



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Aplicación para la obtención de un mapa
3D de una escena con un dispositivo
Android

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Ruoff, Ruben

Tutor: Atienza Vanacloig, Vicente Luis

2015-2016

Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

Resumen

Se ha desarrollado una aplicación para dispositivos Android que realiza la reconstrucción en 3D mediante la toma de imágenes usando la cámara incorporada. Se ha implementado la posibilidad de tomar las fotografías en el instante o cargarlas desde la galería del dispositivo. Una vez las imágenes han sido obtenidas, se calcula una nube de puntos a la cual se le puede aplicar un filtro para descartar puntos erróneos, procediendo posteriormente a una reconstrucción de la superficie, mostrando en cualquier momento el resultado por pantalla.

Palabras clave: Android, reconstrucción, 3D, opencv, cámara, opengl, pcl

Abstract

An application for Android devices has been developed, which performs a 3D reconstruction using images provided by the integrated camera. The photographs can be obtained by taking the pictures in the moment or loading them from the gallery. Once the images are loaded, a point cloud is calculated, which can be filtered to remove erroneous points. Afterwards a reconstruction of the surface is performed. The results are shown all the time on the screen of the device.

Keywords: Android, reconstruction, 3D, opencv, camera, opengl, pcl

Tabla de contenidos

1.	Introducción	9
1.1	Presentación del problema.....	9
1.2	Motivación personal	9
1.3	Objetivos del proyecto	9
1.4	Descripción de la estructura de la memoria.....	10
2.	Antecedentes	11
2.1	Herramientas empleadas	11
2.1.1	Eclipse ADT	11
2.1.2	Android	12
2.1.3	Android NDK	15
2.1.4	OpenCV	16
2.1.5	Point Cloud Library.....	16
2.1.6	OpenGL	17
2.2	Estado actual.....	17
3.	Análisis	18
3.1	Descripción de la solución	18
3.1.1	Conceptos de Structure from Motion.....	18
3.1.2	Estimación del movimiento de la cámara.....	19
3.1.3	Reconstrucción mediante dos imágenes.....	23
3.1.4	Reconstrucción mediante múltiples imágenes	25
3.1.5	Refinamiento de la reconstrucción	26
3.1.6	Reconstrucción de la superficie.....	27
3.2	Descripción del flujo de la aplicación	28
3.2.1	Módulo de entrada	29
3.2.2	Módulo SfM	30
3.2.3	Módulo OpenGL.....	32
3.4	Diagrama de clases	33
4.	Diseño	35
4.1	Plataforma software y hardware.....	35
4.2	Funcionamiento de la aplicación.....	36

4.2.1	Toma de fotografías	36
4.2.2	Algoritmos de Structure from Motion	37
4.2.3	Representación de los resultados	38
4.3	Metodología de desarrollo	40
4.4	Casos de uso	41
5.	Resultados	43
5.1	Ejecución de un ejemplo	43
5.2	Reconstrucción de la superficie con Meshlab	51
6.	Conclusiones	52
6.1	Problemas encontrados	52
6.2	Mejoras futuras	53
6.3	Conclusiones finales	53
Anexo I - Manual de usuario		55
Anexo II - Instalación y configuración		62
Bibliografía y recursos		78



Tabla de contenidos

Ilustración 1 - Logo Eclipse	11
Ilustración 2 - Logo Android	12
Ilustración 3 - Arquitectura Android.....	12
Ilustración 4 - Ciclo de vida de Android.....	14
Ilustración 5 - Esquema NDK.....	15
Ilustración 6 - Logo OpenCV	16
Ilustración 7 - Logo PCL.....	16
Ilustración 8 - Logo OpenGL ES	17
Ilustración 9 - Comparación entre las Pipelines de OpenGL.....	17
Ilustración 10 - Ejemplo de keypoints.....	20
Ilustración 11 - Resultado de aplicar el matching	21
Ilustración 12 - Matriz de la cámara.....	22
Ilustración 13 - Matriz de cámara inicial.....	23
Ilustración 14 - Triangulación a partir de dos imágenes. Fuente: [13]	24
Ilustración 15 - Reconstrucción incremental. Fuente: [13].....	25
Ilustración 16 - Bundle Adjustment Fuente: [13].....	26
Ilustración 17 - Aplicando el filtro estadístico	26
Ilustración 18 - Reconstrucción de una superficie mediante Poisson	27
Ilustración 19 - Diagrama de flujo de la aplicación simplificado	28
Ilustración 20 - Diagrama de flujo del módulo de entrada.....	29
Ilustración 21 - Diagrama de flujo del módulo SfM	31
Ilustración 22 - Diagrama de flujo del módulo OpenGL.....	32
Ilustración 23 - Diagrama de clases	34
Ilustración 24 - Metodología espiral	41
Ilustración 25 - Caso de uso Cámara	41
Ilustración 26 - Caso de uso galería	41
Ilustración 27 - Caso de uso nube de puntos filtrada	42
Ilustración 28 - Caso de uso nube de puntos	42
Ilustración 29 - Caso de uso superficie reconstruida	42
Ilustración 30 - Ejecución - Gal.....	43
Ilustración 31 - Ejecución - Selección múltiple	44
Ilustración 32 - Ejecución - Pantalla de carga.....	44
Ilustración 33 - Ejecución - Nube de puntos.....	45
Ilustración 34 - Ejecución - Ajustes filtro 1	46
Ilustración 35 - Ejecución - Nube filtrada 1	46
Ilustración 36 - Ejecución - Ajustes filtro 2.....	47
Ilustración 37 - Ejecución - Nube filtrada 2	47
Ilustración 38 - Ejecución - Ajustes superficie 1	48
Ilustración 39 - Ejecución - Reconstrucción 1.....	48
Ilustración 40 - Ejecución - Ajustes superficie 2.....	48
Ilustración 41 - Ejecución - Reconstrucción 2.....	49
Ilustración 42 - Ejecución - Ajustes superficie 3.....	49
Ilustración 43 - Ejecución - Reconstrucción 3-1	50
Ilustración 44 - Ejecución - Reconstrucción 3-2.....	50

Ilustración 47 - Reconstrucción MeshLab - 3	51
Ilustración 46 - Reconstrucción MeshLab - 2	51
Ilustración 45 - Reconstrucción MeshLab - 1.....	51

Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

1. Introducción

Hoy en día el sistema operativo más utilizado en el mercado de la telefonía móvil es Android. La gran mayoría de estos dispositivos incorporan diversos elementos como cámaras, micrófonos o giroscopios. Dado que casi cualquier persona tiene un dispositivo móvil como los descritos anteriormente, se ha considerado interesante desarrollar una aplicación para la reconstrucción de una escena en 3D mediante la toma de fotografías de dicha escena desde diferentes perspectivas utilizando la cámara de dichos dispositivos.

1.1 Presentación del problema

En la actualidad, las técnicas para la obtención de modelos 3D están en auge por las infinitas aplicaciones de estos. Tradicionalmente se han utilizado equipos especializados para esta tarea, resultando en un alto costo que no todo el mundo puede afrontar.

Para la reconstrucción mediante *Structure from Motion* (SfM) lo único que es necesario es una cámara de fotografía, la cual cualquier persona lleva encima en forma de teléfono inteligente. Por lo que el desarrollo de una aplicación para la plataforma Android pone a disposición de mucha gente la reconstrucción en 3D de una escena a un coste muy reducido.

1.2 Motivación personal

La motivación inicial fue la posibilidad de trabajar en el campo de la visión por computador ya que parece un tema muy interesante y en el grado no se ha tenido ninguna asignatura relacionada con ello. También era una buena posibilidad para consolidar conocimientos en programación para Android ya que se considera una habilidad muy útil y demandada en el mercado laboral.

Una vez realizada la primera reunión con el tutor y conociendo con mayor detalle la idea fueron surgiendo nuevas motivaciones, como trabajar con OpenCV, una de las librerías más utilizadas en este campo.

1.3 Objetivos del proyecto

El objetivo principal de este proyecto es el desarrollo de una aplicación Android que permite a los usuarios reconstruir una escena en 3D mediante la utilización de fotografías. Aparte de esta descripción general del proyecto se definieron una serie de

objetivos primordiales para que la aplicación fuera lo suficientemente funcional e interesante. Estos objetivos son:

- Ofrecer la posibilidad de que el usuario haga las fotografías mediante la misma aplicación o que pueda elegir utilizar imágenes ya guardadas en el dispositivo.
- Mostrar el resultado, el cual en este caso es representado mediante una nube de puntos en 3D, por pantalla en la misma aplicación.
- Permitir el refinamiento del resultado para mejorar el modelo obtenido.
- Posibilitar la reconstrucción de una superficie a partir de la nube de puntos.
- Poder guardar el resultado en el dispositivo.

1.4 Descripción de la estructura de la memoria

En este apartado se procede a explicar el contenido de cada uno de los principales bloques de la memoria.

El primer bloque consiste de cada uno de los apartados anteriores formando la introducción.

El segundo bloque se centra en los antecedentes del tema del proyecto y se describirán las herramientas empleadas para el desarrollo de la aplicación. Es un bloque muy importante ya que es fundamental tener claro cómo se quiere realizar el desarrollo de la aplicación. El trabajo de investigación llevado a cabo se expone en este bloque.

En el tercer bloque se detalla el análisis. Se describe detalladamente la solución implementada así como los algoritmos utilizados. Además se especifica el diagrama de clases en UML así como el diagrama de flujo.

En el cuarto bloque se especifica el diseño del programa. Se expone la metodología de desarrollo utilizada así como el funcionamiento de la aplicación. También se hace una puntualización en el diseño de la interfaz de usuario ya que es un elemento crítico en los dispositivos móviles.

En quinto bloque consiste en mostrar algún ejemplo completo del uso de la aplicación así como los resultados intermedios e finales.

En el último bloque se exponen las conclusiones acerca del desarrollo así como trabajos futuros posibles en esta área de trabajo.

Finalmente se adjuntan dos apéndices considerados necesarios para complementar el proyecto. El primero es el manual de usuario, el cual es considerado fundamental para explicar cómo se utiliza la aplicación y para explicar con más detalle todas las posibles opciones que son posibles ajustar para mejorar o refinar los resultados. El segundo apéndice explica el proceso para la instalación y configuración del entorno de desarrollo ya que se trata de un proceso bastante complejo y que ha requerido mucho tiempo y esfuerzo.

2. Antecedentes

Para la realización de este proyecto se han estudiado con antelación una serie de herramientas para su correcta utilización. Se ha intentado escoger aquellas con un uso simple, buen rendimiento y con más comunidad detrás. A continuación se expondrán las tecnologías escogidas así como el estado actual de estas en el mundo móvil.

2.1 Herramientas empleadas

Aquí se enumeran las herramientas más importantes utilizadas en el proyecto.

2.1.1 Eclipse ADT

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto



multiplataforma, también conocido como IDE (*Integrated development environment*). Eclipse fue desarrollado originalmente por IBM como sucesor de su familia de herramientas para VisualAge. Hoy en día es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto.

Ilustración 1 - Logo Eclipse

Para extender sus capacidades para funcionar bajo Android se desarrolló un complemento llamada ADT. Esta combinación de software fue durante años la solución oficial y recomendada para el desarrollo de software en Android. A día de hoy, el soporte oficial hacia eclipse por parte de Android ha terminado, y se utiliza una nueva plataforma llamada Android Studio. Por desgracia, este último aún no está completamente desarrollado y algunas funcionalidades aún se encuentran en estado beta, como por ejemplo, el soporte NDK. Unas líneas más abajo se explicará con más detalle en que consiste el NDK, pero por ahora es suficiente saber que en el proyecto se le da un uso intenso y por lo tanto Android Studio fue descartado por aún no tener este funcionamiento a la par con Eclipse.

Aunque Eclipse ya no sea soportado oficialmente, aún posee muchas funciones excelentes para permitir el desarrollo de proyectos en Android. Entre ellos destacan la fácil inclusión de librerías, el potente depurador o su soporte para la utilización del NDK.

2.1.2 Android

Android es un sistema operativo basado en el núcleo Unix. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes, tablets o tabléfonos; y también para relojes inteligentes, televisores y automóviles. Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde compró.



Ilustración 2 - Logo Android

Una de sus más grandes ventajas es que es un sistema libre y multiplataforma. Android utiliza el lenguaje de programación Java, pero en vez de utilizar las máquinas virtuales de Java utiliza una específicamente desarrollada para este sistema operativo llamada Dalvik. La cual utiliza un *bytecode* diferente, optimizado especialmente para funcionar con muy poca memoria, por lo que es ideal para dispositivos móviles.

Arquitectura Android

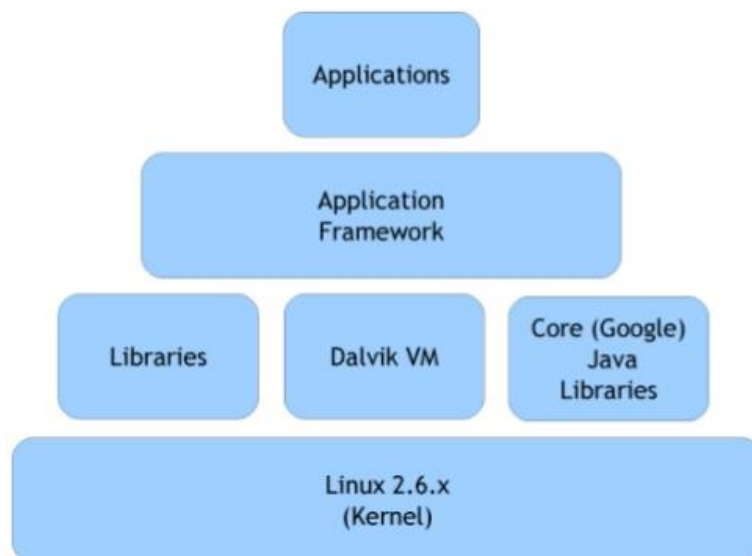


Ilustración 3 - Arquitectura Android

Cabe destacar que el sistema Android está en una constante evolución, lanzándose nuevas versiones cada año donde muchas veces se cambian completamente algunas funciones dificultando el soporte para todas las versiones. Por lo que probablemente el código desarrollado en este proyecto quedará obsoleto en poco tiempo.

Para el desarrollo de aplicaciones se utiliza el SDK de Android, el cual incluye un conjunto de herramientas de desarrollo, un simulador de teléfono y documentación. El SDK viene incluido en el complemento ADT de Eclipse descrito anteriormente.

Una aplicación desarrollada para Android está normalmente compuesta por actividades. El concepto de actividad en Android representa una unidad de interacción con el usuario, es lo que coloquialmente se llama pantalla de la aplicación. Cada actividad ha de tener una vista asociada, que será utilizada como interfaz de usuario. Son las actividades las que realmente controlan el ciclo de vida de la aplicación, dado que el usuario no cambia de aplicación, sino de actividad. El sistema mantiene una pila con las actividades previamente visualizadas, de forma que el usuario puede regresar a la actividad anterior pulsando la tecla “retorno”.

Una característica importante, y poco usual, es que la destrucción de un proceso no es controlada directamente por la aplicación, sino que es el sistema el que determina cuándo destruir el proceso. Lo hace basándose en el conocimiento que tiene de las partes de la aplicación que están en ejecución, en la importancia de dichas partes para el usuario y en cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esa aplicación. En estos casos, será responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sean reiniciadas conserven su estado.

Una actividad en Android puede estar en unos de estos cuatro estados:

Activa (*Running*): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.

Visible (*Paused*): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a actividad otra actividad con alguna parte transparente o que no ocupa toda la pantalla.

Parada (*Stopped*): Cuando la actividad no es visible.

Destruída (*Destroyed*): Cuando la actividad termina al invocarse el método *finish()*, o es destruida por el sistema.



Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación se muestra un esquema que ilustra los métodos que capturan estos eventos.

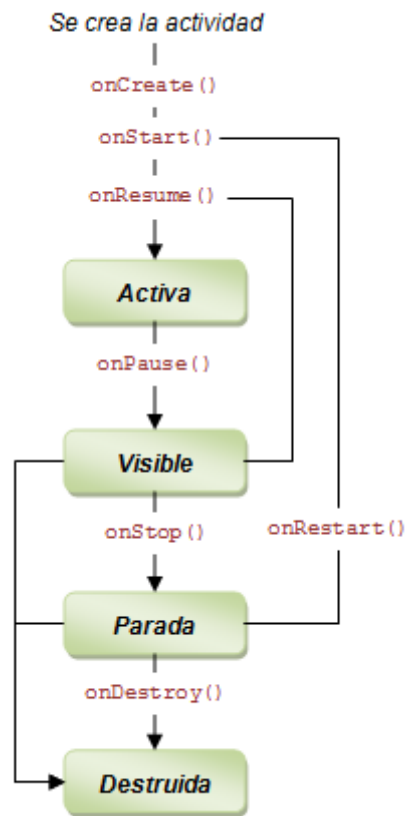


Ilustración 4 - Ciclo de vida de Android

onCreate(Bundle): Se llama en la creación de la actividad.

onStart(): Indica que la actividad está a punto de ser mostrada al usuario.

onResume(): Se llama cuando la actividad va a comenzar a interactuar con el usuario.

onPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada.

onStop(): La actividad ya no va a ser visible para el usuario. Si hay poca memoria, cabe la posibilidad de que la actividad sea destruida.

onRestart(): Indica que la actividad va a volver a ser representada después de haber pasado por onStop().

onDestroy(): Se llama antes de que la actividad sea totalmente destruida. Si existe una falta de memoria es posible que la actividad sea destruida sin llamar este método.

2.1.3 Android NDK

El Android NDK (*Native Development Kit*) permite a los desarrolladores reutilizar código escrito en C/C++ introduciéndolo a las aplicación a través de JNI (*Java Native Interface*). El NDK hace que la ejecución de las aplicaciones sea en cierto modo más rápida, ya que pasará a ejecutarse directamente en el procesador y no es interpretado por una máquina virtual.

El NDK puede ser útil y beneficioso para ciertos tipos de aplicaciones, pero también trae consigo una desventaja bastante importante: la complejidad. Las aplicaciones que suelen utilizar el NDK son aquellas que harán un uso intenso de la CPU o quieran utilizar librerías escritas en C/C++. Como en este proyecto se hace un uso intenso de la CPU mediante los algoritmos de SfM, además de utilizar librerías escritas en C++, no cabe duda que es una buena decisión emplear el NDK.

Una de las dificultades que se han presentado en el desarrollo de este proyecto ha sido hacer funcionar correctamente el NDK, en el Anexo II se describe el proceso detalladamente.

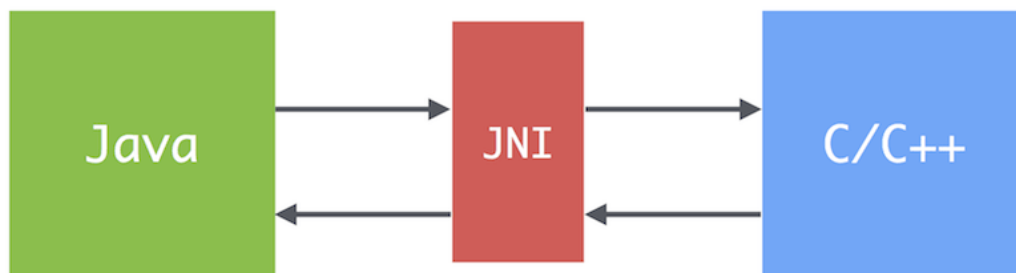


Ilustración 5 - Esquema NDK

Java Native Interface (JNI) es un mecanismo que nos permite interactuar con aplicaciones nativas desde un programa escrito en Java. Las aplicaciones nativas son bibliotecas o funciones escritas en lenguajes como C, C++ y ensamblador para un sistema operativo donde se ejecuta la JVM. La interfaz es bidireccional, permite la comunicación de Java con las aplicaciones nativas y viceversa.

