universidad del país vasco

euskal herriko unibertsitatea

Facultad de Informática **Informatika Fakultatea**

TITULACIÓN:
Ingeniería Informática

Desarrollo en Android para aplicaciones de realidad aumentada

Alumno: Francisco Recio Lecuona

Director: Alejandro García-Alonso Montoya

Proyecto Fin de Carrera, 9 de julio de 2012

Agradecimientos

Lo primero de todo quiero agradecer a mis padres, Julián Teodoro y María Elena, y a mi hermana Malen por el gran apoyo que me han dado a lo largo de la carrera y sin el cual no habría podido terminar. También porque cada uno a su modo me ha servido de ejemplo en la vida.

También quiero agradecer a Diego Borro la oportunidad de realizar este Proyecto Fin de Carrera en el CEIT y a Alejandro García-Alonso por los consejos y orientación con esta memoria.

Por último no me quiero olvidar de todos aquellos amigos y compañeros que han estado ahí ofreciendo su ayuda, ya fuera técnica o sobre todo anímica, en esos momentos de bloqueo.

RESUMEN

El presente proyecto tiene como finalidad realizar aplicaciones de realidad aumentada en Android. Como al desarrollar dichas aplicaciones se han reutilizado librerías existentes implementadas en C++, también se ha realizado un estudio de las alternativas para poder incluirlas en una aplicación Android.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

En este documento se describe el Proyecto Fin de Carrera realizado en el Centro de Estudios e Investigaciones Técnicas de Guipúzcoa (CEIT) bajo la supervisión de Diego Borro y dirigido por el profesor de la UPV/EHU Alejandro García-Alonso. El proyecto consiste en la realización de aplicaciones de Realidad Aumentada (AR por Augmented Reality) en dispositivos móviles Android.

La memoria tiene la estructura que se detalla a continuación:

El segundo capítulo muestra el Documento de Objetivos del Proyecto en el que se presenta la planificación de las fases del proyecto. También se presentan los aspectos relacionados con la gestión de riesgos.

En el tercer capítulo se analizan las opciones que se han analizado para reutilizar el código C++ existente en la empresa dentro de una aplicación de Android.

En el cuarto capítulo se presenta una aplicación que analiza la viabilidad de la opción elegida en el capítulo 3.

En el quinto capítulo se diseña e implementa una aplicación para Realidad Aumentada con marcadores.

En el sexto capítulo se presentan las conclusiones obtenidas con la realización de este proyecto, además de futuras líneas de trabajo

El séptimo capítulo se presenta un listado de la bibliografía que se ha usado durante la realización del proyecto y la memoria.

2. Documento de Objetivos del Proyecto

Seguidamente se expone la motivación para la realización del proyecto, así como los objetivos del mismo para fijar su alcance.

Además se incluyen todas las cuestiones relacionadas con la organización del proyecto y se analizan los riegos que podrían afectar a la realización de este PFC.

2.1. Motivación

En la actualidad el mercado de los dispositivos móviles ha experimentado un gran crecimiento que continúa en expansión. Algunas de las causas de esto son el abaratamiento de los precios de los dispositivos, un mayor número de propiedades y que cada día surgen multitud de aplicaciones para estos dispositivos. Todo esto ha provocado un nuevo mundo de oportunidades para las empresas que desarrollan software.

En nuestro caso se desarrollan aplicaciones de realidad aumentada y para ello se intentará reutilizar la mayor cantidad de código y librerías desarrolladas previamente en el CEIT.

A la hora de realizar el proyecto el problema que surge es que el lenguaje de programación de Android es Java, mientras que las librerías de AR disponibles están programadas en C++. Por este motivo necesitamos investigar las opciones de las que disponemos para poder utilizar ese código C++ en Android.

2.2. Objetivos

El objetivo principal que se plantea con la realización de este proyecto es la realización de aplicaciones de realidad aumentada para dispositivos móviles con sistema Android, reutilizando librerías desarrolladas en el CEIT.

Además surgen una serie de objetivos secundarios que se deben cumplir:

→ Adquirir conocimientos de la plataforma Android para ser capaces de desarrollar aplicaciones en este sistema.

→ Ampliar la destreza en los lenguajes de programación Java y C++ no sólo para programar las aplicaciones, sino además para entender el código de las librerías que se van a reutilizar.

→ Estudio de las librerías OpenGL ES para su correcta utilización.

2.3. Alcance

En este apartado se indica el trabajo que se realizará para lograr los objetivos marcados en el punto anterior.

Se va a realizar un estudio de las alternativas que existen para incluir librerías desarrolladas en C++ dentro de aplicaciones para la plataforma Android.

Una vez escogida la opción que más se adecúe a nuestras necesidades desarrollaremos dos aplicaciones que mostrarán la forma de aplicar la técnica elegida. La primera servirá para validar la propuesta, la segunda mostrará las capacidades de AR.

Esto implica formarse en los sistemas de desarrollo para Android, incluyendo captura de cámara, dibujado 3D, interfaces y comunicaciones.

2.4. Herramientas utilizadas

Para realizar este proyecto se ha utilizado un equipo de desarrollo equipado de la siguiente forma:

→ Sistema Operativo: Windows XP en el sobremesa del CEIT y Windows 7 en el ordenador personal de casa.

→ Librerías de desarrollo: Android SDK, OpenCV 2.3.1, Open GL ES 2.0.

→ Herramientas de desarrollo: IDE Eclipse Indigo con el plugin Android Development Tools (ADT).

Para la depuración y pruebas de las aplicaciones se ha utilizado el emulador del ADT y un tablet con la versión 2.2 de Android.

A la hora de redactar la memoria se han usado varias aplicaciones:

- → Microsoft Office Word 2007 para la redacción y edición del texto.
- → Microsoft Office Visio 2007 para la creación de los diagramas.
- → GanttProject para las gráficas de planificación del proyecto.

2.5. Estructura de Descomposición del Trabajo

En este apartado se muestra la estructura de descomposición del trabajo. No es más que una descripción de las fases en las que se divide el proyecto y que se muestra en la Ilustración 1.

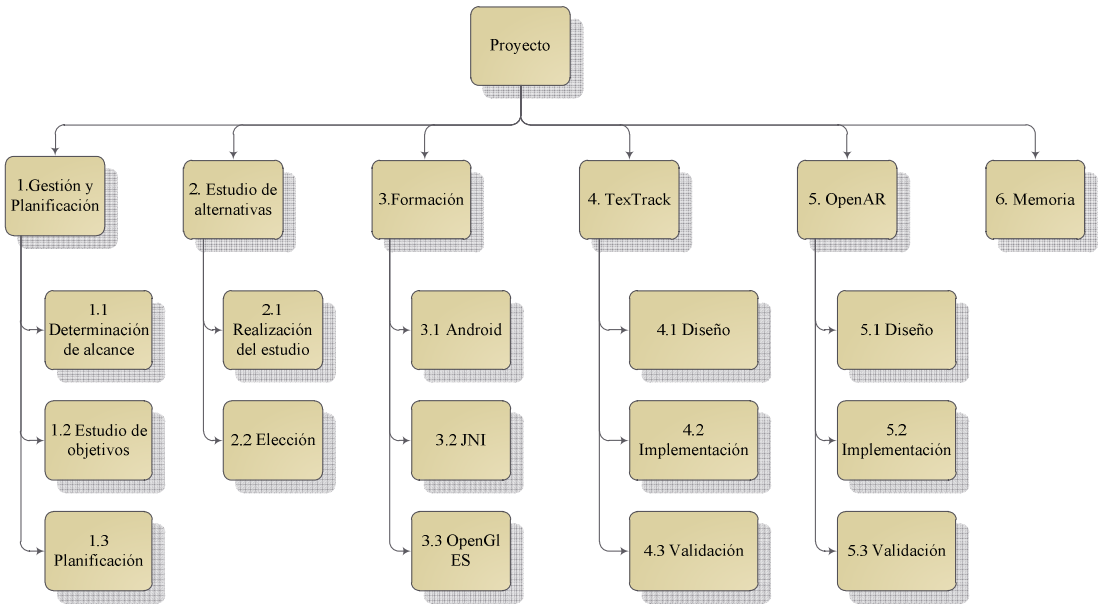

Ilustración 1: EDT.

Las fases que componen el proyecto son las siguientes:

1. Planificación y gestión del proyecto: fase en la que se determinan el alcance del proyecto. Se estudiarán los objetivos que se persiguen con la realización de este proyecto para realizar una correcta planificación.

2. Estudio de alternativas: en esta fase se presenta un análisis de diferentes alternativas para reutilizar en Android librerías desarrolladas en C++ y la justificación de la alternativa elegida para la realización del mismo.

3. Formación: fase en la que se estudiarán las tecnologías necesarias para la realización del proyecto entre las que se encuentran principalmente Android, JNI, OpenGL ES.

4. Textrack: fase en la que se diseña e implementa una primera aplicación que nos servirá para verificar la solución propuesta y que detectará texturas en las imágenes capturadas por la cámara.

5. OpenAR: fase en la que se diseña e implementa una aplicación de realidad aumentada que servirá para detectar marcadores a los que se asocian los objetos 3D a dibujar.

6. Memoria: fase en la que se recopila toda la información del proyecto y se redacta el documento con la memoria del proyecto.

2.6. Estimación de Tiempo

El proyecto comenzó a mediados de octubre y se desarrolló durante el curso hasta principios de julio. Aproximadamente se han dedicado 5 horas diarias a la realización del proyecto.

Hay que tener en cuenta que a la vez que se realizó el siguiente proyecto se compaginaban 3 asignaturas troncales de la carrera que han absorbido mucho tiempo y dedicación. En las épocas de exámenes se ha paralizado el desarrollo del proyecto que el alumno pudiera dedicar más tiempo al estudio de las asignaturas.

También hay que señalar los periodos de vacaciones correspondientes a Navidades (del 21 de diciembre del 2011 al 8 de enero del 2012 y Semana Santa (del 5 al 15 de abril de 2012).

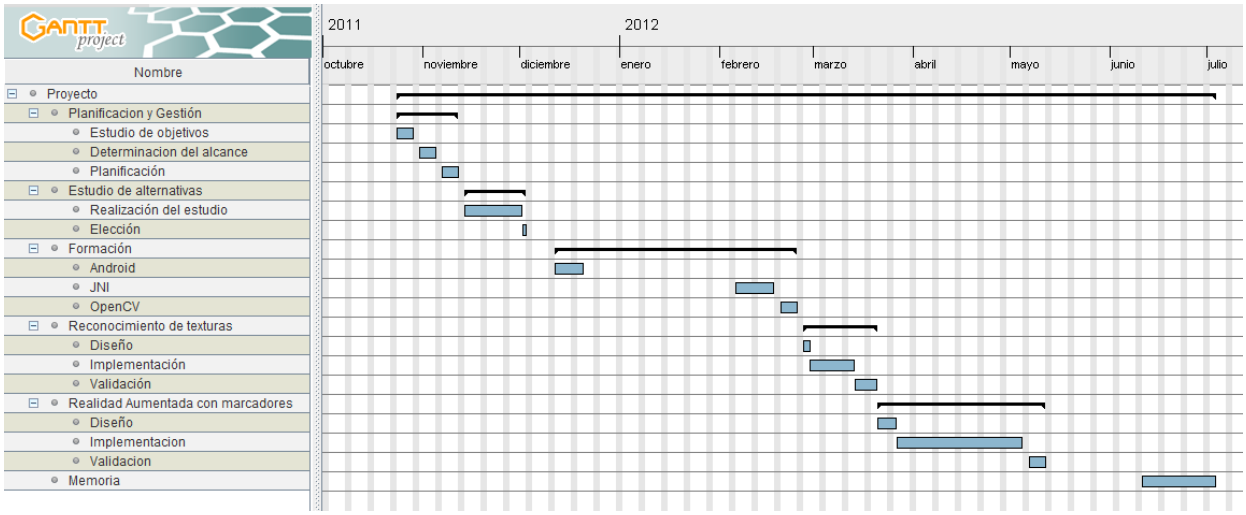

Ilustración 2: Diagrama Gantt del proyecto.

