamos el id de la textura del modelo
        int textPos = 0;
        for(int i =0 ; i<textureIDs.length ; i++, textPos++){
              Log.d("", modelo.getRutaTextura()+" vS ");
              Log.d("", textureNames[i]);
              if(modelo.getRutaTextura().equals( textureNames[i])){
                    break;
              }
        }

        Vertices3 obj = ObjLoader.load(
                    graficos,
                    modelo.getRutaObj() ,
                    textureIDs[textPos],
                    context );
        obj.setRotationAngles(
                    new float[]{modelo.getRotacionXIni() ,
                    modelo.getRotacionYIni(),
                    modelo.getRotacionZIni()});
        obj.setScaleFactor( modelo.getScaleIni() );

        VirtualObject vObj;
        switch( modelo.getClipToPlaneBehavior()){
        case EntityModeloDiccionario.BEHAVIOR_STICK_TO_FLOOR:
              vObj = new VirtualObject(graficos,
                          obj, ambienteEscaneado, Ambiente.PLANO_SUELO);
              break;
        case EntityModeloDiccionario.BEHAVIOR_STICK_TO_ROOF:
              vObj = new VirtualObject(graficos,
                          obj, ambienteEscaneado, Ambiente.PLANO_TECHO);
              break;
        default:// ModeloDiccionario.BEHAVIOR_STICK_TO_WALL:
              vObj = new VirtualObject(graficos,
                          obj, ambienteEscaneado, Ambiente.PLANO_PARED_1);
        }

        vObj.clipToPlane(ambienteEscaneado, instancia.getFaceClipped());
        vObj.setvObjectScaleFactor( instancia.getScaleFactor() );
        vObj.setvObjectNaturalRotationValue( instancia.getRotacion() );
        vObj.setvObjectTranslationMagnitudes(
                    new float[]{instancia.getTraslacionA() ,
                    instancia.getTraslacionB()});
        objetos3D.put( nameInstancia , vObj );

        return;
  }


  /**
   * Agrega un objeto virutal a la escena
   * @param modelo El modelo desde el diccionario de modelos al que
   * corresponde el objeto virtual
   * @param nameInstancia El nombre con el que se identifica al
   * objeto virtual ya en la escena
   * @param isNewInstance Determina si se debe crear un nuevo objeto del tipo
   * EntityModelnstance. Si se llama a addObj para restaurar un estado del
   * ambiente, isNewObject deberï¿½ ser false para evitar duplicar una misma
   * instancia del modelo
   * */
  public synchronized void addObj(EntityModeloDiccionario modelo ,
              String nameInstancia ,
```

```
            boolean isNewInstance ){
        //Determinamos el id de la textura del modelo
        int textPos = 0;
        for(int i =0 ; i<textureIDs.length ; i++, textPos++){
            Log.d("", modelo.getRutaTextura()+" vS ");
            Log.d("", textureNames[i]);
            if(modelo.getRutaTextura().equals( textureNames[i])){
                break;
            }
        }

        Vertices3 obj = ObjLoader.load( graficos,
            modelo.getRutaObj() , textureIDs[textPos], context );
        obj.setRotationAngles(
            new float[]{modelo.getRotacionXIni() ,
            modelo.getRotacionYIni(), modelo.getRotacionZIni()});
        obj.setScaleFactor( modelo.getScaleIni() );

        VirtualObject vObj;
        EntityModelInstance newModelInstance;
        switch( modelo.getClipToPlaneBehavior()){
        case EntityModeloDiccionario.BEHAVIOR_STICK_TO_FLOOR:
            vObj = new VirtualObject(graficos,
                obj, ambienteEscaneado, Ambiente.PLANO_SUELO);
            newModelInstance = new EntityModelInstance(
                        nameInstancia,1f,0f,
                        new float[]{0,0,0},
                        Ambiente.PLANO_SUELO,
                        modelo,
                        ambiente);
            break;
        case EntityModeloDiccionario.BEHAVIOR_STICK_TO_ROOF:
            vObj = new VirtualObject(
                    graficos,
                    obj,
                    ambienteEscaneado,
                    Ambiente.PLANO_TECHO);
            newModelInstance = new EntityModelInstance(
                    nameInstancia,1f,0f,
                    new float[]{0,0,0},
                    Ambiente.PLANO_TECHO,modelo,ambiente);
            break;
        default:// ModeloDiccionario.BEHAVIOR_STICK_TO_WALL:
            vObj = new VirtualObject(
                    graficos, obj, ambienteEscaneado, Ambiente.PLANO_PARED_1);
            newModelInstance = new EntityModelInstance(
                    nameInstancia,1f,0f,
                    new float[]{0,0,0},Ambiente.PLANO_PARED_1,modelo,ambiente);
        }
        objetos3D.put( nameInstancia , vObj );
        if(isNewInstance)
            instanciasObjetos3D.add( newModelInstance );
        return;
    }