Cuando Java llama a funciones implementadas en código nativo se habla de llamar a métodos nativos. Los métodos nativos se implementan por separado en archivos .c o .cpp, por los cuales se generará una biblioteca de enlace dinámico.

Una biblioteca de enlace dinámico es un fichero con código ejecutable el cual se carga en tiempo de ejecución bajo demanda de un programa por parte del SO. Aportan una gran ventaja con respecto al aprovechamiento de la memoria del sistema, dado que gran parte del código ejecutable puede encontrarse en bibliotecas, pudiendo así ser compartido entre distintas aplicaciones.

Para conseguir dicha librería dinámica se utiliza un archivo llamado `Android.mk` donde al igual que en los archivos *makefile* de Unix se especifica la manera de compilar los

archivos nativos. La librería es compilada con una herramienta llamada JAVAH proporcionada en el SDK de Android. El resultado de dicha compilación es un archivo de cabecera que contiene la declaración de las funciones nativas que se van a utilizar. Por último se compila todos los ficheros mediante la herramienta *ndk-build* proporcionada en el NDK de Android. Como ya dicho anteriormente, en el ANEXO II se encuentra una descripción detallada de este proceso.

2.1.4 OpenCV

OpenCV es una biblioteca libre de visión artificial originalmente desarrollada por Intel para proporcionar un entorno de desarrollo fácil y altamente eficiente. Desde que apareció su primera versión, se ha utilizado en infinidad de aplicaciones. Desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación.



Ilustración 6 - Logo OpenCV

Existen diferentes versiones para lenguajes como Java o Python. Aunque existe una versión para Android llamada OpenCV4Android se ha tomado la decisión de utilizar la librería nativa escrita en C++ para obtener un resultado lo más eficiente posible, además facilitar la reutilización de código. Como ya se ha explicado en el apartado anterior, se utiliza el NDK para que esto sea posible.

En el proyecto se utiliza OpenCV para analizar las fotografías y a partir de ellas obtener los puntos característicos y calcular la nube de puntos que representa el modelo 3D.

2.1.5 Point Cloud Library

Point Cloud Library (PCL) es una librería libre para el procesamiento de imágenes y nubes de puntos. También cuenta con una licencia BSD la cual le permite ser usado libremente para uso comercial o de investigación. Es una librería escrita en C++ la cual en este proyecto se utiliza para aplicar un filtro estadístico a la nube de puntos resultante proporcionada por OpenCV. También es la encargada de aplicar las funciones para llevar a cabo la reconstrucción de la superficie del modelo 3D mediante el algoritmo *Poisson*.



Ilustración 7 - Logo PCL

2.1.6 OpenGL

OpenGL (*Open Graphics Library*) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que



produzcan gráficos 2D y 3D. **Ilustración 8 - Logo OpenGL ES**

OpenGL ES (*OpenGL for Embedded Systems*) es una variante simplificada diseñada para dispositivos integrados tales como teléfonos móviles, PDAs y consolas de videojuegos. Es desarrollada por Khronos, un consorcio entre varias empresas de hardware y software gráfico.

A partir de la versión 2 de esta especificación se deja atrás la programación “fija” dando el salto al uso de *Shaders*, lo cual minimiza el coste de los subsistemas gráficos programables avanzados.

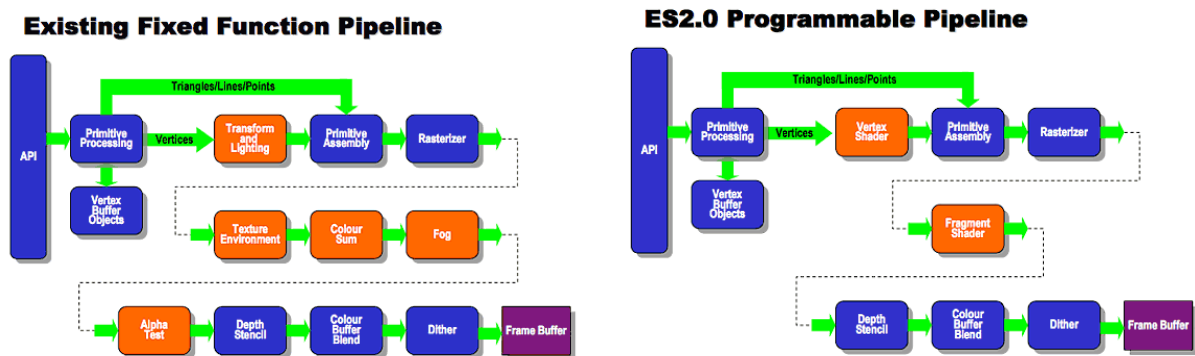


Ilustración 9 - Comparación entre las Pipelines de OpenGL

En este proyecto se emplea para mostrar al usuario la nube de puntos resultante del módulo SfM, así como el modelo 3D final creado con la reconstrucción de la superficie.

2.2 Estado actual

A día de hoy no existen muchas aplicaciones o programas que realizan la tarea que se quiere llevar a cabo con este proyecto. Para el entorno de ordenadores de escritorio la más conocida es VisualSfM. Si se expande la búsqueda a las plataformas móviles, se encuentra apenas una aplicación llamada 123D Catch, creada por Adobe.

Esta aplicación permite tomar fotos a un modelo, las cuales posteriormente son enviadas a un servidor donde se realiza el proceso de reconstrucción en 3D y envía el resultado de vuelta al dispositivo. Esta aproximación conlleva una serie de ventajas e inconvenientes. Por un lado se puede implementar con más libertad el proceso de

reconstrucción al no estar limitado al sistema operativo Android. Por otra parte, el realizar el cálculo en un servidor ajeno, hace necesario que el usuario disponga de conexión a internet en todo momento con el dispositivo. También puede ser problemático en cuanto a consumo de datos que conlleva esto, ya que las imágenes a subir a internet pueden llegar a ser de gran tamaño.

Por tanto se ha considerado apropiado realizar todo el proceso en el mismo dispositivo para ofrecer una alternativa a la aplicación anteriormente mencionada.

3. Análisis

En este apartado se pretende dar diferentes respuestas a los aspectos planificados previamente a su implementación. Es un apartado muy importante ya que un buen análisis puede ahorrar mucho tiempo de desarrollo.

3.1 Descripción de la solución

A continuación se pretende contestar la pregunta ¿Qué hace la aplicación? Para contestar esto, hace falta explicar los conceptos de SfM así como los algoritmos que están presentes en el proceso. Se asume que la cámara que ha tomado las imágenes es siempre la misma y, por lo menos al principio de la explicación del proceso, que se usan solamente dos imágenes.

Para el método utilizado en este trabajo se ha seguido el capítulo 4 de [13] el cual se apoya en el trabajo de Hartley y Zisserman expuesto en los capítulos 9 a 12 de su libro [12].

3.1.1 Conceptos de Structure from Motion

Primeramente cabe destacar la diferencia entre reconstrucción estéreo usando equipos calibrados y SfM. Mientras que en un equipo estéreo ya se sabe la distancia que hay entre las diferentes cámaras, en SfM no se sabe y hay que determinarla. Por lo que las reconstrucciones con equipos estéreo son mucho más precisas al no tener error en la estimación de la distancia y la rotación entre las cámaras. El primer paso en implementar un sistema de SfM es encontrar este desplazamiento entre las vistas.

Si se asume que ya sabemos la distancia y la rotación, el siguiente paso es reconstruir la geometría de la escena. En visión por computador a esto se le llama triangulación, y hay muchas formas de hacerlo. Desgraciadamente la versión actual de OpenCV no contiene una API para realizar la triangulación, por lo que hay que implementarla a mano.

Después de obtener la geometría 3D de dos imágenes hay que incorporar más vistas en el modelo para conseguir una reconstrucción más densa y precisa. En este punto, se suele hacer una optimización de las posiciones de las cámaras y los puntos 3D mediante un proceso llamado *bundle adjustment (BA)*. A pesar de que OpenCV contiene un

método de BA en el módulo de *Image Stitching*, éste solamente contempla la rotación de la vista y no la traslación, por lo que no sirve en este proyecto. Desgraciadamente no se ha encontrado un módulo de BA que funcione en Android y la aplicación se queda sin su implementación. En el apartado posterior de problemas encontrados se expone con más detalle dicho caso.

3.1.2 Estimación del movimiento de la cámara

Antes que nada, viene bien analizar los datos de entrada y las herramientas que se disponen. Se tiene a dos imágenes de una misma escena en una posición no muy diferente. En cuanto a herramientas, existen dos objetos matemáticos que imponen una serie de restricciones sobre las imágenes, las cámaras y la escena. Estos objetos son la matriz fundamental y la matriz esencial. Son muy parecidas entre sí y lo único que los diferencia es que la matriz esencial asume una cámara calibrada, y como este es el caso, se usa esta última. Cabe destacar que se utiliza una calibración de la cámara estimada, la cual se refina en el proceso BA. OpenCV solamente nos permite calcular la matriz fundamental con la función *findFundamentalMat*, pero es sencillo obtener la esencial a partir de la fundamental:

$$E = K' * F * K$$

Donde K es la matriz de la cámara calibrada, F la matriz fundamental y E la matriz esencial.

La matriz esencial es una matriz de 3x3, imponiendo una restricción entre un punto de una imagen y un punto de la otra imagen con $x'Ex = 0$, donde x es un punto en la primera imagen y x' es el punto correspondiente en la segunda imagen.

La matriz de calibración suele definirse de la siguiente manera:

$$K = \begin{bmatrix} fm_x & 0 & c_x \\ 0 & fm_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Donde fm_x y fm_y son la longitud focal en píxeles y c_x y c_y son los puntos principales en el centro de la imagen. Estos valores son desconocidos por lo que se bosquejan unos valores aproximados que posteriormente se refinan mediante el BA.

Por tanto, si se obtiene la matriz esencial se puede saber la posición de las cámaras y hacia dónde están mirando. Si se tienen suficientes de estas ecuaciones de restricciones se puede calcular dicha matriz con relativa facilidad. En teoría con siete de estas ecuaciones ya se puede obtener la matriz esencial, pero cuantas más se utilicen, más robusto será el resultado.



3.1.2.1 Feature Descriptors

A continuación se explica cómo se obtienen las ecuaciones para poder calcular la matriz esencial. Afortunadamente esto se puede hacer de manera bastante sencilla gracias al extenso sistema de *feature-matching* incorporado en OpenCV.

La extracción de *features* (elementos característicos o singulares) y el *matching* (establecimiento de correspondencias) de descriptores son procesos esenciales en el campo de la visión por computador. En esencia, extracción de *features* se refiere a la selección de puntos de una imagen que hacen un buen *feature*, así como el cálculo de un descriptor para cada punto seleccionado. Un descriptor es un vector de números que describe el entorno de un punto en una imagen. El *matching* es el proceso de encontrar un *feature* de una imagen en otra comparando su descriptor.

En la siguiente imagen se muestra un ejemplo de los *features* seleccionados por el algoritmo.

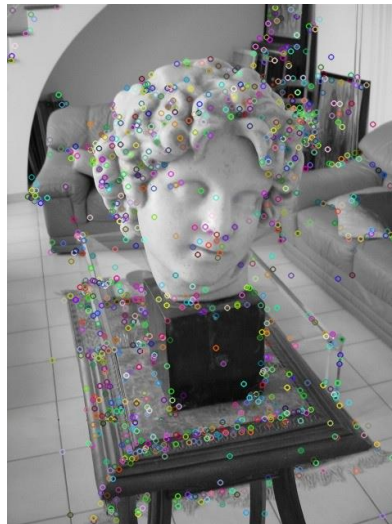


Ilustración 10 - Ejemplo de selección de *features*

Si se aplica el *matching* directamente probablemente se obtienen muchos *matches* erróneos. Para remediar esto, se aplican una serie de filtros por los que se pasan los *matches* y mediante los cuales los erróneos son eliminados.

Un filtro, el cual está implementado en OpenCV, es el *cross-checking* o prueba de bidireccionalidad. Consiste en que un *match* es considerado válido si un *feature* de la primera imagen es seleccionado con uno de la segunda, y el *matching* reverso también selecciona el mismo *feature* de la segunda imagen con el de la primera. Otro filtro que se emplea en este proyecto es aplicar el hecho de que las dos imágenes son de la misma escena y tienen un cierto grado de relación estéreo. Esto se consigue calculando la matriz fundamental, la cual se explicará más detalladamente en futuras secciones, y retiene las parejas de *features* que corresponden a este cálculo con un error pequeño.

Una vez realizado el *matching* y aplicados los filtros se obtiene lo siguiente:



Ilustración 11 - Resultado de aplicar el *matching* y el filtrado

3.1.2.2 Optical Flow

Una alternativa al proceso de *feature-matching* es el *optical flow*. Este método es parecido al anterior pero se diferencia en que para cada punto o píxel en la imagen A, se selecciona un pequeño área alrededor de dicho punto y se compara con la misma región en la imagen B. Esta comparación utiliza la técnica llamada consistencia de brillo, la cual se basa en que estas pequeñas áreas no se diferenciarán mucho entre ambas imágenes y por tanto la resta del brillo de ambas zonas tiende a ser cercana a 0.

Esta técnica presenta varias desventajas como por ejemplo el coste computacional, ya que se realiza un cálculo para cada píxel de cada imagen. Como este proyecto se centra en un dispositivo móvil, se ha optado por centrarse en el *feature-matching*.

3.1.2.3 Estimación de las matrices de la cámara

Una vez obtenidos los pares de *keypoints* o *features*, se procede al cálculo de las matrices de las cámaras. Antes que nada, viene bien examinar la estructura de la matriz que se va a usar.

$$P = [R|t] = \begin{bmatrix} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{bmatrix}$$

Ilustración 12 - Matriz de la cámara

Como se puede observar en la ilustración anterior, el modelo de cámara que se utiliza está compuesto por dos componentes, la rotación y la traslación. Determina una ecuación esencial: $x = PX$, donde x es un punto 2D en la imagen y X es el punto 3D en el espacio. Por lo tanto esta matriz proporciona una relación muy importante entre los puntos de la imagen y los puntos de la escena.

Después de calcular la matriz fundamental y la matriz esencial, el objetivo es obtener los componentes de rotación y traslación a partir de estas matrices. Existen varias formas de realizar esto, pero en este proyecto se ha utilizado la descomposición en valores singulares o SVD, proporcionada por OpenCV. La rotación se obtiene con la siguiente ecuación:

$$R = svd.u * W * svd.vt$$

Donde W es una matriz especial

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Mientras que la traslación es la segunda columna de la matriz resultante U del SVD.

$$t = svd.u.col(2)$$

Después de este proceso se obtiene una de las dos matrices de cámara, para conseguir la segunda matriz hay que asumir que una de las cámaras es fija y canónica (no tiene rotación ni traslación), la siguiente matriz representa a esta cámara:

$$P_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Ilustración 13 - Matriz de cámara inicial

Por lo tanto se puede decir que la matriz de cámara que se ha recuperado a partir de la matriz esencial se ha movido en relación a la fija.

Desafortunadamente esto no es la solución completa, ya que al realizar la descomposición de la rotación y traslación se obtiene cuatro resultados posibles, y solamente una es la correcta. La matriz correcta será la que produce los puntos reconstruidos con un valor en la Z positivo (puntos que se encuentran enfrente de la cámara). La obtención de la matriz correcta se calculará con el proceso de triangulación explicado más adelante.

Es recomendable realizar una comprobación de que la matriz fundamental sea correcta. Muchas veces el cálculo de esta matriz es erróneo y si se da el caso es inútil realizar la triangulación con una matriz errónea. Se puede comprobar si la matriz de rotación es realmente una matriz de rotación calculando el determinante, ya que siempre debería tener un determinante de 1 ó -1.

3.1.3 Reconstrucción mediante dos imágenes

En esta sección se explica cómo utilizar los datos obtenidos anteriormente para obtener una reconstrucción de la escena. Para conseguir esto, se va a emplear un método lineal de triangulación expuesto en [23].

Como se ha visto anteriormente, para cada punto 2D escogido por el algoritmo de *feature-matching* (x) y de las matrices de cámara (P), se tiene dos ecuaciones $x = PX$ y $x' = P'X$, donde x y x' son los *keypoints* que han hecho *matching* y X es el punto 3D representado en ambas imágenes. Reescribiendo estas ecuaciones se puede formular un



Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

sistema lineal el cual puede ser resuelto para el valor de X . Asumiendo que $X = (x, y, z, 1)t$ se crea un sistema lineal no homogéneo con la forma $AX = B$.

Resolviendo este sistema se obtiene una aproximación de los puntos 3D a partir de los puntos 2D. En la siguiente imagen se muestra el resultado de realizar la triangulación a partir de dos imágenes.

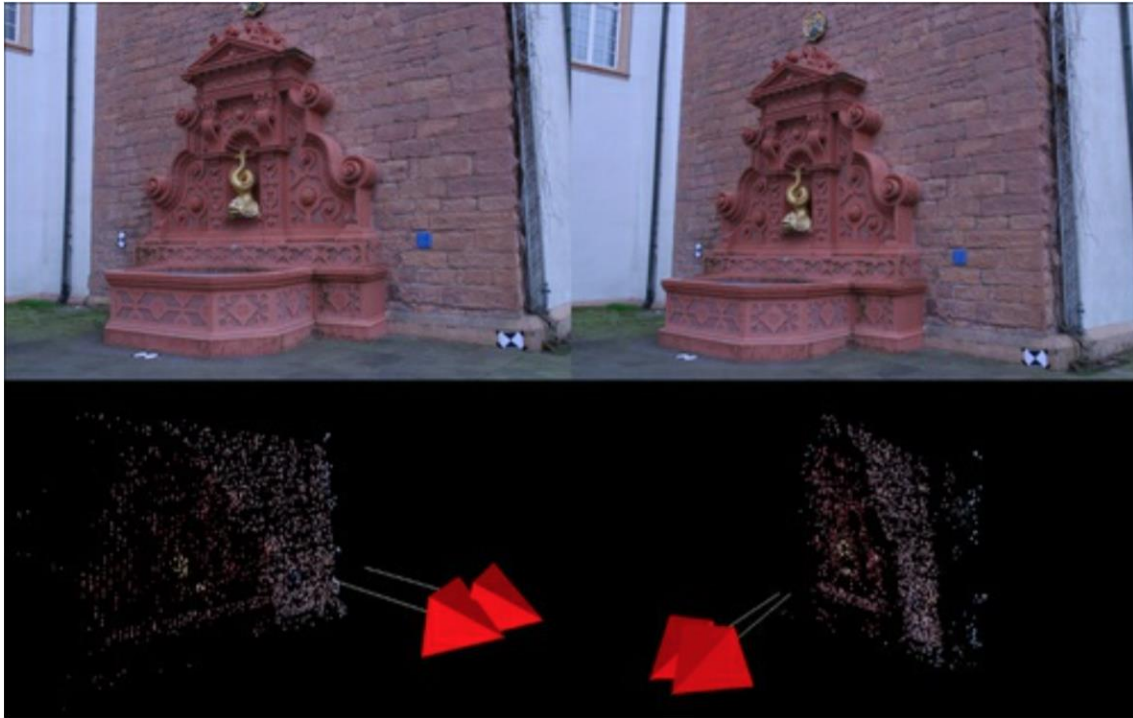


Ilustración 14 - Triangulación a partir de dos imágenes. Fuente: [13]

Cabe destacar un problema que se presenta y que hay que tener en cuenta si se quiere utilizar más de dos imágenes para la reconstrucción. Este problema es la escala de la reconstrucción. El movimiento que se ha obtenido entre las dos vistas tiene una escala arbitraria, no es ninguna unidad como centímetros o milímetros, es simplemente una unidad de escala.

