Desarrollo de una aplicación de realidad aumentada para dispositivos móviles

PROYECTO FINAL DE CARRERA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE VALENCIA

Autor: Miguel Medina Carda
Director: Antonio Cano Gómez
Titulación: Ingeniería Informática
Fecha: Octubre de 2011

ÍNDICE DE CONTENIDOS

1.- RESUMEN

El trabajo realizado en este proyecto se enmarca dentro del ámbito de los dispositivos móviles y de la realidad aumentada. La memoria de este proyecto final de carrera puede servir para hacerse una idea de cual es el estado actual del uso de esta tecnología, que se puede esperar de ella en un futuro y que herramientas están al alcance de los desarrolladores para poder usarla en sus aplicaciones. Sobre todo nos ha interesado como está influyendo en los dispositivos móviles y que utilidades puede tener para los usuarios.

En el siguiente documento se explicará, por tanto, de que se trata esta tecnología, como funciona y qué usos se le puede dar. A continuación, veremos también cuales son las herramientas actuales de las que disponen los programadores, centrándonos en aquellas que se difunden bajo la Licencia Publica General de GNU. Éste es el denominado software libre cuya principal característica es que el código fuente de las aplicaciones está disponible para ser modificado y distribuido, bajo unas ciertas condiciones.

En cuanto al tema de los dispositivos móviles nos centraremos en el Nokia N900, ya que es el modelo del terminal con el que hemos estado trabajando. Se incluye en este documento una descripción de sus características y prestaciones, así como la explicación de como portar algunas de las herramientas a este dispositivo para hacerlas funcionar. En concreto, haremos hincapié en la instalación, uso y funcionamiento de una aplicación de realidad aumentada para este dispositivo llamada “Araap”.

2.- INTRODUCCIÓN

2.1.- OBJETIVOS Y MOTIVACIÓN

El objetivo principal del proyecto es investigar la situación actual de la realidad aumentada en el mundo de los dispositivos móviles, centrándonos en el Nokia N900. Hoy en día los teléfonos móviles nos ofrecen una serie de servicios que van mucho más allá de poder hacer y recibir llamadas. Sobre todo los *smartphones* disponen de una serie de características como la conexión a Internet o la localización geográfica que, teniendo en cuenta que podemos llevarlos encima a todas horas, están llamadas a revolucionar la forma en que vivimos y percibimos la realidad.

Por su parte, la realidad aumentada es una tecnología muy llamativa a nivel visual y que provoca curiosidad en cada usuario que la utiliza. Por eso está popularizándose cada vez más. Además de eso, el imaginar todas las utilidades que puede tener y de que manera nos puede facilitar la vida hacen de este campo un área muy atrayente para los desarrolladores de software. Por tanto, poder basar el PFC en la realidad aumentada sobre un dispositivo móvil es una gran oportunidad ya que se está actuando sobre un área de futuro pero de actual expansión.

Se podría decir que este proyecto pretende ser un primer paso para el desarrollo de aplicaciones de realidad aumentada para el N900, sirviendo de guía inicial de cara a futuros proyectos que puedan aprovechar la información aquí presente. Más concretamente, se analizará la aplicación *Arapp* para conocerla, saber como funciona y analizar que posibles utilidades puede tener su futuro desarrollo.

Consideramos un aspecto importante el hecho de trabajar con software libre ya que proporciona una mayor libertad para los programadores. Además, el tener acceso al código fuente de una aplicación y la posibilidad de modificarla permite el desarrollo de nuevos productos o mejoras de los ya existentes sin la necesidad de comenzar todo el proceso partiendo de cero. Sin olvidar, por supuesto, las ventajas económicas que proporciona el software libre que hacen que, generalmente, la expansión de un producto sea mayor que la de uno de software propietario.

2.2.- REALIDAD AUMENTADA

La investigación y creación de software donde se hace uso de la realidad aumentada está creciendo en los últimos tiempos. Sus múltiples aplicaciones y su atractivo aspecto visual está provocando que cada vez más los desarrolladores del campo de la informática y de las nuevas tecnologías se interesen por ella y la incluyan en sus proyectos.

Por otro lado, el aumento de, no sólo el número de dispositivos móviles existentes en la actualidad, sino también de sus prestaciones hacen que las aplicaciones disponibles sean cada vez mejores y más numerosas. Teniendo en cuenta además que la práctica totalidad de estos dispositivos disponen de cámaras de vídeos de buena calidad, es inevitable pensar que adaptar la realidad aumentada a los dispositivos móviles es un campo de gran interés y de grandes oportunidades.

De hecho así se está produciendo pues ya podemos encontrar algunas aplicaciones interesantes que, aunque aún lejos de ofrecer lo que se puede esperar de una tecnología de estas características, ya nos permiten comprobar que no es una utopía el hecho de pensar que en un futuro no muy lejano la realidad aumentada estará muy presente en nuestra vida diaria.

El campo donde, probablemente, haya progresado más su uso es en el de los videojuegos y entretenimiento. Las videoconsolas ya ofrecen la posibilidad de interactuar (incluso con el propio cuerpo del jugador) con objetos que parece que se mueven sobre el escenario que capta la cámara, viendo además el resultado de sus acciones en una pantalla donde se mezclan la imagen real y la virtual.

Por supuesto, el uso de la realidad aumentada en los dispositivos móviles también irá encaminado hacia el ocio pero no de manera exclusiva. No hay que olvidar que una de las principales ventajas de estos dispositivos es que podemos hacer uso de ellos en cualquier lugar. Por tanto, se pueden crear aplicaciones donde el principal interés sea el mostrar una interacción del mundo virtual con el mundo real, teniendo en cuenta la movilidad que un dispositivo móvil proporciona. Por ejemplo, se podría sobreimprimir en la pantalla información sobre un edificio famoso que estás grabando o dibujar la forma de las constelaciones mientras enfocas a un cielo estrellado.

La mayoría de los dispositivos de última generación incluyen también un receptor GPS que permite ubicar la localización del aparato. Esta cualidad puede ser utilizada por aplicaciones que hagan uso, también, de la realidad aumentada y que mediante la utilización de la cámara pueden mostrar información sobre el lugar donde te encuentras dependiendo incluso de la orientación espacial que tenga el dispositivo. Por ejemplo, una aplicación de navegación que nos indica la ruta a seguir en tiempo real hacia un destino podría hacerlo mostrando las indicaciones sobreimpresas en la propia imagen real que vas captando de la carretera.

Estos son sólo algunos de los usos que una aplicación de realidad aumentada puede tener. Muchas de estas aplicaciones ya comienzan a ser una realidad. Gracias a la incipiente expansión en el desarrollo de esta tecnología y al interés de un público que cada vez dispone de más facilidades para acceder a ella, el crecimiento está asegurado.

2.2.1.- DESCRIPCIÓN

La realidad aumentada consiste en sobreponer una capa de contenido virtual sobre la imagen del mundo real, complementándolo así con información del entorno que estamos visualizando[1]. Entonces, lo primero que necesitamos será un dispositivo capaz de captar la información del mundo real. Esto puede ir desde una simple cámara Web conectada a un PC hasta un dispositivo móvil con todo tipo de sensores (además de la cámara) que sean capaces de medir cosas como la inclinación, ubicación, orientación o aceleración del dispositivo. Existe una gran variedad de sistemas de este tipo en el mercado, aunque nosotros nos centraremos en el teléfono N900 de la marca Nokia.

En el caso de los dispositivos móviles, el propio aparato lleva incorporado también el dispositivo de salida donde se visualizan los resultados y que no es más que la pantalla del teléfono. En el caso de un ordenador será el monitor donde se muestra, generalmente en tiempo real, la mezcla de la información real y virtual.

Para poder realizar este proceso se hace necesario un software de realidad aumentada capaz de, a partir de los datos de entrada del mundo real, generar los datos virtuales que se van a mostrar entremezclados. En el cuarto punto del documento se enumeran algunas de estas herramientas.

En muchos de los casos, este software de realidad aumentada se apoyará en los llamados marcadores, que no son más que símbolos que ayudan al programa de realidad aumentada a interpretar la información del mundo real indicándole, por ejemplo, donde debe ubicar una imagen tridimensional o un texto informativo.

La realidad aumentad es una tecnología emergente, pero con multitud de aplicaciones en muy diversos campos. Desde, como se comentó anteriormente, el mundo de los videojuegos y entretenimiento hasta el de la educación o el ámbito militar. Según un estudio de 2010 de la consultora tecnológica Gartner sobre tecnologías emergentes, la realidad aumentada es una de las tecnologías por la cual se tiene una mayor expectativa de cara al futuro y será dentro de 5 o 10 años cuando alcance su nivel óptimo de productividad[2].

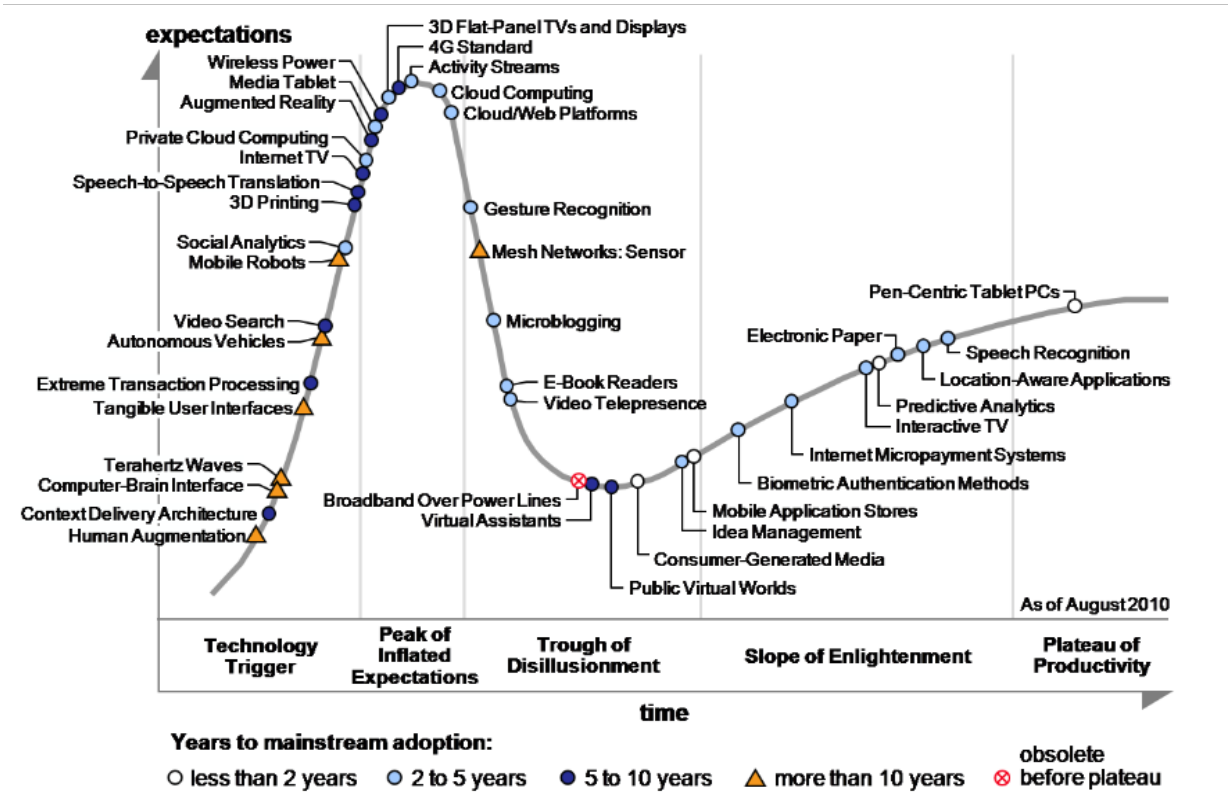

Figura 1: Ciclo de expectativas para tecnologías emergentes.

En el gráfico se puede apreciar el contexto en que se encuentra la realidad aumentada. La expansión de la tecnología será, principalmente, en los dispositivos móviles ya que podemos observar que otra tecnología con un fuerte futuro impacto será el estándar 4G, usada para la conexión de alta velocidad en los teléfonos móviles.

2.2.2.- APLICACIONES

En este punto vamos a describir algunas de las aplicaciones que ya existen actualmente y que hacen uso de la realidad aumentada. Es una manera de ver el punto de partida de la tecnología, sus primeros usos y sus distintos campos de aplicación. Nos centraremos sobre todo en aplicaciones para dispositivos móviles ya que son más útiles y accesibles para el usuario. Algunas son, además, de código abierto lo que facilita a los desarrolladores su mejora y expansión

Una interesante y popular aplicación es el Layar Reality Browser[3]. Se trata de una aplicación multiplataforma disponible para la mayoría de teléfonos móviles que dispongan de GPS, acelerómetro y brújula digital. Consiste en un navegador que, a partir de tu ubicación, te muestra las aplicaciones de realidad aumentada de su catálogo disponibles para la zona donde te encuentres. Se puede decir que esta tecnología es un soporte para hacer llegar distintas aplicaciones (en el programa se refieren a ellas como capas) de realidad aumentada basadas en el posicionamiento espacial a los usuarios.

Encontramos de distintos ámbitos, la mayoría de ellas para indicarnos la posición de lugares de interés cercanos como restaurantes, paradas de metro, farmacias, etc... Sin embargo, existen otras con algunas funciones más. Como por ejemplo, en el campo de la arquitectura existe una capa llamada UAR (Urban Augmented Reality) cuya función es mostrar modelos 3D de edificios de Amsterdam y Rotterdam.

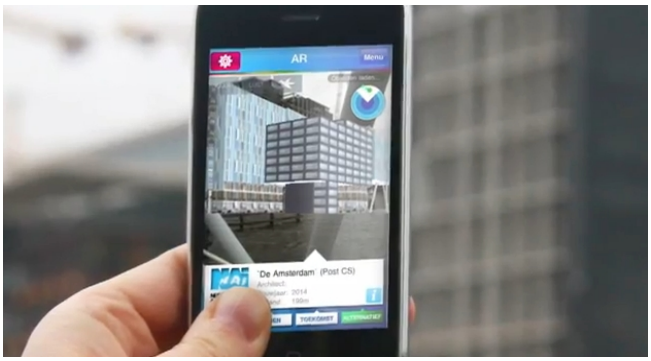

Figura 2: Futura construcción vista en UAR.

Esta guía arquitectónica del Instituto de Arquitectura de Holanda te permite, enfocando hacia una localización de estas ciudades, observar como era un edificio antiguamente antes de ser reformado, visualizar como sería una construcción que se planeó pero nunca se llegó a edificar o como se verá un edificio que aún tiene que terminar de construirse. Como se puede comprobar, es una aplicación de gran utilidad para fines turísticos.

Otra aplicación útil para turistas, esta vez disponible sólo para iPhone es Nearest Tube[4]. En este caso lo que obtenemos es información sobre el metro de Londres (existen versiones paralelas para otras ciudades como Washington, París, Tokio o Madrid) de una manera tridimensional. Utilizando la cámara y la orientación del dispositivo, el programa nos sobreimprime en la pantalla información sobre la distancia a la parada más cercana y que dirección hay que tomar para llegar a ella.

Como se puede apreciar, la combinación entre realidad aumentada y localización por GPS es de gran utilidad y de ahí la ventaja de los dispositivos móviles. Otro ejemplo de ello es TwittARound[5] para iPhone que nos muestra, enfocando hacia una dirección, los mensaje publicados en la red social Twitter de personas que se encuentren cerca en esa dirección. Se puede elegir la distancia a rastrear y veremos sobreimpresa la información correspondiente a cada usuario sobre la pantalla.

En el ámbito del entretenimiento encontramos LevelHead[6]. Se trata de un juego de plataformas que hace uso de la realidad aumentada para mostrar habitaciones sobre cubos, de manera que el usuario tiene que mover un personaje a través de ellos para conseguir completar el puzzle. Cada cubo representa un nivel con distintas puertas y escaleras y para completar el juego el jugador ha de pasar el personaje de una habitación a otra descubriendo donde está la puerta que las conecta.

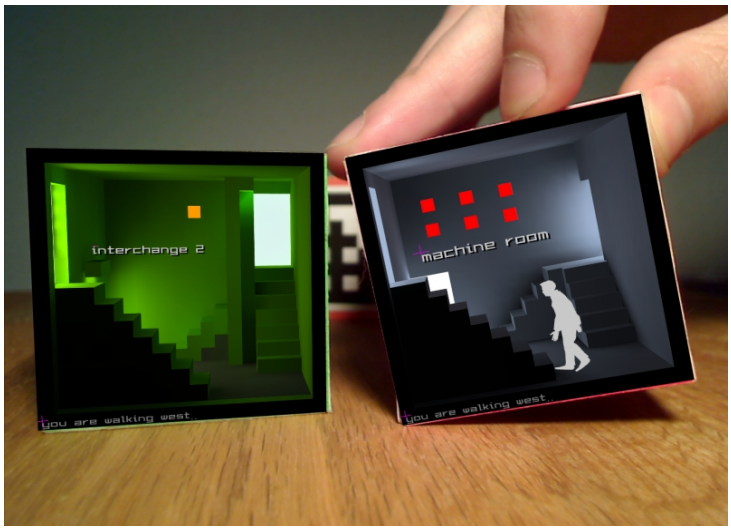

Figura 3: Usuario jugando a LevelHead.

El juego está desarrollado para plataformas Linux, aunque a día de hoy no existen ejecutables disponibles para ningún sistema. El autor proporciona libremente el código fuente e invita a los desarrolladores expertos a intentar compilarlo para el dispositivo que deseen. Se trabaja para que en el futuro dispongamos de una versión final completa para alguna plataforma concreta.

También en el ámbito del ocio encontramos un juego llamado ARQuake[7], que viene a ser una versión del popular juego Quake pero donde se usa la realidad aumentada. Cada uno de los jugadores va equipado con un equipo que incorpora, entre otros elementos, un visor, un acelerómetro y un localizador GPS. Entonces, al ir caminando normalmente, vas viendo los enemigos que vienen hacia ti sobreimpresos en la pantalla.

Figura 4: ARQuake.

Debido al alto coste del equipo necesario para jugar, el uso de la aplicación está reducido a los propios desarrolladores (El Laboratorio de Ordenadores portátiles de la Universidad de Sur Australia). Actualmente, el desarrollo del producto se encuentra parado, pero nos sirve para hacernos una idea de la evolución que la realidad aumentada va teniendo.

Esto es sólo una muestra de algunas aplicaciones de realidad aumentada que existen hoy en día. Vemos que es un campo con mucho potencial y que se puede aplicar a muy diversos ámbitos. Las ideas siguen surgiendo y existen un gran número de proyectos en desarrollo. Muchos de ellos son sólo de investigación, pero con el tiempo esta tecnología se irá incorporando cada vez más al mundo comercial. El número de aplicaciones de realidad aumentada va a seguir creciendo en el futuro y serán cada vez mejores y más útiles.