En las siguientes páginas se realizará una descripción más detallada del trabajo realizado dentro de cada fase, acompañada del diagrama de Gantt correspondiente.

Fase 1 Gestión y Planificación Inicial.

En esta fase se estudia y planifica la realización del proyecto. Se establecen los objetivos y el alcance.

1.1**Estudio de Objetivos:** se establecen los objetivos que se quieren lograr con la realización del proyecto.

1.2 **Determinación del Alcance:** se fija el trabajo que será necesario para lograr los objetivos establecidos.

1.3 **Planificación: O**rganización global de las fases del proyecto y la estimación del tiempo dedicado a cada una de ellas.

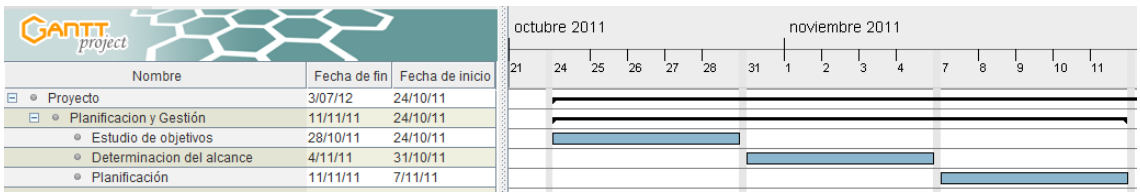

Ilustración 3: Diagrama Gantt: Gestión y Planificación.

Fase 2 Estudio de alternativas.

En esta fase se hace un estudio de las alternativas que existen actualmente para la realización de aplicaciones Android incluyendo código C++.

2.1 Realización del estudio: Se reúne la mayor cantidad posible de información para poder decidir qué tecnologías se escogerán para la realización del proyecto y cómo se utilizarán.

2.2 Elección: el alumno y el tutor de la empresa escogen la opción que mejor se amolda a las necesidades del proyecto.

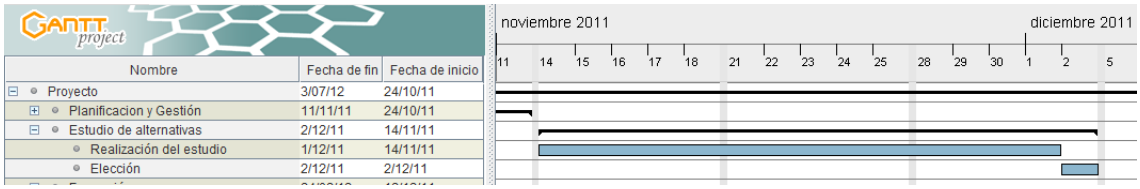

Ilustración 4: Diagrama Gantt: Estudio de alternativas.

Fase 3 Formación.

Estudio de las tecnologías y herramientas que se utilizan en el proyecto. Consistente en la lectura de documentación y en la realización de tutoriales para familiarizarse con el entorno de desarrollo.

Principalmente se ha estudiado el sistema Android y el modo en el que se podrían desarrollar las aplicaciones para la realización del proyecto mediante la creación de ejemplos básicos.

Por otro lado, también se han estudiado ejemplos y realizado pruebas para Java Native Interface (JNI) y la librería de visión por computador OpenCV para poder utilizarlos a lo largo de la realización del proyecto.

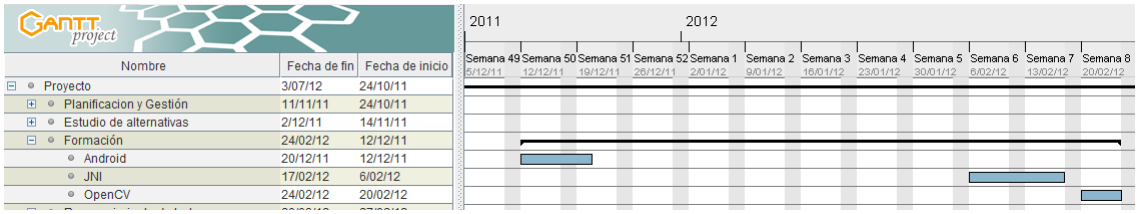

Ilustración 5: Diagrama Gantt: Formación.

Fase 4 TexTrack

Una primera aplicación implementada en la que se usará las técnicas acordes a la elección hecha en la Fase 2

4.1 Diseño: diseño de la arquitectura y los módulos necesarios para implementar la aplicación.

4.2 Implementación: una vez realizado un correcto diseño de la aplicación se procede a desarrollar el código para obtener la aplicación completa.

4.3 Validación: se prueba el funcionamiento de la aplicación mediante unas pruebas preestablecidas.

Ilustración 6: Diagrama Gantt: Reconocimiento de texturas.

Fase 5 Realidad aumentada con marcadores

Desarrollo de una segunda aplicación implementada que, al igual que la anterior, se divide en tres fases:

5.1 Diseño: se realiza un diseño detallado de las clases que conforman la aplicación.

5.2 Implementación: cuando ya tenemos el diseño apropiado de la aplicación, se implementa la solución propuesta.

5.3 Validación: se realizarán pruebas para comprobar que la aplicación funciona correctamente.

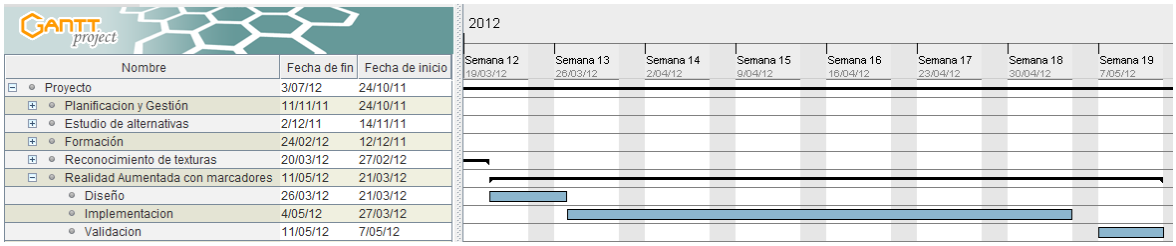

Ilustración 7: Diagrama Gantt: Realidad Aumentada con marcadores.

Fase 6 Memoria

Recopilación de toda la información obtenida, utilizada y generada así como la elaboración del documento de la memoria del proyecto.

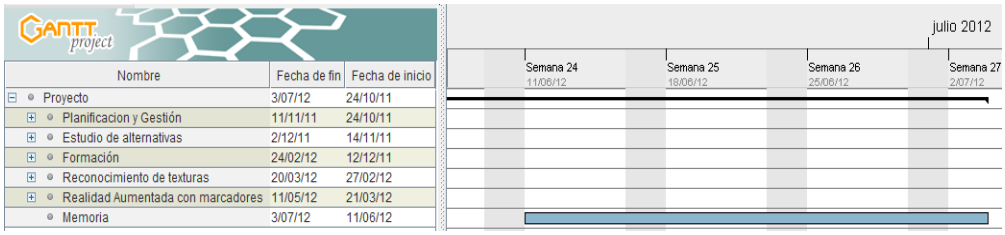

Ilustración 8: Diagrama Gantt: Memoria

2.7. Análisis de Riesgos.

A continuación se analizarán los riesgos identificados, la probabilidad de que sucedan y cómo podrían afectar al proyecto. Así mismo, se describirán las medidas que se han adoptado para evitarlos y el plan de contingencia en caso de que sucediesen.

2.7.1. Riesgos.

- → **Indisponibilidad de alguna de las partes**: la ausencia del alumno o del tutor de la empresa obligada por enfermedades u otros motivos puede ocasionar graves retrasos o incluso la cancelación del proyecto.

- → **Pérdida de información**: La pérdida de datos o cualquier otro tipo de información debida a diferentes motivos, como averías en el equipo o borrados accidentales, que pueden ocasionar graves retrasos.

- → **Problemas de desarrollo**: falta de experiencia por parte del alumno en la utilización de alguna de las herramientas y tecnologías aplicadas en el proyecto.

→ **Estimación errónea**: la inexperiencia en la planificación y gestión de proyectos, puede provocar una estimación de incorrecta que afecte a la duración prevista del proyecto.

→ **Compaginación de asignaturas y proyecto**: En paralelo a la realización del proyecto se han cursado 3 asignaturas obligatorias. Un suspenso en alguno de los exámenes puede provocar tener que dedicar mucho tiempo extra a recuperarlo e incluso tener que retrasar la entrega del proyecto al siguiente curso.

2.7.2. Cuantificación de riesgos.

En la siguiente tabla tenemos una enumeración de los riesgos junto a una estimación de la probabilidad de que ocurra. También se valora el impacto que pueden tener en la realización del proyecto en caso de que alguno de ellos sucediera.

Riesgo	Probabilidad	Impacto	Valoración
Indisponibilidad de alguna de las partes	Baja	Alto	Medio
Pérdida de Información	Baja	Alto	Alto
Problemas en el desarrollo	Alta	Medio	Medio
Estimación errónea	Media	Medio	Medio
Compaginación de asignaturas y proyecto	Media	Alta	Alta

Tabla 1: Cuantificación de riesgos

2.7.3. Plan de contingencia y actuación.

Indisponibilidad de alguna de las partes:

→Si el alumno sufre una indisponibilidad moderada, éste deberá recuperar las horas perdidas.

→En caso de que la indisponibilidad se alargara en el tiempo demasiado habría que revisar la planificación de la fecha de entrega.

→Si la indisponibilidad es causada por alguno de los tutores durante un largo periodo de tiempo, se procederá a buscar un sustituto.

Pérdida de información: Para tratar de evitar esta situación se llevará a cabo una política de copias de seguridad consistente en:

→ Realizar copias diarias de las aplicaciones y de la memoria.

→ Dichas copias se almacenarán en un pendrive, en el PC en casa y en un servidor privado del CEIT.

Problemas en el desarrollo: Para evitar este riesgo se ha tratado de reunir previamente la mayor información posible orientada a cumplir los objetivos de este proyecto. Si esta información no fuera suficiente durante el desarrollo, se procederá a buscar la información necesaria durante la fase de desarrollo o tomar otras alternativas.

Estimación errónea: si se produce esta situación, se replanificará la fase en la que se haya producido y las posteriores fases del proyecto.

Compaginación de asignaturas y el proyecto: Para evitar los efectos causados por este motivo, se dedicará una gran cantidad de tiempo a la preparación de los exámenes.

3. Estudio de alternativas

El centro en el que se ha realizado el proyecto tiene una gran cantidad de aplicaciones y librerías desarrolladas en C/C++. Ahora se quiere aprovechar ese código para crear aplicaciones en Android, evitando en lo posible modificarlo o reescribirlo en Java.