También hay que realizar una manera de medir el error de la reconstrucción para poder obtener mejores resultados. Primero hay que definir el significado del término *reproyección*. La *reproyección* es simplemente escoger un punto 3D y volver a proyectarlo en una imagen, a partir de ahí se calcula la distancia de dónde está ahora el punto reproyectado y donde estaba originalmente. Si la distancia es grande significa que el error es grande y el punto triangulado es erróneo y no se quiere incluir en el resultado. La medida global será la media de las distancias de reproyección y nos indica el rendimiento de la triangulación en general. Si la media global es alta, puede deberse a problemas con las matrices de cámara, y por tanto con el cálculo de la matriz esencial o los *features* encontrados.

Ahora que se ha explicado la triangulación, ya se pueden poner a prueba las matrices obtenidas en el apartado anterior y seleccionar la matriz que más valores positivos tenga en los valores Z de los puntos y desechar las demás matrices.

3.1.4 Reconstrucción mediante múltiples imágenes

Puede parecer trivial añadir más vistas a la reconstrucción, pero no lo es, ya que solamente se puede conseguir una reconstrucción con la misma escala en todas las vistas, y como se ha expuesto anteriormente, esto no es así.

Existen varias formas de conseguir que la escala sea la misma en todas las vistas. Una es la estimación de la pose de la cámara, también conocido como *Perspective N-Point* (PNP), donde se intenta estimar la posición de una cámara mediante los puntos de la escena ya encontrados. Otro método es encontrar más puntos y ver cómo encajan en la geometría actual de la escena. Dado que en OpenCV se dispone de una función para solucionar el caso PNP llamada *solvePnP*, se ha decidido tomar esta aproximación.

El primer paso en la reconstrucción mediante múltiples imágenes en este proyecto consiste en encontrar una estructura base de la escena. Para esto se comprueban todas las imágenes entre ellas para encontrar el par que más puntos triangulados obtenga. A partir de esta base, se van añadiendo más vistas hasta que estén todas en el modelo.

A continuación se muestra una reconstrucción incremental.

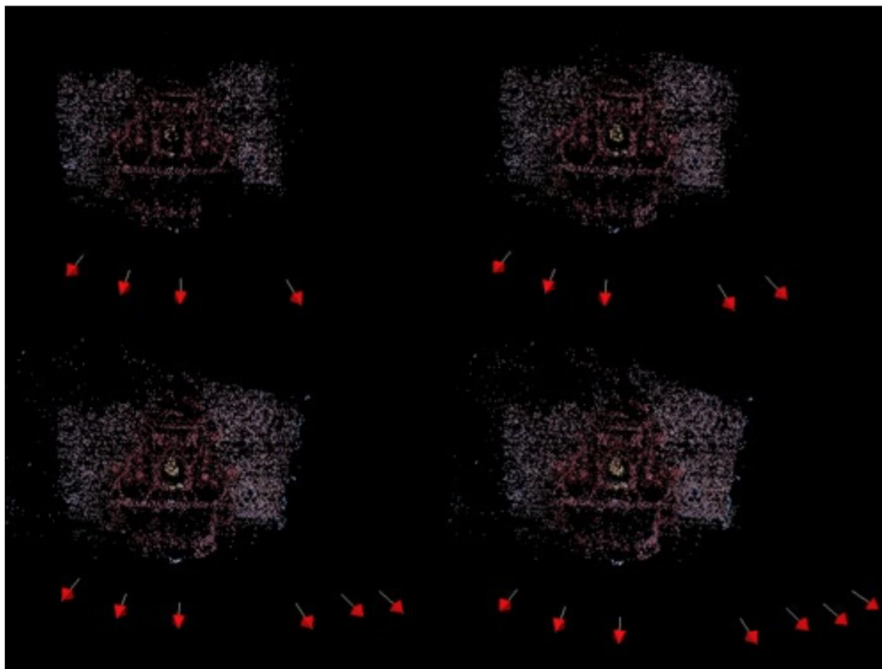


Ilustración 15 - Reconstrucción incremental. Fuente: [13]

3.1.5 Refinamiento de la reconstrucción

Aunque en este proyecto no está incluido el paso de *Bundle Adjustment* (BA) sí que merece la pena dar una pequeña explicación de lo que trata. El BA es un paso de optimización el cual consiste en modelar todos los datos obtenidos de los pasos anteriores en un único modelo. Las posiciones de los puntos así como las posiciones de las cámaras son optimizadas minimizando el error de reproyección. Este proceso suele basarse en solucionar enormes ecuaciones con miles de parámetros, por lo que suele ser un proceso con un alto coste de tiempo computacional. En la siguiente ilustración se muestra el efecto de aplicar la optimización del BA.



Ilustración 16 - *Bundle Adjustment* Fuente: [13]

A pesar de los filtros implementados, hay bastantes puntos erróneos que hacen necesario aplicar algún método para mejorar el resultado. Por lo que se ha aplicado un filtro estadístico presente en la librería PCL. Dicho filtro se basa en la computación de la distribución de la distancia entre puntos y sus vecinos. Asumiendo que la distribución resultante es de carácter gaussiana con media y derivación estándar, todos los puntos cuya distancia media este fuera de un intervalo establecido por la media y desviación estándar global son desechados, resultado en un resultado mucho más refinado. A continuación se ilustra la aplicación de dicho filtro.

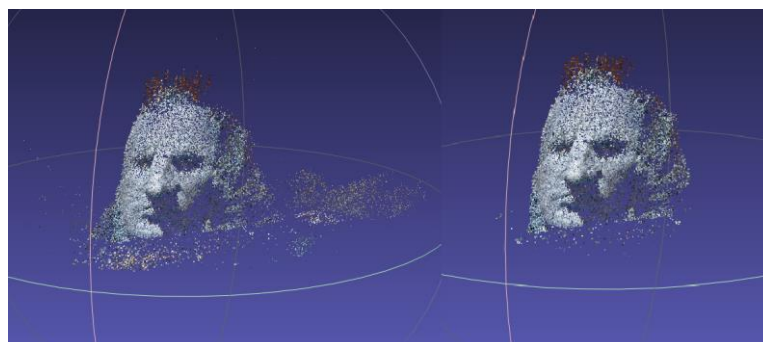


Ilustración 17 - Aplicando el filtro estadístico

3.1.6 Reconstrucción de la superficie

Para este último paso de la solución, se emplea la librería PCL la cual ofrece una implementación del algoritmo de reconstrucción de superficies *Poisson*.

Este algoritmo utiliza una nube de puntos junto a una estimación de la normal de la superficie local de cada punto. Se reconstruye la función implícita f con valor 0 en los puntos P_i y donde el gradiente en este punto P_i es igual al vector normal N_i . Por lo que el par P_i y N_i , es una muestra del campo de vector continuo V . La función f se obtiene calculando la integral del campo de vector V . Dado que no todo campo de vector es el gradiente de una función, el problema puede no tener solución: el requerimiento y condición para que un vector de campo sea un gradiente de una función f , es que el rotacional de V tiene que ser cero. En caso de que esta implicación sea imposible de imponer, es posible realizar la técnica de los mínimos cuadrados para minimizar la diferencia entre V y el gradiente de f .

A continuación se muestra una ilustración después de realizar la reconstrucción de la superficie aplicada a una nube de puntos filtrada:



Ilustración 18 - Reconstrucción de una superficie mediante *Poisson*

3.2 Descripción del flujo de la aplicación

En este apartado se expone como fluye la información a través de la aplicación. El programa está dividido en tres grandes bloques, los cuales serán explicados por separado. A continuación se ilustra el diagrama de flujo general de la aplicación, exponiendo cada uno de los módulos en más detalle posteriormente.

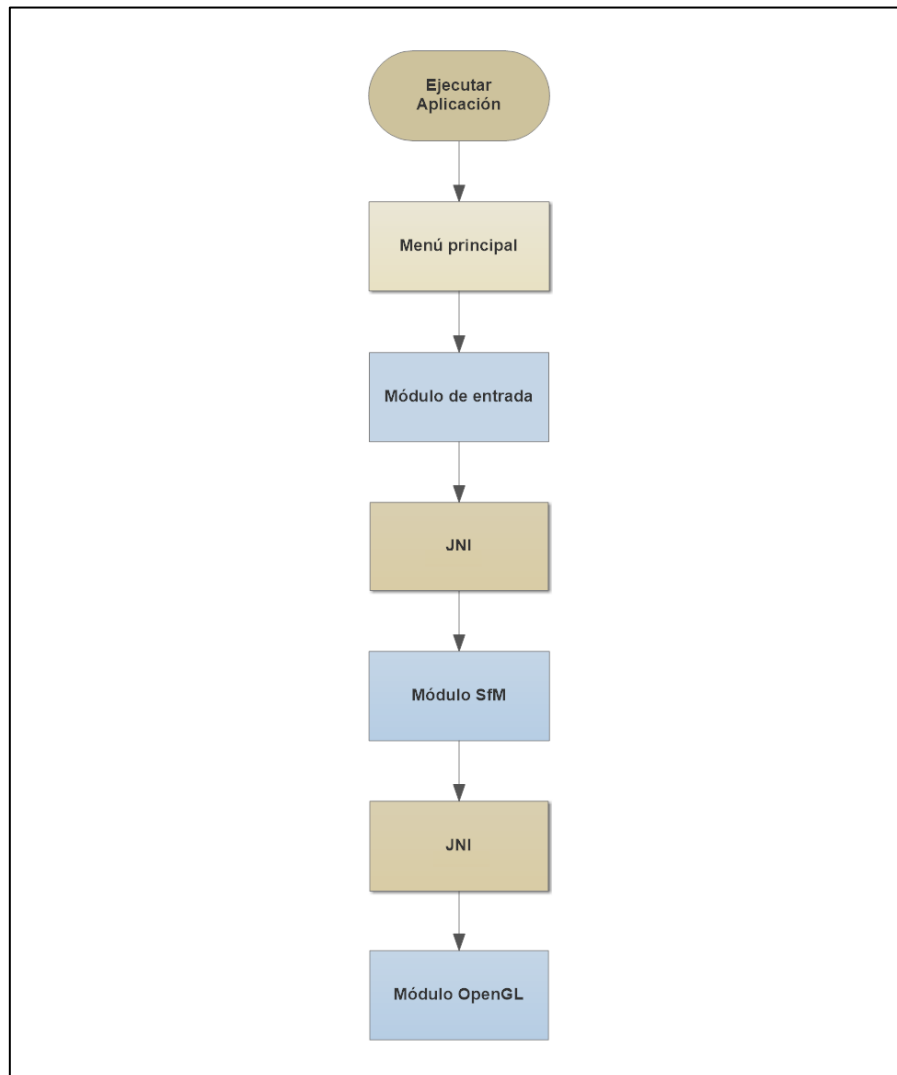


Ilustración 19 - Diagrama de flujo de la aplicación simplificado

Como se puede observar, a partir del menú principal se tienen los tres módulos, entrada, SfM y OpenGL. Cabe destacar que el módulo SfM está rodeado por el *wrapper* de Java a C++ llamado JNI. Cada vez que la información entre o salga de dicho módulo, tiene que pasar por la parte JNI para que ambas partes del código puedan comunicarse.

3.2.1 Módulo de entrada

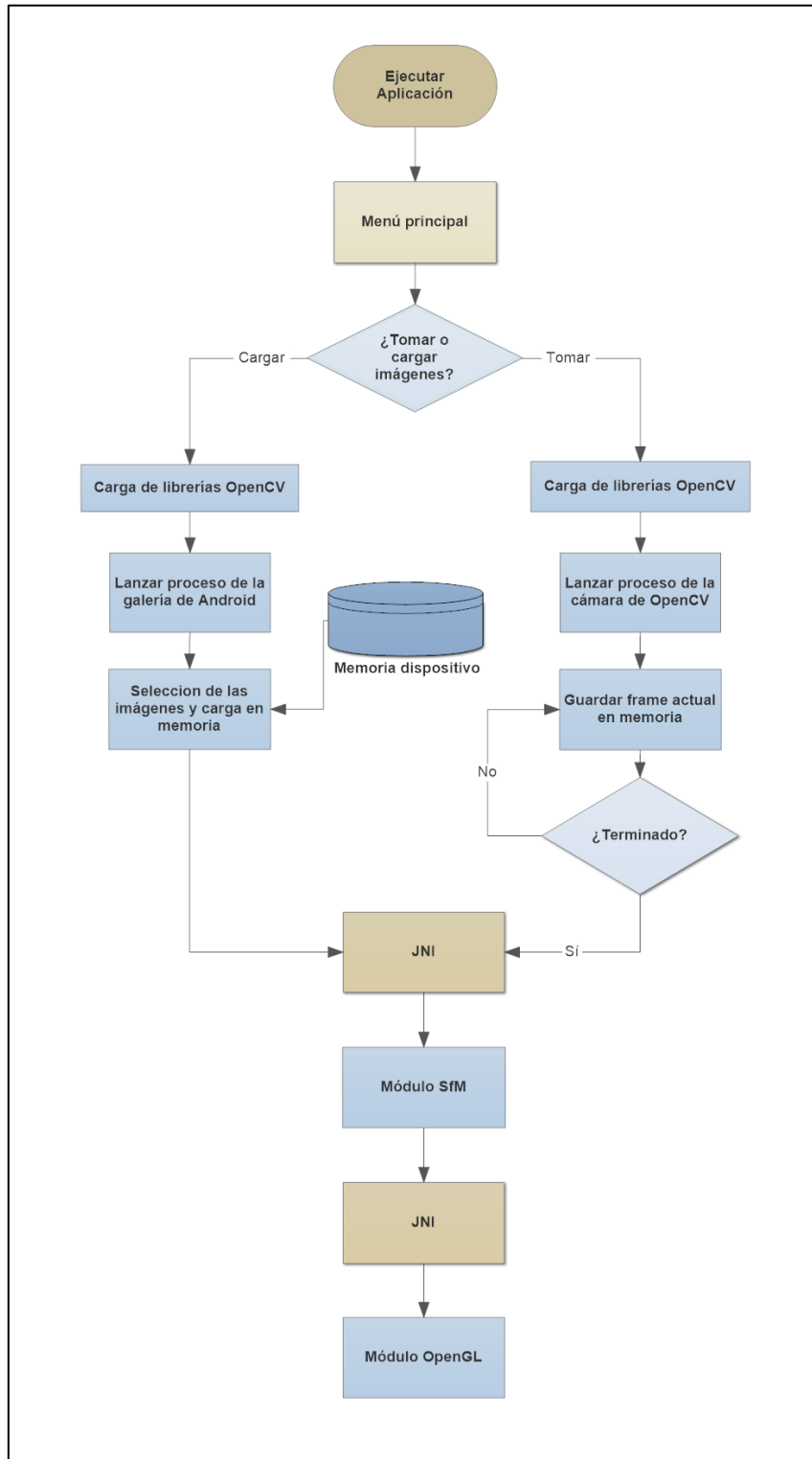


Ilustración 20 - Diagrama de flujo del módulo de entrada

Una vez ejecutada la aplicación y en el menú principal, se le da al usuario la opción de obtener las imágenes para la reconstrucción de dos maneras: realizar fotografías con el mismo móvil o seleccionar imágenes ya presentes en la galería del dispositivo.

En caso de querer cargar las imágenes desde el dispositivo, se crea una ventana con el contenido de la galería donde el usuario puede seleccionar múltiples fotografías. Para poder activar la selección múltiple es necesario mantener pulsado durante unos pocos segundos una imagen. Posteriormente se pueden seleccionar todas las fotografías con el simple hecho de presionarlas.

Si el usuario prefiere tomar las imágenes en el mismo momento con su dispositivo, se carga el contexto de OpenCV junto con el módulo de la cámara. Una vez aquí, el usuario puede realizar fotografías pulsando la pantalla. En el menú se encuentra una opción para indicar que el usuario ha terminado.

Cargadas las imágenes, se envían en forma de matrices al módulo de SfM para iniciar el proceso de reconstrucción.

3.2.2 Módulo SfM

Una vez obtenidos los datos en el módulo SfM se procede a encontrar los *features* y realizar el *matching* entre todas las imágenes. Posteriormente se selecciona la mejor pareja de vistas para crear la reconstrucción base. En caso de no obtener ninguna base se vuelve al menú principal, ya que si se falla en este paso, existe un problema con las imágenes seleccionadas y no se puede proceder con la reconstrucción. Si se ha obtenido una base, se procede a crear la nube de puntos inicial basándose solamente en las dos fotografías de la base.

A continuación se selecciona una vista adicional y se comprueba si es una vista válida para añadir al modelo base. Si es válida, se procede a triangular la vista actual con las vistas anteriores añadiendo el resultado a la nube de puntos actual, si no es válida, se descarta.