3.- DISPOSITIVO NOKIA N900

El dispositivo móvil N900[8] es un teléfono con acceso a Internet fabricado por la empresa Nokia y que fue lanzado al mercado a finales del 2009. El N900 posee las funciones clásicas de un teléfono móvil, además de acceso a servicios en línea como gestión de correo electrónico o navegación web.

Viene por defecto con el sistema operativo Maemo 5[9] (también conocido como Fremantle) instalado, el cual es un SO basado en la distribución Debian de Linux. Maemo es un SO multitarea y multiventana con el que disponemos de hasta cuatro escritorios. Excepto algunos componentes de software propietario, la mayoría del código de Maemo es Open Source.

Figura 5: Nokia N900.

3.1.- CARACTERÍSTICAS

El teléfono dispone de un diseño deslizante con un teclado QWERTY y una pantalla táctil de 3,5''. Incorpora un procesador AMR Cortex A8 a 600 Mhz para las aplicaciones y el sistema operativo y un PowerVR SGX 540 GPU que proporciona aceleración de gráficos 3D compatibles con OpenGL ES 2.0. Su memoria interna es de hasta 32 GB y tiene lector de tarjeta microSD. En cuanto a la memoria virtual, tiene una memoria RAM de 256 MB a la que hay que sumarle 768 MB que el SO gestiona como espacio de intercambio. En total 1GB de memoria para aplicaciones.

Su cámara de fotos es de 5 megapíxeles con flash LED y graba vídeos con una resolución de hasta 800x480 píxeles. En cuanto a la conectividad, el teléfono es compatible con las redes de datos GPRS, EDGE, WCDMA, HSPA y puede conectarse a redes Wi-Fi (WLAN IEEE 802.11b/g). Además, también incorpora Bluetooth 2.1 y receptor GPS integrado. Conectándolo por USB, puede ser utilizado como módem.

Como se puede comprobar, el Nokia N900 es un *smartphone* con muchas posibilidades y que teniendo a Maemo como sistema operativo, hacen de este teléfono un dispositivo ideal para desarrolladores. No hay que olvidar que dispone de una terminal desde la que podemos acceder a todos sus recursos.

Sin embargo, hay que tener cuidado con lo que se hace ya que, aunque este SO es muy similar a Debian, no todas las aplicaciones son totalmente compatibles e instalar en el móvil programas incompatibles con el N900 pueden provocar un bloqueo permanente del sistema que obligaría a volver a reinstalar el sistema operativo.

Las capturas de la pantalla del teléfono que se pueden encontrar en este documento han sido realizadas mediante la pulsación simultánea de las teclas “Ctrl”, “⇧” (Shift) y la letra “P”. Esta combinación genera una imagen con una captura de lo que se está visualizando en ese momento y la guarda en la galería de fotos.

3.2.- MODO PASTILLA ROJA

Durante el proceso de instalación de algunas herramientas que se describen en esta memoria, necesitamos acceder a archivos que nos son accesibles por defecto al usuario común. El modo normal de funcionamiento, llamado Modo Pastilla Azul, bloquea algunos directorios y paquetes para evitar un daño por un mal uso accidental, eliminación de archivos del sistema o instalación de paquetes incompatibles. Para desactivar estar restricciones hay que activar el Modo Pastilla Roja[10]. Para ello, hay que abrir una terminal en el N900 y teclear:

```
vi /home/user/.osso/hildon-application-manager
```

Con esto abrimos este archivo de configuración utilizando el editor *vi*, aunque se puede abrir con cualquier otro. Una vez abierto, pulsando al tecla *i* entramos en el modo de edición y en él basta con cambiar el 0 de la línea siguiente línea a 1:

```
red-pill-mode 0
```

Si aparece la línea

```
red-pill-permanent 0
```

también la podemos modificar para hacer que el Modo Pastilla Roja quede activado de forma permanente. Para guardar y salir volvemos al modo de entrada de comando mediante *Esc* y utilizamos *:w* para guardar y *:q* para salir. Para deshacer el proceso y volver al Modo Pastilla Azul basta con volver a editar el archivo y cambiar los 1's a 0's.

3.3.- ADMINISTRADOR DE APLICACIONES Y REPOSITORIOS

Bajo el nombre de “Administrador de aplicaciones” encontramos al gestor de programas para el Nokia N900, mediante el cual podemos descargar nuevas aplicaciones o comprobar y desinstalar las ya existentes. Dado que Maemo es un SO basado en Linux también utiliza repositorios[11] para la instalación de los programas. Se trata de catálogos en línea con la información necesaria sobre aplicaciones, dependencias y compatibilidad de las aplicaciones. Nos permiten descargar nuevos programas de forma automática.

Por defecto, dos repositorios vienen incluidos: “Aplicaciones Nokia” y “Actualizaciones del software del sistema Nokia”. Pero existen algunos más que podemos agregar manualmente. Para ello hay varias formas de hacerlo. La más común es ir a los catálogos de aplicaciones, pulsar en Nuevo e introducir los datos correspondientes.

La otra es pulsar sobre un enlace que contenga el archivo de instalación (.install), que hará que se añada automáticamente el catálogo al administrador. Se ha incluido en el nombre del catálogo de la lista que viene a continuación, un enlace a las referencias de este documento donde se incluye la dirección con el archivo de instalación correspondiente para cada catálogo.

Entre los repositorios que podemos añadir destacamos tres que nos van a ser útiles para la descarga de aplicaciones que hemos utilizado y que no han sido admitidas en los repositorios por defecto.

- Maemo Extras[12]
  En este catálogo encontramos aplicaciones finales, 100% fiables y que han sido probadas y optimizadas correctamente. De hecho, a partir del firmware 2.2009-51-1, este repositorio ha sido renombrado como Maemo y viene incluido por defecto. Los datos necesarios para añadirlo son:
  Nombre de catálogo: Extras
  Dirección web: http://repository.maemo.org/extras/
  Distribución: fremantle
  Componentes: free non-free

Figura 6: Añadiendo el repositorio Extras.

- Extras-testing[13]
  Si añadimos este catálogo podremos descargar aplicaciones en estado de prueba y que aún están por optimizar. Son las futuras aplicaciones que estarán en el catálogo Extras cuando la comunidad Maemo las apruebe, pero mientras tanto pueden producir algunos problemas en el teléfono tales como excesivo consumo de batería, rendimiento deficiente del sistema o uso del espacio total del disco. Por ello, las instalación de aplicaciones de este repositorio se realiza en el directorio */opt* en lugar de en la memoria principal. Así se evitan problemas en caso de que no hubiera suficiente memoria disponible, ya que las aplicaciones no son del todo fiables.
  En este repositorio encontraremos la aplicación arapp de la que se habla en el punto quinto de este documento.
  Dirección web: http://repository.maemo.org/extras-testing/
  Distribución:fremantle
  Componentes: free non-free

- Extras-devel[14]
  En este repositorio encontramos aplicaciones que están en una primera fase de desarrollo, por lo que hay que ser consciente de que instalar software de este catálogo puede provocar daños en el dispositivo. Está orientado a desarrolladores que quieren hacer pruebas con los inicios de sus proyectos. Es recomendable hacer una copia de seguridad del sistema antes de usar estos programas, pues puede que sea necesario volver a instalar el sistema operativo y devolver al dispositivo a la configuración inicial[15].
  Dirección web: http://repository.maemo.org/extras-devel/
  Distribución:fremantle
  Componentes: free non-free

Figura 7: Añadiendo el repositorio Extras-devel.

Existen otros repositorios no oficiales creados por terceros, los cuales hay que tener cautela a la hora de añadirlos pues podemos estar instalando software que nos cause daños o haga que el dispositivo funcione de una manera inestable. Una vez un repositorio haya sido añadido, sus aplicaciones nos aparecerán automáticamente en el apartado de Descargar.

Tenemos la posibilidad además de descargar nuevas aplicaciones desde la consola en modo *root* mediante *apt-get*. Esto puede ser bastante interesante, porque al utilizar el "Administrador de aplicaciones" hay algunas librerías o paquetes que no son visibles al usuario por ser dependencias de otros o porque no se consideran preparados para el usuario común. Sin embargo, si que podremos acceder a ellos y gestionarlos de forma autónoma utilizando este comando de consola.

En el caso de que nos fuera necesario, también podemos añadir nuevos repositorios o direcciones de descarga mediante la consola en el archivo */etc/apt/sources.list*, pero es más recomendable utilizar el gestor oficial. La manera de incluirlos aquí es añadir al fichero la información sobre el catálogo que se quiere incluir. Por ejemplo, para añadir el catálogo *Extras-Testing* habría que introducir:

```
deb http://repository.maemo.org/extras-testing/ fremantle free non-free
deb-src http://repository.maemo.org/extras-testing/ fremantle free
```

Tras añadir algún repositorio nuevo mediante este método, conviene hacer un *apt-get update* para actualizar el catálogo con las nuevas aplicaciones.

Existen además un repositorio sólo para desarrolladores llamado Maemo SDK, que contiene herramientas para programadores que solamente pueden ser descargadas utilizando *apt-get*, no desde el "Administrador de aplicaciones". De aquí necesitaremos instalar el paquete *build-essential*, que contiene algunas librerías de compilación como la herramienta *make* o un compilador de C++.

Este repositorio existe para las distintas versiones de Maemo. En nuestro caso con Maemo 5.0 Fremantle, para añadirlo hay que introducir en el archivo *sources.list* nombrado anteriormente las siguiente direcciones:

```
deb http://repository.maemo.org/ fremantle/sdk free non-free
deb-src http://repository.maemo.org/ fremantle/sdk free
deb http://repository.maemo.org/ fremantle/tools free non-free
deb-src http://repository.maemo.org/ fremantle/tools free
deb http://repository.maemo.org/ fremantle/<token-code> nokia-binaries
```

Donde el campo <token-code> de la última línea se puede conseguir en la siguiente dirección Web tras aceptar la licencia de usuario necesaria para añadir ese paquete:

```
http://tablets-dev.nokia.com/eula/index.php
```

Una vez incluido, se puede instalar el paquete con:

```
apt-get install build-essential
```

Hay que tener en cuenta que con todos los catálogos que hemos añadido, el "Administrador de aplicaciones" puede funcionar de una manera bastante lenta. Por tanto, una vez instalados los paquetes necesarios puede ser conveniente quitar los repositorios que no se vayan a utilizar más para obtener un mejor rendimiento a la hora de abrir el administrador.

3.4.- CONEXIÓN Y TRANSFERENCIA DE ARCHIVOS

Tanto para ejecutar la aplicación que se ha estudiado, como para ver y editar el código de la misma, lo hacemos directamente sobre el dispositivo móvil. Sin embargo, para obtener una mayor comodidad a la hora de trabajar nos conectamos al mismo mediante el protocolo SSH desde un PC. Así, estamos directamente ejecutando las ordenes en el Nokia pero utilizando el teclado del ordenador.

En particular, utilizamos el software OPEN SSH[16] junto con el Mad-Developer. OPEN SSH es una versión gratuita de un software que implemente la conexión mediante el protocolo SSH. Tanto la versión servidor como la cliente están disponibles para su descarga, aunque en este caso solamente necesitamos la de servidor. Obviamente, en el PC que utilizamos deberá estar instalada la versión cliente para poder realizar la conexión. Por su parte, Mad-developer es un software de Maemo para facilitar el acceso de los desarrolladores al dispositivo. Ambas aplicaciones están disponibles para su descarga e instalación en el *Administrador de aplicaciones* del teléfono.

Para realizar la conexión seguiremos los siguientes pasos. En primer lugar, hay que conectar el Nokia N900 al PC utilizando el cable USB que viene disponible en la compra del terminal. Al realizar la conexión, nos pregunta el modo de conexión. Tenemos que seleccionar *Modo PC Suite*.

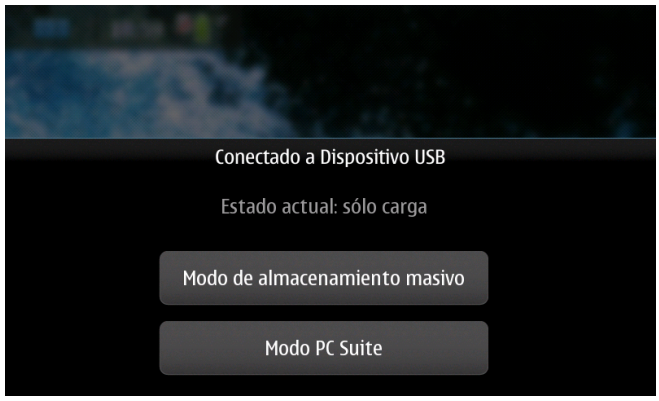

Figura 8: Selección del modo de conexión.

Después, hay que abrir el Mad-developer en el dispositivo y pulsar en botón *Edit* para así asignar una IP a la conexión. Si aparecen correctamente los números, entonces se pulsa *Configure* para fijar los valores. En la imagen se pueden ver los valores que se utilizan usualmente.

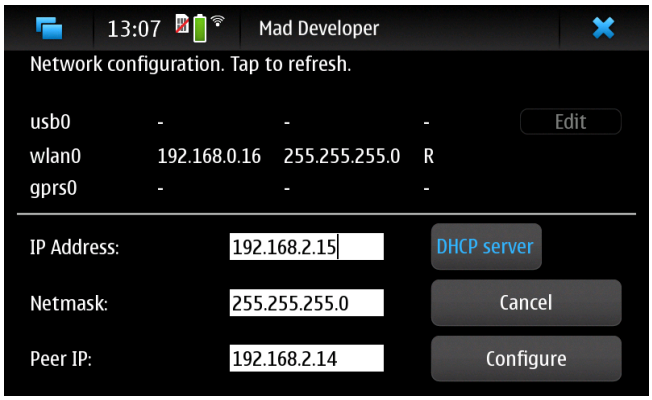

Figura 9: Edición de la dirección IP.

Entonces ya podremos conectarnos, pero dado que lo vamos a hacer en modo desarrollo es necesario primero obtener el password. Como Mad-Developer es un programa pensado especialmente para crear una conexión robusta de los desarrolladores al dispositivo, nos permite obtener una clave válida para la cuenta *developer*.

Para ello, pulsamos el botón *Developer Password* y se nos proporcionará contraseña a utilizar. Es importante siempre obtener primero la clave y luego realizar la acción de conexión puesto que si lo hacemos en orden inverso, la contraseña no servirá. Solamente es válida desde el momento en que se visualiza en la pantalla.

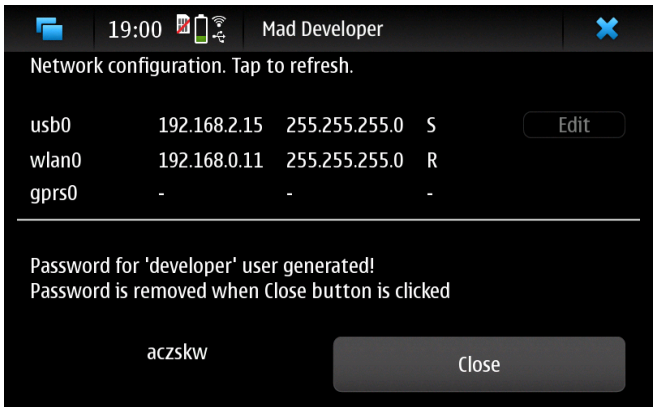

Figura 10: Clave para desarrollador.

Una vez realizado el paso anterior, conectaremos desde una terminal del PC mediante la orden SSH de la siguiente manera.

```
ssh developer@192.168.2.15
```

Estamos conectando a la IP 192.168.2.15 (que corresponde al Nokia) como desarrollador. Introducimos el password y se completa la conexión. Ahora ya estamos trabajando sobre el dispositivo en el directorio de desarrollo. Solamente falta identificarnos como usuario *root*, ya que necesitamos tener los permisos adecuados para realizar algunas acciones. Todos los comandos y acciones realizadas que se describen en este documento se suponen hechos en modo root. Así que será lo primero que tenemos que hacer nada más conectarnos. Para ello, tecleamos la orden siguiente orden sudo:

```
sudo gainroot
```

Para cuando tengamos la necesidad de transmitir archivos desde el PC al teléfono podemos utilizar la orden SCP. Secure Copy o SCP es un medio de transferencia segura de archivos informáticos entre un host local (el PC en nuestro caso) y otro remoto (N900), usando el protocolo Secure Shell (SSH). Un ejemplo de su uso desde la terminal del PC sería el siguiente:

```
scp imagen.png developer@192.168.2.15:imagen.png
```

Si en algún momento pasamos algún archivo ZIP comprimido necesitaremos instalar alguna utilidad para descomprimir ese tipo de archivos ya que no viene incluida ninguna por defecto. Por ejemplo, podemos conseguir del *Administrador de aplicaciones* InfoZip[17] y ya tendremos una orden de comando para comprimir y descomprimir archivos ZIP.

4.- HERRAMIENTAS DE REALIDAD AUMENTADA

Las funciones básicas que una aplicación de realidad aumentada debe realizar son dos. En primer lugar, es necesario un reconocimiento de imágenes. Bien sea para poder localizar un marcador o para detectar alguna particularidad en la imagen que nos indique qué estamos viendo y donde se debe proyectar la parte virtual. Es necesario pues, disponer de una herramienta de reconocimiento y orientación espacial. En segundo lugar, hay que superponer en tiempo real las imágenes virtuales sobre las reales.

Por ello, listamos a continuación una serie de herramientas software que se hacen necesarias para las dos tareas nombradas anteriormente. Muchas de estas herramientas son de código abierto, lo que permiten una sencilla difusión y utilización, lo que está provocando que la realidad aumentada esté entrando en una era de importante crecimiento. Se nombran las más conocidas y accesibles, pero nos centraremos en dos de ellas para explicar, además, su uso e instalación dentro del marco que supone este proyecto.

- El SDK para realidad aumentada de la compañía Qualcomm[18] es un kit de desarrollo de software para los sistemas operativos móviles Android e iOS. En un principio fue lanzado en 2010 para Android 2010 y posteriormente, en julio de 2011, la versión para iOS fue publicada. Este kit pretende ser una herramienta de ayuda para desarrolladores que quieren construir aplicaciones de realidad aumentada para estos sistemas sin tener que crear todo el código desde cero.

- Atomic Authoring Tool[19] es un software para la creación de escenas de realidad aumentada cuyo funcionamiento es el mismo que el de la mayoría de aplicaciones de RA; el programa detecta el patrón del marcador y muestra el objeto 3D. Su principal ventaja es que es multiplataforma, ya que está disponible tanto para Ubuntu, Windows y Mac. Hace uso de ArToolKit para la parte de reconocimiento y salida de vídeo.