    float aspect;
    // Call back after onSurfaceCreated() or whenever the window's size changes
    @Override
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        if (height == 0) height = 1;   // To prevent divide by zero
```

```
        aspect = (float)width / height;

        // Set the viewport (display area) to cover the entire window
        gl.glViewport(0, 0, width, height);

        // Setup perspective projection, with aspect ratio matches viewport
        gl.glMatrixMode(GL10.GL_PROJECTION); // Select projection matrix
        gl.glLoadIdentity();                 // Reset projection matrix
        // Use perspective projection
        GLU.gluPerspective(gl, fovYAngle, aspect, zNear, 30.f);


        gl.glMatrixMode(GL10.GL_MODELVIEW);  // Select model-view matrix
        gl.glLoadIdentity();

    }

    // Call back to draw the current frame.
    @Override
    public synchronized void onDrawFrame(GL10 gl) {
        // Limpia la pantalla y el buffer de profundidad
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT | GL10.GL_DEPTH_BUFFER_BIT);
        // Setup perspective projection, with aspect ratio matches viewport
        gl.glMatrixMode(GL10.GL_PROJECTION); // Select projection matrix
        gl.glLoadIdentity();                 // Reset projection matrix
        // Use perspective projection
        GLU.gluPerspective(graficos.getGL(), fovYAngle, aspect, zNear, 30.f);
        gl.glMatrixMode(GL10.GL_MODELVIEW);  // Select model-view matrix
        gl.glLoadIdentity();               // Reset the model-view matrix

        if(outterView){
            // Translate left and into the screen ( NEW )
            gl.glTranslatef(0f, 0.0f, -18.0f);
        }

        gl.glMultMatrixf(Globals.SENSOR_ROTATION_MATRIX, 0);
        if(renderViewMode){
            //Dibujamos los ejes
            ejes.draw(gl);
        }

        gl.glPushMatrix();

        //Ajustamos a la posición del usuario
        //El ambiente es trasladado en direccion inversa al
        //vector de desplazamiento del usuario
        gl.glTranslatef(
            -desplazamientoUsuario[0] ,
            -desplazamientoUsuario[1], -desplazamientoUsuario[2]);

        if(renderViewMode){
            //Dibujamos el ambiente
            ambienteEscaneado.draw(gl);
        }
        //Dibujamos los objetos asociados al ambiente
        for( Vertices3 obj : objetos3D.values() ){
            obj.bind();
            obj.draw(GL10.GL_TRIANGLES, 0, obj.getNumVertices());
            obj.unbind();
        }
```

```
		gl.glPopMatrix();
	}



	public void adjustUserPosition(float angle, float aMagnitude, float bMagnitude){

		float gama = angle, beta=0, alpha=0;

		//Calcular |c| (formado por los vertice 4-5 o bien 0-1)
		float[][] verticesAmbiente = ambienteEscaneado.getRealVertices();
		float[] c = MiMath.restaVect3(verticesAmbiente[1], verticesAmbiente[0]);
		float cMagnitude = MiMath.vectorMagnitud(c);
		Log.v("","cMagnitude: "+ cMagnitude);
		Log.v("","aMagnitude: "+ aMagnitude);
		Log.v("","bMagnitude: "+ bMagnitude);

		//Calcular angulos
		alpha = (float) Math.asin( aMagnitude * Math.sin(Math.toRadians(gama)) /
cMagnitude);
		alpha = (float) Math.toDegrees( alpha );
		beta = 180 - alpha - gama;