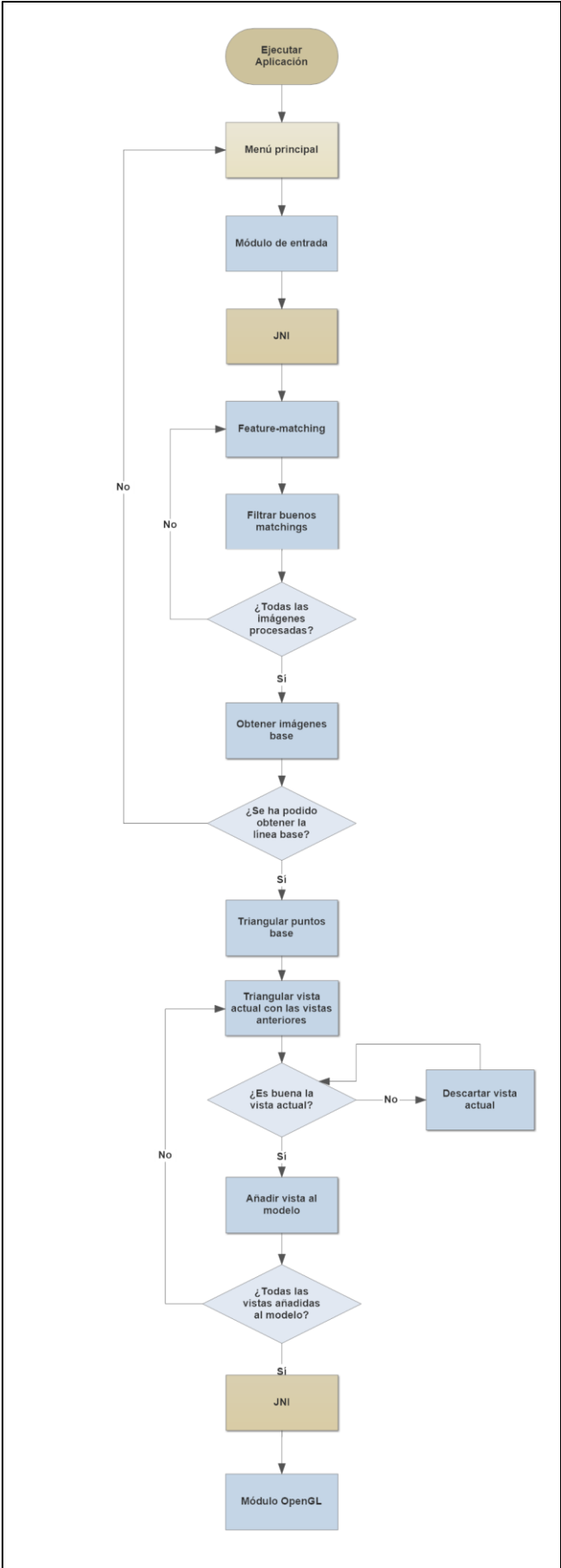


Ilustración 21 - Diagrama de flujo del módulo SfM

3.2.3 Módulo OpenGL

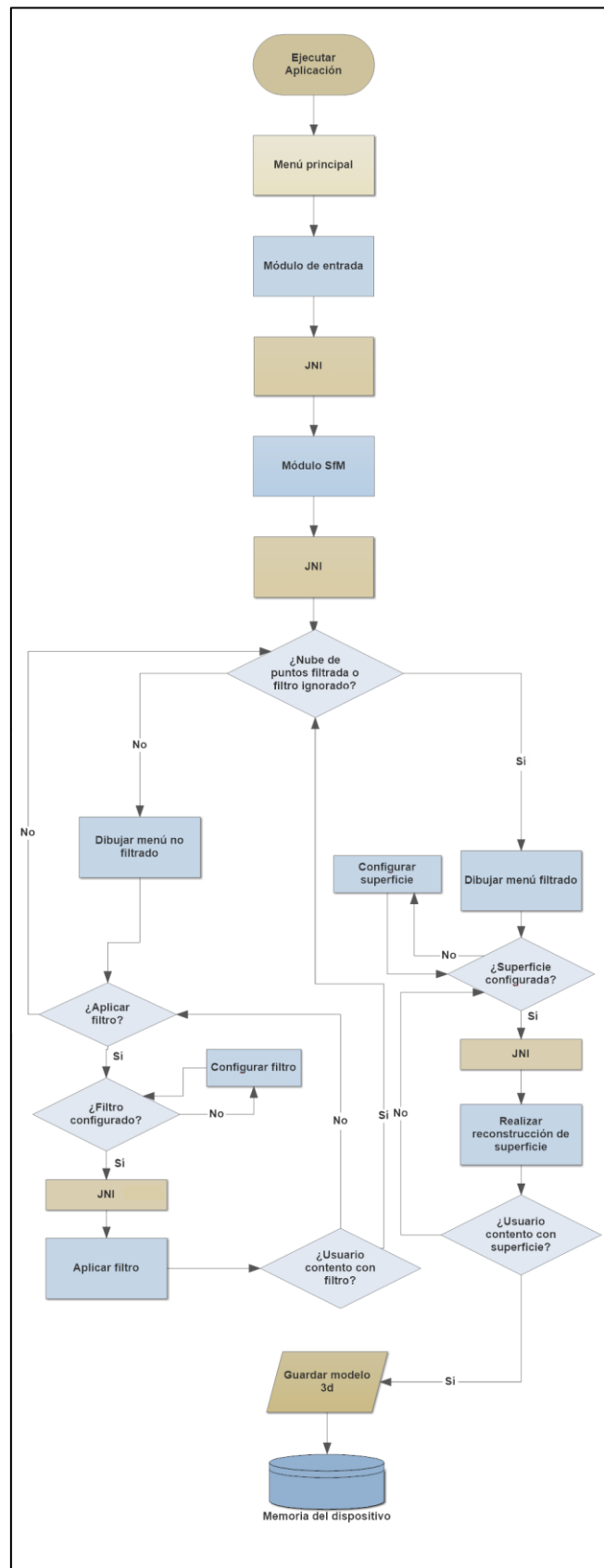


Ilustración 22 - Diagrama de flujo del módulo OpenGL

La nube de puntos calculada por el módulo SfM es representada en este módulo mediante OpenGL. Normalmente es recomendado aplicar un filtro para borrar puntos erróneos y conseguir una nube de puntos más precisa. De todos modos, es posible saltarse el paso del filtrado. Si en cambio, se decide aplicar el filtro, se dibuja en pantalla un menú donde se puede acceder a la configuración de las opciones del filtro. Al seleccionar la opción de realizar el filtrado, los datos de la nube de puntos son enviados al JNI donde son convertidos para ser procesados por el algoritmo de filtrado. El resultado del filtro es devuelto y representado al usuario. Aquí puede darse el caso de que el resultado obtenido no es el deseado, por lo que se permite volver al estado anterior al filtrado y reajustar las opciones del filtro.

Una vez se obtiene el resultado esperado, se realiza un ciclo parecido para la reconstrucción de la superficie. Es posible ajustar los parámetros y realizar la reconstrucción. Si el resultado no es el deseado, se puede volver al estado anterior y llevar a cabo los reajustes necesarios. Cuando finalmente se obtiene ya el modelo 3D, se ofrece la opción de guardar el resultado en la memoria del dispositivo.

3.4 Diagrama de clases

Los diagramas de clases muestran las diferentes clases y las relaciones entre ellas de un sistema. Son muy útiles para entender el esquema de la aplicación de manera rápida y facilita posibles modificaciones futuras.

En la ilustración más abajo se muestra el diagrama de clases de la aplicación. Como ya explicado anteriormente, en Android las clases utilizadas para mostrar información por pantalla son llamadas actividades, siendo hijos de la clase Activity. Dichas clases utilizan unos métodos básicos para poder funcionar, Eclipse introduce estos métodos de manera automática.

En cada clase se muestra el nombre, seguido de los atributos y los métodos más importantes. Se puede observar que muchas de las clases operan de manera independiente y no hay estructuras complejas de objetos.



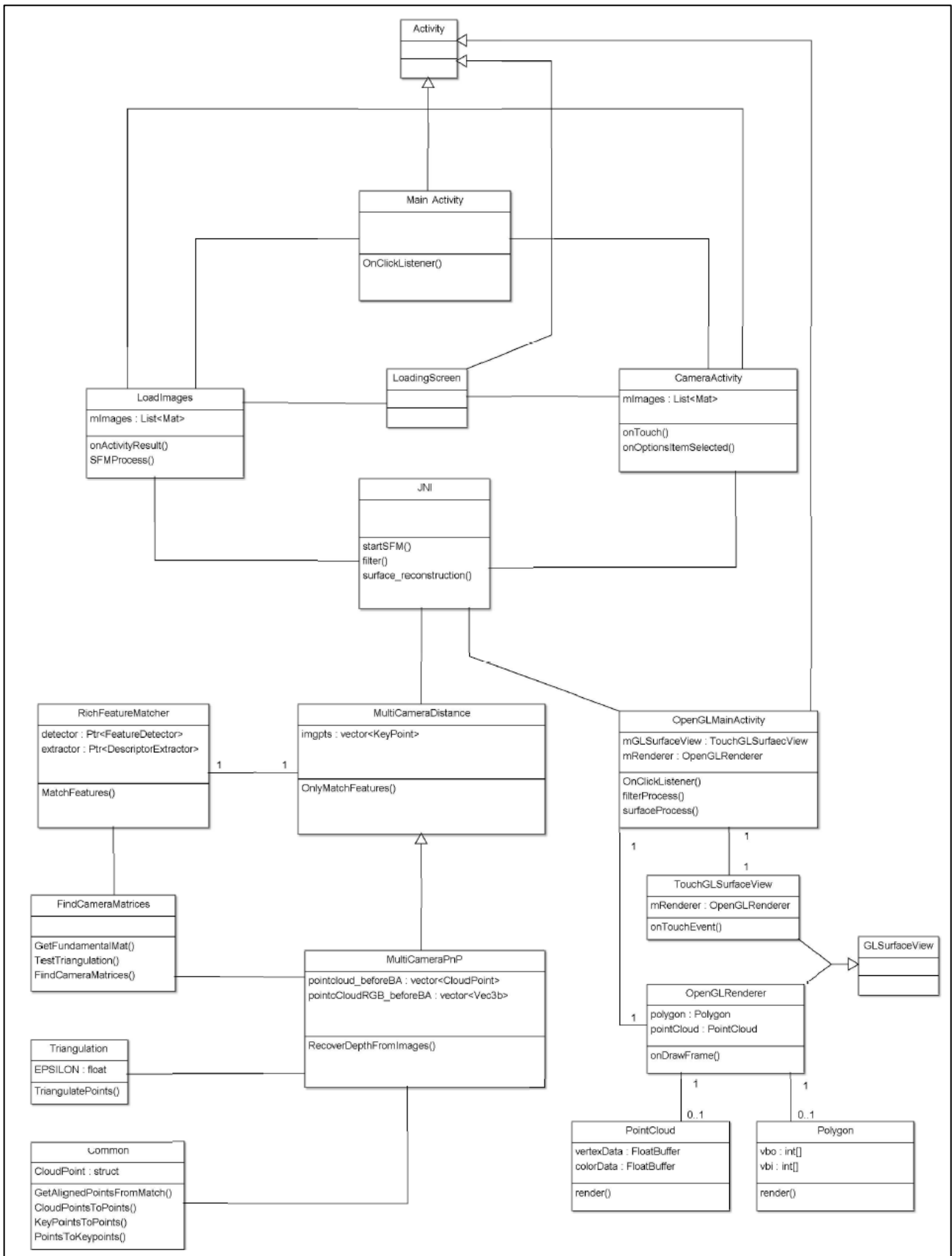


Ilustración 23 - Diagrama de clases

4. Diseño

En este apartado se pretende explicar detalladamente cómo la aplicación realiza las tareas, evitando todo lo posible la mención de código ya que muchas veces puede complicar demasiado la lectura de esta memoria.

4.1 Plataforma software y hardware

En esta sección se expondrán brevemente las limitaciones de software y hardware que se han tenido que tener en cuenta en el desarrollo de la funcionalidad de este proyecto.

Esta aplicación está diseñada para dispositivos móviles usando el sistema operativo Android. Esto implica una serie de limitaciones como puede ser que el dispositivo solamente dispone de una única cámara (en el mismo lado). Lo ideal habría sido disponer de dos cámaras desplazada ligeramente por una distancia conocida. Esto hubiera facilitado enormemente el desarrollo de la aplicación así como conseguir un resultado más exacto. En un primer momento se jugó con la idea de utilizar el GPS o el acelerómetro para determinar la distancia y rotación de la cámara entre diferentes fotografías, pero como demostró una investigación más profunda sobre dichas características, las estimaciones obtenidas no eran lo suficientemente precisas para esta tarea. Por lo que se tuvo que recurrir al método de *Structure from Motion* expuesto en el apartado de análisis.

Otra limitación de hardware es la reducida capacidad de proceso de los dispositivos móviles. Sí que es cierto que se ha conseguido un rendimiento impresionante para el tamaño de dichos aparatos, pero el problema que se trata en este proyecto requiere de mucha potencia de cómputo. Esto dio nacimiento a la idea de que se podrían enviar las imágenes tomadas por el dispositivo a un servidor donde se ejecutara el método de reconstrucción. Pero debido a que no siempre se dispone de conexión a internet y a la gran cantidad de datos, que implicaría subir mucha información a internet, se decidió centrarse en realizar todo el proceso en el mismo dispositivo.

A parte de las limitaciones de hardware se tienen las limitaciones de software, que también han tenido impacto en el diseño de este proyecto. Aunque sí que es verdad que existen muchas librerías soportadas por Android, ya sea directamente en Java o en C/C++ mediante el NDK, palidece en comparación a un entorno de escritorio. Uno de los impactos más severos que ha tenido esta incompatibilidad de librerías es que no se ha podido implementar un método de BA para el refinamiento de la reconstrucción 3d.

4.2 Funcionamiento de la aplicación

Con la información establecida en el análisis se realiza el diseño de los componentes de la aplicación, para facilitar su posterior implementación. A continuación se explicará la forma de funcionar de las diferentes partes del proyecto.

4.2.1 Toma de fotografías

La primera parte es la toma de datos. Como ya se ha indicado anteriormente, esta se puede conseguir de dos maneras, cargando las imágenes directamente del dispositivo o utilizando el módulo de la cámara de OpenCV. Para esta selección se han colocado dos botones en el menú de inicio. Para saber si un botón es presionado en Android se le asigna un *listener* al botón mediante el método *setOnClickListener()*. Una vez el botón es presionado y el método ejecutado se llama la siguiente actividad mediante la función *startActivity()*.

El primer proceso que se explica es el de la carga de las fotografías. Una vez instanciada la nueva actividad, se carga el módulo OpenCV ya que se requiere una estructura de datos proporcionada por esta librería. Esta estructura es *Mat*, la cual se puede entender como una matriz donde cada elemento es un píxel de una imagen dada, por lo que resulta perfecta para el propósito que se requiere: almacenar una imagen. Para guardar varias fotografías se utiliza un *ArrayList* de elementos *Mat*.

Para cargar múltiples imágenes, se llama al método *startActivityForResult()* con un *Intent* que indica que se quiere seleccionar varios elementos del tipo de contenido *PICTURE*. Si se ejecuta correctamente se guarda cada imagen en un elemento de la *ArrayList* mencionada anteriormente. Una vez terminado esta parte, se ejecuta un método de manera asíncrona, el cual envía la información guardada al JNI. Es importante ejecutar el proceso de reconstrucción de forma asíncrona para que el hilo principal de ejecución no se quede bloqueado. Cuando se ejecuta un método de manera asíncrona, se crea un nuevo hilo para su ejecución. En el hilo principal se carga una actividad que muestra una pantalla de carga para informar al usuario que se están procesando los datos.

Para cuando el usuario pulsa el botón para realizar las fotografías con el dispositivo se ejecuta una nueva actividad en la cual se carga OpenCV para hacer uso de su módulo de cámara. Una vez se ha cargado el módulo de la cámara de OpenCV, se muestra por pantalla lo que se observa por la cámara. Mediante la función *onTouch()* se escucha cualquier gesto que se realiza en la pantalla. Cuando el usuario toca la pantalla se guarda la imagen actual en la misma estructura de datos mencionada anteriormente. Esto se puede repetir hasta que se selecciona la opción de Done en el menú de la actividad. El menú también permite cambiar la resolución de las imágenes capturadas por la cámara.

Una vez terminado, al igual que en el método de carga directa del dispositivo, se llama a la función para enviar los datos obtenidos al JNI.

4.2.2 Algoritmos de Structure from Motion

Cabe destacar que esta parte se ha extraído de [13], simplemente se han llevado a cabo unas pequeñas modificaciones para adaptar el código al entorno Android. Aun así, se explica a continuación el funcionamiento.

Cuando los datos del apartado anterior son recibidos en el JNI, lo primero que se hace es convertir la estructura de datos de Java a una de C++. La estructura *Mat* es la misma en ambas versiones de OpenCV (java y C++) por lo que solamente se tiene que convertir el contenedor. La estructura que más se adapta a las necesidades es el vector, por lo que el *ArrayList* de *Mat* es convertido en un vector de *Mat*. El siguiente paso es pasar los datos a una función que empieza el proceso de reconstrucción. Aquí es buena idea aprovecharse de las ventajas que presenta el paso por referencia de C++ para mantener un uso de memoria moderado.

Para empezar la reconstrucción se instancia un nuevo objeto de tipo *MultiCameraPnP*, el cual es un hijo de *MultiCameraDistance*, por lo que antes que nada se ejecuta el constructor de esta clase donde se realiza una comprobación del tipo de las imágenes y la creación de variables necesarias para proceder como por ejemplo la matriz de calibración de la cámara. Al no tener la cámara calibrada se realiza una aproximación.

Una vez creado el objeto con éxito, se llama al método que empieza la reconstrucción. Primeramente se llama al constructor de la clase *RichFeatureMatcher* donde se realiza la extracción y computación de descriptores de los puntos clave de cada imagen. El resultado es guardado en un vector de vectores de puntos clave. Posteriormente se realiza el *matching* entre todas las vistas, aplicando el filtro de podado mediante la matriz fundamental. Gracias a la función *findFundamentalMat()* proporcionada por OpenCV se puede obtener la matriz fundamental con facilidad. Dicha función también ofrece la posibilidad de obtener un vector el cual indica si el punto tiene o no sentido estéreo con la matriz calculada, filtrando los puntos probablemente erróneos.

A continuación se procede a obtener la línea base o lo que es lo mismo, una nube de puntos 3D inicial a partir de solamente dos imágenes. Para esto hay que ordenar las parejas de vistas dependiendo de cuantos puntos en común tienen. Una vez se ha establecido dicho orden, se prueba a usar la primera pareja. Primero se calcula la matriz fundamental entre ellos y si resulta no ser buena, se prueba con otra pareja. La manera de saber si la matriz fundamental es buena o no, se puede determinar, por ejemplo, por el número total de *matchings* que quedan después de realizar el cálculo y quedarse con los que tienen sentido. También se puede saber si la matriz esencial resultante es correcta o no calculando el determinante, el cual debería ser muy próximo a 0. Realizando el cálculo del determinante sobre la matriz de rotación obtenida se puede saber si es correcta o no, ya que las matrices de rotación siempre tendrán un determinante próximo a 1 o -1.



En el caso de que la pareja escogida resulte ser buena, se procede a triangular los puntos entre las imágenes.

Para realizar la triangulación simplemente hay que solucionar el sistema planteado en el apartado de análisis para cada punto. OpenCV ofrece un método llamado *solve()* el cual permite hacer esto con facilidad. Cabe destacar que los puntos en 2D están representados en forma de coordenadas homogéneas $(x,y,1)$. Hay que asegurarse de que las coordenadas estén normalizadas, lo cual se consigue multiplicándolos por la matriz de calibración. En cada iteración sobre un punto se calcula el error de reproyección para posteriormente calcular la media y poder desechar puntos mal triangulados. Todo este proceso se lleva a cabo en el fichero *Triangulation*.

Como ya indicado en el análisis, para realizar la reconstrucción incremental se utiliza la función *solvePnP* de OpenCV. Esta función intenta solucionar un sistema dado los puntos que ya se han reconstruido y una nueva cámara, por lo que necesita dos vectores alineados entre los puntos 2D y 3D. Vectores alineados significa que la el elemento en la posición n de un vector se alinea con el elemento en la misma posición del otro. Por lo que hace falta encontrar los puntos 2D en los ya triangulados puntos 3D. La forma más simple de hacer esto es añadir a cada punto 3D su origen. Por esta razón se ha utilizado una estructura especial para un punto 3D donde se almacena el mismo punto y un vector de enteros con los índices (para cada vista) de su origen en 2D. Esta información es necesaria inicializarla cuando se triangula un punto, guardando que cámara ha estado involucrada. Una vez obtenido el resultado de *solvePnP* y éste sea aceptable, se puede obtener la nueva matriz estimada de la posición para la cámara y utilizar nuevamente la triangulación entre esta y todas las vistas anteriores para popular la nube de puntos con más puntos. Todo esto se repite hasta haber iterado sobre todas las imágenes disponibles.

4.2.3 Representación de los resultados

Al terminar el proceso de reconstrucción, los datos son devueltos al JNI donde son convertidos a estructuras de datos del JNI para poder transformarlos a JAVA. En un principio no se realizaba este paso, y se guardaba el resultado en un fichero en la memoria física del dispositivo, pero por temas de rendimiento se cambió finalmente. El tener los datos en la memoria virtual del dispositivo disminuye enormemente la velocidad de carga de los mismos, sobre todo a la hora de mostrarlos por pantalla.

Cuando el hilo de la reconstrucción termina, se llama a una nueva actividad llamada *OpenGLMainActivity* encargada del proceso de mostrar los resultados por pantalla. Esta actividad es necesaria para inicializar la actividad que realiza el renderizado de lo que se mostrará. También incluye los métodos de escucha de los botones del menú, así como crea una instancia de la actividad encargada de leer los gestos táctiles que se producen en la pantalla por el usuario, llamada *TouchGLSurfaceView*.