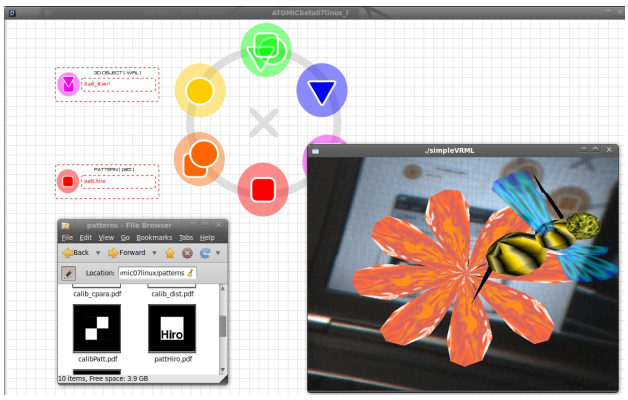

Figura 11: Aplicación Atomic funcionando.

– Atomic Web Authoring Tool[20] es un proyecto derivado del anterior, cuya particularidad reside en que está enfocado a la creación de aplicaciones de realidad aumentada para insertarlas en sitios Web. Su objetivo principal es proporcionar a la comunidad una herramienta, de código abierto, que se pueda modificar con facilidad y que no exija demasiados conocimientos técnicos para integrar la tecnología de la realidad aumentada dentro de cualquier sitio Web.

  Disponible para Windows y Ubuntu, su código está escrito en el lenguaje de programación Processing[21], especialmente pensado para el desarrollo de animaciones, imágenes e interacciones. Hace uso de una librería GPL, también basada en ArToolKit, llamada FLARToolKit[22].

– Look![23] es un *framework* para el desarrollo de aplicaciones de RA para Android. De código abierto, contiene herramientas para el dibujado de objetos simples con colores y texturas. Además, permite utilizar la conexión Wi-Fi del móvil para obtener la localización exacta en un espacio cerrado donde, en ocasiones, el GPS no es lo suficientemente preciso.

  Es un software de reciente creación y, por tanto, pocas aplicaciones lo han utilizado pero permite a los desarrolladores de Android ahorrar trabajo en la creación de sus aplicaciones.

– SSTT[24] es una biblioteca para el posicionamiento en la realidad aumentada mediante marcadores, creada por Hartmut Seichter y cuya principal ventaja es la versatilidad que tiene, ya que está disponible para Windows, Mac y Linux. Además, tiene versiones optimizadas para plataformas móviles como Android, Maemo o iOS (SSTT Mobile).

  En el ejemplo de la siguiente figura, creado con SSTT, se ve como se puede utilizar la realidad aumentada para poder visualizar en 3D un edificio cuya información estás viendo en un libro sobre arquitectura.

Figura 12: Realidad aumentada con SSTT.

– Osgart[25] es una biblioteca que mezcla ArToolKit con OpenSceneGraph[26] para intentar simplificar el desarrollo de las aplicaciones de RA. La herramienta GNU GPL OpenSceneGraph se usa para realizar gráficos en 3D de alta calidad, de manera que Osgart aprovecha sus ventajas y las de ArToolKit para crear una herramienta de desarrollo para aplicaciones que combine las ventajas de las dos herramientas. Dispone también de una versión profesional paralela, enfocada para la creación de aplicaciones comerciales.

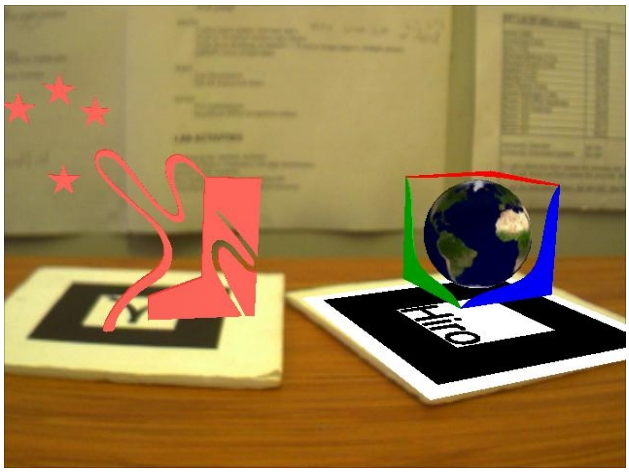

Figura 13: Visualización de objetos con Osgart.

4.1.- ARTOOLKIT

ArToolKit[27] es una librería software que se emplea en el desarrollo de aplicaciones de realidad aumentada. Tal y como se ha podido comprobar en la descripción de las anteriores, es usada en multitud de proyectos de diversa índole.

Su desarrollo inicial fue realizado en 1999 por el Doctor Hirokazu Katoe, mientras que su mantenimiento actual está respaldado por el Laboratorio Tecnológico de Interfaz Humana de la Universidad de Washington[28], el HIT Lab NZ de la Universidad de Catenbury de Nueva Zelanda y la compañía ARToolworks[29] de Seattle.

Además de que su código completo sea distribuido bajo licencia GNU, otra ventaja de ArToolKit es que tiene distribuciones para Linux, IRIX, Windows y Mac. Lo cual, unido a sus múltiples funciones, hace que sea una de las herramientas más utilizadas.

Como función principal ArToolKit permite calcular, en tiempo real, la posición y orientación de la cámara respecto a un marcador que aparece en la imagen. También posee algunas funciones de calibrado.

Dado que ARToolKit no tiene una versión específica para Maemo, hemos intentado portar la versión de Linux disponible en su Web. Tras pasar el código descargado al dispositivo N900, y antes de instalarlo, hay que asegurarse de que tenemos las dependencias de paquetes necesarios.

La mayoría de ellos los podremos instalar mediante la orden *apt-get install* y el nombre del paquete. Son los siguientes:

- freegult3
- freegult3-dev
- libxi-dev
- libxmu-dev
- libv4l-dev
- libgstreamer0.10-dev

Si tenemos algún problema con los catálogos o alguno de los paquetes no está disponible en los repositorios también podemos instalarlo manualmente. Para ello hay que buscar el nombre del paquete en la página oficial de Maemo y descargar el fichero en formato *.deb* disponible. Para instalar un paquete basta con teclear:

```
dpkg –i nombre_del_paquete
```

También necesitamos instalar OpenGL, pero el problema es que no lo tenemos disponible para Maemo. En su lugar, haremos uso de OpenGL-ES[30], que es una variante simplificada de OpenGL desarrollada por el Grupo Khronos y diseñada específicamente para dispositivos empotrados como teléfonos móviles, PDAs, videoconsolas, etc...

Afortunadamente, de OpenGL-ES si disponemos una versión en Maemo lista para instalar. La instalación de los paquetes requeridos para compatibilidad tanto con la versión OpenGL-ES 2.0 como con la 1.0 y 1.1 es la siguiente:

```
apt-get install libgles2-sgx-img-dev
apt-get install opengles-sgx-img-common-dev
apt-get install libgles1-sgx-img-dev
apt-get install libgles1-sgx-img libgles2-sgx-img opengles-sgx-img-common
```

Tras esto, encontramos un par de errores más al hacer el *make* de la aplicación. Pese a haber instalado anteriormente el paquete *freeglut3*, el proceso de compilación nos indica que se necesitan un par de ficheros de cabecera más: *glut.h* y *egl.h*. Al igual que ocurre con OpenGL, tenemos que descargar la versión paralela para dispositivos portables llamada Glut-ES[31]. Podemos hacerlo desde su página oficial y tras ello, copiar el contenido de la carpeta “Inc” en “*/usr/include”*. Este directorio es la ruta donde los programas de compilación buscan las bibliotecas incluidas y los ficheros de inclusión que le son necesarias. En el caso de C++, vienen indicadas en el código mediante la línea

```
#include<nombre_de_la _biblioteca>
```

El *egl.h* lo bajamos[32] de la página de Khronos y lo copiamos en el mismo directorio dentro de la carpeta EGL.

Después de todo este proceso, ya podremos compilar ARToolKit sin problemas. Para ello, en primer lugar, es recomendable utilizar la orden “make clean” para eliminar ficheros sobrantes que puedan ocasionar algún conflicto. Después ya podemos escribir “./Configure”, tras lo cual nos hará algunas preguntas previas de configuración.

```
Select a video capture driver.
1: Video4Linux
2: Video4Linux+JPEG Decompression (EyeToy)
3: Digital Video Camcoder through IEEE 1394 (DV Format)
4: Digital Video Camera through IEEE 1394 (VGA NONCOMPRESSED Image Format)
5: GStreamer Media Framework
```

Elegiremos la opción 5, ya que hemos instalado la librería multimedia Gstreamer. En las siguientes dos preguntas respondemos afirmativamente con la opción “y”.

```
Do you want to create debug symbols? (y or n)
Enter : y

Build gsub libraries with texture rectangle support? (y or n)
GL_NV_texture_rectangle is supported on most NVidia graphics cards and on ATi
Radeon and better graphics cards
Enter : y
```

Si todo ha ido bien, podemos ejecutar la orden “make” y el proceso de instalación de ARToolKit se completará sin problemas. Adicionalmente, podemos utilizar *gstreamer* para comprobar si éste procesa correctamente la entrada desde la cámara y si ArToolKit sabe como tratar la fuente de vídeo.

Con el siguiente comando se verifican todos los parámetros y se abre una ventana donde se visualiza el vídeo final mostrado por *gstreamer*:

```
gst-launch-0.10 v4l2src device=/dev/video0 ! ffmpegcolorspace ! identity
                        name=artoolkit ! xvimagesink
```

En lo referente a la manera de funcionar de esta herramienta, podemos distinguir una serie de etapas en su proceso de funcionamiento hasta que se visualiza el objeto virtual sobre la imagen real. En primer lugar, la fuente de vídeo desde la cámara llega al ordenador donde el software analiza la imagen en busca de algún marcador.

Estos marcadores son unas imágenes cuadradas de color negro sobre las que hay un código formado por cuadros blancos correctamente posicionados para formar una identidad reconocible por el programa. Se analiza cada fotograma de vídeo entrante transformándolo en imagen binaria para buscar en ella alguna forma cuadrada.

Dentro de cada cuadrado hay un símbolo identificativo que va asociado al objeto a representar. Suelen ser símbolos bastante simples que simplemente ayudan a discernir entre los objetos que tengamos cargados en memoria.

Figura 14: Marcadores utilizados por Arapp.

Una vez localizado un marcador, se realizan una serie de cálculos matemáticos para calcular la posición y orientación de la cámara respecto al cuadrado detectado. Se utiliza el sistema de matrices de OpenGL para representar la posición del marcador y realizar las transformaciones necesarias para adaptarlo al sistema de coordenadas de la cámara.

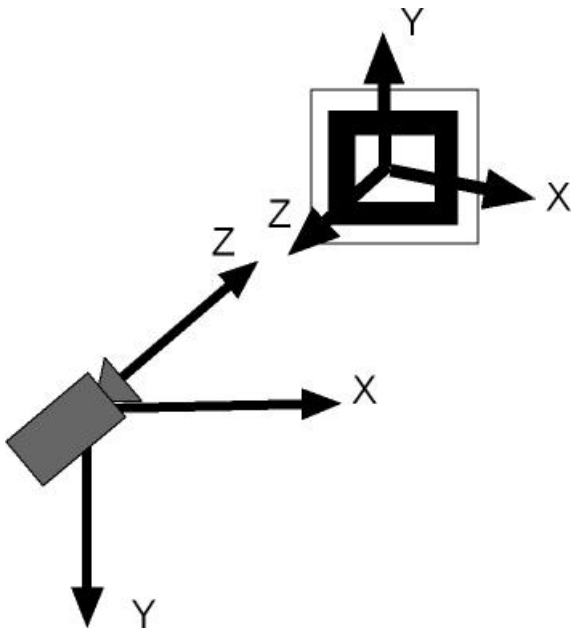

Figura 15: Sistemas de coordenadas.

El siguiente paso, una vez reconocido el marcador, es dibujar el modelo 3D de manera que tenga la misma orientación y posición que el patrón. Tras aplicarle las transformaciones 3D, el objeto queda dibujado directamente sobre la imagen de vídeo real, creando así la realidad aumentada. Todo este proceso se realiza en tiempo real mientras se va procesando el flujo de vídeo recibido de la cámara.

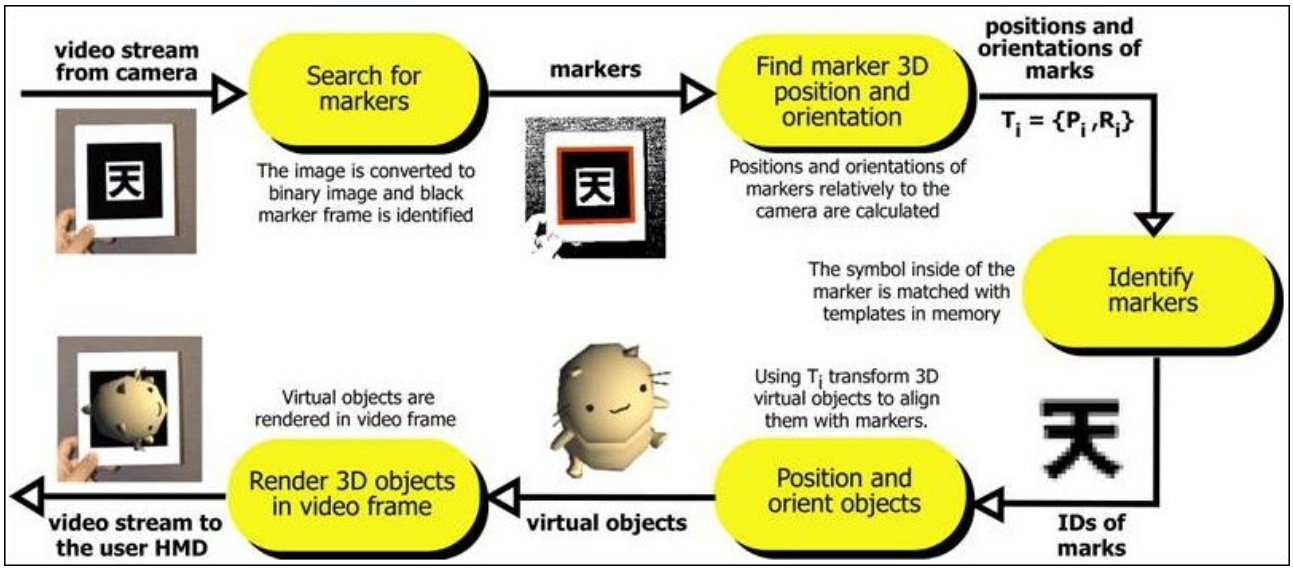

Figura 16: Resumen de los pasos que realiza ARToolKit.

4.2.- ARTOOLKITPLUS

ArToolKitPlus[33] nace como una extensión al código de ArToolKit, al que no sólo se le añaden nuevas características, sino que además se optimiza para dispositivos móviles. Fue desarrollado por el Christian Doppler Laboratory de la Universidad Tecnológica de Graz como parte de sus proyectos "Handheld AR" que pretenden acercar la realidad aumentada a los dispositivos móviles. Debido a las peticiones recibidas, ArToolKitPlus fue hecho publico bajo licencia GPL, aunque sus creadores advierten que no está pensado para desarrolladores que se están iniciando en la realidad aumentada, sino que está recomendado para programadores avanzados en C++.

Pese a que es una mejora de ArToolKit y aporta nuevas funcionalidades, la creación de una nueva interfaz de programación de la aplicación (API) respecto al original hace que no todas las funciones de ArToolKitPlus sean compatibles con las de su predecesor.

Además, es importante destacar que esta librería lleva desde junio de 2006 sin ser actualizada, lo que puede aumentar las incompatibilidades y hacer que, con el paso del tiempo, algunas de sus funciones vayan quedando obsoletas. Aún así, es una librería muy utilizada debido a sus prestaciones. Por ejemplo, la aplicación *arapp* tratada en el quinto punto de este documento hace uso de ArToolKitPlus para el cálculo de la posición y orientación de la cámara respecto a los marcadores.

A la hora de instalar ARToolKitPlus en el N900 no encontramos los mismos problemas de compatibilidad que con QtCreator, porque el mismo autor de la aplicación de *arapp* se ha encargado de mantener la versión de la librería de *ArToolKitPlus* para Maemo[34], eliminando así las posibles incompatibilidades. Podemos encontrar el programa en el repositorio de Extras-devel y descargarlo mediante *apt-get* con la orden:

```
apt-get install artoolkitplus-dev
```

Con esto, la herramienta quedará instalada en el sistema.

5.- APLICACIÓN ARAPP.

5.1.- DESCRIPCIÓN

Arapp[35] (Augmented Reality APPlication) es el nombre de una aplicación desarrollada por Pavel Rojtberg[36]. Está escrita utilizando Qt[37] y se encuentra todavía en fase de desarrollo. Qt es una biblioteca multiplataforma, desarrollada por la empresa de telefonía móvil Nokia, que mediante el uso del lenguaje C++ da soporte a la creación de aplicaciones. Está enfocado hacia el desarrollo de programas con interfaz gráfica de usuario, pero también se puede usar para programas sin interfaz gráfica que funcionen mediante consola.

Esta aplicación de realidad aumentada posee una interfaz táctil de manera que el usuario puede interactuar con los objetos virtuales y otros parámetros del sistema. Los objetos virtuales son representados en tiempo real en relación con el entorno físico utilizando la cámara del dispositivo. Todo ello con una iluminación realista y la representación de modelos de alta resolución virtual, lo cual se logra a través de aceleración hardware para los gráficos y sus mapas de sombra.

En cuanto a su programación, nos centraremos ahora en describir de que se encarga cada una de las clases que encontramos en el código. La aplicación utiliza la biblioteca Qt de C++, por lo que lo primero que se hace en el *main* es crear una *Qapplication que es* la clase que maneja la interfaz gráfica del usuario. A partir de ahí, tenemos dos clases principales: *View* y *Scene*. El main se encarga de crear una escena vacía y se la pasa al constructor de la clase *View*.

La clase *View* es la que se encarga de los menús, de gestionar los eventos que se producen y de llamar a los métodos de la clase *Scene* para indicar cuando se tiene que generar cada cosa. Entre sus funciones está, además, la de inicializar la librería gráfica y detectar cuando el usuario pulsa en algún punto de la pantalla o realiza alguna interacción utilizando las teclas.

Por su parte, la clase *Scene* es la que se encarga de gestionar lo que vemos y representar los gráficos que conforman la realidad aumentada sobre la imagen real. Para ello, en primer lugar hay que crear el sistema de coordenadas y adaptarlo a la imagen de vídeo con la que se está trabajando. Se calculan los vectores de rotación y la distancia de la cámara a los marcadores detectados. En esta fase es donde *arapp* hace uso de la librería *ARToolKitPlus* mediante el uso de la clase *TrackerSingleMarker*. Con los métodos de esta clase se obtiene también la matriz de proyección, necesaria para representar los objetos correctamente acorde a la perspectiva que tenga la cámara respecto a los marcadores en ese instante.