Deberemos tener en cuenta también la necesidad de usar un terminal físico para probar las aplicaciones. Nuestro objetivo final es desarrollar aplicaciones de Realidad Aumentada que harán uso de la cámara y soporte para gráficos de dicho terminal.

Al comenzar la realización de este estudio los emuladores no soportaban aplicaciones que usaran la cámara ni emulaba la aceleración de gráficos a través de la GPU. En posteriores versiones del SDK de Android se han ido subsanando algunas de estas debilidades.

A continuación se muestra un breve análisis de las alternativas para acceder a librerías C++ lo cual es la piedra angular para la realización del presente proyecto:

- → Android Native Development Kit (JNI y Native Activities)
- → Mono for Android
- → Arquitectura cliente-servidor.

3.1. Android Native Development Kit

El Android Native Development Kit (NDK) es un conjunto de herramientas que permite el uso de código nativo en las aplicaciones Android. Se entiende por código nativo aquel que está compilado para una arquitectura concreta en la que se vaya a trabajar. En este caso las librerías C/C++ utilizadas en el centro. Este hecho puede provocar la pérdida del componente multiplataforma de Android ya que tendremos que compilar las librerías para cada arquitectura en la que se quiera trabajar.

Gracias al NDK podemos efectuar llamadas a funciones de librerías desarrolladas en C/C++ desde Java, e incluso a partir de la versión 2.3 (Gingerbread) también nos permite implementar aplicaciones Android enteramente en C/C++.

Al principio el compilador del NDK sólo trabajaba en sistemas Unix por lo que había que instalar algún emulador para poder utilizarlo en sistemas Windows. A partir de la versión r7 se incluye un nuevo comando que nos permite compilar el código nativo directamente desde la consola de Windows.

El NDK nos permite usar dos métodos distintos que se describen a continuación.

3.1.1. Java Native Interface (JNI)

Java Native Interface es un interfaz que funciona como puente entre el Java y otros lenguajes de programación. Permite que un programa escrito en Java y que se esté ejecutando en la máquina virtual java pueda interactuar con métodos escritos en otros lenguajes.

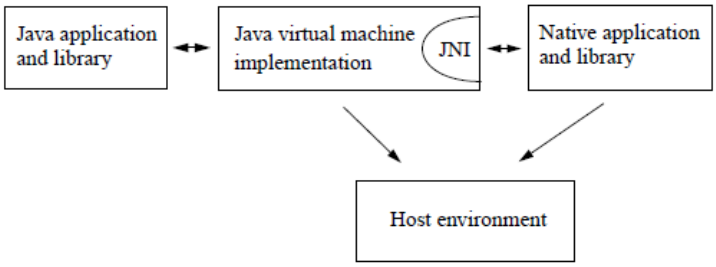

Ilustración 9: JNI como puente entre Java y C++

El desarrollo es igual que cualquier otra aplicación Android, pero utilizaremos JNI para realizar llamadas a aquellas funciones de las librerías nativas que necesitemos utilizar.

3.1.2. Native activities

La clase NativeActivity se encarga de manejar la comunicación entre el framework de Android y el código nativo.

Las native activities se pueden usar a partir de la versión Gingerbread (2.3) de Android. Con ellas se pueden implementar aplicaciones de Android enteramente en código nativo.

También se pueden usar para implementar el ciclo de vida de una activity y los diferentes eventos que se produzcan.

Para utilizar las native activities, basta con declarar en el Android Manifest que lo que se va a desarrollar es una aplicación nativa.

Aunque en la clase NativeActivity se nos ofrece cada vez más funcionalidades desde la parte nativa, es necesario seguir usando JNI para acceder a varias características de las APIs de Android.

3.2. Mono for Android

Mono es una plataforma de desarrollo que utiliza C# para programar las aplicaciones Android. Ejecuta las aplicaciones a través de una instancia del Android runtime VM, que coexiste con la máquina virtual Dalvik. Mono permite importar librerías desarrollados en C++ fácilmente de la misma forma que se realiza en la plataforma .Net.

Para el desarrollo se pueden utilizar Microsoft Visual Studio 10 o el propio de la plataforma, MonoDevelop. Ambos entornos requieren adquirir licencias de desarrollo.

También existe una versión de prueba que tiene funciones limitadas. Por ejemplo, las aplicaciones sólo se pueden probar en unidades simuladas y no en terminales físicos. La licencia de uso es de duración ilimitada. Aunque si se quiere mantener actualizado, habrá que renovar la suscripción a las actualizaciones anualmente.

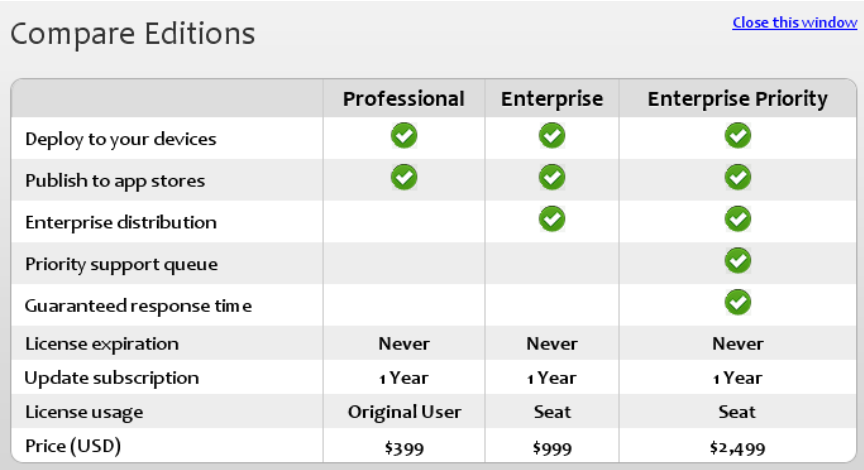

Ilustración 10: Comparativa de las licencias de Mono for Android.

3.3. Arquitectura cliente-servidor

Otra alternativa para hacer uso de librerías nativas auxiliares, es utilizar una arquitectura cliente-servidor que se comunicarían mediante una conexión de red. Esto provoca que además haya que programar una cama de comunicación entre las dos partes.

La parte cliente estaría desarrollada enteramente en Java y se ejecutaría en el dispositivo móvil. Mientras que la parte del servidor se implementaría en C++ y se desplegaría en un ordenador remoto.

En el caso de una aplicación de realidad aumentada el cliente se encargaría de la captura de imágenes a través de la cámara del dispositivo. Seguidamente se las enviaría al servidor para que las procese y aplique las transformaciones necesarias a la imagen. Finalmente se devolvería el resultado al terminal cliente.

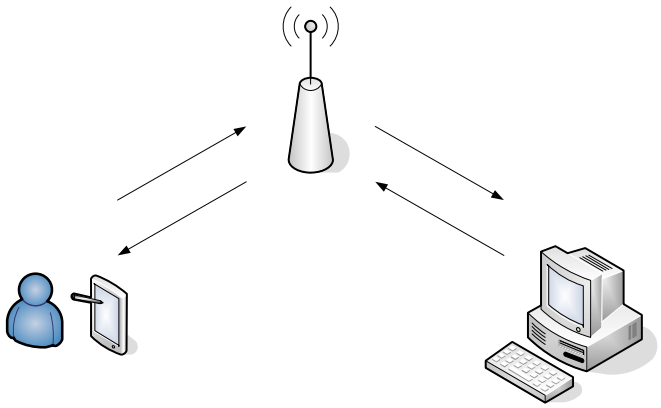

Ilustración 11: Arquitectura cliente-servidor

3.4. Opción elegida

Después de considerar las alternativas estudiadas nos decidimos por usar el Android NDK con JNI. Las razones para decantarnos por esta opción son las siguientes:

- → Se puede utilizar Eclipse como entorno de desarrollo. Además de ser fiable y tener una gran comunidad de apoyo, es gratuito. Mientras que para los entornos de desarrollo que utiliza Mono hay que obtener una licencia para disfrutar de todas sus opciones.

- → Mayor disponibilidad de documentación. En el momento en el que se hizo el estudio, la información sobre las native activities y Mono for Android se limitaba prácticamente a lo que ofrecían sus desarrolladores y unos sencillos tutoriales. En cambio la documentación de JNI es amplia y variada.

- → Aunque se estudió la adquisición de un terminal más moderno, finalmente se ha utilizado una tablet Samsung Galaxy de la que ya disponía el departamento. Dicha tablet tenía instalada la versión Froyo de Android (2.2) lo que provocó la imposibilidad de utilizar las native activities.

- → En el mundo real no siempre se puede tener acceso a una red, lo que nos llevó a descartar la arquitectura cliente-servidor. Además sería complicado obtener un buen sincronismo Real Virtual

4. Reconocimiento de texturas

Una vez elegida la opción para utilizar código nativo en Android, se procede a la implementación de una primera aplicación. Esta aplicación realizará el seguimiento (tracking) 3D de texturas. La información para el seguimiento es obtenida a través del procesamiento de imágenes, y permitirá alinear objetos virtuales con sus correspondientes homónimos reales haciendo uso de la tecnología de la Realidad Aumentada.

4.1. Objetivo de la aplicación

Dado un conjunto de imágenes de referencia de varios objetos, el objetivo consiste en determinar si en la imagen actual capturada por la cámara aparece alguno de estos objetos. Se debe hacer un emparejamiento entre las imágenes de referencia y las nuevas imágenes de entrada. Además, en caso de encontrar una correspondencia positiva, se deberá conocer de qué objeto se trata y cual es su posición y orientación 3D.

Otro objetivo de la aplicación es comprobar el funcionamiento del Android NDK y JNI para la reutilización de las librerías de código nativo ya desarrolladas en el CEIT.

4.2. Requisitos

En este apartado describiremos los requisitos mínimos que se deben cumplir para que la aplicación funcione correctamente.

El dispositivo móvil en el que se instale la aplicación deberá contar con la versión de Android 2.2 (Froyo) o superior. Obviamente también tendrá que disponer de una cámara para la captura de imágenes.

Además para el funcionamiento de la aplicación se necesita que se encuentren en la memoria del dispositivo varios archivos:

- → **camera_file**: fichero (.txt) que almacena los parámetros intrínsecos de la cámara.

- → **config_file**: fichero (.txt) que indica cuales son las imágenes de referencia. El formato es el que se muestra a continuación, donde se indica el factor de escala, el número de imágenes de referencia y el nombre de las imágenes de referencia

Ilustración 12: Formato del configFile.txt

→ **training_file**: fichero (.bin) que contiene los datos generados en el proceso de entrenamiento. Si no se especifica ningún fichero (valor NULL), entonces se lleva a cabo el entrenamiento.

4.3. Diseño

La aplicación consta de una sola activity llamada TextrackActivity. Una activity básicamente es una pantalla de una aplicación Android. Esta activity cargará la clase TrackingView.

TrackingView es la clase encargada de tomar las imágenes de la cámara y mostrarlas en la pantalla una vez tratadas. También hará las llamadas a las librerías para procesar las imágenes.

Finalmente tendremos un módulo formado por las librerías desarrolladas en la empresa y por la parte de JNI que hará de puente entre la parte Java y las librerías.

4.4. Implementación

Una vez realizado el diseño, vamos a especificar los detalles más importantes para la implementación de la aplicación. La implementación la revisaremos en dos partes. Por un lado se explicarán los aspectos pertenecientes a la parte de Android y por otro los correspondientes a la parte del código nativo

4.4.1. Android

En la parte Android hay que implementar el AndroidManifest y las clases Java que utiliza la aplicación.