Los datos son enviados a la clase *OpenGLRenderer* en formato de un vector *float*. Dependiendo de si se quiere dibujar una nube de puntos o un polígono, cambiará la forma en que se guarda el contenido. En caso de tratarse de una nube de puntos, los

tres primeras posiciones tendrán asignadas las coordenadas X, Y, Z de la posición del primer punto, seguido por su información de color, representado también por tres valores, RGB. En cambio si se trata de un polígono, los primeros dos valores del vector indican la cantidad de vértices que tiene el modelo, y el segundo valor indica el número de caras que componen el modelo. Las siguientes posiciones son las coordenadas de todos los puntos, seguido de las normales de todos los puntos y finalmente el orden de utilizar los puntos anteriores para crear las caras de la figura.

A la vez que se extraen estos datos, se obtiene el punto medio de cada modelo calculando la media entre el valor más alto y el más bajo de cada coordenada en cada eje (X, Y, Z). Esto es necesario para posteriormente poder rotar el objeto sobre sí mismo, consiguiendo una rotación mucho más natural e imposibilitando al modelo salirse del área visible.

Para poder renderizar en OpenGL ES 2.0, hace falta encapsular la información en buffers. Estos buffers son enviados directamente a la memoria de la GPU y guardados ahí, ofreciendo un considerable incremento en el rendimiento. Creados estos buffers se envía la información para que sean representados, mediante `GL_POINTS` en el caso de tratarse de una nube de puntos o `GL_TRIANGLES` en caso de ser un polígono. También hace falta multiplicar las matrices de modelo, vista y proyección para conseguir el resultado deseado. Aquí es donde entra en juego la clase mencionada anteriormente de *TouchGLSurfaceView*, ya que cuando el usuario desliza el dedo por la pantalla, esos valores son recogidos por esta clase y enviados a la clase de renderizado modificando la matriz de modelo para representar la rotación correspondiente del modelo.

En todo momento se exponen una serie de botones en la parte inferior de la pantalla, para poder seguir con el proceso o configurar lo que se vaya a realizar en el siguiente paso. Para mostrar el menú del filtro o la reconstrucción de la superficie se llama a otra actividad para exponer todas las opciones de configuración. En caso de que se presione el botón de filtrar o reconstrucción, los datos son enviados nuevamente al JNI para que se realice la acción correspondiente.

En caso de realizar el filtrado los datos son convertidos a una estructura de datos propia de la librería PCL llamada *PointCloud<PointXYZRGB>*. Una vez realizada esta conversión se crea un objeto *StatisticalOutlierRemoval*, al cual se le pasa los datos junto a los parámetros de configuración para que realice el filtrado. Posteriormente se extraen los puntos filtrados de la estructura anteriormente mencionada y se guardan en un vector, devolviendo los datos al módulo de representación.

En la reconstrucción de la superficie los datos viajan de la misma manera que en el filtrado, aunque la estructura de datos esta vez es ligeramente diferente ya que no permite una nube de puntos de color, si no, simplemente una nube de puntos. Por lo tanto la estructura es la siguiente: *PointCloud<PointXYZ>*. Para aplicar la reconstrucción de *poisson*, hace falta realizar una estimación de las normales de cada punto presente en la nube. Esto se realiza con un objeto llamado *NormalEstimation*, el cual recibe los parámetros de configuración y los datos. El resultado es una estructura *PointCloud<Normal>*, la cual solamente contiene las normales estimadas, por lo que es



necesario concatenarla con la nube de puntos original. Ahora los datos están preparados para ser enviados al objeto de reconstrucción de la superficie llamado *Poisson<PointNormal>*. Una vez calculada la superficie, es devuelta en una estructura *PolygonMesh*, la cual está compuesta por los vértices y las caras del modelo. Cabe destacar que la información de las normales se ha perdido, por lo que es necesario recalcularlas para poder representar la figura posteriormente con OpenGL. Extrayendo los datos de sus respectivas estructuras, los datos son devueltos al JNI y de ahí a la actividad de OpenGL.

En la clase de renderizado se cuenta con tres variables donde se almacena la nube de puntos cruda, la nube de puntos filtrada y el polígono final. Esto facilita volver a un estado previo si no se está contento con el resultado, evitando tener que recalcular algo ya obtenido. Mediante el uso de variables booleanas se controla lo que es mostrado por pantalla en cada momento. Una vez llegado al punto de la reconstrucción completa, es posible guardar el modelo actual en la memoria física del dispositivo pulsando el botón *save*.

4.3 Metodología de desarrollo

La metodología empleada en este proyecto es el modelo en espiral. Se basa principalmente en la reducción del riesgo asumido y posibilita la mejora y la introducción de nuevos elementos durante el desarrollo.

El modelo consiste de una serie de campos, los cuales son recorridos en forma de espiral hasta alcanzar todos los requerimientos establecidos. Las diferentes etapas suelen ser las siguientes:

1. **Análisis:** Suele ser la primera fase y en ella se establecen detalladamente los requerimientos de cada objetivo así como los riesgos y las posibles soluciones.
2. **Diseño:** En esta etapa se especifica el diseño de la aplicación basándose en los datos obtenidos en la parte del análisis.
3. **Implementación:** Aquí se realiza la implementación del diseño con código.
4. **Evaluación:** Esta es una fase vital que muchas veces no es tomada en cuenta lo suficiente. Aquí se revisa el funcionamiento de lo implementado y se comprueba que se hayan alcanzado todos los requerimientos establecidos. También sirve para encontrar problemas que puedan llevar a un cambio en el diseño del proyecto.

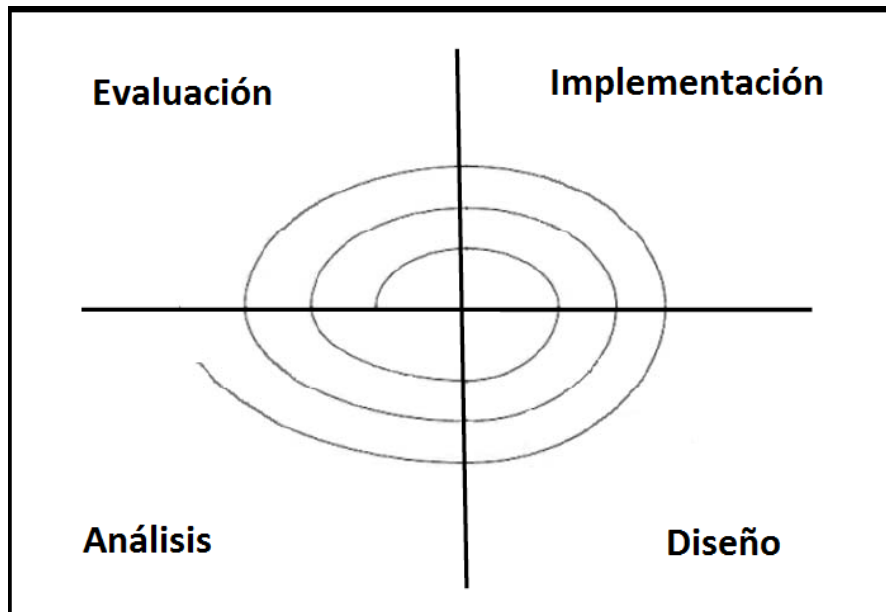


Ilustración 24 - Metodología espiral

4.4 Casos de uso

Los casos de uso son una herramienta útil en el proceso del diseño de una aplicación mostrándonos cuando el usuario interactúa con una determinada parte del sistema. Suelen ser acciones realizadas por el usuario y no por el sistema, realizando una acción completa en un tiempo corto desde el punto de vista del usuario.

También muestran en cierta medida una funcionalidad, aunque en este caso se refiere a lo que se espera que la interacción realice y no a como lo realiza.

A continuación se mostrarán los diferentes casos de usos que se presentan en este proyecto. Dado que el módulo de SfM no requiere de interacciones, no existe ningún caso de uso para esta etapa. En módulo de entrada de datos, se puede dar el caso de dos interacciones diferentes:

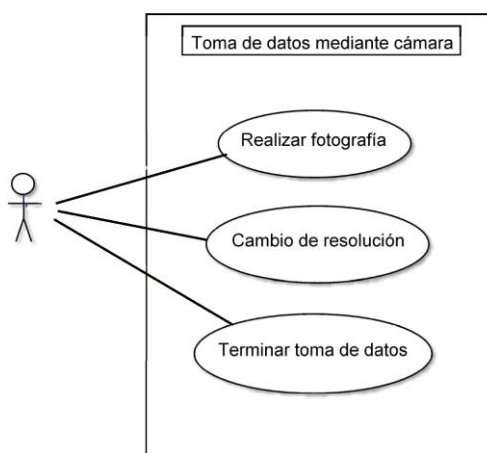


Ilustración 25 - Caso de uso Cámara

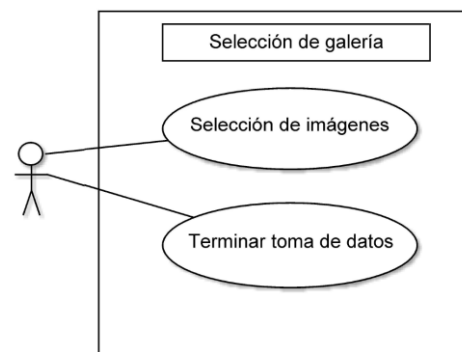


Ilustración 26 - Caso de uso galería

En el módulo de representación de los resultados existen un total de tres casos posibles, mostrados a continuación:

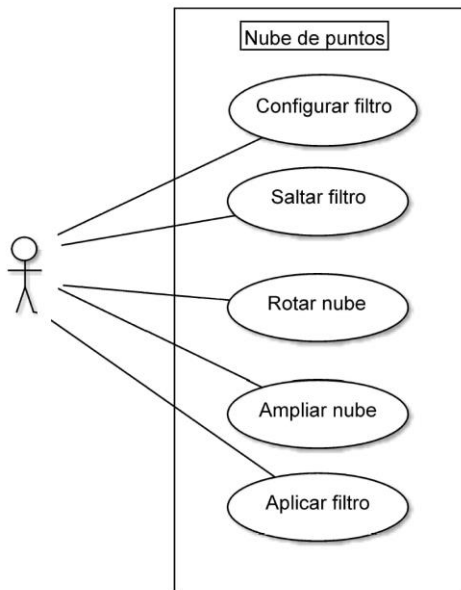


Ilustración 28 - Caso de uso nube de puntos

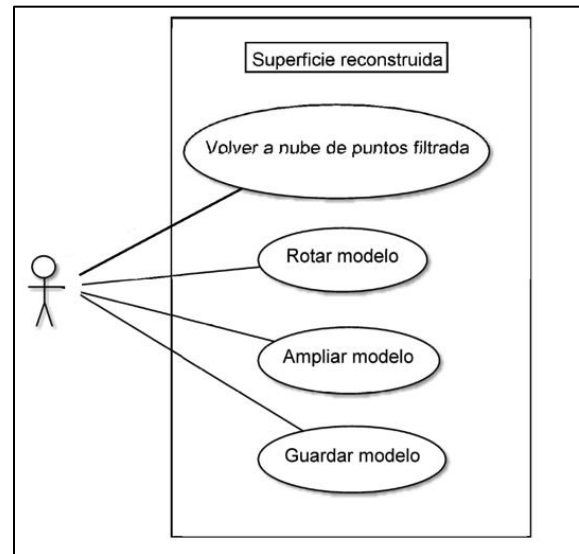


Ilustración 27 - Caso de uso nube de puntos filtrada

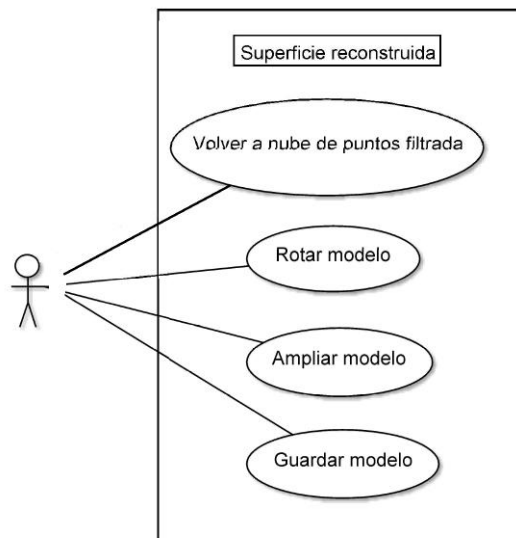


Ilustración 29 - Caso de uso superficie reconstruida

Como se puede observar, se ha intentado mantener un bajo número de acciones posibles en cada sección para crear una aplicación simple de manejar y no confundir al usuario.

5. Resultados

En este apartado se pretende mostrar los resultados obtenidos. Por una parte se va a realizar una ejecución completa de la aplicación desarrollada mostrando paso a paso los resultados utilizando unas imágenes guardada en la galería del dispositivo. Por otra parte se va a mostrar un ejemplo de reconstrucción de superficie un poco más completo usando la herramienta MeshLab. La nube de puntos de la cual se parte en este último ejemplo, será la que ha sido calculada por la aplicación desarrollada.

Si se quiere conocer la parte de toma de fotografía mediante la cámara incorporada, es recomendable consultar el ANEXO I presente al final de este documento.

5.1 Ejecución de un ejemplo

Pulsando el botón correspondiente en el menú principal se abrirá la vista mostrando todas las imágenes disponibles en la galería.



Ilustración 30 - Ejecución - Gal

Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

Pulsando sobre una de las imágenes que se desean utilizar, se activará el modo selección múltiple. En este caso se han elegido tres fotografías.

Una vez seleccionadas las imágenes, comienza el proceso donde se calcula la nube de puntos en 3D. Mientras se lleva a cabo el proceso, se muestra un pantalla de carga indicando que la aplicación está trabajando.

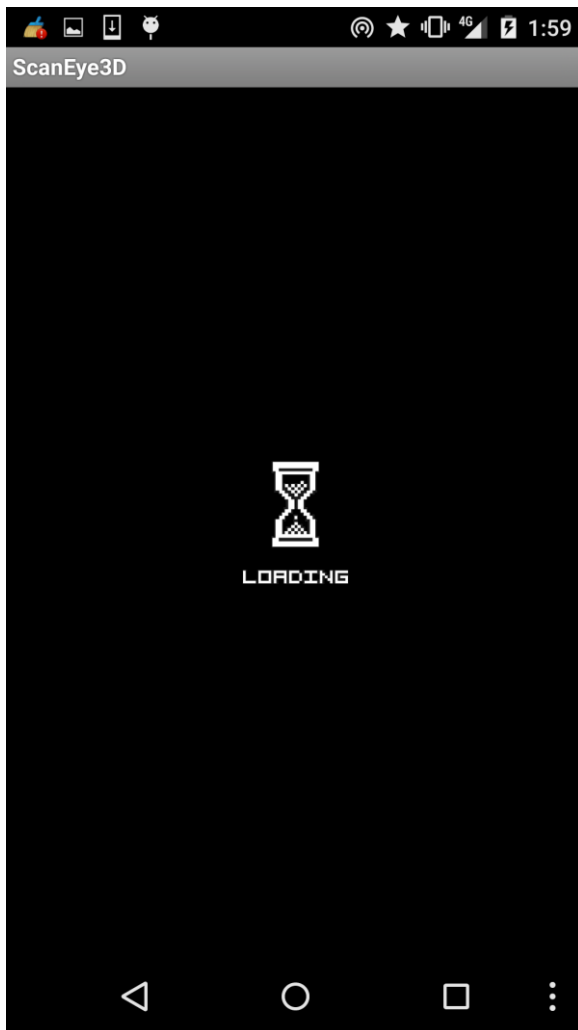


Ilustración 32 - Ejecución - Pantalla de carga

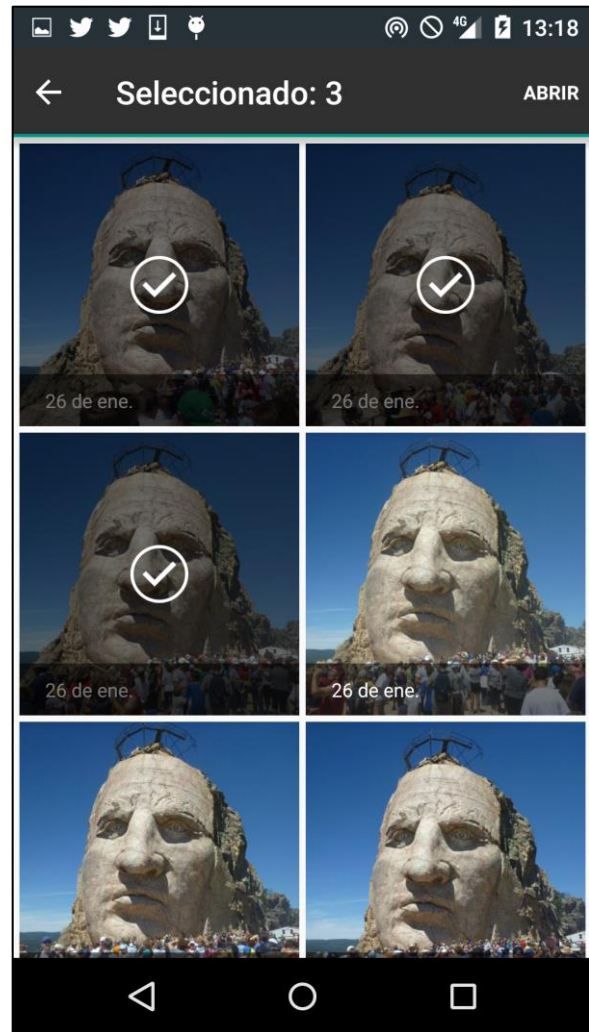


Ilustración 31 - Ejecución - Selección múltiple

Dado que la potencia de procesamiento de un dispositivo móvil es limitada, este proceso puede tardar bastante tiempo. En este ejemplo, ejecutándolo en un Nexus 4, se ha tardado alrededor de un minuto para las tres imágenes mostradas.

Cuando este proceso finaliza se abre una nueva vista con el resultado calculado. Como se puede observar, la nube de puntos guarda cierta relación con el objeto fotografiado. Desgraciadamente el proceso de obtención de la nube de puntos no es perfecto y por tanto la mayoría de las veces existirán muchos puntos erróneos.

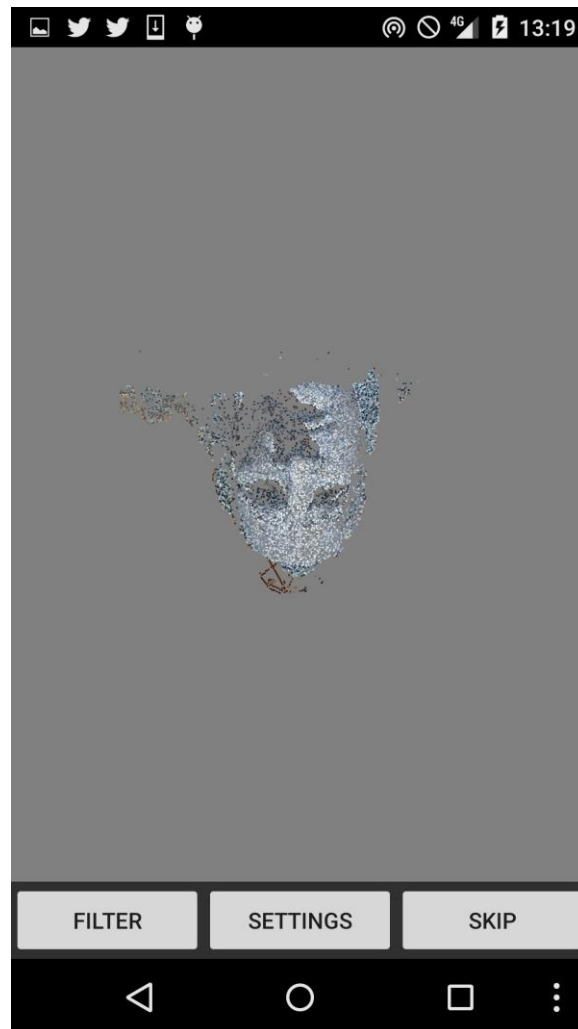


Ilustración 33 - Ejecución - Nube de puntos

En el ejemplo se pueden observar estos puntos en los laterales de la figura, por lo que es buena idea utilizar el filtro implementado para remediar esto. Entrando en la configuración del filtro, se procede a probar con los valores por defecto.