En la clase *Scene* también se indican las rutas donde se encuentran los modelos 3D definidos en archivos externos, se cargan los datos y se representa toda la escena incluyendo los objetos y las anotaciones que el usuario haya podido realizar. Existe una clase *Object* para las operaciones relacionadas con los objetos donde están los métodos para cargar las texturas, almacenar y modificar la matriz de transformación, etc... En la clase *ObjReader* es donde se lee línea a línea los ficheros OBJ y MTL para obtener todos los datos necesarios para el renderizado del objeto.

Por su parte, las funciones de transformación y representación de las anotaciones vienen en la clase *CanvasPlane*, mientras que para el cálculo de las sombras se apoya en las clases *VideoQuad* y *ShadowQuad*. El flujo de entrada de vídeo y los parámetros de la cámara los controla la clase VideoSrc.

5.2.- INSTALACIÓN

A la hora de intentar compilar está aplicación en el N900, la intención inicial era hacerlo utilizando el software QtCreator desde el PC mediante compilación cruzada. De esta forma, estaríamos compilando la aplicación en el PC, pero se ejecutaría en el dispositivo móvil. Este método nos proporcionaría varías facilidades, ya que es bastante cómodo trabajar desde el PC con el QtCreator y por otra parte, podríamos hacer uso de la cámara del N900 (que estaría conectado mediante USB) mientras está la aplicación ejecutándose en él.

Uno de los problemas encontrados en este punto es que *arapp* hace uso de ArToolKitPlus, por lo que también había que incluirlo en el proyecto de Qt y compilarlo para que se pudiera usar. Sin embargo, esta librería lleva sin ser actualizada desde el 2006 por lo que encontramos algunos problemas de incompatibilidad con la versión de QtCreator (2.0.95 con Qt 4.7.1) que estábamos utilizando. Concretamente, en la librería ArToolKitPlus algunas funciones del fichero *TrackerSingleMarker.h* de las que hacía uso *arapp* estaban declaradas como funciones virtuales y no eran referidas como tal.

Estas funciones tenían su definición en otro fichero llamado *TrackerSingleMarkerImpl.h*, donde venía también declarado el constructor de la clase. Por tanto, había que modificar la cabecera de algunas funciones para que no fueran virtuales y traer algunas definiciones de métodos del fichero *Impl.h* al fichero *.h* para que fuera encontrado durante el proceso de compilación. Así, la clase dejaba de ser abstracta, podía ser instanciada por el código de *arapp* y eliminábamos el error. Se incluye en el apéndice 1 el código del fichero modificado.

De todas formas, tras seguir intentando arreglar los problemas que surgían nos dimos cuenta de que iba a ser imposible hacer una compilación tal y como la estábamos planteando debido a incompatibilidades entre la arquitectura del procesador del PC que utilizamos y la del dispositivo Nokia. Mientras que por un lado utilizamos un procesador Intel con arquitectura x86, el N900 posee un procesador ARM con arquitectura Cortex ARMv7-A, lo que supone un problema al intentar compilar de la manera en que la que lo hacíamos con QtCreator.

Llegados a este punto y buscando posibles alternativas para continuar trabajando con la aplicación, podíamos utilizar el kit de desarrollo Scratchbox o intentar compilar la aplicación directamente trabajando sobre el N900. Scratchbox[38] es un conjunto de herramientas para las distribuciones Linux (especialmente destinado para Debian) que fueron diseñadas para el desarrollo en Maemo y que se puede utilizar para realizar compilación cruzada con un dispositivo móvil. Es un software distribuido bajo licencia GNU GPL (General Public License) y es compatible tanto con la arquitectura ARM como con la x86.

La otra alternativa era, teniendo el código fuente de la aplicación, introducirlo en el N900 e intentar hacerlo funcionar desde allí. Finalmente, nos decantamos por esta opción para poder trabajar sobre el dispositivo móvil directamente y no depender de otro software adicional. Además, conectándonos mediante *ssh* (tal y como se explica en el punto 3.4) podemos seguir utilizando el teclado del PC y ahorrarnos la incomodidad de estar tecleando desde el teclado del móvil.

La aplicación ya compilada se encuentra en el repositorio Extra-Devel, pero como queremos poder estudiar su código y modificarla tendremos que bajar el código fuente y compilarlo nosotros mismos. Este código está disponible para bajar en la página Web de la aplicación y de la sección de paquetes de la Web de Maemo[39]. Hay que tener en cuenta que, ya que la aplicación se encuentra aún en fase experimental, pueden producirse cambios en el código al tiempo que se elabora este documento.

Una vez pasadas las fuentes al N900 y antes de hacer el make, tenemos que instalar las librerías de las que tiene dependencia la aplicación para poder luego compilarla sin errores. En primer lugar, necesitamos tener el paquete *ArToolKitPlus*, el cual podemos conseguir fácilmente mediante el comando *apt-get*, tal y como se indica en el punto 4.2. Por su parte, si no tenemos OpenGL-ES, la información al respecto viene en el apartado 4.1 del documento.

Como hace uso de Qt, tenemos que instalar también las librerías correspondientes de Qt4. Utilizando *apt-get* podemos instalar tanto este paquete como todos los demás necesarios. Se listan a continuación la serie de ordenes que realizan esa tarea:

```
apt-get install libqt4-dev
apt-get install debhelper
apt-get install libgstreamer0.10-dev
apt-get install libgstreamer-plugins-base0.10-dev
apt-get install libx11-dev
apt-get install x11proto-core-dev
apt-get install mesa-dev
```

Algunas de estas librerías se encuentran en el repositorio de desarrollo, así que se tiene que tener agregado para poder usarlo. Llegados a este punto ya tenemos las librerías necesarias pero aún no vamos a poder ejecutar la orden *make* sin que nos de errores.

Ocurre que dado que la aplicación es para Maemo, está preparada para compilarla a través del Scratchbox, no directamente sobre el sistema como estamos haciendo nosotros. Entonces hay que modificar unas cuantas líneas del fichero Makefile para indicar la ruta correcta de algunos archivos.

Concretamente, en el archivo de compilación vienen indicada la ruta de algunos ficheros necesarios de la librería Qt4 que hemos instalado anteriormente, como si se encontraran dentro de un directorio con la siguiente ruta:

```
/targets/maemo5-arm-21f2fec326c8f7ade2255079f2e72585f0974e47/usr/share/qt4/
```

El directorio *targets* es el que utiliza Scratchbox para indicar donde se encuentra el dispositivo sobre el que se está haciendo la compilación cruzada. Como nosotros estamos trabajando directamente sobre el dispositivo, el programa de compilación producirá un error con esta ruta. Para solucionarlo, simplemente tenemos que eliminar la parte de delante y dejarlo para que empiece directamente con el directorio */usr*. Hay que hacerlo por cada línea donde se haga referencia a un fichero que esté dentro de la ruta indicada anteriormente. En el apéndice 2 se incluye el código con las modificaciones del fichero Makefile.

Por último, otro error que nos puede aparecer al ejecutar la orden make es el siguiente:

```
/usr/bin/ld: cannot find -lgstapp-0.10
```

El problema aquí es que no encuentra un archivo indicado por el flag *gstapp-0.10*. Los archivos necesarios ya fueron instalados con la orden *apt-get* anteriormente al instalar los distintos paquetes, pero si nos fijamos en el make lo que se necesita es un archivo llamado *libgstapp-0.10.so*. Es un fichero correspondiente a la biblioteca Gstreamer y que se debería encontrar en el directorio */usr/bin/ld*.

Si buscamos dentro de ese directorio, lo que encontramos es un archivo llamado *libgstapp-0.10.so.0*, es decir, el fichero que nos falta pero con un 0 al final del nombre. Esta clase de terminación numérica es utilizada en los nombres de los archivos de biblioteca por algunos sistemas Unix para indicar el número de revisión dentro de las biblioteca de enlace dinámico a la que hacen referencia.

La solución es sencilla, pues basta con crear un enlace simbólico del archivo con terminación *.so.0* al que termina en *.so*. Así no eliminamos el fichero del directorio, pero el programa de compilación será capaz de encontrarlo. Basta con introducir el siguiente comando:

```
ln -s libgstapp-0.10.so.0 libgstapp-0.10.so
```

Con todo esto, ejecutando la orden *make* del directorio de la aplicación el proceso de compilación se realizará correctamente, generando así el ejecutable llamado *arapp*.

5.3.- USO Y FUNCIONAMIENTO

Una vez la aplicación ha sido correctamente configurada y compilada, basta con ejecutar el archivo *arapp* para ponerla en funcionamiento. Podemos distinguir dos etapas durante la ejecución del programa. La primera es para la creación de la escena, que incluye la lectura de los objetos a representar, el posicionamiento de la cámara, la detección de los marcadores (mediante ARToolKitPlus) y el renderizado de los objetos en la escena. De esta parte se encarga la clase *Scene*.

En segundo lugar, la clase *View* mantiene el render actualizado y se encarga de controlar los eventos de entrada tanto por pulsación de teclas como de interacción directamente sobre una posición en la pantalla.

Por defecto, la aplicación viene con los modelos de dos edificios; una casa y una granja. Podemos encontrar sus definiciones, respectivamente, dentro de las carpetas *casa* y *farm* del directorio *models*. Para cada uno de ellos encontramos un archivo de extensión *.obj* y otro *.mtl*, ambos pertenecientes al formato de descripción de modelos Waveform, el cual viene detallado en el punto siguiente. Se incluye también una serie de archivos de imagen en formato PNG donde se almacenan las texturas de los objetos. Hay 20 para la casa, mientras que para la granja se dispone de uno donde se condensa toda la información necesaria.

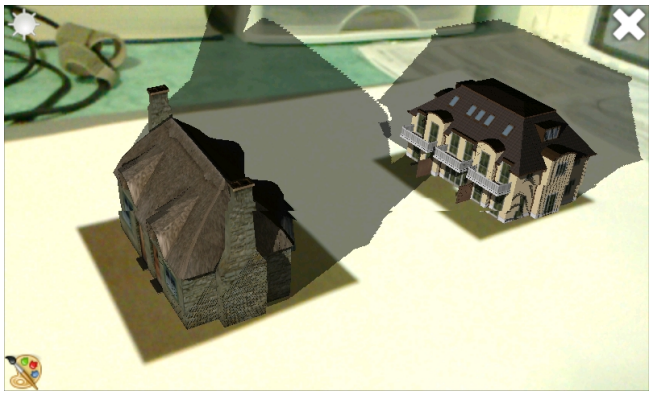

Figura 17: Aplicación Arapp funcionando

Cuando ya está en funcionamiento, podemos ver en la pantalla la imagen que se recibe de la cámara de vídeo del teléfono mientras la aplicación intenta detectar algún marcador o marcadores donde ubicar los objetos. Una vez detectado, el objeto correspondiente es representado en el lugar, adecuándose a la posición e inclinación de la cámara. De este modo, cuando varía la orientación del teléfono también cambia la forma en que vemos al objeto representado. Además, se muestra una sombra para cada uno de ellos.

También podemos interactuar para hacer que cambie la dirección de la sombra, como si variara la dirección de la luz. Para ello, primero hay que pulsar en el icono del sol que viene en la esquina izquierda superior de la pantalla. Con esto, se actualiza la posición de la sombra con la ubicación actual de la cámara respecto al objeto. Después se pueden usar las teclas ← y → del teclado para ir variando la incidencia de la luz a un lado u a otro.

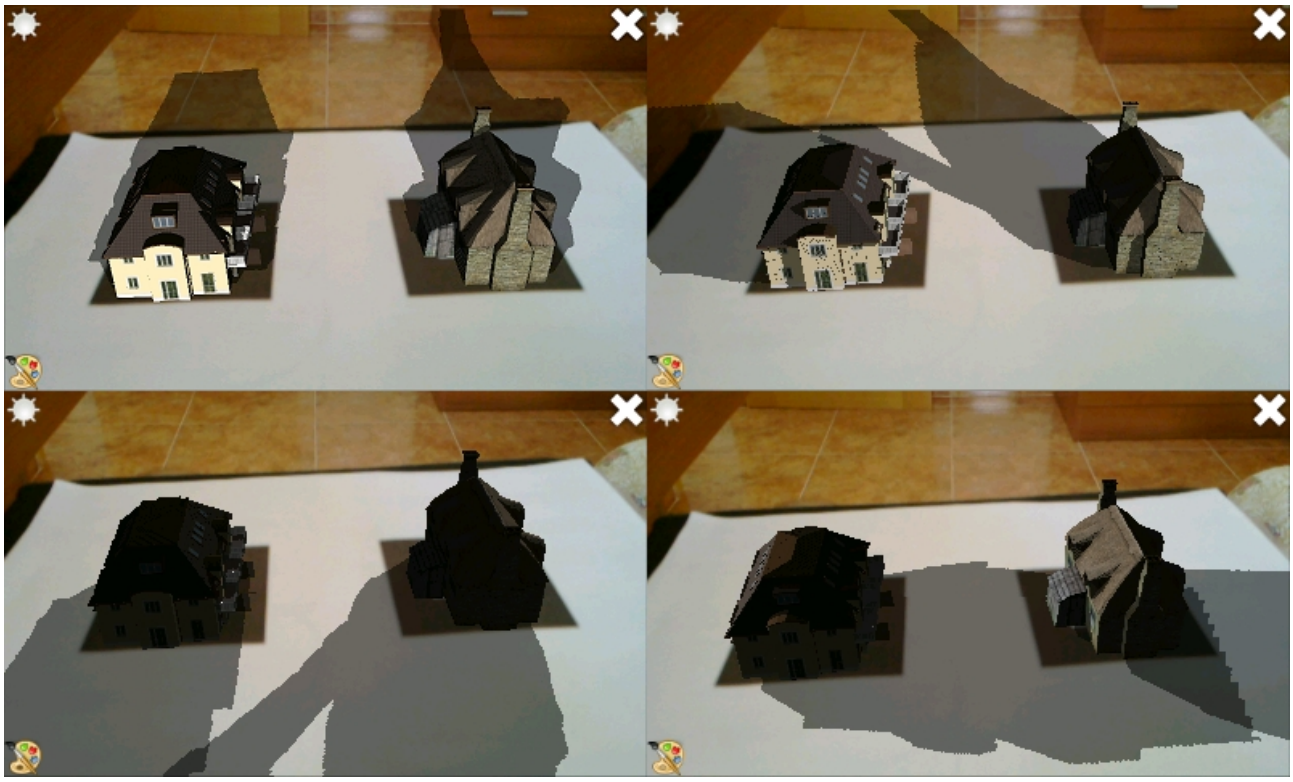

Figura 18: Secuencia de imágenes donde se varía la dirección de la sombra

Cada uno de los objetos representados tiene asociado un índice dentro del vector de objetos que representa el orden en que el programa ha cargado los modelos. Al pulsar sobre cada uno de ellos en la pantalla, dicho índice aparece por la salida estándar de la aplicación. En nuestro caso, la consola desde donde la hemos ejecutado. El último seleccionado se convierte en el *objeto activo* (por defecto el 0) y podemos interactuar con él para variar el tamaño en que aparece.

Cuando se manejan los eventos de pulsación de teclas en el archivo *View.cpp*, las teclas que se utilizan para variar la escala son *F7* y *F8*. Dado que dichos botones no aparecen en el teclado del N900, las hemos cambiado por *A* y *S* para que la interacción sea posible (Apéndice 3).

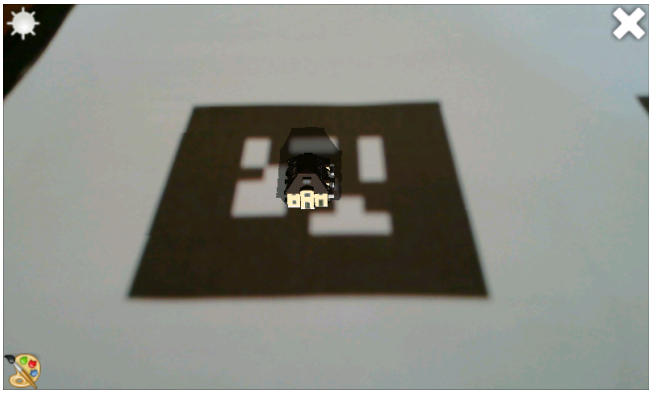

Figura 19: Reducción de la escala de la casa.

Así, tras marcar el objeto que queramos como objeto activo, pulsamos repetidas veces las teclas mencionadas anteriormente para aumentar o reducir el tamaño del objeto 3D.

En resumen, el objeto representado se posiciona sobre el marcador, ajustándose a la posición e inclinación de la cámara y a las características que indiquemos tales como la dirección de la luz y la escala.

Figura 20: La casa tras aumentar su tamaño.

En la esquina inferior izquierda de la pantalla se ubica una paleta de pintura que sirve para poder realizar anotaciones o dibujos sobre la imagen que se está visualizando. Su modo de funcionamiento es simple, basta con pulsar sobre dicho dibujo y en ese momento la imagen se queda congelada. Entonces, sobre la pantalla táctil se puede hacer el dibujo o trazado deseado.

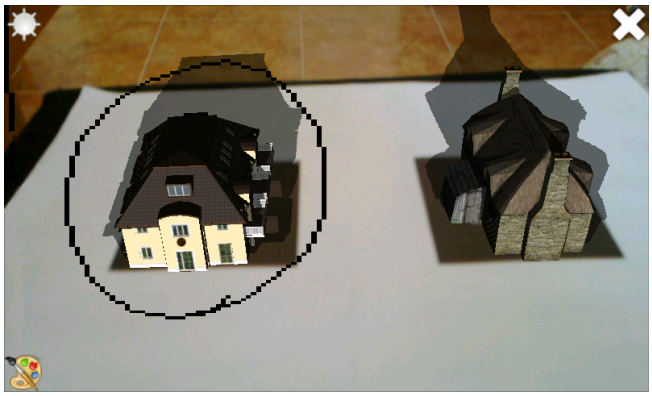

Figura 21: Circulo marcando uno de los edificios.

Una vez realizado, se pulsa de nuevo la paleta y la imagen de la cámara junto a los renders correspondientes aparecen de nuevo. Se muestra también la anotación que se ha añadido en la posición donde se creo, de manera que al mover la cámara también se visualiza desde otra perspectiva.

Estas anotaciones son temporales y se eliminan al terminar la ejecución de la aplicación.