AndroidManifest.xml

El AndroidManifest le indica al sistema Android los recursos que la aplicación necesita para poder ejecutarse. En nuestro caso necesitamos permisos para utilizar la cámara y para escribir en la memoria del dispositivo, ya que bajo determinadas condiciones tendremos que crear el fichero con los datos de entrenamiento. Para ello añadiremos las siguientes líneas al manifiesto:

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

TextrackActivity

Esta activity es el punto de entrada de la aplicación. Sólo se ha implementado el método onCreate que se ejecutará al iniciarse la aplicación. Le indica a Android que no se apague la pantalla automáticamente y que elimine la barra del título. Además también se ocupa de cargar la vista de la cámara.

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature( Window.FEATURE_NO_TITLE );

        getWindow().setFlags( WindowManager.LayoutParams.FLAG_FULLSCREEN,
                              WindowManager.LayoutParams.FLAG_FULLSCREEN );

        getWindow().setFlags( WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
                              WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        setContentView(new TrackingView(this));
    }
```

TrackingView

La clase TrackingView implementa la cámara. Para poder visualizar las imágenes de la cámara debemos tener una superficie de la clase SurfaceView para mostrar los fotogramas. Esta superficie es controlada por una interfaz de tipo SurfaceHolder, que permite, entre otras cosas, controlar el formato de la superficie, o modificar los píxels de la superficie. De esta forma, la clase extenderá a la clase SurfaceView e implementará la interfaz SurfaceHolder.

Para desarrollar esta clase hemos partido de la ofrecida en los ejemplos de la librería OpenCV que toma, procesa y muestra las imágenes de la cámara. Se ha modificado añadiendo las llamadas a las librerías con el código nativo.

Los métodos nativos que se vayan a utilizar en esta clase hay que especificarlos, añadiéndoles el modificador “native” y cargar la librería donde se encuentra su código. Aunque la implementación de estos métodos la explicaremos en la sección correspondiente al código nativo, a continuación se muestra su especificación y la carga de la librería.

```
private native void inicializarTexTrack();
private native void TexTrack(int width, int height, byte yuv[], int[] rgb);
private native void finalizarTexTrack();



//carga de la librería
    static {
       System.loadLibrary("TexTrack");

    }
```

4.4.2. Parte nativa

Esta sección describe la implantación de los módulos necesarios para exportar los métodos que se vayan a usar desde las clases Java. Estos módulos se compilan a través del Android NDK para generar la librería. Además del código programado en C++, hay que crear los ficheros **Android.mk y Aplication.mk** para indicar al compilador qué recursos tiene que compilar y también otras opciones de compilación.

Seguidamente se describe el módulo que se ha implementado

jni_part

Esté modulo define e implementa las funciones puente entre Java y las librerías C++. Está desarrollado en C++ y siguen los convenios de JNI (ver Anexo II). Es llamado por el módulo de la cámara y llama a las funciones de las librerías C++ que se van a reutilizar.

Tiene un método que llama al constructor del objeto encargado de realizar el tracking de las texturas. Al hacer esta llamada si no encuentra el archivo training.bin se produce el entrenamiento.

```
JNIEXPORT void JNICALL Java_com_tracking_SampleViewBase_inicializarTexTrack(
      JNIEnv env, jobject thiz);
```

Como hemos dicho también implementamos un método que será el encargado de eliminar el objeto encargado de realizar el tracking de las texturas.

```
JNIEXPORT void JNICALL Java_com_tracking_SampleViewBase_finalizarTexTrack(
   JNIEnv env, jobject thiz);
```

También se ha implementado un método que procesa la imagen. Recibe dos números que indica la altura y la anchura de la imagen. También se le pasa como parámetro el contenido de la imagen de entrada y dejará en rgb la información de la imagen una vez procesada. Hacemos esto ya que la imagen captada por la cámara está codificada con el estándar YUV, mientras que la librería está diseñada para trabajar con codificación RGB. La conversión de un formato al otro la realiza internamente usando la librería OpenCV. Si detecta alguna de las imágenes de referencia dibuja un marco alrededor de la textura.

```
JNIEXPORT void JNICALL Java_com_tracking_TrackingView_TexTrack(
      JNIEnv env, jobject thiz, jint width, jint height,
      jbyteArray yuv, jintArray rgb);
```

Librerías utilizadas:

- **ARTrackPlanarTexture**: Es el módulo principal creado previamente en el CEIT escrita en C/C++. Está encargado de realizar dos tareas:
    - *Entrenamiento*: Pre-procesa las imágenes de referencia y almacena toda esta información, dejándola accesible en tiempo de ejecución. Esta información es almacenada en un fichero (Training.bin).
    - *Tracking*: Procesar las imágenes de la cámara en tiempo de ejecución y extraer toda la información de tracking. Hace uso de los datos pre-procesados (Training.bin).

- **OpenCV:** es una librería de Visión por Computador desarrollada en C++. También tiene una versión para programar en Android.
- **Log:** Librería que nos ofrece el Android NDK para mostrar mensajes de información en la consola.

Application.mk

En el fichero Application.mk indicamos al compilador qué versión de la librería estándar de C++ vamos a utilizar y la arquitectura para la que queremos compilar la librería nativa.

```
APP_STL := gnustl_static
APP_ABI := armeabi-v7a
```

Android.mk

En el Android.mk le indicamos al compilador cuáles son los ficheros con el código fuente, su ubicación y el nombre que queremos dar a la librería compilada y qué otras librerías utiliza.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

#Opciones para la carga de los módulos de OpenCV
OPENCV_LIB_TYPE:=STATIC
OPENCV_INSTALL_MODULES:=on
OPENCV_CAMERA_MODULES:=off
```

```
include includeOpenCV.mk
ifeq ("$(wildcard $(OPENCV_MK_PATH))","")
      include $(TOOLCHAIN_PREBUILT_ROOT)/user/share/OpenCV/OpenCV.mk
else
      include $(OPENCV_MK_PATH)
endif


NDK_MODULE_PATH:= $(LOCAL_PATH)

#Nombre de la librería creada
LOCAL_MODULE    := TexTrack

LOCAL_SRC_FILES := \
      #listado de ficheros fuente


LOCAL_CPP_EXTENSION := .cpp

LOCAL_CPP_INCLUDES := \
      #listado de los archivos

LOCAL_LDLIBS +=  -llog


include $(BUILD_SHARED_LIBRARY)
```

4.5. Validación

En este apartado se describe la prueba realizada para comprobar el funcionamiento de la aplicación y el resultado obtenido, mostrando que funciona cumpliendo los objetivos.

4.5.1. Descripción

Para realizar esta prueba se ha iniciado la aplicación sin guardar el fichero "training.bin". Con esto forzamos que al no detectar el archivo, se ejecute la fase de entrenamiento. Una vez terminada, se procederá a la captura de datos de la cámara y a efectuar el tracking. Si detecta una de las imágenes de referencia dibujará un marco a su alrededor.

La siguiente imagen muestra la textura que se quiere detectar:

Ilustración 13: Textura de muestra

4.5.2. Resultado

La siguiente captura muestra el resultado de la prueba. El funcionamiento es el esperado, produciéndose el reconocimiento de la textura de forma satisfactoria e ignorando otras texturas que aparecen en la imagen.

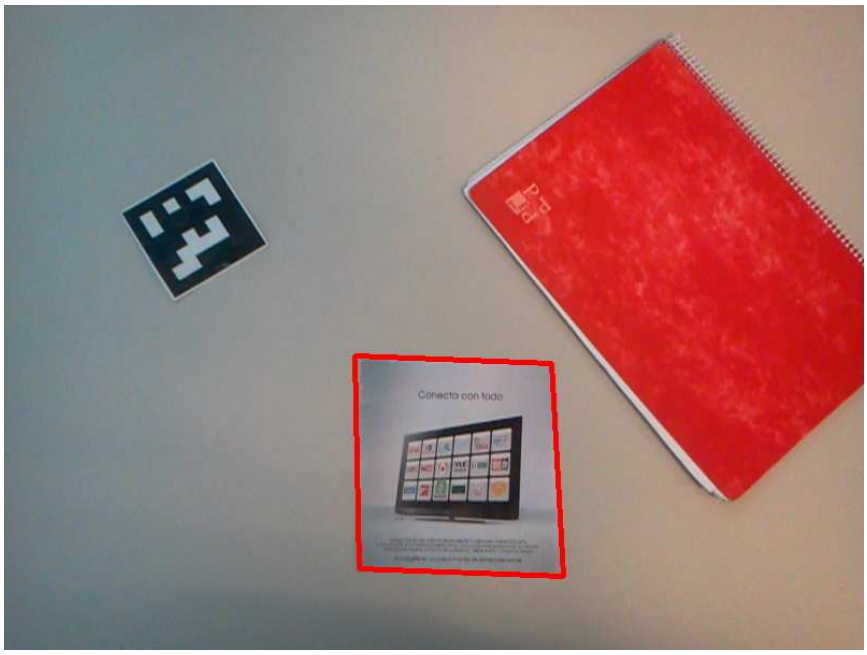

Ilustración 14: Resultado de la prueba.

5. Realidad aumentada con marcadores

La segunda aplicación desarrollada es una aplicación de realidad aumentada con marcadores. Se analizarán las capturas realizadas por la cámara para ver si aparece en ellas algún marcador. A continuación se muestran dos ejemplos de marcadores que se utilizan en realidad aumentada.

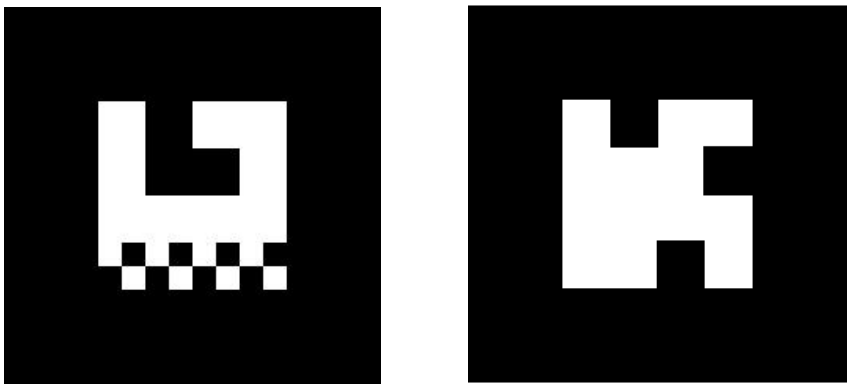

Ilustración 15: Ejemplo de marcadores.

En el siguiente capítulo se describe el diseño e implementación de la aplicación

5.1. Objetivos de la aplicación

El objetivo de la aplicación es detectar si en la imagen capturada aparece un marcador y superponer en la imagen del "mundo real" un modelo en 3D colocado de acuerdo a la posición del marcador.

Finalmente se quiere comprobar el funcionamiento de Android NDK y JNI para reutilizar código C++.

5.2. Requisitos

En este apartado describiremos los requisitos mínimos que se deben cumplir para que la aplicación funcione correctamente.

El dispositivo móvil en el que se instale la aplicación deberá contar con la versión de Android 2.2 (Froyo) o superior y disponer de una cámara para la captura de imágenes.

Además para el funcionamiento de la aplicación se necesita que se encuentren en la memoria los siguientes archivos:

- → **camera_file. txt**: almacena los parámetros intrínsecos de la cámara.
- → **Object_file.obj**: contiene la información del modelo que se quiere dibujar. Se rige según el estándar de Wavefront.