		//Aqui debemos de usar la magnitud que más nos convenga,
		// |a| ó |b|. Dependiendo
		//cuál magnitud es confiable en el método trigonométrico

		//el vector a se pude obtener a partir del vector b
		float[] cUnit = MiMath.normalizaVect2( c );
		MiMath.logVector(cUnit);
		float[] aUnit = MiMath.rotateVector2(cUnit, beta);
		MiMath.logVector(aUnit);
		float[] negA = MiMath.scaleVect2(aMagnitude, aUnit);
		MiMath.logVector(negA);

		float[] newPosition = MiMath.sumaVect2(
			new float[]{verticesAmbiente[0][0],verticesAmbiente[0][1]} , negA);
		desplazamientoUsuario[0] = newPosition[0];
		desplazamientoUsuario[1] = newPosition[1];
		//desplazamientoUsuario[2] <-- El desplazamiento es sobre X y Y, no en Z
	}


	public void changeFOV(float angle){
		fovYAngle += angle;
		Log.d("", "New Angle: " + fovYAngle );
	}
	public float getFovAngle(){
		return fovYAngle;
	}

	public void changeRotZ(float angle){
		environmentZRot+=angle;
		Log.d("", "New Angle: " + environmentZRot );
	}
	public float getRotZ(){
		return environmentZRot;
	}

	public void changeZNear(float deltaZNear){
		zNear+=deltaZNear;
```

```
}
public float getZNear(){
	return zNear;
}

//Método para obtener todas los identificadores (llaves) de
//los objetos virtuales creados
public String[] getVirtualObjectKeys(){
	Set<String> keys = objetos3D.keySet();
	String[] keysArr = new String[ keys.size() ];

	int i=0;
	for (String key : objetos3D.keySet()) {
		keysArr[i] = key;   i++;
	}
	return keysArr;
}

//Metodo para seleccionar
public boolean selectObject(String key){
	if(!objetos3D.containsKey( key ) ){
		return true;
	}
	else{
		objSelected = (VirtualObject) objetos3D.get( key );
		objSelectedKey = key;
		return true;
	}
}

public String getObjectSelectedKey(){
	return objSelectedKey;
}



//---------------Transformaciones aplicadas a algun objeto seleccionado---
public void scaleObjectSelected( float deltaValue ){
	if( objSelected == null ){ return;	}
	objSelected.setScaleFactor( objSelected.getScaleFactor() + deltaValue );
}

public void rotateObjectSelected( float deltaValue ){
	if(objSelected == null ){  return;	}
		objSelected.setvObjectNaturalRotationValue(
			objSelected.getvObjectNaturalRotationValue() + deltaValue );
}

public void translateObject(float deltaDir1 , float deltaDir2){
	objSelected.deltaTranslation(deltaDir1, deltaDir2);
}


public void changePlaneObjectSelected(int newPlane){
	objSelected.clipToPlane(ambienteEscaneado, newPlane);
}

//--------------- Persiste el ambiente en la BD -----------------
//---------------------------------------------------------------
public void persistAmbiente(){
	try {
```

```
                    DBHelper helper = OpenHelperManager.getHelper(
                          context, DBHelper.class);
                    Dao<EntityModelInstance, Integer> dao =
                                helper.getModelInstanceDao();

                    for(EntityModelInstance mI : instanciasObjetos3D){
                          VirtualObject vO = objetos3D.get( mI.getNombre());

                          mI.setFaceClipped( vO.getPlaneClipped() );

                          mI.setScaleFactor( vO.getvObjectScaleFactor() );

                          mI.setRotacion( vO.getvObjectNaturalRotationValue() );
                          mI.setTraslacionA( vO.getvObjectTranslationMagnitudes()[0]);
                          mI.setTraslacionB( vO.getvObjectTranslationMagnitudes()[1]);

                          dao.update( mI );
                    }

             } catch (SQLException e) {
                    e.printStackTrace();
             }
      }

      private DBHelper getHelper() {
             if (mDBHelper == null) {
                    mDBHelper = OpenHelperManager.getHelper(context, DBHelper.class);
             }
             return mDBHelper;
      }

}
```