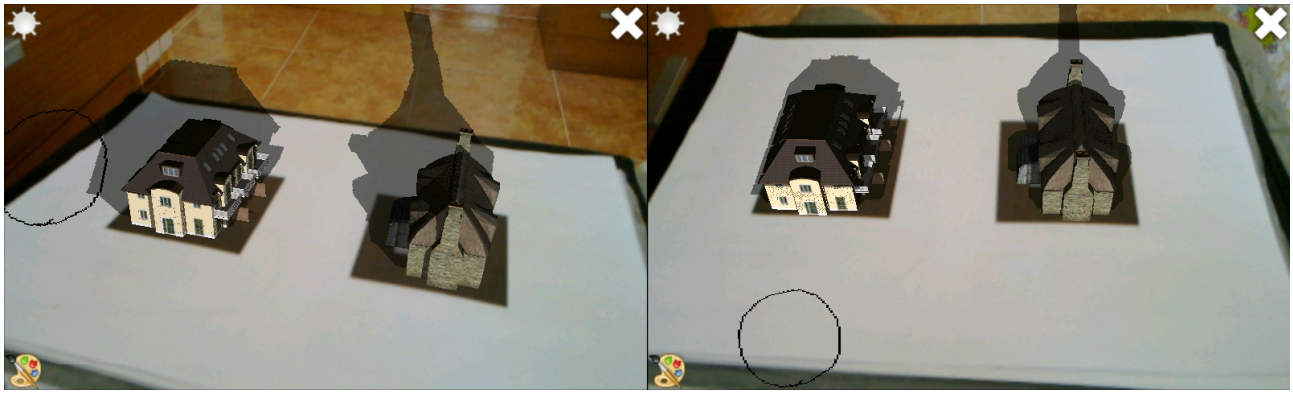

Figura 22: Variación de la anotación según la perspectiva de la cámara.

5.4.- FORMATO WAVEFRONT OBJ

Algunos de los inconvenientes que nos podemos encontrar a la hora de usar la aplicación *araap*, pueden venir a la hora de querer personalizar la representación de los objetos para cambiar los que vienen por defecto por otros. Si son modelos 3D de objetos que ya tenemos, puede que tengamos que adaptarlos a las necesidades que la aplicación nos exige. Mientras que si la intención es integrar objetos propios, éstos deben ser creados siguiendo unas pautas que permitan que el formato con que se definan sea leído por el programa.

Para ello, se define a continuación el formato .OBJ, tanto sus características generales como los requisitos propios necesarios para que sea aceptado por la aplicación.

5.4.1.- HISTORIA

El formato de definición de geometrías en tres dimensiones Wavefront OBJ[40] fue desarrollado en 1989 por la compañía de gráficos por computador “Wavefront Technologies” para su paquete de software gráfico “The Advanced Visualizer”. En sus orígenes, este software era un conjunto de programas independientes que hacían uso del formato OBJ. Entre ellos, había un modelador 3D, un visor, un editor de imágenes de mapa de bits y un programa de renderizado. Al usar todos el mismo archivo de formato, se podía transferir la información geométrica de unos programas a otros.

Fue un paquete precursor en la tecnología de gráficos por computador y fue usado para la elaboración de algunas películas como “Aladdín”, “Stargate: Puerta a las estrellas” o “Estallido”. En palabras del director de efectos especiales de Electronic Arts, Richard Taylor, los programas de Wavefront estaban “*tan bellamente diseñados que incluso una persona sin conocimientos técnicos podría aprenderlos. Wavefront fue la principal razón de que los gráficos por computador dieran un paso adelante*”[41].

En 1995 “Wavefront Technologies” fue comprada por “Silicon Graphics”[42] y fusionada con “Alias Research”, otra empresa del sector de los gráficos 3D, formando así “Alias Systems Corporation”. Esta nueva empresa creo el famoso programa de modelado 3D y animación “Maya”, usado en multitud de efectos visuales, videojuegos, películas animadas y series de televisión. Tal es así, que el 1 de Marzo del 2003, la compañía fue galardonada por la Academia de las Artes y las Ciencias[43] de EEUU con un Oscar por logros científicos y técnicos por su desarrollo del software Maya. Los fundadores de “Wavefront Techonologies”, Bill Kovacs y Roy A. Hall ya habían recibido en 1997 un premio de la academia por el que se les reconocía su trabajo en el desarrollo del visualizador avanzado de Wavefront.

En 2005, “Alias Systems Corporation” fue comprada por el creador de AutoCad y 3DS Max, Autodesk[44]. Por su parte, el formato de archivo OBJ está abierto y ha sido adoptado por muchos otros proveedores de aplicaciones gráficas en 3D. En la actualidad, se puede considerar como un formato universalmente aceptado aunque cada aplicación lo haya adaptado a sus necesidades. En el siguiente punto se va a describir el funcionamiento del formato centrándose en la variante que acepta la aplicación *arapp*.

5.4.2.- DESCRIPCIÓN

La descripción en este formato se realiza en un fichero ASCII con extensión .obj, en el que viene definida la geometría del objeto. Dentro de esta definición podemos distinguir varias secciones que conforman cada uno de los aspectos que son necesarios para la especificación. En primer lugar se define la información referente a los vértices, teniendo en cada línea las características de cada uno de ellos, acorde al siguiente formato:

```
v x y z w
```

Con la 'v' indicamos que se está definiendo un vértice, mientras que la 'x', 'y' y 'z' son número reales con las coordenadas geométricas del vértice en cuestión. La 'w' hace referencia a un peso necesario para la definición de curvas y superficies racionales. Es un campo no obligatorio cuyo valor por defecto es 1.0. Para curvas o superficies paramétricas también se pueden definir vértices siguiente el patrón:

```
vp u v w
```

Los campos 'u' y 'v' son los valores de los puntos de control de la curva o superficie, mientras que 'w' es, al igual que antes, el peso. Este tipo de vértices no los encontramos en los ficheros OBJ de la casa y la granja que vienen con la aplicación, ya que dichos objetos están definidos con geometría poligonal. Hay también que definir los vectores normales que son necesarios para un correcto renderizado. Vienen definidos acorde al formato:

```
vn i j k
```

Los tres parámetros son los números reales que forman la normal. Al igual que los vértices normales, hay que indicar uno por línea.

Respecto a la textura del objeto, hay que definir las coordenadas (generalmente en dos dimensiones) de los vértices dentro del fichero de imagen. De esta manera, al definir las caras cada uno de los vértices que la formen tendrá su correspondiente coordenada en la textura. El patrón para los vértices de la textura es:

```
vt u v w
```

Con 'vt' se indica que se está definiendo una coordenada en la textura, mientras que 'u', 'v' y 'w' son los valores de las coordenadas de dicho vértice. Si la textura es unidimensional solamente será necesario el primer parámetro, mientras que si es 2D será necesario que aparezca tanto el valor de 'u' como el de 'v'. Si la textura con la que trabajamos tiene profundidad, ésta viene indicada en el valor de 'w'. Por defecto vale 0. Si el objeto representado tiene varios fichero de textura, se definen aquí todas las coordenadas de los vértices de todos los archivos. Al definir la cara ya se indica a que archivo de imagen hace referencia el vértice utilizado.

Usando listas de índices de los vértices, de las coordenadas de textura y de las normales podemos definir las caras. Al igual que con el resto de elementos, cada cara viene representada en una línea que comienza, en este caso, con la letra 'f'. Se usa el siguiente formato:

```
f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 ...
```

Observamos que tenemos una serie de elementos separados con barras, que a su vez se separan unos de otros con espacios. Cada uno de estos bloques de elementos entre espacios representa un vértice de la cara, donde (para el primer trinomio) 'v1' es el índice de un vértice definido anteriormente, 'vt1' es el índice de una coordenada de textura y 'vn1' el índice de la normal correspondiente, también ya definida. Estos índices son independientes según el tipo de elemento (vértice, vértice de textura o normal) y van desde 1 hasta tantos elementos del tipo como hayan definidos en el fichero. Después viene un espacio y la información de otro vértice. Y así sucesivamente para tantos vértices como formen la cara.

Según la especificación del formato OBJ no existe ningún límite en cuanto al número de vértices que puede tener una cara. Sin embargo, la variación del formato aceptada por la aplicación *arapp* para representar los objetos solo acepta caras formadas por tres vértices. Si incluimos en la definición caras con cuatro o más vértices solamente tomará los tres primeros, despreciando el resto.

Si queremos incluir en la aplicación un modelo OBJ independiente ya creado o que generemos con algún programa de diseño, uno de los problemas que podemos tener al obtener un fichero mediante este método es que, como se ha indicado, la aplicación *arapp* solamente acepta caras triangulares y en la especificación obtenida es posible que existan caras definidas con más de tres vértices. En el apartado de caras del siguiente punto se presenta una posible solución para arreglar este problema.

Además, es posible definir caras que no contengan información de texturas o de normales simplemente omitiendo el dato en la línea. Las barras separadoras no se quitan y es necesario respetar el orden de la especificación aunque no se hayan completado todos los datos. Por ejemplo;

```
f 1 2 3
f 3/1 4/2 5/3
f 3//2 4//1 5//4
f 6/4/1 3/5/3 7/6/5
```

Observamos aquí cuatro formas distintas de definir una cara de tres vértices. En la primera se hace simplemente indicando los vértices, mientras que en la segunda se añade también información de textura para cada vértice. En la tercera opción se han omitido las coordenadas de textura y se ha añadido la información de la normal (Nótese que las barras separadoras persisten pese a faltar el vértice de la textura). Por último, la cuarta fila nos muestra una cara definida con todas sus propiedades. Esta última opción es la aceptada por la aplicación *arapp* para la representación de la información. Si a las caras les falta información no se podrá representar correctamente el objeto.

Con todo esto tenemos definida la geometría de la figura, pero para indicar algunas opciones sobre la apariencia y las características del material hay que utilizar una serie de etiquetas que hacen referencia a un fichero ASCII externo llamado MTL[45]. Éste contiene la información necesaria sobre el sombreado de las superficies para hacer el renderizado y en él viene también indicado el nombre del fichero de la textura a la que hacen referencia los vértices de coordenadas de textura.

Cada fichero MTL puede contener una o más definiciones de materiales. Para indicar el nombre de este fichero que acompaña al OBJ se utiliza la etiqueta `mtllib` seguida del nombre del archivo MTL. Esta línea viene escrita al comienzo del OBJ. Cada material especificado en el MTL (Material Template Libray) define las propiedades de reflexión de la luz en una superficie según el modelo de reflexión de la luz de Phong.

Antes de ver como se indica el uso de estos materiales en el OBJ, vamos a pasar a describir el contenido de los ficheros MTL. Cada una de las definiciones de los materiales que se definen comienzan por una línea donde pone `newmtl`, un espacio y el nombre que se le quiera dar a ese material. En las siguientes líneas (hasta el siguiente `newmtl` o el fin del fichero) se indican los cantidades propias del material que se está definiendo y que vienen a ser los parámetros del modelo de iluminación de Phong. La primera palabra de la línea indica la característica y a continuación vienen los valores. Para el color ambiente:

```
Ka r g b
```

Donde 'r', 'g' y 'b' son las números reales que representan las componentes de color rojo, verde y azul para esta propiedad del material. El valor por defecto, por si no se indica ninguno, es 0.2,0.2,0.2. Para definir el ratio de reflexión de la componente difusa de la iluminación se usa:

```
Kd r g b
```

El valor por defecto en este caso es 0.8,0.8,0.8. Para la componente especular:

```
Ks r g b
```

Este valor afectará a las zonas donde la intensidad de la luz sea más alta. Su valor predeterminado es 1.0,1.0,1.0. El siguiente parámetro es para indicar cual es la transparencia del material:

```
d alfa
```

El valor de alfa iría de 0.0 a 1.0, siendo 0.0 un material totalmente transparente y 1.0 completamente opaco (valor por defecto). Algunas implementaciones del formato Wavefront OBJ usan *Tr* en lugar de *d* como constante para indicar el valor de transparencia. *Arapp* utiliza la *d*. Para el definir el brillo del material se utiliza:

```
Ns s
```

El valor predeterminado para *s* es 0. Con los parámetros que estamos añadiendo, podemos apreciar que el modelo de iluminación de Phong esta formado por varios niveles de iluminación. Para calcular el color que se tiene que visualizar en un punto se tienen un cuenta la intensidad provocada por la luz ambiente, por la luz difusa y por la especular. Podemos indicar en las propiedades del material si queremos o no que se utilice la componente especular del modelo de iluminación:

```
illum n
```

En el caso de que *n* valga 1 estamos definiendo un material plano sin reflejos especulares. Por tanto, no será necesario haber definido un valor para el parámetro *Ks*. Si por el contrario el valor de *n* es 2, estamos ante el modelo de iluminación completo y si es necesario que exista la línea de *Ks*.

Por último, se pueden utilizar mapas de texturas que se encuentren en ficheros externos para indicar como se debe visualizar el material del siguiente modo:

```
map_Ka lenna.tga
map_Kd lenna.tga
map_Ks lenna.tga
map_d lenna_alpha.tga
map_bump lenna_bump.tga
```

Con cada una de estas líneas se indica el nombre del fichero externo donde se encuentre un mapa de textura para la luz ambiente (*map_Ka*), un mapa de textura para la componente difusa (*map_Kd*), la especular (*map_Ks*), un mapa de textura para la transparencia (*map_d*) o un mapa de textura donde se indiquen relieves (*map_bump*). En la mayoría de los casos en que se usan, el de la luz ambiente y el de la luz difusa suelen coincidir.

En estas líneas es donde indicamos el nombre del archivo de imagen externo que se usa como textura para el objeto. Por ejemplo, en la definición de un material empleado en la casa que viene con arapp, se indica el uso de una imagen PNG como textura de la siguiente forma:

```
map_Kd white_file1.PNG
```

Por tanto, teniendo los distintos materiales definidos en el fichero MTL falta saber como indicar en el OBJ que material se usa. Habiendo previamente incluido el nombre del fichero MTL con *mtllib*, hay que poner en la definición de las caras una línea con el siguiente formato:

```
usemtl nombre
```

En *nombre* aparece el nombre del material definido en el MTL y dicho material se aplicará a todas las caras que vengan definidas a continuación de esta línea y hasta que aparezca otro *usemtl*. Otra etiqueta que se puede utilizar en la definición de las caras es:

```
g nombre del grupo
```

Sirve para ponerle un nombre al grupo de caras de la figura que viene a continuación. Generalmente se usa para agrupar un conjunto de polígonos que pertenecen a la misma parte del objeto o que tienen unas mismas propiedades de visualización.

Tanto en el fichero OBJ como en el MTL, podemos insertar comentarios haciendo que la línea comience con el símbolo #. Todo lo que venga a continuación en esa misma línea sera tomado como una anotación que no afecta a la definición del modelo 3D.

5.4.3.- MODELADO DE UN DADO

Para poder aprender a introducir un nuevo objeto en la aplicación, se expone a continuación la definición e integración de un dado. La creación de este objeto simple puede servir como base para conocer como se ha de realizar la definición de manera correcta de cara a otros objetos más complejos que se puedan crear en el futuro.

Tal y como se puede observar en la siguiente figura el nuevo objeto consiste en un cubo con una textura correctamente ajustada, de manera que quede un número para cada una de las caras. El modelo ha sido totalmente diseñado manualmente escribiendo la definición en los ficheros de texto.

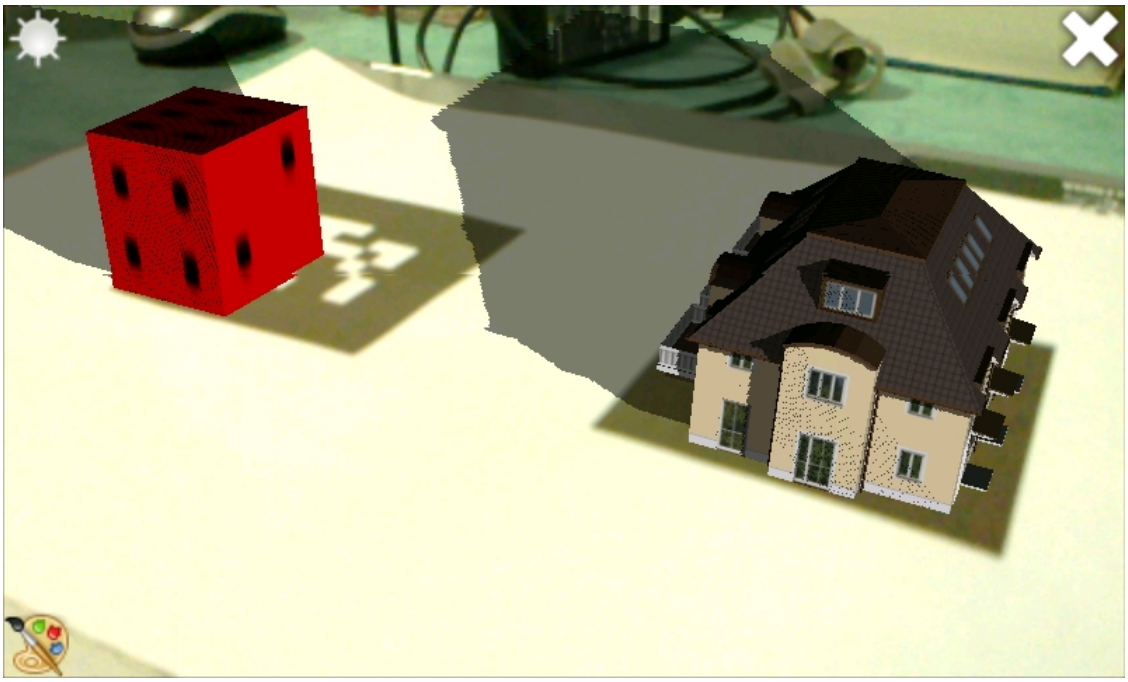

Figura 23: Visualización del dado y la casa

También se puede apreciar como, a diferencia de la casa de la derecha, el dado rojo no aparece centrado en el marcador sino que comienza desde una de las esquinas. Esto se debe a que en la especificación del cubo uno de los vértices de la esquina coincide con el punto (0,0,0) del sistema de coordenadas. Por tanto, si quisiéramos que el objeto se mostrase centrado habría que escribir las coordenadas correspondientes acorde con el tamaño del objeto para que quedara posicionado en el lugar deseado.

Una vez tenemos los ficheros con la definición del dado creados, hay que modificar el código de la función *setupObjects* de la clase *Scene* para sustituir uno de los objetos que vienen por defecto por el propio. Basta cambiar la ruta del que se desee para hacer referencia al archivo OBJ del dado, tal y como queda en el código que se muestra en el apéndice 4.

5.4.3.1.- FORMATO Y DESCRIPCIÓN

Tal y como indica la especificación, hemos trabajado creando dos ficheros ASCII con las extensiones .OBJ y .MTL donde se indican la descripción del objeto y del material, respectivamente. Como el dado es un objeto único y está envuelto todo por la misma textura sólo ha sido necesaria la definición de un material en el fichero MTL.

El contenido de los archivos *cubo.mtl* y *cubo.obj* puede consultarse en los apéndices 5 y 6 del final del documento.