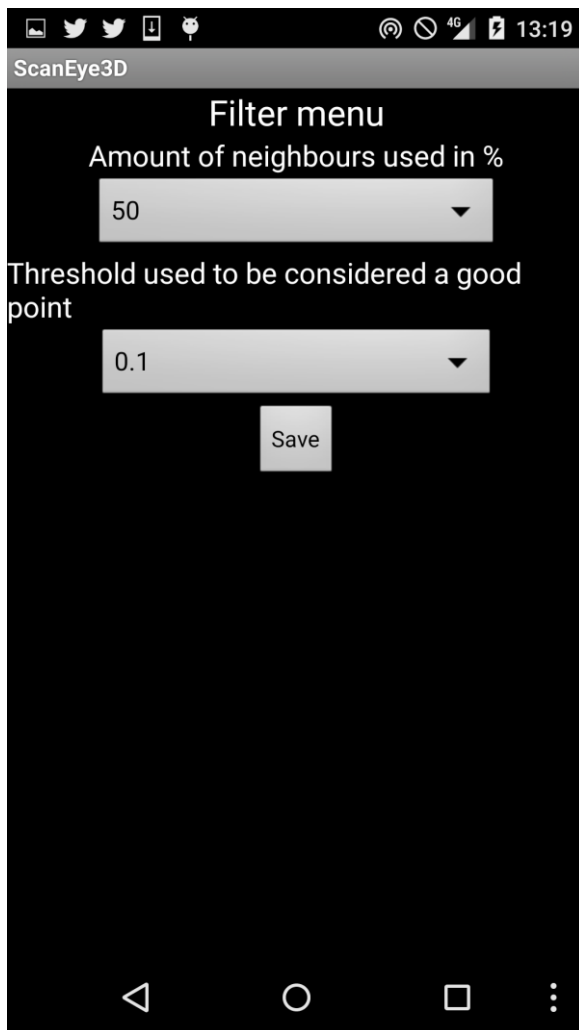


Ilustración 34 - Ejecución - Ajustes filtro 1

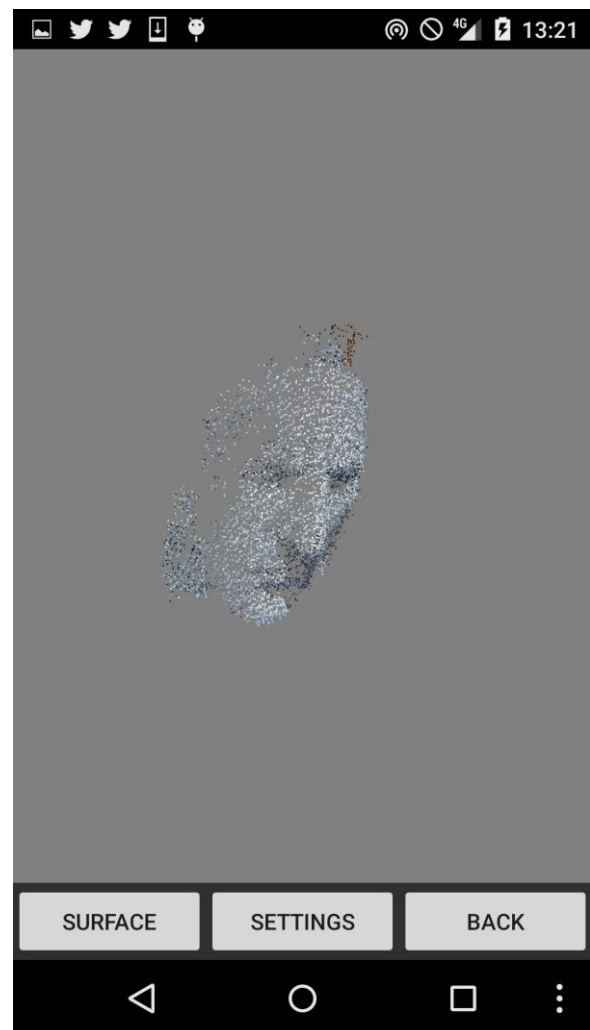


Ilustración 35 - Ejecución - Nube filtrada 1

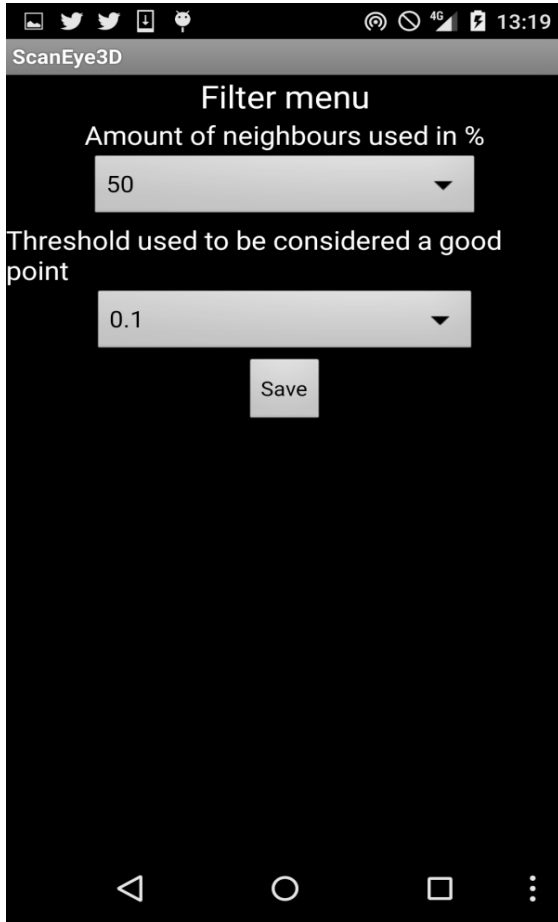


Ilustración 36 - Ejecución - Ajustes filtro 2

La mejora obtenida respecto a la primera vez es poca, por lo que en este caso no era necesario realizar un reajuste de los parámetros. Procediendo a la reconstrucción de superficie, se prueba primeramente con los valores por defecto.

Comparando ambos resultados se aprecia la mejora que ha supuesto la aplicación del filtrado. Aunque la nube de puntos actual ya es aceptable para proceder a la reconstrucción de la superficie, se va a intentar mejorarlo más aún. Reajustando los ajustes del filtro como muestra la imagen, se consigue:



Ilustración 37 - Ejecución - Nube filtrada 2

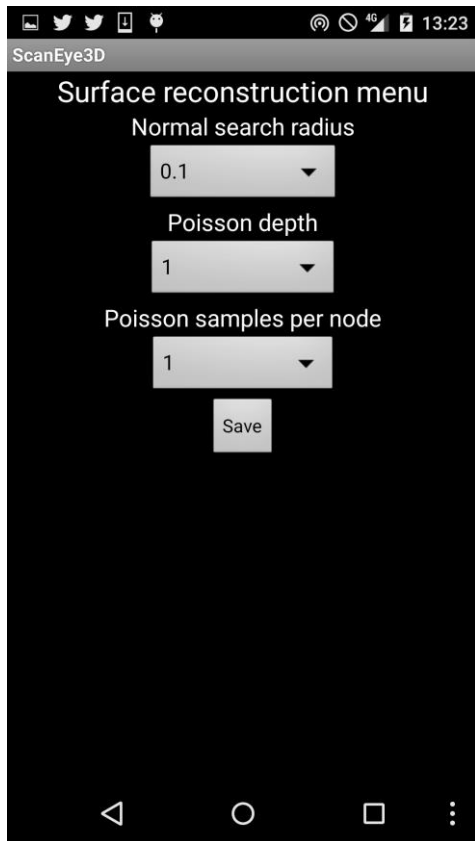


Ilustración 38 - Ejecución - Ajustes superficie 1



Ilustración 39 - Ejecución - Reconstrucción 1

Lo cual no corresponde en absoluto con el objeto inicialmente fotografiado. Por lo tanto es un buen momento de volver atrás y reajustar las opciones del algoritmo.

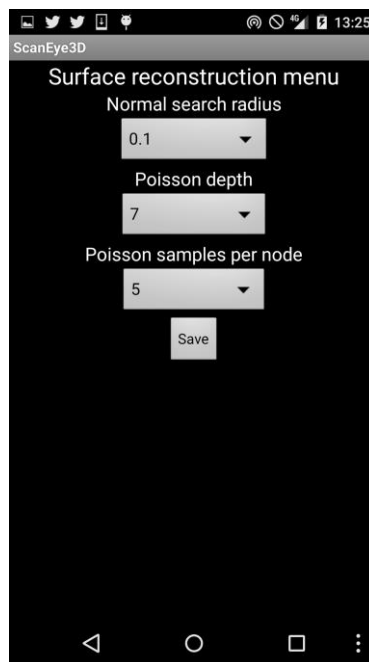


Ilustración 40 - Ejecución - Ajustes superficie 2

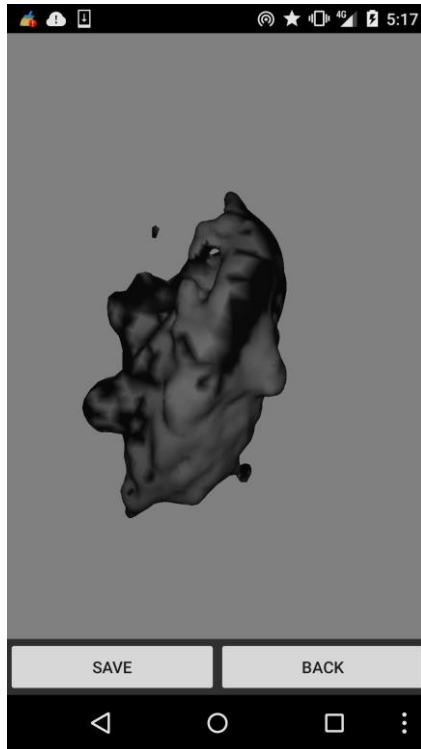


Ilustración 41 - Ejecución - Reconstrucción 2

Comparado con el primer resultado es mejor, pero sigue siendo un mal resultado. Volviendo otra vez y ajustando aún más los parámetros:

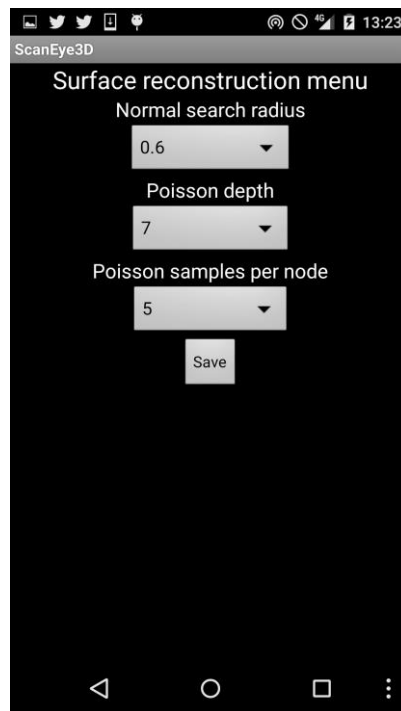


Ilustración 42 - Ejecución - Ajustes superficie 3

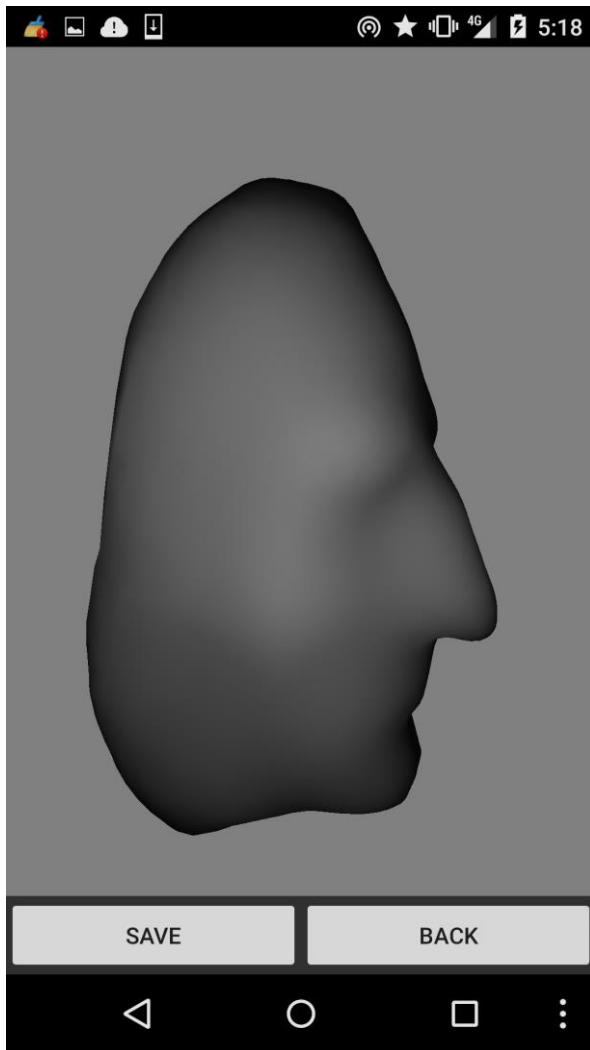


Ilustración 43 - Ejecución - Reconstrucción 3-1

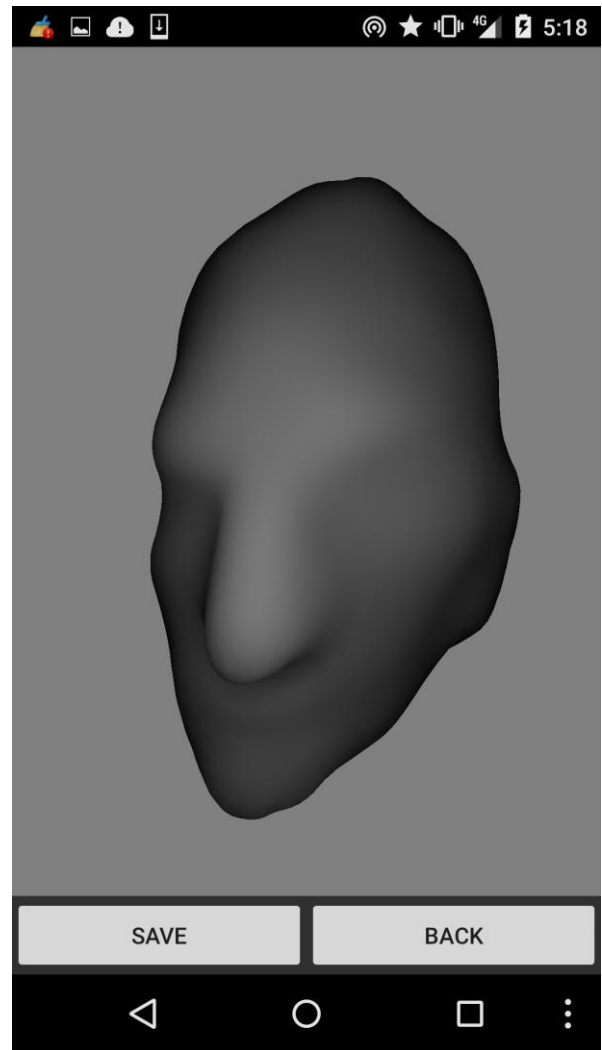


Ilustración 44 - Ejecución - Reconstrucción 3-2

Lo cual ya parece un resultado mucho más acertado. Al disponer ya de un modelo decente, se procede a guardarlo en el dispositivo pulsando el botón correspondiente.

5.2 Reconstrucción de la superficie con Meshlab

Dado que Meshlab ofrece la posibilidad de crear una superficie con textura a partir de una nube de puntos, se ha considerado interesante utilizar la nube de puntos que se ha obtenido con la aplicación y realizar una reconstrucción con Meshlab. A continuación se explicará brevemente cómo se consigue esto y después se expondrán los resultados obtenidos.

Una vez cargado la nube de puntos en Meshlab, el primer paso es calcular las normales. Para esto hay que dirigirse a:

Filters > Normals, Curvatures and Orientation > Compute normals for point sets

El segundo paso es reconstruir la superficie, esto se hace en:

Filters > Remeshing, Simplification and Reconstruction > Surface Reconstruction: Poisson

El siguiente paso es conseguir la textura, para ello hay que dirigirse a:

Filters > Texture > Parametrization: Trivial Per-Triangle

Ahora hay que guardar el proyecto en:

File > Save Project

En las opciones de guardado hay que seleccionar “Flags” y “Normal” bajo “Vert” y “Flags” bajo “Face”.

A continuación hay que transferir el color a la textura:

Filters > Texture > Transfer Vertex Attributes to Textures

En “Source Mesh” hay que seleccionar la nube de puntos de origen, y habilitar la opción de “Assign Texture”.

Con esto el programa procederá a aplicar la textura a la superficie, en este caso obteniendo lo siguiente:

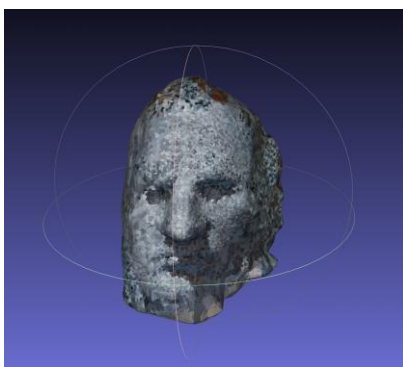


Ilustración 47 - Reconstrucción MeshLab - 1



Ilustración 46 - Reconstrucción MeshLab - 2



Ilustración 45 - Reconstrucción MeshLab - 3

6. Conclusiones

6.1 Problemas encontrados

Durante el desarrollo de este proyecto han aparecidos muchos problemas que con el tiempo se han solucionado o se ha encontrado una manera de evitarlos. Desafortunadamente también hay unos pocos problemas los cuales no se han podido solucionar del todo. A continuación se exponen los dos problemas con más impacto y las aproximaciones que se han intentado para solucionarlos.

Por un lado está el paso adicional en el refinamiento de la nube de puntos resultante llamado *Bundle Adjustment* (BA). Como ya se ha explicado en el apartado de análisis, este proceso conlleva una gran mejora a los resultados obtenidos. El libro del cual se ha obtenido el código del módulo de SfM [13] utiliza una librería llamada SSBA (*simple sparse bundle adjuster*) para este cometido. Dado que este proyecto está escrito para dispositivos móviles Android y la librería mencionada está escrita en C, no es trivial utilizarla. Se intentó compilar la librería utilizando el NDK de Android sin éxito. Incluso se creó un post pidiendo ayuda en *StackOverflow*, una de las páginas más utilizadas en cuanto a dudas sobre desarrollo de software se refiere. El post se encuentra en [16] y a día de escribir esta memoria no cuenta con solución.