5.3. Diseño

Este apartado realizaremos un resumen del diseño de la aplicación y cada una de sus partes.

Para realizar el renderizado de los objetos y texturas se va a utilizar OpenGL ES 2.0. En el Anexo III se explican con mayor detalle algunos los aspectos importantes al usar OpenGL ES 2.0.

La clase principal de la aplicación es OpenArActivity que extiende la clase Activity y que cargará las clases CameraPreview y GLLayer.

La clase CameraPreview captura las imágenes de la cámara y las manda a la clase GLLayer que es la superficie en la que se renderizarán el modelo que se quiera dibujar y una textura de fondo con las imágenes reales tomadas por la cámara.

Para dibujar el modelo usaremos la clase Model, que utiliza la clase ObjLoader para cargar y guardar la información necesaria para renderizar el objeto.

Un módulo formado por las librerías C++ del centro gestionará la realidad aumentada. JNI hará de puente entre la parte Java y dichas librerías.

5.4. Implementación

Una vez realizado el diseño, vamos a especificar los detalles más importantes para la implementación de la aplicación. Primero se explicarán las clases propias de Android y seguidamente las del código nativo.

5.4.1. Android

En la parte Android hay que implementar el AndroidManifest y las clases Java que utiliza la aplicación

AndroidManifest.xml

En el AndroidManifest basta con indicar que se necesitan los permisos para el uso de la cámara, ya que a diferencia de la aplicación anterior no realizamos escrituras en la memoria del dispositivo. Así mismo, como utilizamos OpenGL ES 2.0 también debemos señalarlo. La forma de indicarlo es a través de las siguientes líneas:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-feature android:required="true" android:glEsVersion="0x00020000"/>
```

OpenArActivity

OpenArActivity es la única activity de la que consta la aplicación. A través del método onCreate, cargamos la capa de la cámara y la capa de OpenGL en la que se visualizarán las imágenes.

```
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        this.setRequestedOrientation
             (ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        mGlLayer = new GlLayer(this);
        mCamPrev = new CameraPreview(this, mGlLayer);

        setContentView(mGlLayer);

        addContentView(mCamPrev,new LayoutParams
             (LayoutParams.FILL_PARENT, LayoutParams.FILL_PARENT));
    }
```

CameraPreview

La clase CameraPreview se ocupará de tomar las imágenes de la cámara y enviarlas a la clase GLLayer. Esto lo logramos implementando el callback onPreviewFrame. Mientras esté la cámara tomando imágenes, se ejecutará cada vez que detecta un nuevo fotograma.

```
public void surfaceCreated(SurfaceHolder holder) {
	try {
		mCamera = Camera.open();
		mCamera.setPreviewDisplay(null);
		mCamera.startPreview();

		mCamera.setPreviewCallback(new PreviewCallback() {
		  public void onPreviewFrame(byte[] data, Camera camera) {
			mGLLayer.camData(data, getFrameWidth(),
				getFrameHeight());
		  }
	});
}
```

GLLayer

Como hemos dicho en el diseño de la aplicación usaremos OpenGL ES 2.0 para el renderizado de los objetos y texturas. En Android esto supone que se necesitará una GLSurfaceView y un interfaz. Renderer. Para lograr esto crearemos la clase GLLayer de la aplicación.

La función de GLLayer es procesar y renderizar los fotogramas que recibe de la cámara. Extiende de la clase GLSurfaceView para mostrar las imágenes renderizadas. También implementa la interfaz GLSurfaceView.Renderer que define los métodos para dibujar gráficos.

Al implementar la interfaz GLSurfaceView.Renderer se deben implementar los siguientes métodos:

- → Un método que se llama una sola vez cuando se crea la superficie GLSurfaceView. Se utiliza para las acciones que sólo se ejecutan una vez como iniciar el contexto de OpenGL y cargar los datos referentes al objeto

```
public void onSurfaceCreated(EGLConfig config)
```

→ El siguiente método se ejecuta cada vez que se redibuja la GLSurfaceView y desde aquí llamaremos a las funciones que dibujan el modelo.

```
public void onDrawFrame()
```

→ El último método obligatorio de esta clase es OnSurfaceChanged y se llama cada vez que se produzca un cambio en el tamaño u orientación del dispositivo

```
public void onSurfaceChanged(GL10 unused, int width, int height)
```

En GLLayer también se declaran los métodos nativos que se van a llamar y se carga la librería que los implementa.

```
private native void calc(long addrImage);
private native boolean getMarkerInfo(int marker_Id, float[] m);
private native float[] getProjectionMatrix();

static {
      System.loadLibrary("ARToolKitPlus");
      System.loadLibrary("wrapper");
   }
```

ObjLoader

La clase ObjLoader carga desde el archivo .obj, la descripción del objeto que se quiere renderizar. Guarda los datos de los vértices, vectores normales y caras del objeto que se quiere renderizar. Para conseguir esto se ha implementado la siguiente función que dado el nombre del archivo nos devuelve verdadero en caso de que se haya conseguido cargar el objeto:

```
public boolean loadOBJ(FileInputStream in)
```

Model

Esta clase tiene un ObjLoader para cargar el objeto, así como los métodos y shaders necesarios para dibujar el objeto en la pantalla.

El método initShaders carga los shaders en el pipeline de OpenGL

```
private void initShaders()
```

Para dibujar los objetos tenemos el método drawModel que dados la matriz de proyección y del modelo así como un factor de escala para las dimensione en x e y, dibuja el objeto en pantalla.

```
public void drawModel(float[] projection, float[] modelview, float scalex,
float scaley)
```

5.4.2. Parte nativa

Esta sección describe la implantación de los módulos desarrollados en C++ y que se compilarán con el Android NDK para generar la librería. También se detallan los archivos **Android.mk y Aplication.mk** en los que se informa al compilador qué recursos se tienen que compilar.

jni-part

Esté modulo se implementan los métodos que sirven de puente entre la parte de java y los módulos desarrollados en C++.

El primer método llama al constructor del objeto que analiza la imagen en busca de marcadores.

```
JNIEXPORT void JNICALL
      Java_com_Artkpexp_GlLayer_inicializarARToolKitPlus
            (JNIEnv* env, jobject thiz, jbyteArray);
```

También se ha implementado un método para procesar las imágenes. Dada una imagen RGB como parámetro de entrada (image), la función procesa la misma para determinar

la presencia de marcadores. Finalmente guarda la información de posicionamiento (ubicación en la imagen) para cada uno de los marcadores detectados.

```
JNIEXPORT void JNICALL Java_com_openar_GlLayer_calc(JNIEnv* env,
 jobject thiz, jlong image);
```

Se ha diseñado una función que a partir de un identificador de marcador (id) devuelve verdadero en caso de que el marcador asociado a dicho identificador haya sido detectado en la imagen procesada por la función calc (véase arriba). En caso contrario devuelve falso. Además, en caso de una detección positiva, también devuelve en la variable modelView la posición y orientación de la cámara respecto al marcador (una matriz 4x4 válida para fijar la matriz de modelado en OpenGL), permitiendo renderizar objetos virtuales correctamente alineados con el marcador.

```
JNIEXPORT jboolean JNICALL Java_com_openar_GlLayer_getMarkerInfo
      (JNIEnv* env, jobject thiz, jint marker_Id, jfloatArray modelView);
```

La función getProjectionMatrix devuelve la matriz de calibración de la cámara utilizada para capturar las imágenes. Esta matriz depende de los parámetros intrínsecos de la cámara, y define la proyección de los objetos 3D al plano de imagen 2D. Asimismo, se devuelve una matriz 4x4, válida para fijar la matriz de proyección de OpenGL.

```
JNIEXPORT jfloatArray JNICALL
      Java_com_openar_GlLayer_getProjectionMatrix
            (JNIEnv* env, jobject thiz);
```

El último método implementado elimina el objeto que realiza la detección de marcadores.

```
JNIEXPORT void JNICALL
       Java_com_Artkpexp_GlLayer_finalizarARToolKitPlus
            (JNIEnv* env, jobject thiz),
```

Librerías utilizadas:

ARToolkitPlus: es una librería para la creación de aplicaciones de realidad aumentada implementada en C++.

OpenCV: es una librería de Visión por Computador desarrollada en C++. También tiene una versión para programar en Android.

Log: Librería que nos ofrece el Android NDK para mostrar mensajes de información en la consola.

Application.mk

En fichero **Application.mk** indicamos al compilador la versión de la librería estándar de C++ utilizada y la arquitectura para la que queremos compilar la librería nativa.

```
APP_STL := gnustl_static
APP_ABI := armeabi-v7a
```

Android.mk

En el Android.mk le indicamos al compilador cuáles son los ficheros con el código fuente, su ubicación y el nombre que queremos dar a la librería compilada y qué otras librerías utiliza.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

OPENCV_CAMERA_MODULES:=off
include includeOpenCV.mk
ifeq ("$(wildcard $(OPENCV_MK_PATH))","")
      #try to load OpenCV.mk from default install location
      include $(TOOLCHAIN_PREBUILT_ROOT)/user/share/OpenCV/OpenCV.mk
else
      include $(OPENCV_MK_PATH)
endif
NDK_MODULE_PATH:= $(LOCAL_PATH)

LOCAL_MODULE    := ARToolkitPlusWrapper

LOCAL_SRC_FILES := \
```

```
      jni_part.cpp \
      ARToolkitPlusWrapper.cpp\


LOCAL_SHARED_LIBRARIES := ARToolKitPlus

LOCAL_LDLIBS +=  -llog -ldl

include $(BUILD_SHARED_LIBRARY)
 $(call import-module,ARToolKitPlus)
```

5.5. Validación

En este apartado se detalla la prueba realizada para comprobar el funcionamiento de la aplicación y el resultado obtenido, mostrando que funciona cumpliendo los objetivos.

5.5.1. Descripción

La validación se ha realizado capturando las imágenes de la cámara y utilizando un marcador de los proporcionados por ArtoolkitPlus. Cuando lo detecta tiene que dibujar el objeto virtual en la posición que señala el marcador.

5.5.2. Resultado

La aplicación detecta correctamente el marcador y renderiza el objeto virtual con respecto a su posición. En este caso se renderizar un objeto sencillo como es una tuerca. A continuación se muestran una serie de imágenes que verifican el funcionamiento de la aplicación. Cabe destacar que la imagen no esté centrada con la realidad debido a que la cámara del dispositivo se encuentra en una esquina del mismo.