5.4.3.2.- VÉRTICES

El objeto consta de ocho vértices y, dado que se definen independientemente uno del otro mediante sus coordenadas, no es necesario seguir ningún orden concreto para su definición. La siguiente figura muestra las coordenadas de cada uno de ellos acorde a la definición realizada en el fichero OBJ:

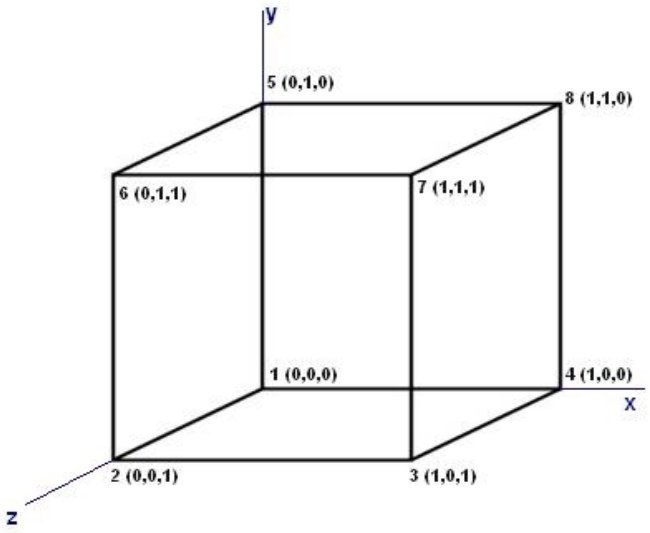

Figura 24: Distribución del los vértices del dado.

Así que basta indicar en el OBJ las coordenadas de los ocho vértices, uno por línea.

5.4.3.3.- NORMALES

Dado que las caras del dado son planas todos los vértices de una misma cara tendrán el mismo vector normal. Por tanto, sólo es necesario definir estas seis normales:
(0,-1,0), (0,1,0), (0,0,1), (1,0,0), (0,0,-1), (-1,0,0).

Estos vectores unitarios se asociarán a cada uno de los vértices de una cara en su definición y nos servirán para indicar la parte exterior de la misma. Este detalle es indispensable para una correcta visualización de la textura del objeto.

5.4.3.4.- TEXTURAS

En el MTL viene indicado el nombre del archivo de textura que se utiliza, mientras que en el OBJ hay que definir los vértices de textura para ese mismo archivo (uno por línea). En el caso del dado se utiliza una imagen donde vienen todo los puntos dibujados para formar las distintas caras. Por la manera de definir los vértices, es recomendable que la imagen que se utilice sea cuadrada.

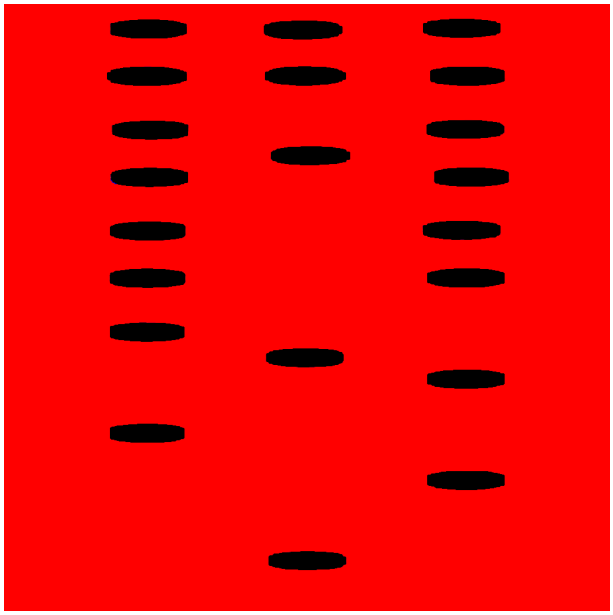

Figura 25: Imagen usada como textura.

Observando la textura se aprecia que va a ser necesario definir 14 vértices, los cuales van a ir ubicados justo en los bordes de la imagen para separar los números unos de otros y luego asociar cada uno de estos puntos a una de las esquinas del cubo que conforma el dado. Las coordenadas varían entre 0 y 1, siendo (0,0) la esquina superior izquierda y (1,1) la esquina inferior derecha.

5.4.3.5.- CARAS

Con la definición de las caras pondremos en común los vértices, las normales y las texturas. Se definen ocho caras, una en cada línea del fichero, indicando sus tres vértices espaciales con sus vértices de textura y normales correspondientes. Las caras deben ser triangulares para poder ser representadas.

En la siguiente figura podemos apreciar tanto la ubicación de los vértices de textura en la imagen como su distribución para formar las caras. Es imprescindible que cada uno de estos vértices se asocie junto al vértice del cubo correspondiente para que la textura encaje con el modelo.

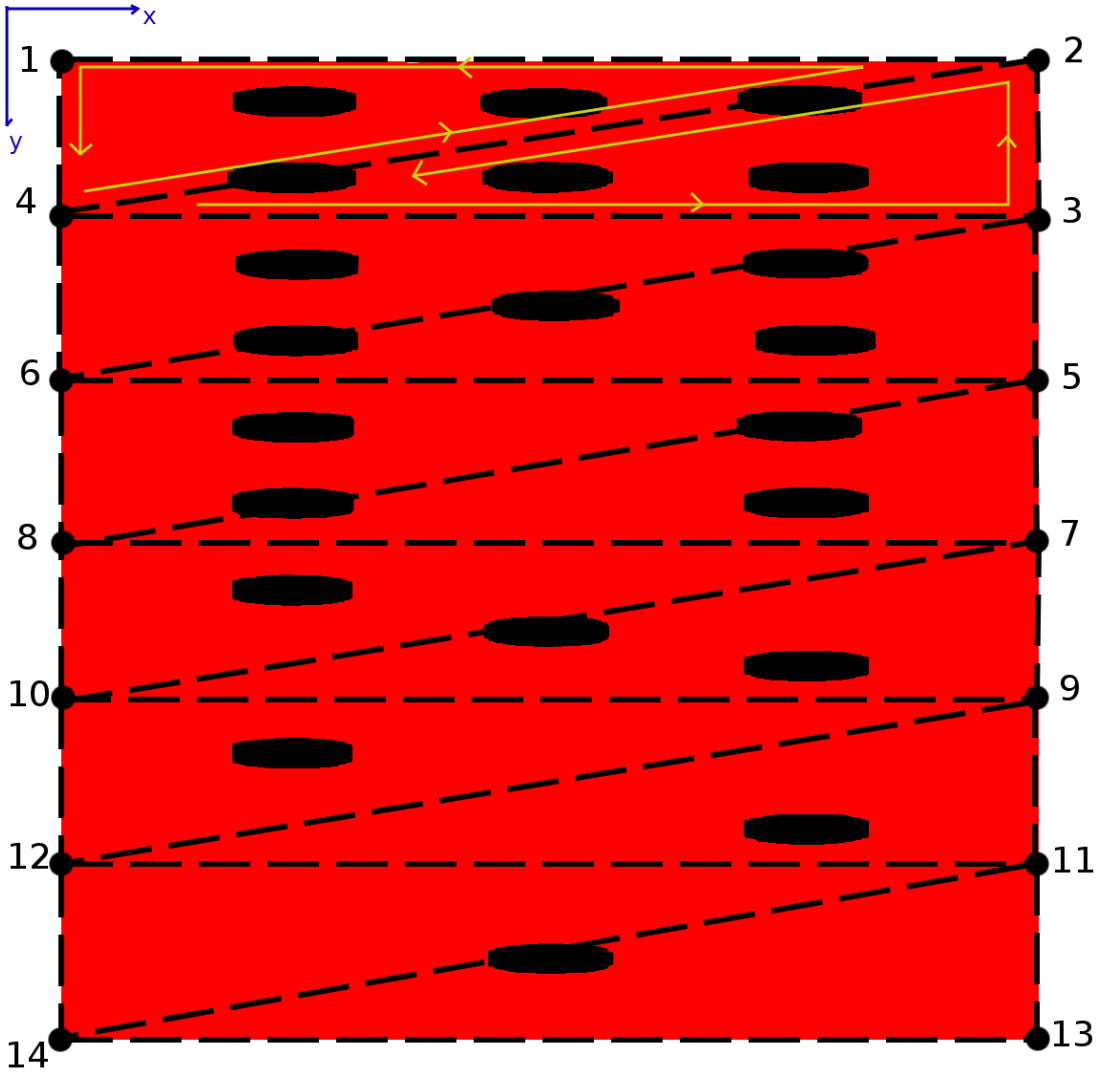

Figura 26: Vértices de textura del dado formando las caras.

La definición de las caras se ha hecho siguiendo el sentido antihorario para el orden de los vértices.

Par el caso de que se disponga de la definición de un modelo OBJ cuyas caras están formadas por más de tres vértices, hemos creado un script en AWK para intentar adaptarlo al formato correcto. AWK es un lenguaje de programación para procesar cadenas de texto. En los sistemas Unix viene integrado un programa (invocado mediante la orden *awk*) para interpretar programas escritos en este código.

La idea es procesar un fichero *.obj*, de manera que cada vez que encuentre una línea de definición de cara la divida en tantas como sean necesarias para que, al final, todas las caras vengan definidas con sólo tres vértices. El código propuesto sirve para transformar caras de cuatro, cinco y seis vértices en caras de tres, pero es fácilmente ampliable para caras de más vértices.

El proceso para hacer la transformación sería escribir en una consola la siguiente orden:

```
cat fichero.obj | ./triangulizar.sh > fichero.obj
```

La orden *cat* pasa mediante una tubería el código del modelo al script, y éste a su vez lo procesa y vuelca la salida en el mismo fichero. Así, lo sobrescribe con los cambios realizados. El contenido del fichero triangulizar.sh se puede consultar en el apéndice 7.

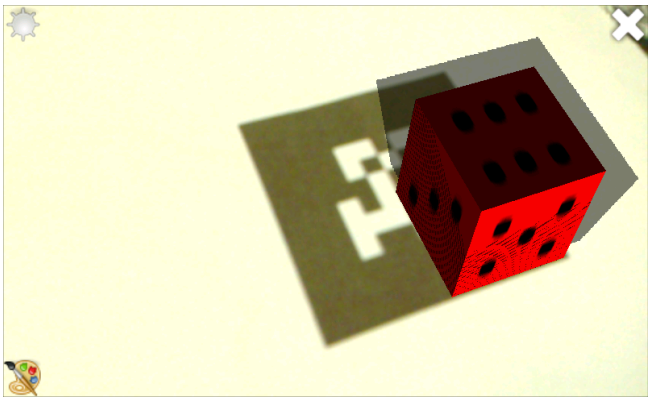

Figura 27: Visualización del dado.

5.4.4 CREANDO UN OBJ CON BLENDER

Algunos programas de diseño son capaces de generar un fichero de salida con la descripción de un modelo 3D en formato Wavefront. Sin embargo, es posible que la definición en los archivos obtenidos no sea compatible con la aplicación con la que estamos trabajando y, por tanto, es posible que se tengan que hacer algunas modificaciones a mano para adaptarla.

Se explica a continuación como generar un fichero OBJ utilizando un objeto 3D creado utilizando el software Blender[46]. Blender es un programa informático multiplataforma dedicado al modelado y animación de gráficos en 3D, que se distribuye gratuitamente bajo licencia GPL. Los ficheros Blender tienen una extensión *.blend* y vamos a explicar los pasos para crear uno[47] y exportarlo como OBJ.

En este ejemplo concreto vamos a utilizar el diseño del planeta Tierra. Para crearlo bastaría con dibujar una malla en forma de esfera y añadirle una textura. Se selecciona *Add*, *Mesh* y después *UV Sphere*. La aplicación *arapp* solamente acepta objetos que tengan texturas porque en caso contrario no tiene ningún material que representar y no se visualiza.

Por tanto, hay que añadir la textura como material de la malla. Para ello, vamos a la pestaña de Materiales de las propiedades del objeto y le damos a crear uno nuevo. Si teníamos seleccionada la esfera, automáticamente se añadirá al objeto. Ahora falta ponerle la textura al material. Vamos a la pestaña de Texturas, seleccionamos nueva textura de tipo “Image”, abrimos el archivo donde tengamos la imagen con la textura que queremos y se añade al objeto.

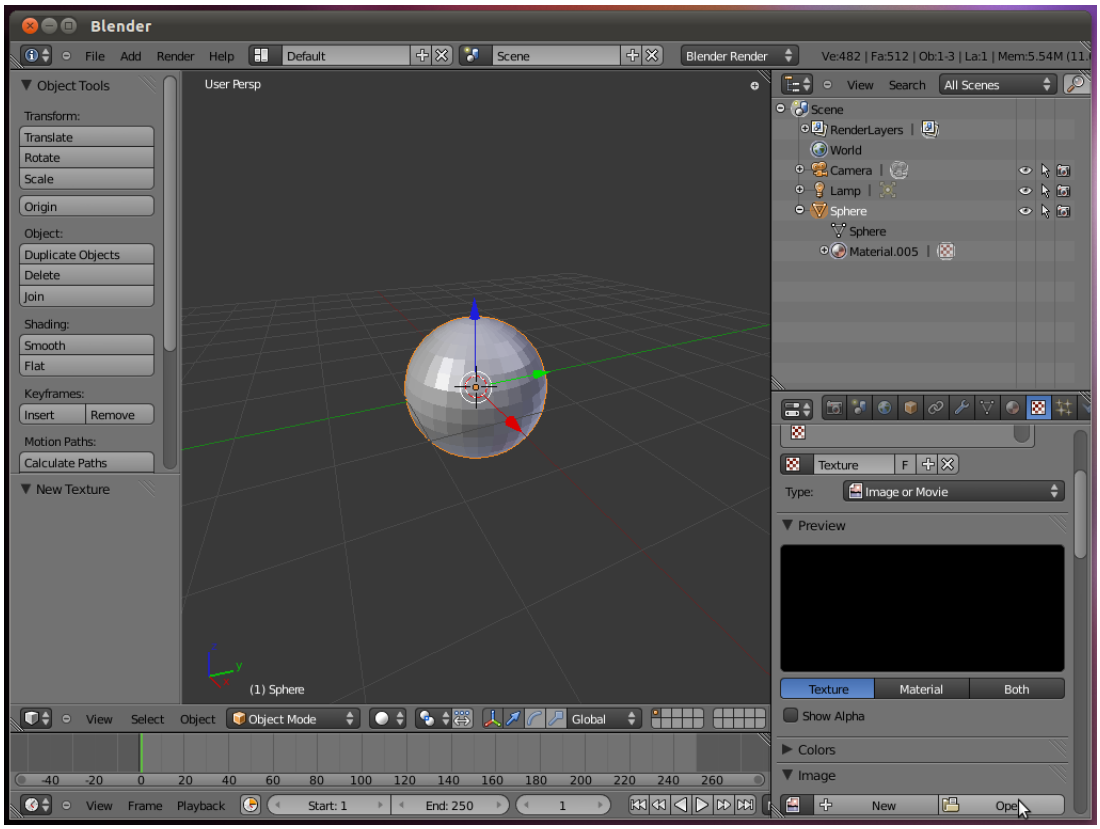

Figura 28: Abriendo el fichero de textura.

Hasta este punto nos sirve cualquier modelo que encontremos para descargar y abrir en Blender. El siguiente paso es trasformar la malla a triángulos para que sea compatible con *arapp* y acoplarle la textura. Cambiamos el tipo de modo para ir al "Edit mode" y allí seleccionamos la opción *Mesh*, luego *Faces* y finalmente hacemos click en *Quads to Tris*.

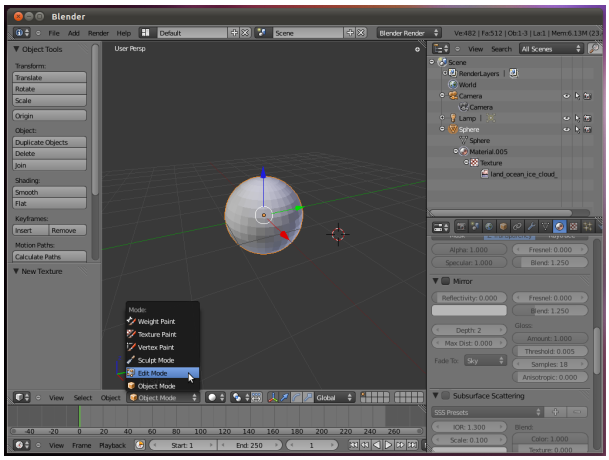

Figura 29: Edit Mode.

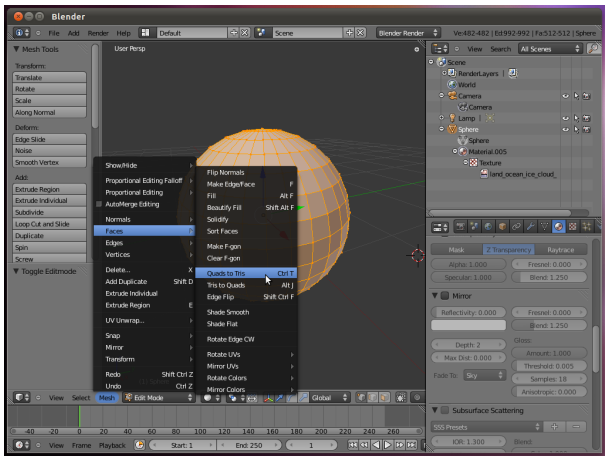

Figura 30: Triangulando la malla.

Podemos apreciar como la malla, que antes estaba formada por cuadriláteros, ahora está formada por triángulos. Recordemos que las caras del fichero OBJ tienen que estar formadas por tres vértices. Una vez triangulada la malla, tenemos que mapear la textura para que se acople a ella. Para ello, en primer lugar hay que seleccionar la opción *Unwrap* del menú Mesh para desenrollar la malla y después cambiamos la vista 3D por *UV/Image Editor*, donde aparecerá la malla estirada.

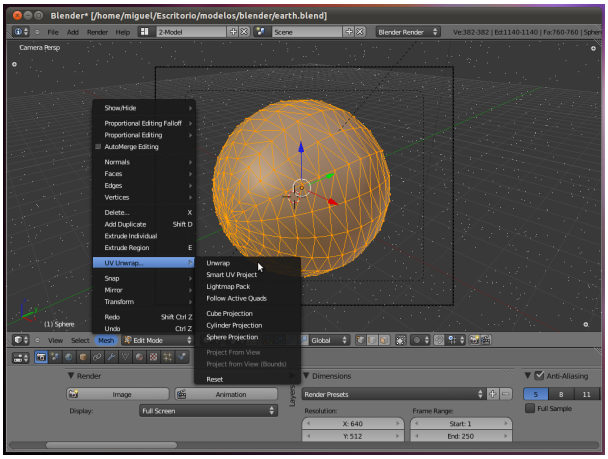

Figura 31: Aplicando Unwrap a la malla.