También se intentó utilizar una herramienta llamada *Ceres Solver*. Esta librería es propiedad de Google y es capaz de realizar BA. Desgraciadamente después de mucho investigar no se consiguió que funcionara en este proyecto, aunque sí que se tiene pensado centrarse en esta librería para implementar el BA en la aplicación en trabajos futuros, ya que es una librería la cual se puede compilar con relativa facilidad para el sistema operativo Android.

El otro gran problema encontrado es que al incluir la librería PCL la aplicación ya no es compilada para todas las arquitecturas de Android existentes. Por esta razón, solamente se compila para arquitecturas ARMEABI-v7a. Aunque sí que es cierto que es la arquitectura más utilizada, siempre se debería intentar cubrir una compatibilidad más grande posible. Probando a compilar PCL para otras arquitecturas como por ejemplo x86 o mips, siempre resultaba en un error como sigue:

Undefined reference to `omp_get_max_threads_`

Esto se consiguió solucionar implementando el parche [23] en el archivo correspondiente, resultando en la compilación correcta de la librería para las diferentes arquitecturas. Desafortunadamente seguían produciéndose muchos errores a la hora de compilar la aplicación con las librerías compiladas anteriormente, los cuales no se han podido solucionar.

6.2 Mejoras futuras

En este apartado se pretende proponer una serie de mejoras que por falta de tiempo o conocimiento no se han podido implementar. Algunas de ellas son:

- Solucionar los errores expuestos en el apartado anterior.
- Mejorar la eficiencia de la aplicación haciendo uso de todos los núcleos de cálculo disponibles mediante técnicas de computación paralela.
- Mejorar la detección de puntos clave para que la aplicación sea más robusta.
- Implementar la técnica de *optical flow* para calcular el movimiento de las cámaras.
- Añadir texturas al modelo final.
- Implementar una eliminación manual para los puntos erróneos una vez obtenida la nube de puntos.
- Permitir guardar el resultado en un formato de impresión 3D.

6.3 Conclusiones finales

Con el desarrollo de este proyecto se ha llegado a comprender mucho mejor la complejidad que supone el desarrollo de un proyecto de software, así como la importancia de seguir una metodología claramente especificada. Se ha observado la gran importancia que posee el realizar adecuadamente una documentación previamente al proceso de desarrollo, ya que en una primera instancia, tras la realización de una investigación previa, se empezó la implementación sin una perfecta comprensión de los conceptos y herramientas a utilizar. Lo cual produjo resultados negativos, permitiendo la obtención de un *feedback* que redirigió el proyecto siendo necesario empezar de nuevo con nuevos conocimientos más adecuados. Un vez el proyecto se encauzó a buen rumbo se desarrolló fluidamente hasta su completa realización. Gracias a dicho *feedback* se ha adquirido una experiencia considerablemente positiva para afrontar futuros proyectos de forma mucho más capacitada.

Además de esta valiosa lección, se han aprendido numerosas habilidades, entre ellas la utilización de uno de los sistemas operativos móviles más utilizados, de los fundamentos básicos de la visión por computador junto a la librería más importante en este campo, así como la creación de gráficos en 3D mediante OpenGL, finalmente cabe remarcar el aprendizaje de la programación en uno de los lenguajes más importantes como es C++.

A parte de todo lo aprendido expuesto anteriormente, cabe destacar que se está satisfecho por haber superado el reto de solucionar de una forma razonable este problema de análisis 3D, computacionalmente tan exigente, en un entorno limitado como lo es un dispositivo móvil. Las buenas capacidades de representación gráfica en estos dispositivos han permitido obtener unos resultados muy vistosos, los cuales pueden tener amplias utilidades de uso, como por ejemplo la impresión 3D o el uso en



nuevos sistemas de entretenimiento como lo es la reciente realidad virtual. Cabe mencionar que esto no es más que un comienzo, ya que se pretende mejorar la aplicación solucionando los errores e implementando las mejoras mencionadas en los puntos anteriores, llevando a publicar la aplicación en un futuro en la tienda de Google Play.

Por último, es importante indicar que tras la realización de este proyecto se ha entendido el valor de los conocimientos adquiridos en el grado siendo estos de gran utilidad en cuanto a la preparación del alumno a la hora de afrontar proyectos futuros.

Anexo I - Manual de usuario

Una vez arrancada la aplicación aparecerá en el menú principal dos botones:

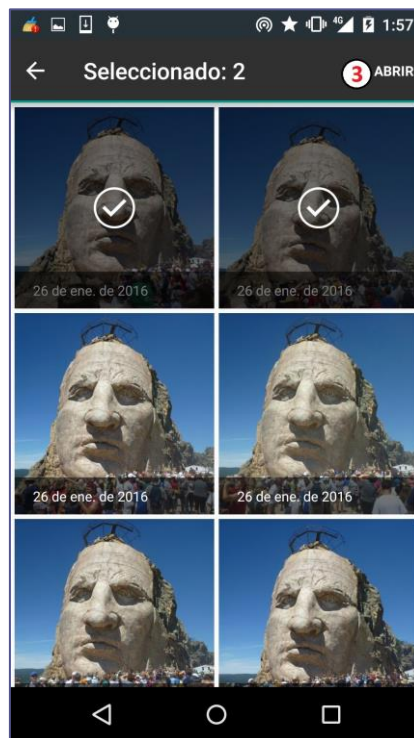


El botón 1 se encarga de abrir la galería del dispositivo para mostrar las imágenes disponibles:



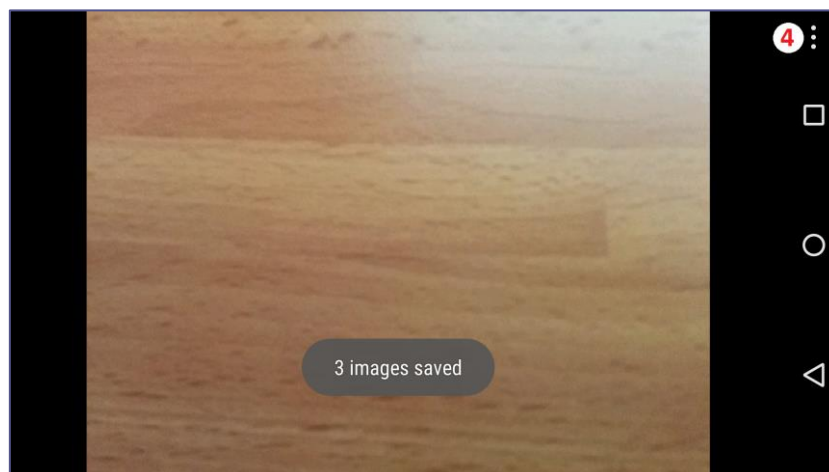
Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

Para seleccionar múltiples imágenes simplemente se tiene que mantener el dedo pulsado sobre una imagen hasta que se active el modo de selección múltiple:

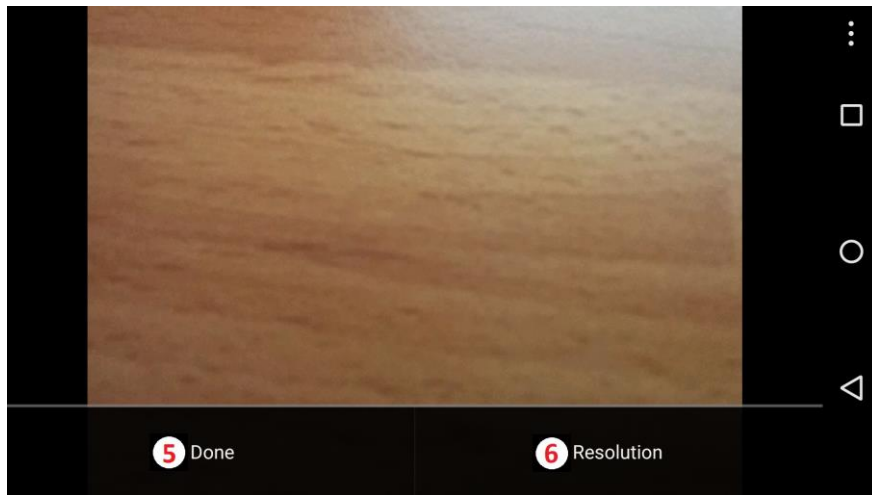


Una vez seleccionadas las fotografías deseadas, se termina el proceso pulsando el botón 3.

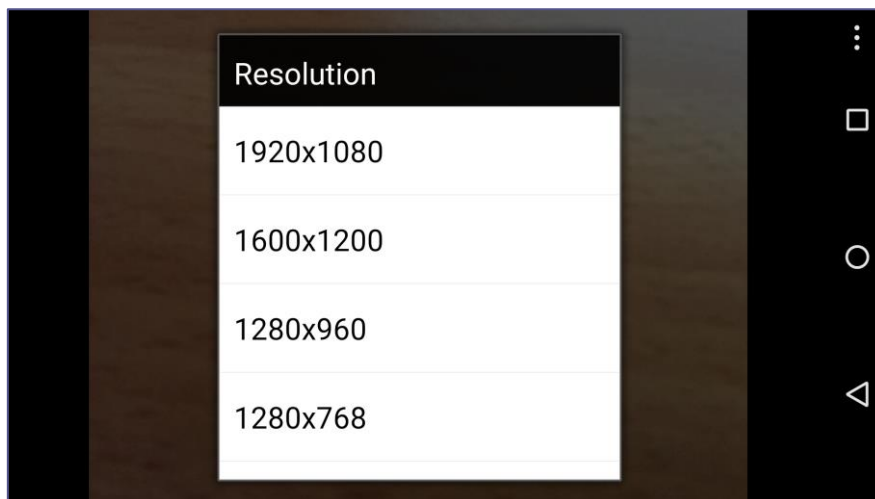
Si en el menú principal se elige el botón 2, se abrirá la vista de la cámara en el dispositivo, donde pulsando la pantalla se realiza una captura notificando al usuario de ello:



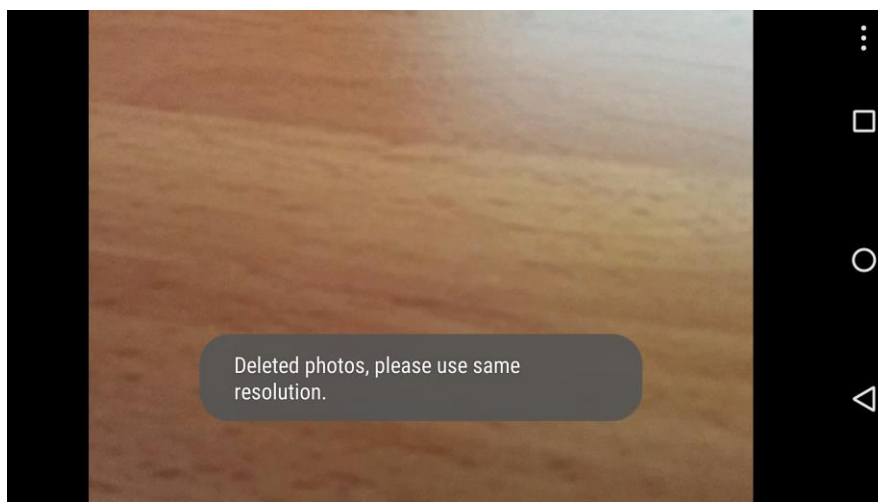
Presionando sobre el botón 4, aparecerá un pequeño menú, el cual permite cambiar la resolución de la cámara o terminar la toma de imágenes.



Si se desea cambiar la resolución de la cámara, mediante el botón 6, aparecerá una ventana con las resoluciones disponibles:

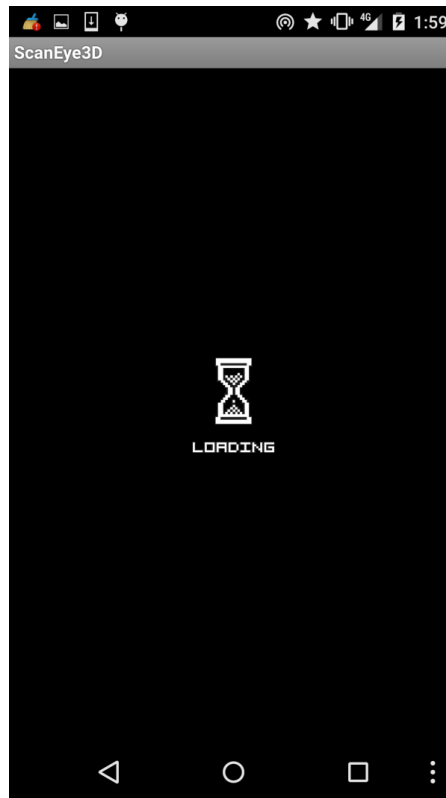


En caso de haber tomado ya alguna fotografía anteriormente con una resolución diferente, las imágenes son borradas ya que el algoritmo de SfM no está preparado para trabajar con resoluciones diferentes.

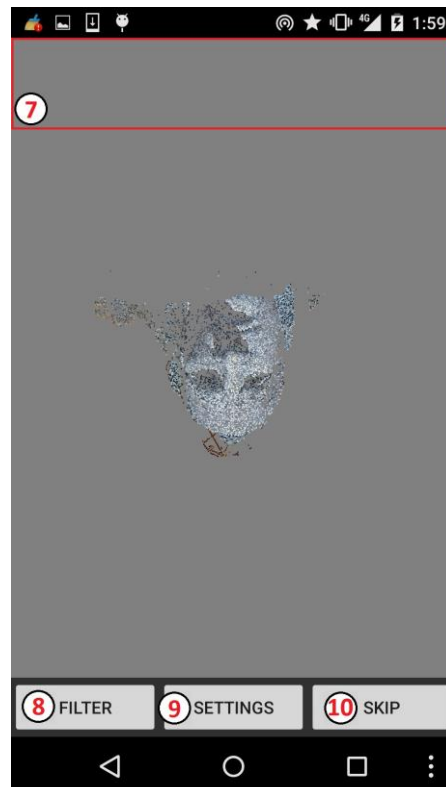


Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

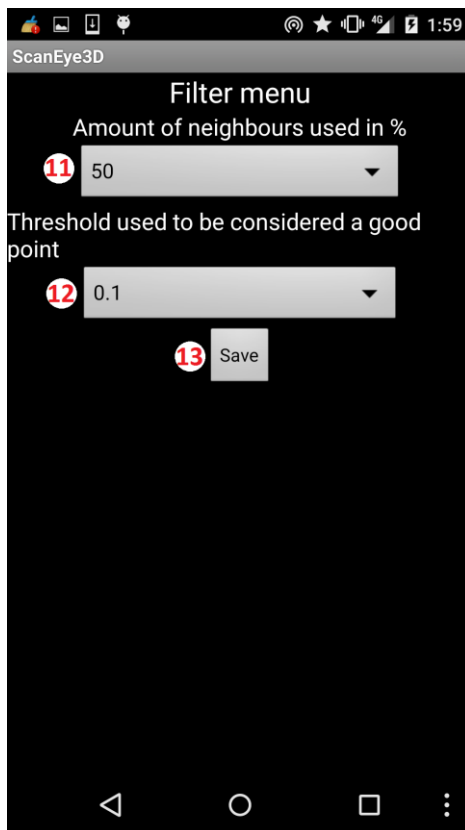
Pulsando el botón 5 se inicia el proceso de SfM mostrando una pantalla de carga.



Una vez terminado se muestra la nube de puntos calculada, permitiendo al usuario rotarla mediante movimientos del dedo por la pantalla. También se puede realizar zoom, moviendo el dedo de derecha a izquierda en la parte superior marcada con el número 7.



En la parte inferior de la pantalla se muestra un pequeño menú con varias opciones, el botón 8 inicia el proceso de filtrado de la nube de puntos, eliminando gran parte de los puntos erróneos que pueden darse. Si antes de pulsar este botón no se ha configurado el filtro mediante el botón 9, se ejecutará el filtro con los valores por defecto. Es recomendable ajustar manualmente estos parámetros para garantizar un resultado más óptimo. Los ajustes mostrados por el botón 9 son los siguientes:



Opción 11: Se refiere a la cantidad de puntos vecinos que se utilizan para realizar el filtrado. Si se selecciona el máximo número posible, se emplean todos los puntos de la nube de puntos. Esto puede producir un aumento notable en el tiempo de computación del filtro, pero también puede ofrecer resultados mejores.

Opción 12: Es el umbral con el que se decide si un punto es válido o no. Cuanto más bajo es, más puntos se eliminan.

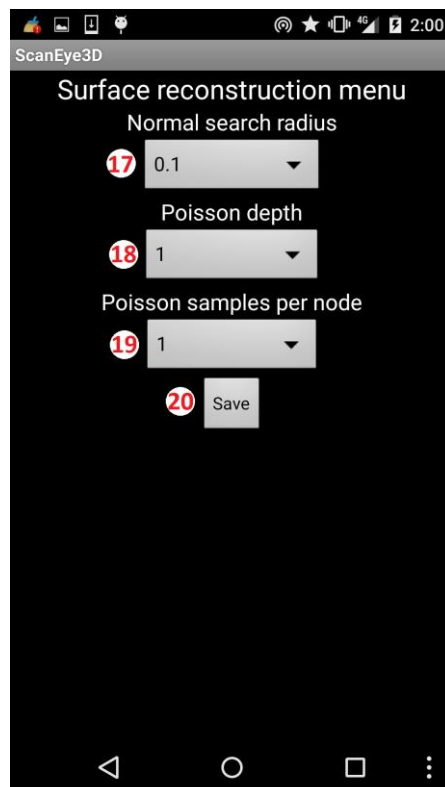
Pulsando el botón 13 se guardarán los ajustes y se volverá a la vista de la nube de puntos.

También es posible saltarse por completo el paso de filtrado pulsando el botón 10. Una vez ejecutado el filtro se mostrará la nube de puntos filtrada.

Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android



Si no se está contento con el resultado, se puede volver al paso anterior mediante el botón 16 y realizar un reajuste del filtro. Los otros botones tienen un comportamiento similar a los del paso anterior. El botón 15 mostrará la configuración de los ajustes para la reconstrucción, y el botón 14 inicializará el proceso. Pulsando 15 aparecerán los siguientes ajustes:



Opción 17: El radio que se aplica para calcular la normal de un punto. Todos los puntos dentro de este radio serán tenidos en cuenta en el cálculo.

Opción 18: Máxima profundidad del árbol que será utilizado para la reconstrucción.

Opción 19: Mínimo número de puntos de muestra que debería haber en un nodo *octree*. Para modelos con poco ruido se recomiendan valores entre 1 y 5, mientras que para modelos con ruido valores más altos son recomendados.

Pulsando 20 se guarda la configuración y se vuelve a mostrar la nube de puntos.

Una vez realizada la reconstrucción se mostrará el modelo calculado. Si no se está satisfecho con el resultado se puede volver con el botón 22. Para guardar el modelo solamente hay que pulsar el botón 21.



Anexo II - Instalación y configuración

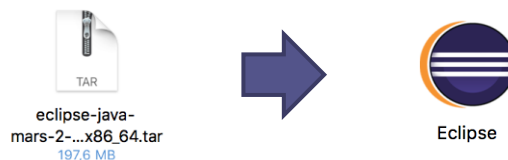
En este apartado se va a explicar la instalación y configuración del entorno de desarrollo utilizado en este proyecto. Dado que se utilizan varias herramientas puede resultar complejo hacerlas funcionar en armonía, por lo que aquí se resume cómo se ha conseguido esto. Los pasos aquí mostrados corresponden al sistema operativo OSX en su versión 10.11 El Capitan.