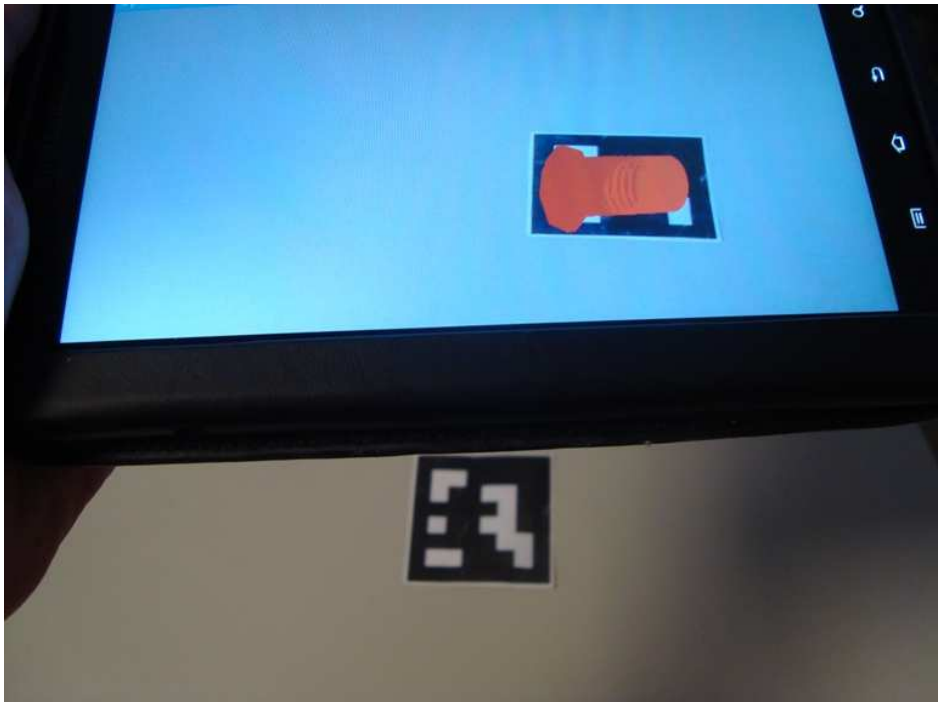

Ilustración 16: Verificación de OpenAr 1

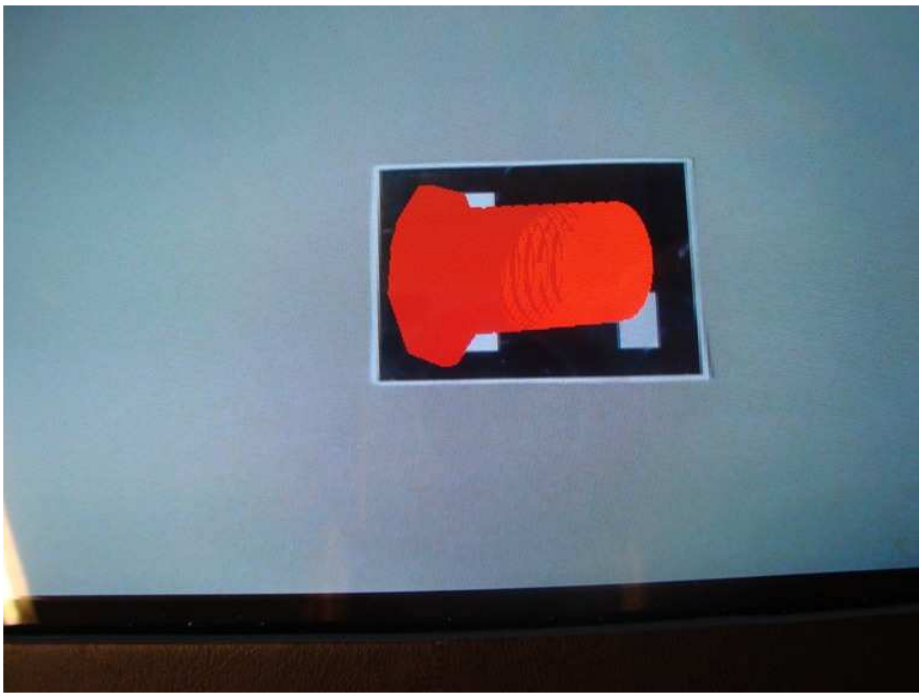

Ilustración 17: Verificación de OpenAr 2

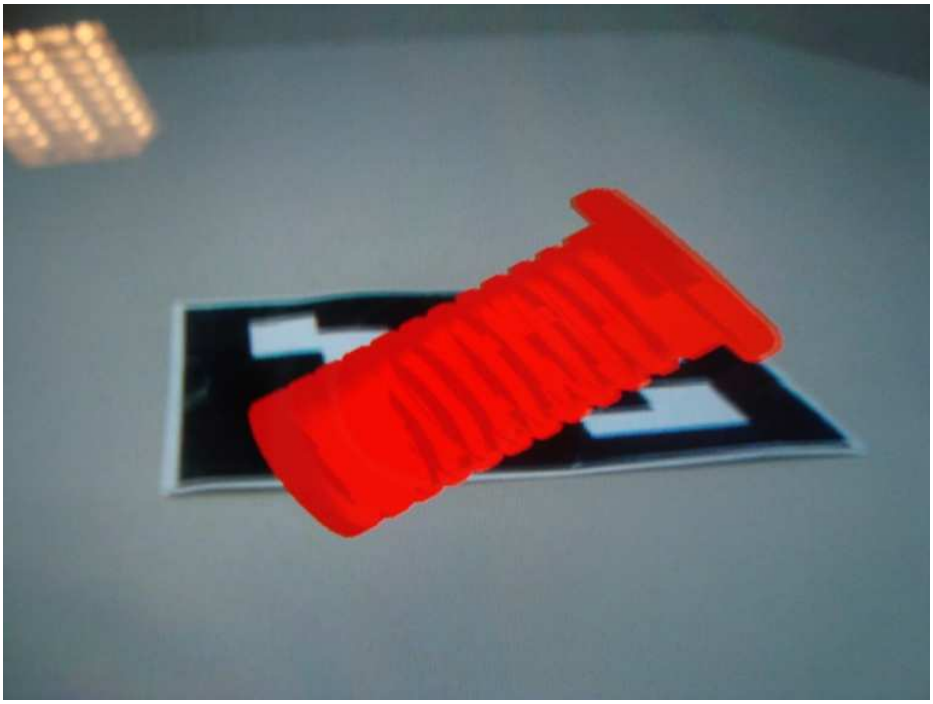

Ilustración 18: Verificación de OpenAr 3

6. Conclusiones y líneas futuras de trabajo

En este último apartado de la memoria del proyecto se detalla la concordancia entre los resultados obtenidos y los objetivos establecidos al principio del proyecto. Por otro lado, se presentan las conclusiones que han derivado de la realización del proyecto. Así como un resumen de las futuras líneas de trabajo en las que se puede hacer hincapié una vez se dé por concluido el desarrollo de esta aplicación.

6.1. Concordancia entre resultados y objetivos.

Una vez que se ha realizado el estudio de alternativas y la implementación de las dos aplicaciones, se procede a comprobar si se han alcanzado los objetivos que se han marcado al principio del proyecto.

El objetivo principal que se planteaba era la realización de aplicaciones de realidad aumentada para dispositivos móviles con sistema Android, reutilizando librerías desarrolladas en el CEIT en C++.

Como hemos visto en los apartados de validación de las aplicaciones desarrolladas, somos capaces de invocar los métodos de esas librerías, por lo que consideramos que se ha cumplido el objetivo.

Los objetivos secundarios también se han logrado ya que para el desarrollo de las aplicaciones hemos tenido que estudiar las características de la plataforma Android.

Para la realización del proyecto se ha tenido que programar una gran cantidad de código tanto en Java como en C++, lo que ha producido una mayor destreza en el uso de estos lenguajes de programación.

Por último, para obtener la aplicación OpenAr se ha tenido que hacer uso de la librería OpenGL ES, por lo que se han aprendido los conceptos básicos de esta librería y su uso en dispositivos Android.

6.2. Líneas futuras

Una vez que hemos conseguido invocar con éxito las funciones y métodos que habían sido desarrollados en la empresa, el siguiente paso que se puede dar es crear aplicaciones más complejas.

Para lograr aumentar la complejidad se puede añadir a las aplicaciones interfaces de usuario más complejos y ambiciosos.

Otra posibilidad que se puede explorar sería programar eventos que respondan a las señales enviadas por los sensores del dispositivo.

6.3. Conclusiones

Una vez terminada la aplicación y después de analizar los resultados obtenidos, se obtienen las siguientes conclusiones a nivel profesional y personal.

En primer lugar, cabe destacar que se han alcanzado todos los objetivos propuestos. Se han conseguido desarrollar dos aplicaciones que haciendo uso del Android NDK y JNI invocan métodos de librerías desarrolladas con el lenguaje de programación C++.

A nivel personal, ha sido gratificante poder realizar el proyecto en un centro de la relevancia del CEIT dentro de un entorno de investigación.

Además de esto, la realización de este proyecto ha traído consigo la adquisición de nuevos conocimientos en un campo, el del desarrollo de aplicaciones Android, con un gran mercado lo que se puede aprovechar para un futuro laboral.

7. Bibliografía

Libros

James Stelee & Nelson To (2010). ***The Android Developer's Cookbook***, Adisson Wesley**.** ISBN-10: 0-321-74123-4

Frank Ableson & Charlie Collins & Robi Sen (2009) ***Unlocking Android – A Developer´s Guide* .** Manning Publications Co. ISBN 978-1-933988-67-2

Reto Meier (2009). ***Android Application Development.,*** Wiley Publishing, Inc. ISBN: 978-0-470-34471-2

Shegn Liang (1999). ***Java Native Interface: Programmer's Guide and Specification***, Adisson Wesley. ISBN 0-201-32577-2

Webs

Android Developers Home Page http://developer.android.com/index.html

Mono for Android: http://xamarin.com/monoforandroid

Learn OpenGL ES: http://www.learnopengles.com/android-lesson-one-getting-started/

Tutoriales

Android Developer Resurces http://developer.android.com/resources/index.html

Videotutoriales sobre Android de Edu4Java http://www.youtube.com/user/edu4java

Edu4Java Home Page http://www.edu4java.com/android.html

Grupo google "desarrolladores-android":
http://groups.google.com/group/desarrolladores-android

Foros de consulta:

Android-Spa es la comunidad oficial de Android en Español
http://www.androidspa.com/

Foro Android: http://www.grupoandroid.com/index

ANEXO I
PREPARACIÓN DE ECLIPSE

En el siguiente anexo se describe la preparación del entorno Eclipse para desarrollar aplicaciones Android con código nativo.

1 INSTALACIÓN DE ANDROID SDK

El primer paso es descargar el Android SDK (del inglés Software Development Kit) de la página web de Android Developers:

http://developer.android.com/sdk/index.html

Una vez terminada la descarga, hay que ejecutar el archivo. Durante la instalación hay que señalar la carpeta en la que se quiere realizar la instalación del SDK. Debemos recordar el nombre de la carpeta, ya que más tarde lo necesitaremos.

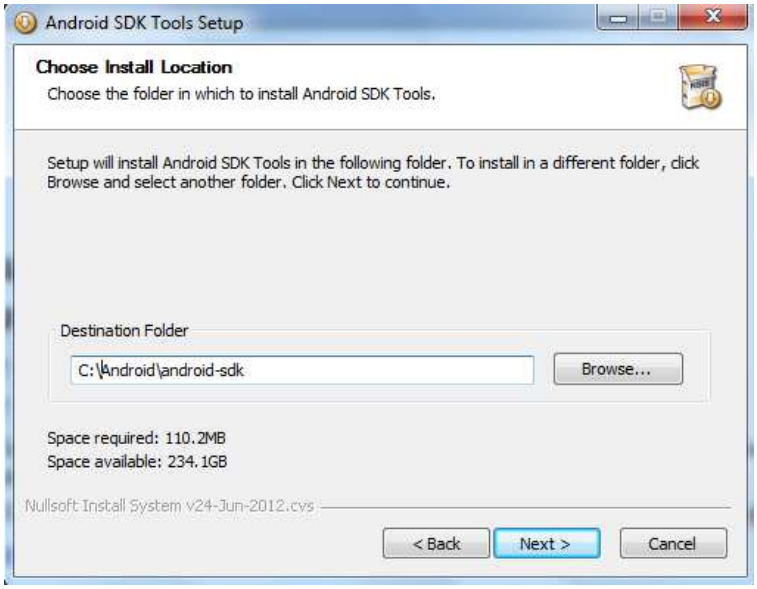

Ilustración 1: directorio de instalación del SDK

Al terminar la instalación de los archivos aparecerá una ventana con una opción para iniciar el SDK Manager al finalizar la instalación. La dejamos activada y pulsamos el botón finish tras lo que se ejecuta el Android SDK Manager.