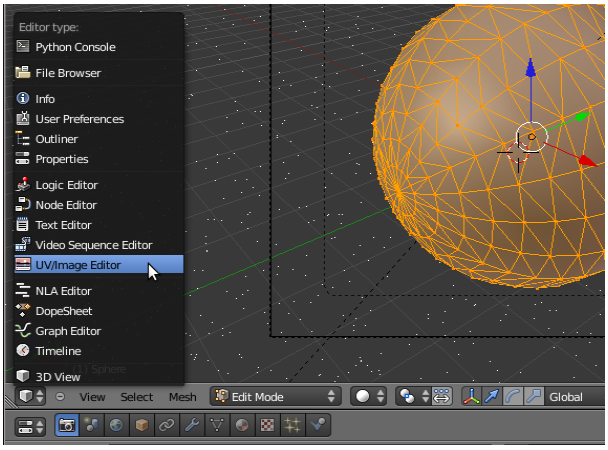

Figura 32: Modo UV/Image Editor.

En este modo tenemos que abrir la imagen de textura (*Image > Open Image*) y realizar las transformaciones necesarias desde el menú *UVs* a la malla para acoplarla de modo que envuelva a la imagen y deje cubiertos todos los polígonos.

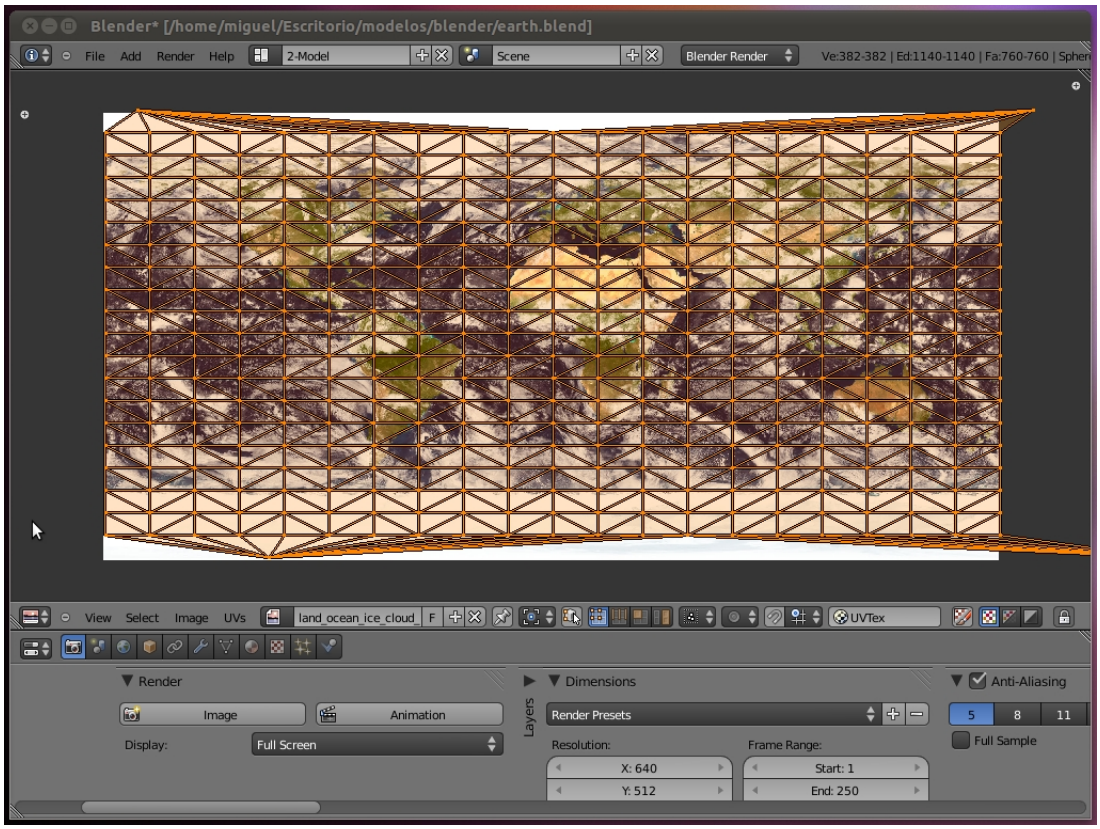

Figura 33: Mapeando la textura.

Con todo esto ya tenemos el modelo compatible y listo para ser exportado en formato OBJ. La exportación se realiza desde el menú *File*, seleccionando *Export* y después *Wavefront (.obj)*.

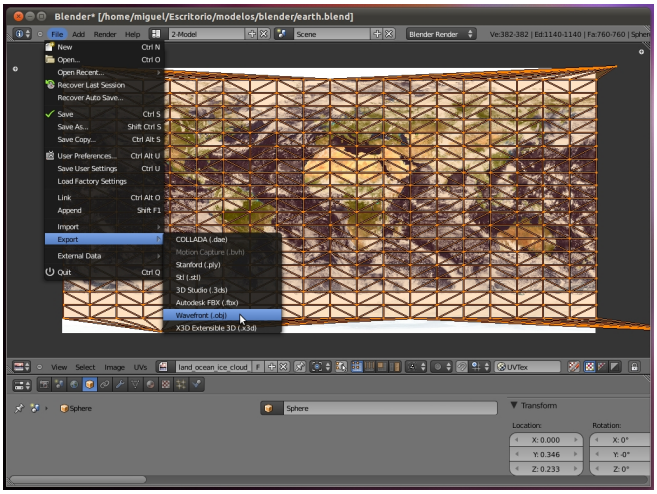

Figura 34: Exportando en formato obj.

En la pantalla de opciones de exportación donde seleccionamos el nombre del archivo y la ruta donde se guardará el fichero es importante marcar las opciones de *Normals* y *High Quality Normals* para que se incluyan las normales de las caras en el fichero y *UVs, Materials* y *Triangulate* para que la información de la textura también vaya correctamente incluida.

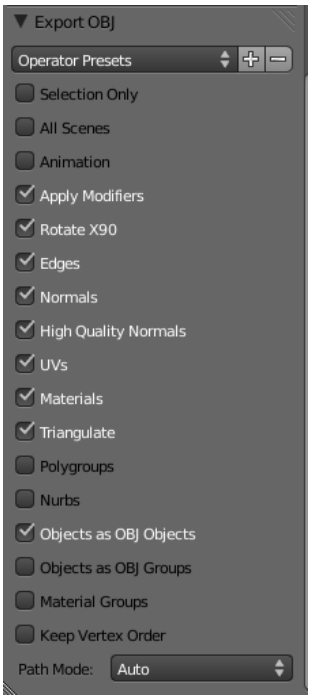

Figura 35: Opciones.

Tras pulsar *Export OBJ* obtendremos los archivos OBJ y MTL con la especificación del modelo. Antes de pasarlos al móvil para incluirlos en la aplicación hay que comprobar un par de detalles. Si estamos trabajando con un modelo descargado de Internet es posible que la imagen o imágenes de la textura vayan empaquetadas en el fichero *.blend*, por lo que tenemos que extraerlas para poder pasarlas también ya que el fichero MTL la buscará.

Para ello, seleccionamos en Blender la opción *External Data* del menú *File* y después *Unpack into Files*. Así obtendremos las imágenes que lleve el modelo.

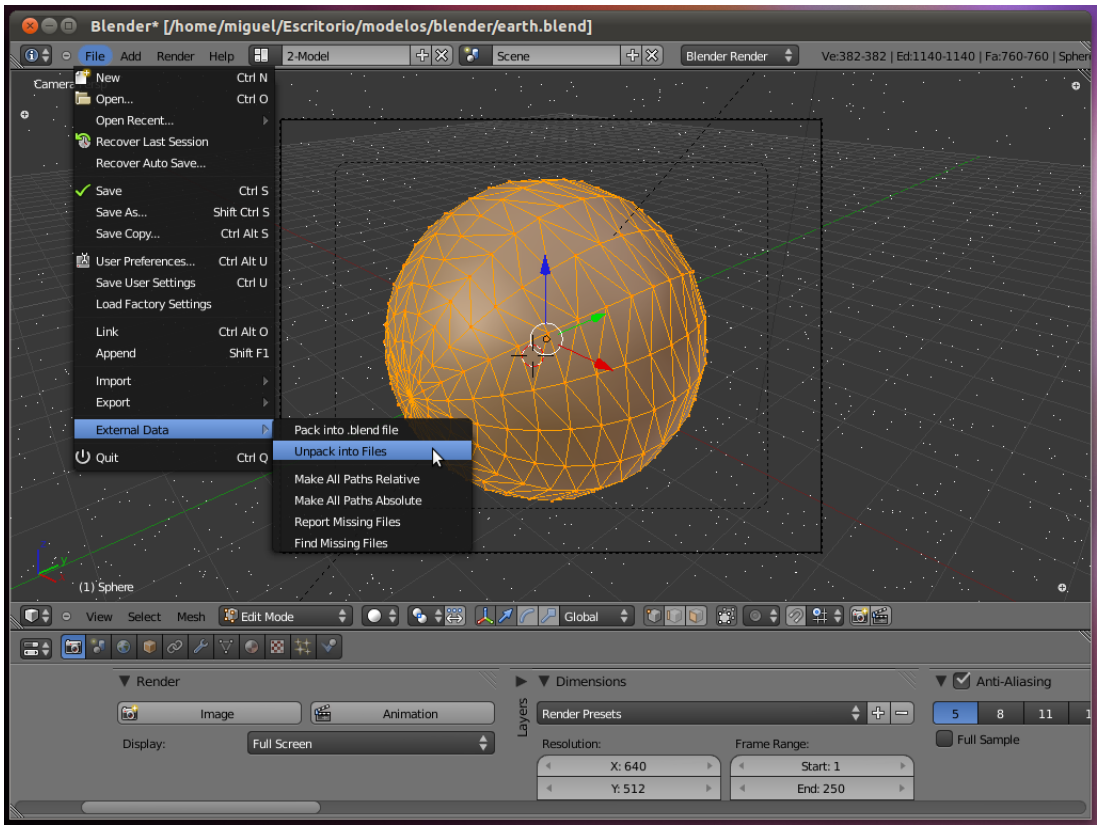

Figura 36: Desempaquetando los archivos.

En segundo lugar, hay que abrir el fichero MTL y modificar la ruta de la imagen ya que normalmente encontraremos una ruta absoluta del archivo dentro del PC donde estamos trabajando. Pero claro, como luego vamos a pasar tanto el OBJ, el MTL y la imagen o imágenes al Nokia hay que modificar la ruta de la opción *map_Kd* para dejarla relativa a la ubicación actual. Es decir, dejar solamente el nombre del archivo de imagen.

Finalmente, pasamos los archivos al N900, modificamos el archivo *Scene.cpp* para incluir la ruta del OBJ, compilamos la aplicación y visualizaremos el nuevo objeto al detectarse el marcador correspondiente.

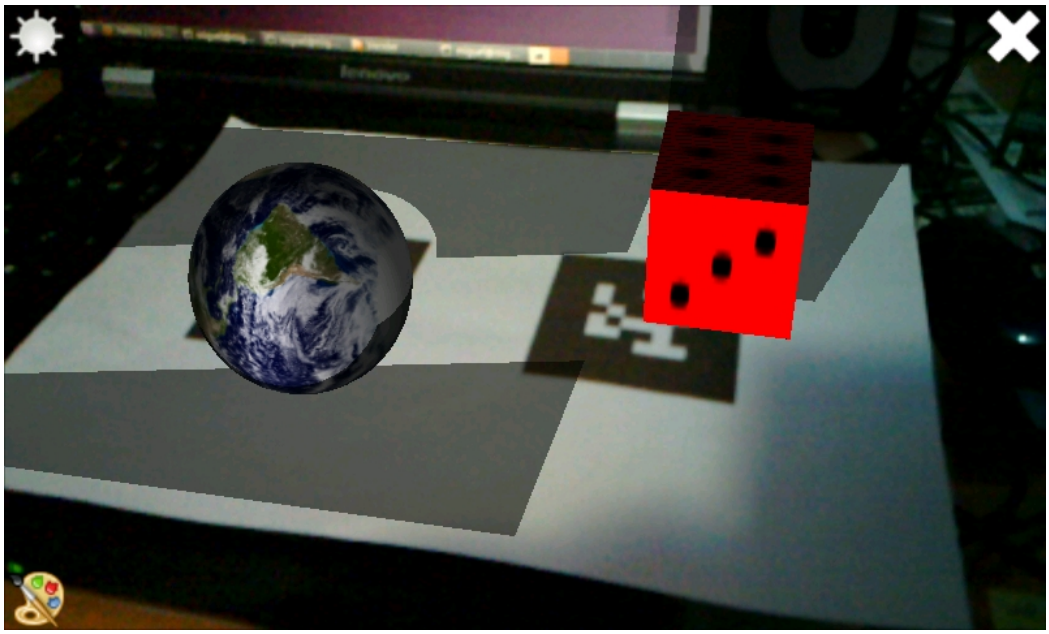

Figura 37: Visualización de la Tierra junto al dado.

Al igual que con los demás, podemos aplicar a los objetos las opciones de interacción ya vistas para modificar su escala o la dirección con la que proyectar la sombra.

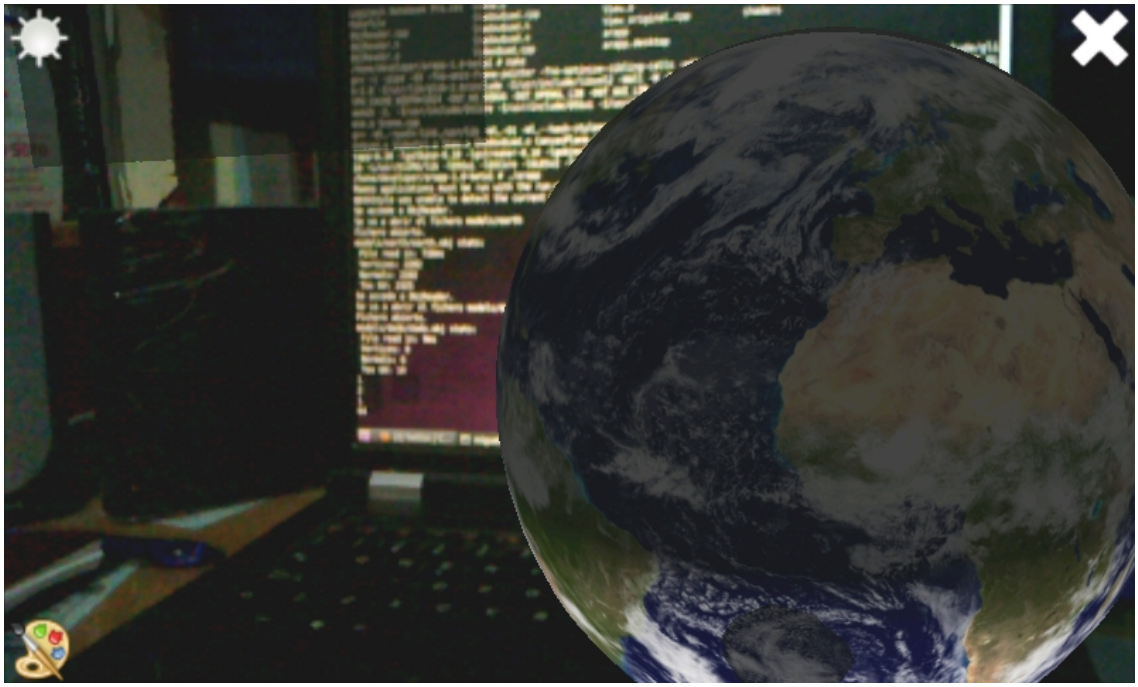

Figura 38: Visualización de la Tierra.

6.- CONCLUSIONES

El desarrollo de este proyecto ha supuesto para el autor un importante aprendizaje no solo en el campo de la realidad aumentada, sino también en el manejo de los sistemas operativos Maemo y la distribución Ubuntu de Linux. Se ha trabajado con ambos en profundidad conociendo sus detalles de funcionamiento y administración.

El sistema operativo Maemo va a ser fusionado con el proyecto Moblin de Intel dando lugar al que será el sustituto natural en los futuros dispositivos móviles de Nokia y que se llamará MeeGo[48]. También es un sistema basado en Linux y que hace uso de las librerías de Qt. Por lo tanto, es de suponer que no se encontrarán grandes problemas de compatibilidad con las herramientas aquí expuestas.

En cuanto a la aplicación *Arapp*, su desarrollo continua a día de hoy y es probable que en el futuro mejore en sus prestaciones y funcionalidades o que sirva de base para otras aplicaciones de realidad aumentada para el N900 o sus sucesores.

7.- REFERENCIAS Y BIBLIOGRAFÍA

[1] “Realidad aumentada: Lo que la realidad esconde”, Blog Consultec, 2009-11-19: http://blog.consultec.es/index.php/2009/11/realidad-aumentada-lo-que-la-realidad-esconde

[2] 2010 Hype Cycle for Emerging Techonologies, Gartner, 2010-09-07: http://blogs.gartner.com/hypecyclebook/2010/09/07/2010-emerging-technologies-hype-cycle-is-here

[3] Layar Reality Browser: http://www.layar.com/browser/

[4] Nearest Tube: http://www.acrossair.com/apps_nearesttube.htm

[5] TwittARound: http://i.document.m05.de/?page_id=700

[6] LevelHead: http://selectparks.net/~julian/levelhead/

[7] Proyecto ARQuake: http://wearables.unisa.edu.au/projects/arquake/

[8] Nokia N900: http://www.nokia.es/productos/moviles/nokia-n900-maemo-pantalla-tactil-teclado-qwerty

[9] Página oficial de Maemo: http://www.maemo.org

[10] Red Pill Mode: http://wiki.maemo.org/Red_Pill_mode

[11] Repositorios de Maemo: http://repository.maemo.org/

[12] Catálogo Maemo Extras: http://repository.maemo.org/extras/install/extras.install

[13] Catálogo Extras-testing: http://my-maemo.com/download/repos/extras-testing.install

[14] Catálogo Extra-devel: http://repository.maemo.org/extras/install/extras-devel-fremantle.install

[15] Actualizar el firmware del N900: http://wiki.maemo.org/Updating_the_firmware

[16] Open SSH: http://www.openssh.org/

[17] Info-Zip: http://www.info-zip.org/

[18] Qualcomm AR SDK: https://developer.qualcomm.com/develop/mobile-technologies/augmented-reality

[19] Atomic Authoring Tool: http://www.sologicolibre.org/projects/atomic/es/

[20] Atomic Web Authoring Tool: http://www.sologicolibre.org/projects/atomicweb/es/

[21] Lenguaje de programación Processing: http://processing.org/

[22] FLARToolKit: http://www.libspark.org/wiki/saqoosha/FLARToolKit/en

[23] Look!: http://www.lookar.net

[24] SSTT - Augmented Reality Tracking Library: http://technotecture.com/projects/sstt

[25] Osgart: http://www.artoolworks.com/community/osgart/

[26] OpenSceneGraph website: http://www.openscenegraph.org/projects/osg

[27] ArToolKit: http://www.hitl.washington.edu/artoolkit/

[28] HITLab de la Universidad de Washington: http://www.hitl.washington.edu/

[29] ARToolworks, Inc: http://www.artoolworks.com/

[30] ARToolKitPlus para Maemo:
http://maemo.org/packages/package_instance/view/fremantle_extras-devel_free_armel/artoolkitplus-dev/2.2.0-0maemo1/

[31] OpenGL-ES: http://www.khronos.org/opengles/

[32] Glut-ES: http://glutes.sourceforge.net/

[33] EGL: http://www.khronos.org/egl/

[34] Web de la aplicación Arapp: https://launchpad.net/arinterface

[35] ARToolKitPlus: http://handheldar.icg.tugraz.at/artoolkitplus.php

[36] Qt: http://qt.nokia.com

[37] Scratchbox: http://www.scratchbox.org/

[38] Arapp en Paquetes de Maemo:
http://maemo.org/packages/source/view/fremantle_extras-devel_free_source/arapp/1.0~beta1-0maemo5/

[39] Pavel Rojtberg: http://www.rojtberg.net/

[40] Especificación del formato OBJ: http://www.martinreddy.net/gfx/3d/OBJ.spec

[41] Ellen Wolff 2006-06-01. "Remembering CG Pioneer Bill Kovacs".