El SDK Manager permite elegir que partes del Android SDK queremos instalar. Algunas opciones son la versión de la plataforma, herramientas, documentación o ejemplos. Al realizar la primera instalación es obligatorio instalar las siguientes opciones: SDK Tools, SDK Platform-tools y al menos una versión de la plataforma Android. En nuestro caso hemos instalado la versión 2.2 (API8) ya que es la que dispone el dispositivo que hemos utilizado en el proyecto.

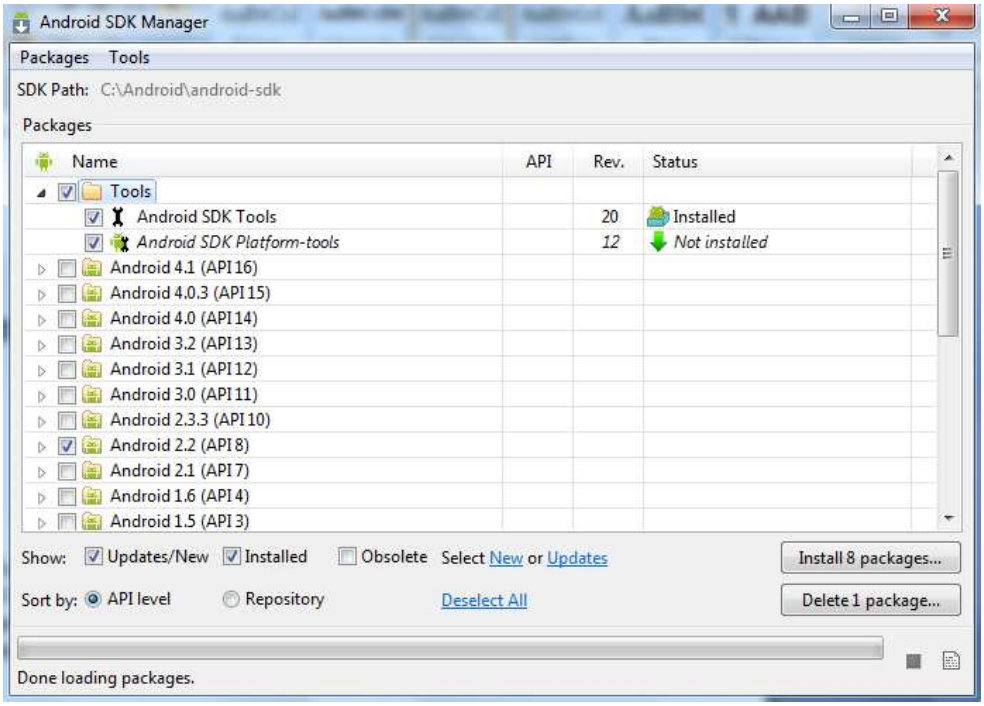

Ilustración 2: Android SDK Manager. Selección de paquetes

Después de seleccionar los paquetes que se deseen instalar pulsamos el botón con Install. Nos aparecerá una nueva ventana en la que deberemos aceptar las licencias de cada paquete. Después de un tiempo se habrán instalado los paquetes elegidos y podremos cerrar el SDK Manager.

2 ANDROID DEVELOPMENT TOOLS

El siguiente paso es instalar el plugin Android Development Tools (ADT) para Eclipse. Es un plugin diseñado para integrar el desarrollo de las aplicaciones Android en el entorno de programación Eclipse. Entre otras características permite la creación de proyectos Android, diseño y creación de interfaces de usuario e incluso la depuración de las aplicaciones.

La secuencia para instalar el ADT es la siguiente:

1. Abrir Eclipse
2. En el menú “Help” seleccionar la opción “Install new software…”
3. Pulsar Add para añadir un nuevo repositorio.
4. Después de escribir un nombre y una de las siguientes direcciones pulsar OK:
   https://dl-ssl.google.com/android/eclipse
   http://dl-ssl.google.com/android/eclipse
5. Seleccionar las herramientas que deseemos instalar y pulsar Next.

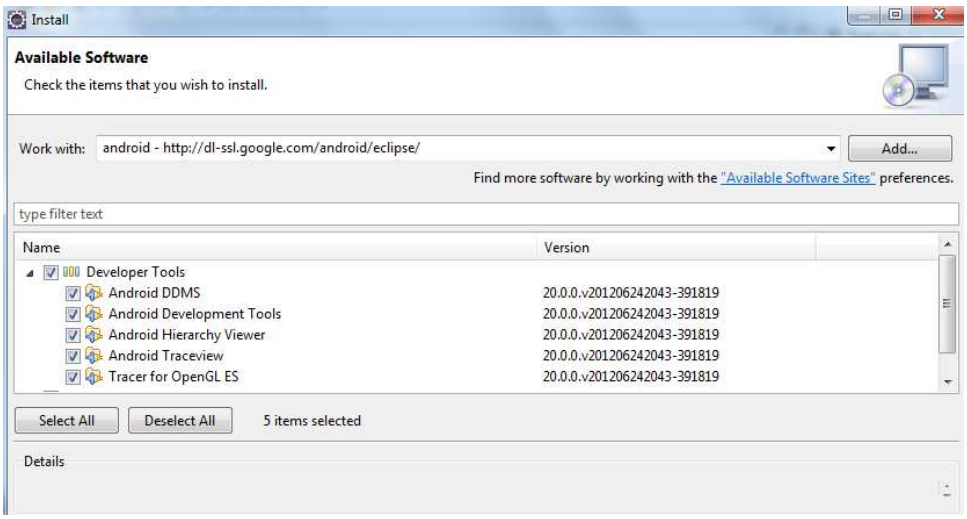

Ilustración 3: Instalación ADT

6. Después de confirmar las herramientas que se instalarán debemos aceptar los términos de la licencia para proceder a su instalación.
7. En caso de que surja algún aviso de seguridad pulsar en OK.
8. Reiniciar Eclipse.

Una vez que se ha instalado el ADT sólo nos queda configurarlo para indicarle el directorio en el que se encuentra instalado el SDK. Para ello basta con los siguientes pasos:

1. Elegir en el menú “Window” la opción “Preferences”
2. En el panel izquierdo de la ventana que surge seleccionar Android
3. Buscar la carpeta en la que hemos instalado el SDK. Si hemos seleccionado la carpeta correcta aparecerán las APIs Android que se encuentran en el directorio.

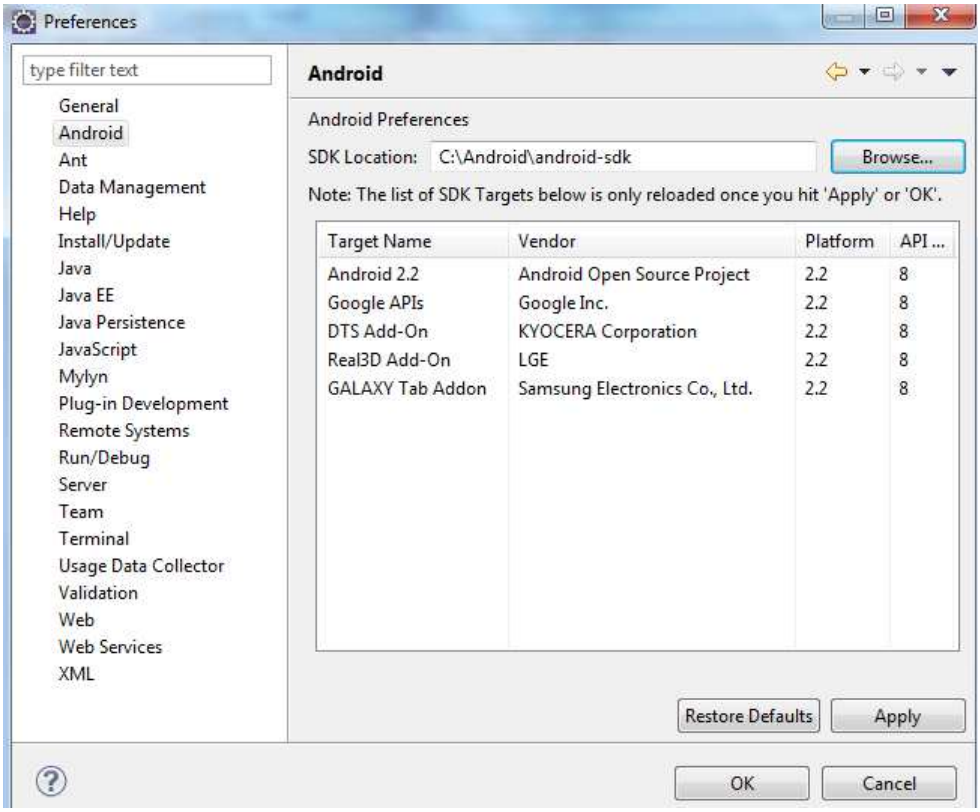

Ilustración 4: Configuración ADT.

3 ANDROID NATIVE DEVELOPMENT KIT

El Android Native Development Kit es el que nos permitirá compilar código C++ para poder utilizarlo en Android. Para instalarlo solo es necesario descargarlo, y una vez hecho esto descomprimirlo en el directorio que deseemos. También es recomendable añadir el directorio a la variable de entorno Path. Se puede descargar de la siguiente dirección: http://developer.android.com/tools/sdk/ndk/index.html

ANEXO 2
EJEMPLO NDK

En este anexo se va a indicar el proceso para crear una aplicación Android que utilice el NDK y JNI para cargar librerías con código C++. Para ello vamos a contar con uno de los ejemplos disponibles con la librería OpenCV

1 ESTRUCTURA DEL PROYECTO

Cuando usamos el NDK para cargar librerías con código nativo, la estructura es la misma que la de un proyecto Android normal. Aparecen dos nuevas carpetas que cabe destacar: jni y libs.

- En la carpeta jni se debe colocar los ficheros con las funciones C++ que deseamos invocar desde Java.
- En la carpeta libs es donde se instalan automáticamente las librerías una vez se han compilado.