[42] Silicon Graphics International: http://www.sgi.com

[43] The Academy Of Motion Pictures Arts And Sciences: http://www.oscars.org/

[44] Autodesk: http://www.autodesk.com/

[45] People.sc.fsu.edu 2004-06-14. “MTL Files - Material Definitions for OBJ Files":

http://people.sc.fsu.edu/~jburkardt/data/mtl/mtl.html

[46] Blender: http://www.blender.org

[47] Repositorio de modelos 3D para Blender:
http://e2-productions.com/repository/modules/PDdownloads/

[48] MeeGo: https://meego.com/

[49] El blog de Parq: Compilando ARToolKit en Ubuntu.
http://elblogdeparq.blogspot.com/2010/11/artollkit-271-ubuntu-1004-funciona.html

[50] Lee G., Nelles C., Billinghurst M. & Kim G. J. 2004. "Immersive Authoring of Tangible Augmented Reality Applications" Proceedings of the International Symposium on Mixed and Augmented Reality 2004:
http://home.postech.ac.kr/~endovert/ismar04-ppt.pdf

[51] Tatzgern M., Kalkofen D., Schamalstieg D., 2010. “Multi-perspective compact explosion diagrams”, Computers & Graphics; extended papers from NPAR 2010.

[52] Wagner D., Schmalstieg D., 2007. “ARToolKitPlus for Pose Tracking on Mobile Devices”, Graz University of Technology.

8.- APÉNDICES

1.- Variante de *TrackerSingleMarker.h* para arreglar el problema de las clases virtuales:

```
#ifndef __TRACKERSINGLEMARKER_HEADERFILE__
#define __TRACKERSINGLEMARKER_HEADERFILE__
//#include <ARToolKitPlus/TrackerSingleMarkerImpl.h>
#include <ARToolKitPlus/Tracker.h>
#include <ARToolKitPlus/Logger.h>
#include <vector>
namespace ARToolKitPlus
{

class TrackerSingleMarker : public Tracker
{
public:
    void cleanup();
    bool setPixelFormat(PIXEL_FORMAT nFormat)  ;
    bool loadCameraFile(const char* nCamParamFile, ARFloat nNearClip, ARFloat
nFarClip)  ;
    void setLoadUndistLUT(bool nSet)  ;
    void setLogger(ARToolKitPlus::Logger* nLogger)  ;
    int arDetectMarker(ARUint8 *dataPtr, int thresh, ARMarkerInfo
**marker_info, int *marker_num) ;
    int arDetectMarkerLite(ARUint8 *dataPtr, int thresh, ARMarkerInfo
**marker_info, int *marker_num);
    ARFloat arMultiGetTransMat(ARMarkerInfo *marker_info, int marker_num,
ARMultiMarkerInfoT *config);
    ARFloat arGetTransMat(ARMarkerInfo *marker_info, ARFloat center[2], ARFloat
width, ARFloat conv[3][4]);
    ARFloat arGetTransMatCont(ARMarkerInfo *marker_info, ARFloat prev_conv[3]
[4], ARFloat center[2], ARFloat width, ARFloat conv[3][4]) ;
    ARFloat rppMultiGetTransMat(ARMarkerInfo *marker_info, int marker_num,
ARMultiMarkerInfoT *config);
    ARFloat rppGetTransMat(ARMarkerInfo *marker_info, ARFloat center[2],
ARFloat width, ARFloat conv[3][4]);
    int arLoadPatt(char *filename) ;
    int arFreePatt(int patno)  ;
    int arMultiFreeConfig(ARMultiMarkerInfoT *config);
    ARMultiMarkerInfoT *arMultiReadConfigFile(const char *filename);
    void activateBinaryMarker(int nThreshold);
    void setMarkerMode(MARKER_MODE nMarkerMode);
    void activateVignettingCompensation(bool nEnable, int nCorners=0, int
nLeftRight=0, int nTopBottom=0);
    void changeCameraSize(int nWidth, int nHeight) ;
    void setUndistortionMode(UNDIST_MODE nMode)  ;
    bool setPoseEstimator(POSE_ESTIMATOR nMethod);
    void setBorderWidth(ARFloat nFraction) ;
    void setThreshold(int nValue) ;
```

```
    int getThreshold() const;
    void activateAutoThreshold(bool nEnable) ;
    bool isAutoThresholdActivated() const ;
    void setNumAutoThresholdRetries(int nNumRetries) ;
    const ARFloat* getModelViewMatrix() const  ;
    const ARFloat* getProjectionMatrix() const  ;
    const char* getDescription() ;
    PIXEL_FORMAT getPixelFormat() const ;
    int getBitsPerPixel() const  ;
    int getNumLoadablePatterns() const  ;
    void setImageProcessingMode(IMAGE_PROC_MODE nMode) ;
    Profiler& getProfiler() ;
    Camera* getCamera()  ;
    void setCamera(Camera* nCamera) ;
    void setCamera(Camera* nCamera, ARFloat nNearClip, ARFloat nFarClip) ;
    ARFloat calcOpenGLMatrixFromMarker(ARMarkerInfo* nMarkerInfo, ARFloat
nPatternCenter[2], ARFloat nPatternSize, ARFloat *nOpenGLMatrix) ;
    ARFloat executeSingleMarkerPoseEstimator(ARMarkerInfo *marker_info, ARFloat
center[2], ARFloat width, ARFloat conv[3][4]) ;
    ARFloat executeMultiMarkerPoseEstimator(ARMarkerInfo *marker_info, int
marker_num, ARMultiMarkerInfoT *config);
    void selectDetectedMarker(const int id);

    //Constructor añadido porque no existía aquí, solo en el Impl.h
    TrackerSingleMarker(int nWidth=DEF_CAMWIDTH, int nHeight=DEF_CAMHEIGHT);
    virtual ~TrackerSingleMarker()
        {}
        /*  nCamParamFile is the name of the camera parameter file
         *  nLogger is an instance which implements the ARToolKit::Logger
interface*/
     bool init(const char* nCamParamFile, ARFloat nNearClip, ARFloat nFarClip,
ARToolKitPlus::Logger* nLogger=NULL) ;
     int addPattern(const char* nFileName);
        /// calculates the transformation matrix
        /*  pass the image as RGBX (32-bits) in 320x240 pixels.
         *  if nPattern is not -1 then only this pattern is accepted
         *  otherwise any found pattern will be used.    */
     std::vector<int> calc(const unsigned char* nImage, int nPattern=-1, bool
nUpdateMatrix=true,ARMarkerInfo** nMarker_info=NULL, int* nNumMarkers=NULL);
        /// Sets the width and height of the patterns.
     void setPatternWidth(ARFloat nWidth);
        /// Provides access to ARToolKit' patt_trans matrix
     void getARMatrix(ARFloat nMatrix[3][4]) const;
        /// Returns the confidence value of the currently best detected marker.
        ARFloat getConfidence() const;
};
};      // namespace ARToolKitPlus
//#include <ARToolKitPlus/TrackerSingleMarkerImpl.h>
#endif //__TRACKERSINGLEMARKER_HEADERFILE__
```

2.- Modificaciones realizada en el fichero Makefile de *Arapp*:

```
CC            = gcc
CXX           = g++
DEFINES       = -DQT_GL_NO_SCISSOR_TEST
-DQT_DEFAULT_TEXTURE_GLYPH_CACHE_WIDTH=1024 -DQT_NO_DEBUG -DQT_OPENGL_LIB
-DQT_GUI_LIB -DQT_CORE_LIB -DQT_SHARED
CFLAGS        = -pipe -O3 -fno-omit-frame-pointer -fno-optimize-sibling-calls
-pthread -I/usr/include/gstreamer-0.10 -I/usr/include/glib-2.0 -I/usr/lib/glib-
2.0/include -I/usr/include/libxml2 -Wall -W -D_REENTRANT $(DEFINES)
CXXFLAGS      = -pipe -O3 -fno-omit-frame-pointer -fno-optimize-sibling-calls
-pthread -I/usr/include/gstreamer-0.10 -I/usr/include/glib-2.0 -I/usr/lib/glib-
2.0/include -I/usr/include/libxml2 -Wall -W -D_REENTRANT $(DEFINES)
INCPATH       = -I/usr/share/qt4/mkspecs/linux-g++-maemo5 -I.
-I/usr/include/QtCore -I/usr/include/QtGui -I/usr/include/QtOpenGL
-I/usr/include -I. -I/usr/X11R6/include -I.
```

[...]

```
DIST          = /usr/share/qt4/mkspecs/common/unix.conf \
            /usr/share/qt4/mkspecs/common/linux.conf \
            /usr/share/qt4/mkspecs/qconfig.pri \
            /usr/share/qt4/mkspecs/features/qt_functions.prf \
            /usr/share/qt4/mkspecs/features/qt_config.prf \
            /usr/share/qt4/mkspecs/features/exclusive_builds.prf \
            /usr/share/qt4/mkspecs/features/default_pre.prf \
            /usr/share/qt4/mkspecs/features/release.prf \
            /usr/share/qt4/mkspecs/features/default_post.prf \
            /usr/share/qt4/mkspecs/features/link_pkgconfig.prf \
            /usr/share/qt4/mkspecs/features/warn_on.prf \
            /usr/share/qt4/mkspecs/features/qt.prf \
            /usr/share/qt4/mkspecs/features/unix/opengl.prf \
            /usr/share/qt4/mkspecs/features/unix/thread.prf \
            /usr/share/qt4/mkspecs/features/moc.prf \
            /usr/share/qt4/mkspecs/features/resources.prf \
            /usr/share/qt4/mkspecs/features/uic.prf \
            /usr/share/qt4/mkspecs/features/yacc.prf \
            /usr/share/qt4/mkspecs/features/lex.prf \
            /usr/share/qt4/mkspecs/features/include_source_dir.prf \
            arapp.pro
QMAKE_TARGET  = arapp
DESTDIR       =
TARGET        = arapp

first: all
####### Implicit rules
.SUFFIXES: .o .c .cpp .cc .cxx .C
.cpp.o:
      $(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"
.cc.o:
```

```
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"
.cxx.o:
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"
.C.o:
	$(CXX) -c $(CXXFLAGS) $(INCPATH) -o "$@" "$<"
.c.o:
	$(CC) -c $(CFLAGS) $(INCPATH) -o "$@" "$<"

####### Build rules
all: Makefile $(TARGET)
$(TARGET):  $(OBJECTS)
	$(LINK) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(OBJCOMP) $(LIBS)

Makefile: arapp.pro  /usr/share/qt4/mkspecs/linux-g++-maemo5/qmake.conf
/usr/share/qt4/mkspecs/common/unix.conf \
		/usr/share/qt4/mkspecs/common/linux.conf \
		/usr/share/qt4/mkspecs/qconfig.pri \
		/usr/share/qt4/mkspecs/features/qt_functions.prf \
		/usr/share/qt4/mkspecs/features/qt_config.prf \
		/usr/share/qt4/mkspecs/features/exclusive_builds.prf \
		/usr/share/qt4/mkspecs/features/default_pre.prf \
		/usr/share/qt4/mkspecs/features/release.prf \
		/usr/share/qt4/mkspecs/features/default_post.prf \
		/usr/share/qt4/mkspecs/features/link_pkgconfig.prf \
		/usr/share/qt4/mkspecs/features/warn_on.prf \
		/usr/share/qt4/mkspecs/features/qt.prf \
		/usr/share/qt4/mkspecs/features/unix/opengl.prf \
		/usr/share/qt4/mkspecs/features/unix/thread.prf \
		/usr/share/qt4/mkspecs/features/moc.prf \
		/usr/share/qt4/mkspecs/features/resources.prf \
		/usr/share/qt4/mkspecs/features/uic.prf \
		/usr/share/qt4/mkspecs/features/yacc.prf \
		/usr/share/qt4/mkspecs/features/lex.prf \
		/usr/share/qt4/mkspecs/features/include_source_dir.prf \
		/usr/lib/libQtOpenGL.prl \
		/usr/lib/libQtGui.prl \
		/usr/lib/libQtCore.prl
	$(QMAKE) -unix -o Makefile arapp.pro
/usr/share/qt4/mkspecs/common/unix.conf:
/usr/share/qt4/mkspecs/common/linux.conf:
/usr/share/qt4/mkspecs/qconfig.pri:
/usr/share/qt4/mkspecs/features/qt_functions.prf:
/usr/share/qt4/mkspecs/features/qt_config.prf:
/usr/share/qt4/mkspecs/features/exclusive_builds.prf:
/usr/share/qt4/mkspecs/features/default_pre.prf:
/usr/share/qt4/mkspecs/features/release.prf:
/usr/share/qt4/mkspecs/features/default_post.prf:
/usr/share/qt4/mkspecs/features/link_pkgconfig.prf:
/usr/share/qt4/mkspecs/features/warn_on.prf:
/usr/share/qt4/mkspecs/features/qt.prf:
```

```
/usr/share/qt4/mkspecs/features/unix/opengl.prf:
/usr/share/qt4/mkspecs/features/unix/thread.prf:
/usr/share/qt4/mkspecs/features/moc.prf:
/usr/share/qt4/mkspecs/features/resources.prf:
/usr/share/qt4/mkspecs/features/uic.prf:
/usr/share/qt4/mkspecs/features/yacc.prf:
/usr/share/qt4/mkspecs/features/lex.prf:
/usr/share/qt4/mkspecs/features/include_source_dir.prf:
/usr/lib/libQtOpenGL.prl:
/usr/lib/libQtGui.prl:
/usr/lib/libQtCore.prl:
```

[...]

3.- Fragmento modificado de *View.cpp* para la interacción por pulsación de teclas:

```
void View::keyPressEvent(QKeyEvent *e) {
    if (e->key() == Qt::Key_Left) {
        s.angle += 10;
    } else if (e->key() == Qt::Key_Right) {
        s.angle -= 10;
    } else if (e->key() == Qt::Key_A) {
        s.activeOb->scale += 0.1;
    } else if (e->key() == Qt::Key_S) {
        s.activeOb->scale -= 0.1;
    } else if (e->key() == Qt::Key_C) {
        s.bg = s.bg == QColor(0,0,0,0) ? Qcolor(255,255,255,128) :
QColor(0,0,0,0);
    }
    e->accept();
}
```

4.- Contenido de la función *setupObjects* en *Scene.cpp*:

```
void Scene::setupObjects() {
    objects[0].setup(ObjReader("models/casa/casa.obj")); // 7 fps
    objects[0].id = 0;

    //objects[1].setup(ObjReader("models/farm/farm.obj")); // 10fps
    objects[1].setup(ObjReader("models/dado/dado.obj"));
    objects[1].id = 1;

    activeOb = &objects[0];
    vq.setup();
    sq.setup();
    intf.setup();
}
```

5.- Fichero *cubo.obj*:

```
mtllib cubo.mtl
o Cube
v 0.000000 0.000000 0.000000
v 0.000000 0.000000 1.000000
v 1.000000 0.000000 1.000000
v 1.000000 0.000000 0.000000
v 0.000000 1.000000 0.000000
v 0.000000 1.000000 1.000000
v 1.000000 1.000000 1.000000
v 1.000000 1.000000 0.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 1.000000 0.166666
vt 0.000000 0.166666
vt 1.000000 0.333333
vt 0.000000 0.333333
vt 1.000000 0.500000
vt 0.000000 0.500000
vt 1.000000 0.666666
vt 0.000000 0.666666
vt 1.000000 0.833333
vt 0.000000 0.833333
vt 1.000000 1.000000
vt 0.000000 1.000000
vn 0.000000 -1.000000 0.000000
vn 0.000000 1.000000 0.000000
vn 0.000000 0.000000 1.000000
vn 1.000000 0.000000 0.000000
vn 0.000000 0.000000 -1.000000
vn -1.000000 0.000000 0.000000
g Cube_Cube_Material
usemtl Material
s off
f 4/4/1 3/3/1 2/2/1
f 4/4/1 2/2/1 1/1/1
f 6/14/2 7/13/2 8/11/2
f 6/14/2 8/11/2 5/12/2
f 2/12/3 3/11/3 7/9/3
f 2/12/3 7/9/3 6/10/3
f 3/10/4 4/9/4 8/7/4
f 3/10/4 8/7/4 7/8/4
f 5/6/5 8/5/5 4/3/5
f 5/6/5 4/3/5 1/4/5
f 1/8/6 2/7/6 6/5/6
f 1/8/6 6/5/6 5/6/6
```

6.- Fichero *cubo.mtl*:

```
newmtl Material
Ns 96.078431
Ka 1.000000 1.000000 1.000000
Kd 1.000000 1.000000 1.000000
Ks 0.500000 0.500000 0.500000
Ni 1.000000
d 1.000000
illum 2
map_Kd cubo.png
```

7.- Código awk para la triangulación de caras:

```
awk 'NF==5 && $1=="f" {print $1" "$2" "$3" "$4"\n"$1" "$2" "$4" "$5} $1!="f"
{print $0}
     NF==6 && $1=="f" {print $1" "$2" "$3" "$4"\n"$1" "$2" "$4" "$5"\n"$1" "$2"
"$5" "$6}
     NF==7 $$ $1=="f" {print $1" "$2" "$3" "$4"\n"$1" "$2" "$4" "$5"\n"$1" "$2"
"$5" "$6"\n"$1" "$2" "$6" "$7}'
```