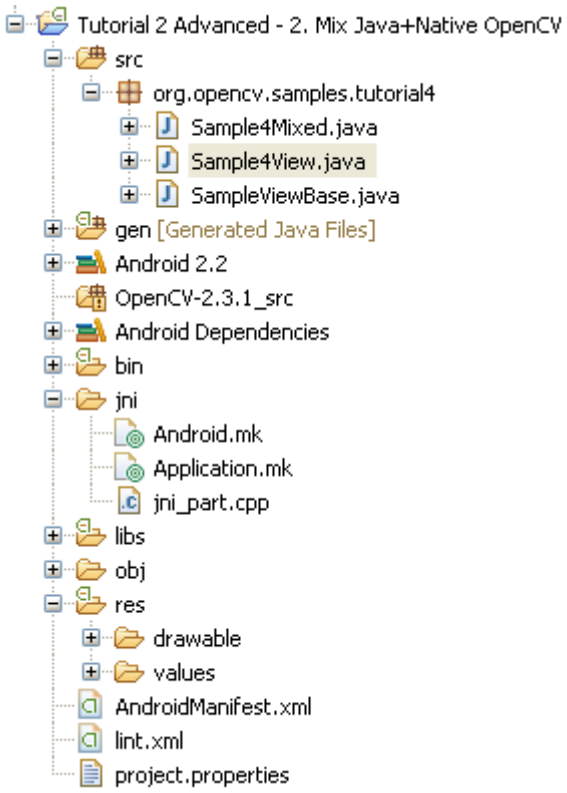

Ilustración 1: Estructura de un proyecto Android con NDK

2 FICHERO CON EL CÓDIGO C++

Debemos incluir los ficheros con el código C++. En este ejemplo tenemos un solo fichero con una única función. El código de este fichero debe seguir los convenios de JNI.

```
extern "C" {
JNIEXPORT void JNICALL
Java_org_opencv_samples_tutorial4_Sample4View_FindFeatures
    (JNIEnv* env, jobject thiz, jlong addrGray, jlong addrRgba)
{
    Mat* pMatGr=(Mat*)addrGray;
    Mat* pMatRgb=(Mat*)addrRgba;
    vector<KeyPoint> v;

    FastFeatureDetector detector(50);
    detector.detect(*pMatGr, v);
    for( size_t i = 0; i < v.size(); i++ )
        circle(*pMatRgb, Point(v[i].pt.x, v[i].pt.y), 10,
             Scalar(255,0,0,255));
}


}
```

Cabe destacar el nombre de la función. Es un nombre compuesto con las siguientes características:

- Java: todos los métodos deben empezar por esta palabra.
- org_opencv_examples_tutorial4: nombre del paquete del proyecto.
- Sample4View: nombre de la clase Java desde la que se invoca el método.
- FindFeatures: método que se implementa.

Los parámetros JNIEnv* env y jobject thiz, tienen que estar presentes cuando implementemos algun método en JNI.

- JNIEnv* env: es un apuntador a una tabla que contiene la interfaz hacia la máquina virtual. Incluye todas las funciones necesarias para interactuar con la maquina virtual de Java y para trabajar con los objetos Java.
- jobject thiz: actúa como una referencia this al objeto Java en el que se invoca el método.

3 APPLICATION.MK Y ANDROID.MK

Application.mk y Android.mk son los ficheros Makefile que controlan la construcción de la librería. Mientras que Application.mk indica los recursos nativos que se van a utilizar por la aplicación, el Android.mk sirve para describir qué ficheros son los que se quieren compilar.

A continuación vemos los archivos correspondientes al ejemplo y una pequeña descripción de su contenido.

Application.mk:

```
APP_STL := gnustl_static
APP_ABI := armeabi-v7a
APP_CPPFLAGS := -frtti –fexceptions
```

- APP_STL: indica la implementación de la librería estandar de C++ que usa la aplicación.
- APP_ABI: señala la arquitectura para la que se compilarán los archivos.
- APP_CPPFLAGS: muestra los flags que se van a pasar al compilador de C++.

Android.mk:

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

OPENCV_CAMERA_MODULES:=off
include ../includeOpenCV.mk
ifeq ("$(wildcard $(OPENCV_MK_PATH))","")
      #try to load OpenCV.mk from default install location
      include
$(TOOLCHAIN_PREBUILT_ROOT)/user/share/OpenCV/OpenCV.mk
else
      include $(OPENCV_MK_PATH)
endif

LOCAL_MODULE    := mixed_sample
LOCAL_SRC_FILES := jni_part.cpp
LOCAL_LDLIBS +=  -llog -ldl

include $(BUILD_SHARED_LIBRARY)
```

Aparte de las líneas de código que se encargan de cargar los módulos de las librerías de OpenCV, merecen una explicación algunas instrucciones:

- LOCAL_PATH: indica la localización de los archivos
- LOCAL_MODULE: nombre que le damos a la librería creada.
- LOCAL_SRC_FILES: ficheros con el código nativo que se van a compilar.
- LOCAL_LDLIBS: librerías auxiliars que utilicemos.

4 COMPILAR EL CÓDIGO NATIVO

Para compilar el código, debemos abrir una ventana de la consola y situarnos en el directorio donde se encuentra el proyecto y ejecutaremos el comando “ndk-build”. Una vez finalizada la compilación se instalara el archivo con la librería en la carpeta libs.

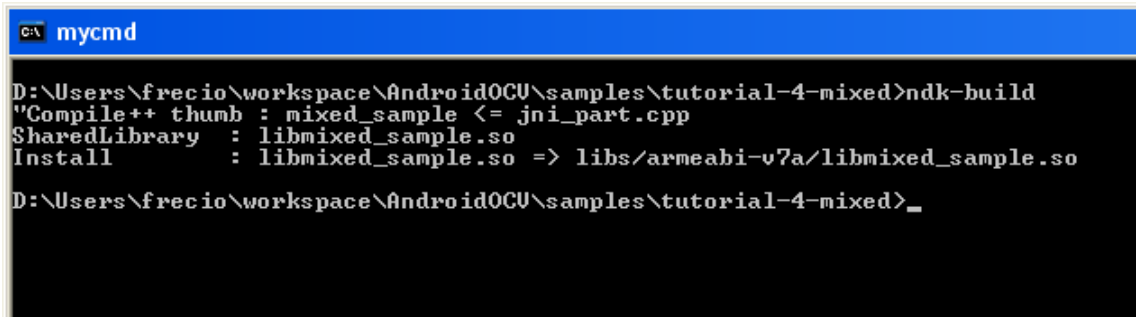

Ilustración 2: Compilación de la librería

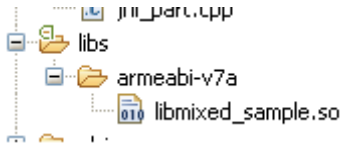

Ilustración 3: Localización de la librería creada

ANEXO 3
OPENGL ES 2.0 EN ANDROID

En este anexo se presentan los conceptos más importantes para el uso de OpenGL ES 2.0 en Android. Entre ellos veremos el uso de buffers para los datos de los vértices y los shaders con las operaciones que se aplican a cada pixel.

1 ANDROID

En este apartado se explican los cambios que hay que hacer en el Android Manifest para poder utilizar OpenGL desde Android y las principales clases implicadas.

Lo primero de todo es indicar al sistema la versión de OpenGL que se va a utilizar. Para ello hay que añadir al Android Manifest:

```
<uses-feature android:glEsVersion="0x00020000"
android:required="true" />
```

Para poder utilizar OpenGL ES se utilizan dos clases básicas para crear y manipular gráficos que se describen a continuación

GLSurfaceView: es una superficie especial de Android en la que se dibujan los gráficos. Hay que añadirle un renderer que se encargue del dibujado y decirle que cree un contexto de OpenGL ES 2.0 mediante las instrucciones:

```
//Indicar el renderer para la GLSurfaceView

setRenderer(new MyRenderer());

//Crear un context de OpenGL ES 2.0

setEGLContextClientVersion(2);
```

GLSurfaceView.Renderer: hay que implementar la interfaz Renderer que es el encargado de lo que se dibuja en la GLSurfaceView. Contiene los métodos para dibujar los modelos y debe implementar al menos los siguientes tres métodos:

```
//Se ejecuta al crearse la superficie. Para inicializar los objetos
//o para establecer el entorno de OpenGL
public void onSurfaceCreated(GL10 unused, EGLConfig config)


//Método que se llama cada vez que se dibuje la superficie
    public void onDrawFrame(GL10 unused

//Método que implementa lo que sucede al cambiar la geometría
// de la superficie
    public void onSurfaceChanged(GL10 unused, int width, int
height)
```

2 BUFFERS

Los datos de los vértices (coordenadas, color…) del modelo los guardamos en arrays. A La hora de pasárselos a OpenGL, lo haremos a través de buffers para maximizar la eficiencia. El orden de los bytes de estos buffers puede no ser el mismo en Java que en el dispositivo en el que se ejecute, debemos ordenar los buffers de acuerdo a este último.

```
        //Asignar al buffer un tamaño correspondiente al vector por
        //el número de bytes de una posición del buffer
        cbb = ByteBuffer.allocateDirect(vertexData.length * 4);

        //indicarle que use el orden de bytes que utilice el
        //dispositivo
        cbb.order(ByteOrder.nativeOrder());

        vertexBuffer = cbb.asFloatBuffer();
        vertexBuffer.put(vertexData).position(0);
```

3 SHADERS

Los shaders son pequeños programas que nos indican los rasgos de los vértices y de los pixels. Utilizamos dos clases de shaders: vertex shader y fragment shader.

- Vertex shader: permite aplicar las transformaciones sobre las propiedades de un vértice, ya sea color, la textura o sus coordenadas.
- Fragment shader: para cada píxel de la imagen calcula el color y otras propiedades.

En Android los shaders debemos definirlos en string.

```
private final String vertexShaderCode =
           "attribute vec4 vPosition; \n"
           + "attribute vec4 SrcColor; \n"
           + "attribute vec2 a_texCoord;   \n"
           + "varying vec4 DstColor; \n"
           + "varying vec2 v_texCoord; \n"
           + "void main(){              \n"
           + " gl_Position = vPosition; \n"
           + " DstColor = SrcColor; \n"
           + " v_texCoord = a_texCoord;  \n"
           + "}

private final String fragmentShaderCode =
           "precision mediump float;  \n"
           + "varying vec4 DstColor; \n"
           + "varying vec2 v_texCoord; \n"
           + "uniform sampler2D s_texture; \n"
           + "void main(){              \n"
           + "gl_FragColor = texture2D( s_texture,v_texCoord );\n"
           + "} \n";
```

Para cargar los shader creamos una función que se encargara de cargar el shader que se le pasa de parámetro. Como parámetro también hay que indicarle si se trata de un vertex o un fragment shader.

```
private int loadShader(int type, String shaderCode) {

    // crear un vertex shader (GLES20.GL_VERTEX_SHADER)
    // o un fragment shader (GLES20.GL_FRAGMENT_SHADER)
    int shader = GLES20.glCreateShader(type);

    // añadir el código de los shaders y compilarlos
    GLES20.glShaderSource(shader, shaderCode);
    GLES20.glCompileShader(shader);

    return shader;
    }
```

Tanto el vertex shader como el fragment shader debemos cargarlos en un “Program” que finalmente será compilado por OpenGL. Un Program es un objeto de OpenGL que contiene los shaders que luego se usarán para dibujar.

```
int vertexShader =
loadShader(GLES20.GL_VERTEX_SHADER,vertexShaderCode);
int fragmentShader =
loadShader(GLES20.GL_FRAGMENT_SHADER,fragmentShaderCode);

// crear un OpenGL Program vacío
mProgram = GLES20.glCreateProgram();

// añadir el vertex shader al Program
GLES20.glAttachShader(mProgram, vertexShader);

// añadir el fragment shader al Program
GLES20.glAttachShader(mProgram, fragmentShader);

// crea los programas que ejecuta OpenGL
GLES20.glLinkProgram(mProgram);
```

Este proceso de carga de shaders y compilación de los Program es bastante costoso por lo que deberá hacerse solo una vez, por ejemplo, al crear el modelo 3D que se quiere renderizar.