1. Configuración de Eclipse para Android

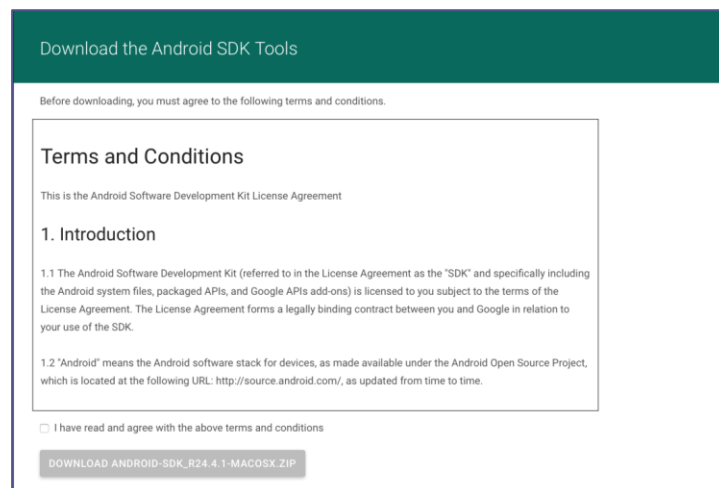


En primer lugar hay que descargar el entorno IDE, en este caso Eclipse. Para esto hay que dirigirse a [2] y seleccionar el sistema operativo correspondiente. A continuación se procede a descargar la versión de Eclipse para Java.

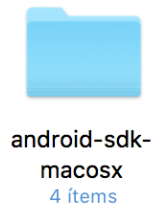
Esto descargará un archivo comprimido, el cual hay que descomprimir para encontrarse con el lanzador de Eclipse.



A continuación es necesario descargar el SKD de Android, el cual se puede encontrar en [1] en la sección “*Get just the command line tools*”. Al presionar el botón de descarga aparecerá una pantalla indicando los términos de uso.



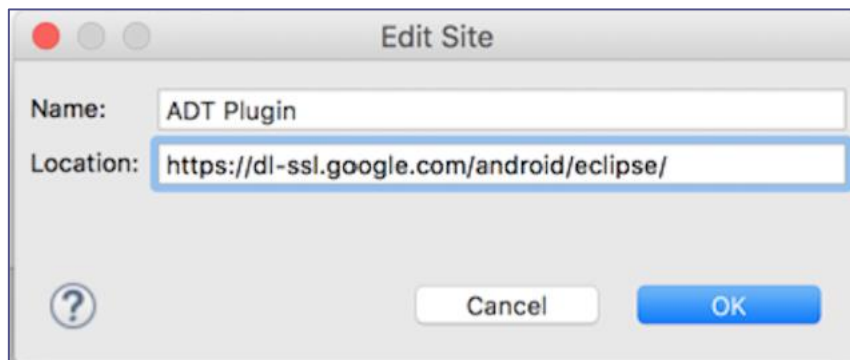
Una vez descargado el SDK, aparecerá una carpeta en nuestro ordenador. Esta carpeta hay que guardarla en una ubicación adecuada ya que contiene todo lo necesario para el desarrollo en Android. En este caso se ha creado una ubicación específica para almacenar todos los ficheros relacionados con el proyecto.



El siguiente paso es abrir Eclipse y seleccionar el *workspace*, o lo que es lo mismo, el sitio donde se quiera guardar los proyectos realizados con Eclipse. Una vez se ha desplegado la interfaz, hay que dirigirse en el menú hasta

Help > Install New Software

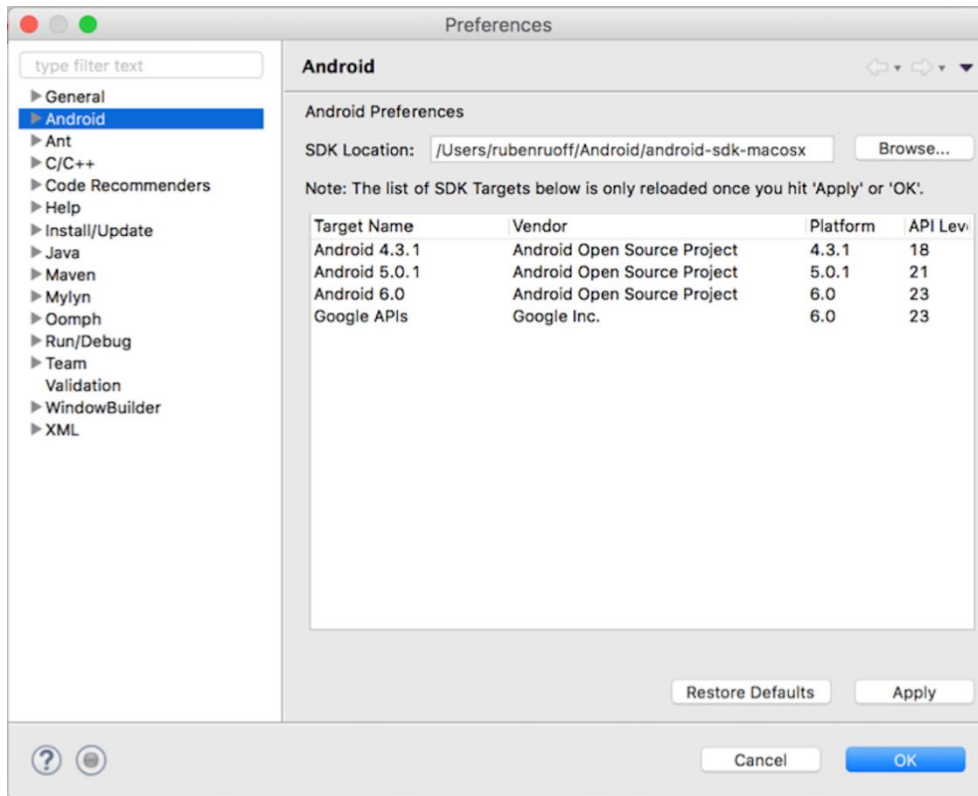
En la siguiente ventana hay que presionar el botón “Add” en la esquina superior derecha y rellenar los campos como se muestran a continuación:



Después de instalar el *plugin* y reiniciar Eclipse hay que indicarle dónde se encuentra el SDK bajado anteriormente. Esto se consigue bajo

Eclipse > Preferencias > Android > SDK Location

Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android



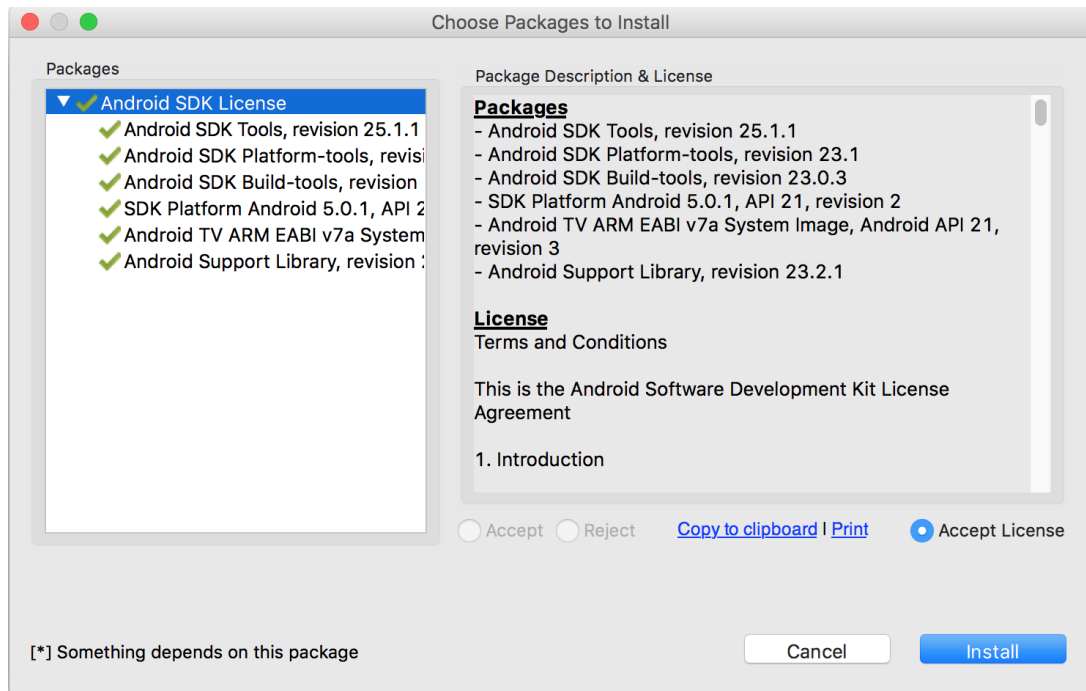
Aparecerá una ventana indicando que faltan componentes por instalar, lo cual se puede hacer abriendo el SDK Manager pulsando el botón que nos muestra. También se puede acceder mediante el menú seleccionando

Window > Android SDK Manager

Seleccionando la API que se quiera usar junto a las herramientas de plataforma y construcción, se procede a instalarlos.

Name	API	Rev.	Status
Android SDK Tools		24.4.1	Update available: rev. 25.1.
Android SDK Platform-tools		23.1	Not installed
Android SDK Build-tools		23.0.3	Not installed
Android SDK Build-tools		23.0.2	Not installed
Android SDK Build-tools		23.0.1	Not installed
Android SDK Build-tools		22.0.1	Not installed
Android SDK Build-tools		21.1.2	Not installed
Android SDK Build-tools		20	Not installed
Android SDK Build-tools		19.1	Not installed

Android 5.0.1 (API 21)			
SDK Platform	21	2	Not installed
Android TV ARM EABI v7a Svstem Imaae	21	3	Not installed
Android TV Intel x86 Atom Svstem Imaae	21	3	Not installed
Android Wear ARM EABI v7a Svstem Imaae	21	3	Not installed
Android Wear Intel x86 Atom Svstem Imaae	21	3	Not installed
ARM EABI v7a Svstem Imaae	21	3	Not installed
Intel x86 / Android SDK Platform 5.0.2	21	3	Not installed
Intel x86 / Revision 3	21	3	Not installed
Goaale AI Requires SDK Platform Android API 21	21	10	Not installed
Goaale AI Size: 177,9 MiB	21	10	Not installed
Goaale AI Provided by dl.google.com	21	10	Not installed
Goaale APIs	21	1	Not installed
Sources for Android SDK	21	1	Not installed



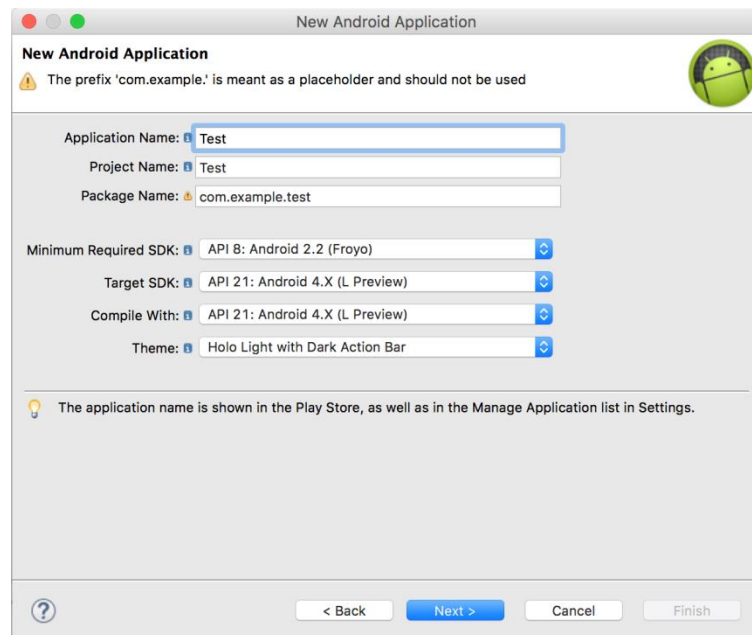
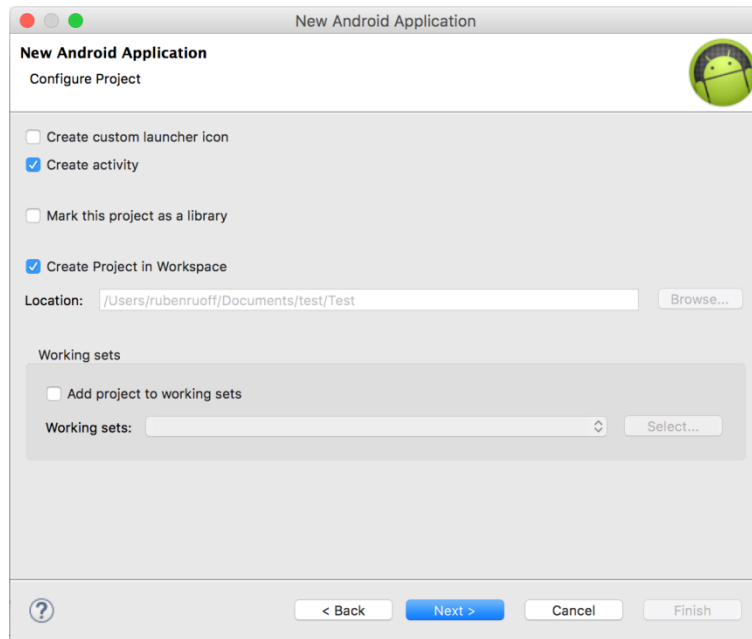
Terminado este proceso ya está correctamente instalado y configurado el SDK de Android.

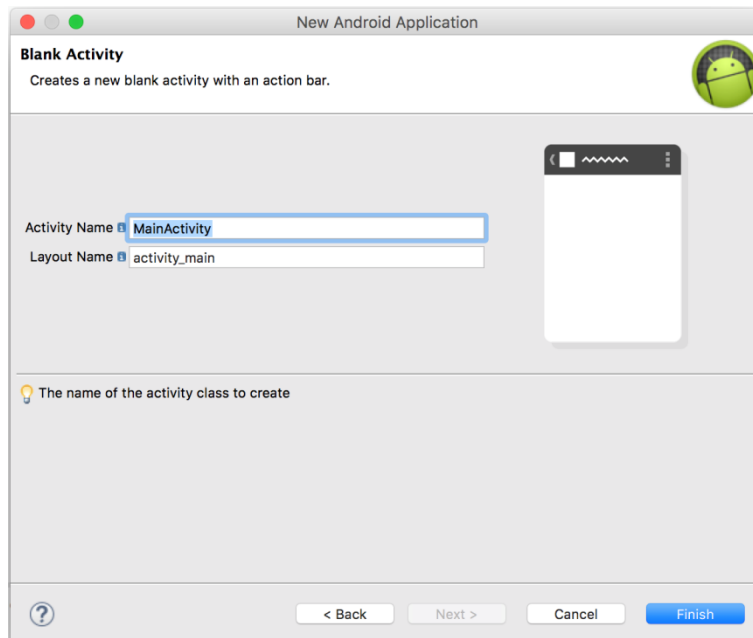
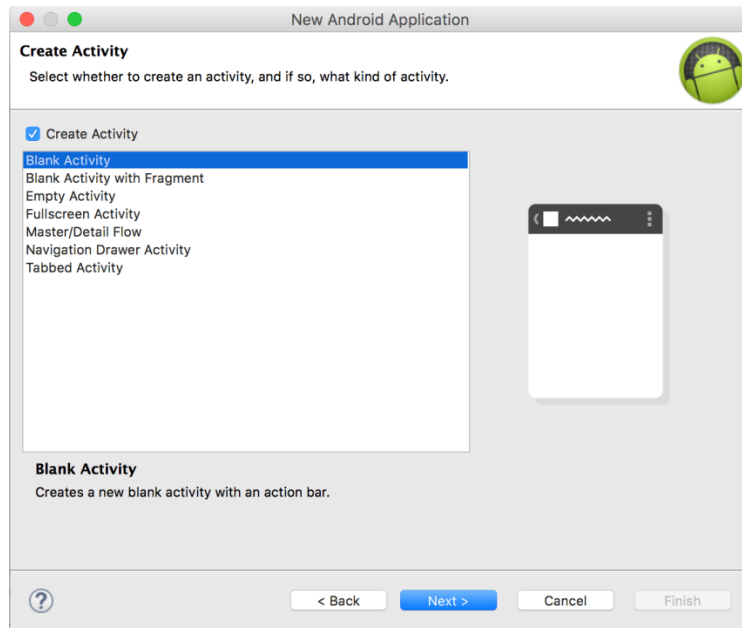
Ahora se puede crear un proyecto Android yendo a:

File > New > Project

Aquí hay que desplegar la opción “Android” y seleccionar “*Android Application*” Project. En la siguiente ventana hay que introducir en nombre de la aplicación así como la versión del SDK con la que se quiere trabajar. La opción “*Minimum Required SDK*” indica la versión más baja soportada por la aplicación, esta opción es interesante para dar un mayor soporte a dispositivos más viejos. Se avanza creando una actividad en blanco y pulsando sobre el botón “*Finish*”. Eclipse creará todos los archivos necesarios y mostrará en el “*Package Explorer*” el contenido del proyecto.

Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android





Cabe destacar que la versión de *Android SDK Tools* y la versión de *Project Build Target* tienen que ser las mismas, si no, se produce uno o varios errores como el que se muestra a continuación:

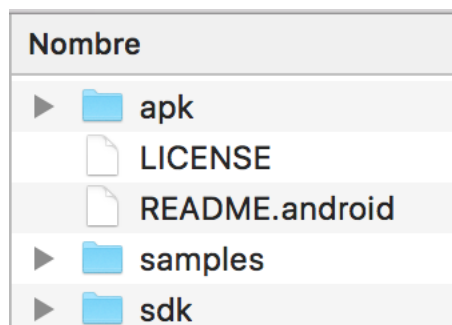
No resource found that matches the given name 'android:Widget.Material.Button.Colored'.

2. Añadir OpenCV para Android a Eclipse

El siguiente paso es añadir OpenCV al proyecto Android creado anteriormente. Para ello hay que dirigirse a [3] y descargar la versión “*OpenCV for Android*”.



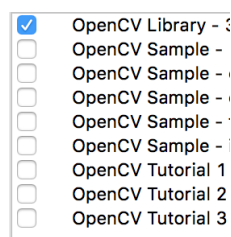
Una vez descargado, aparecerá la carpeta correspondiente en nuestro ordenador, la cual habrá que guardar en una ubicación fácil de localizar.



Para incluir esta librería en Eclipse, hay que dirigirse a:

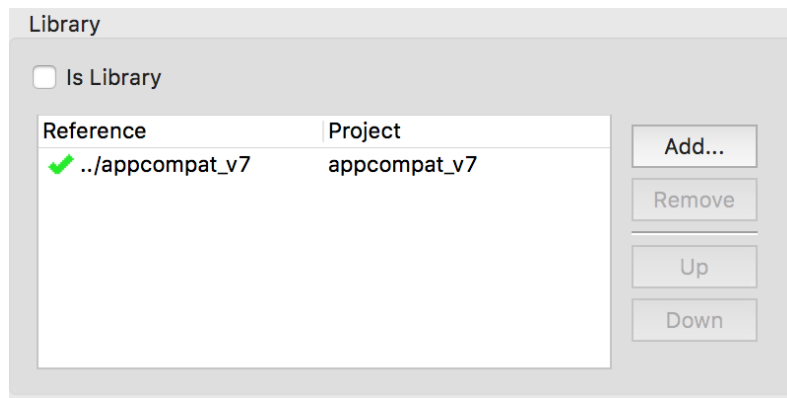
File > Import > General > Existing Project into Workspace

En la opción “*Select root directory*” hay que indicar la ubicación de la carpeta descargada anteriormente. Al pulsar el botón abrir aparecerá junto a la librería una serie de ejemplos y tutoriales. Seleccionando solamente la librería se pulsa el botón de “*Finish*” y OpenCV será añadido al entorno de desarrollo.

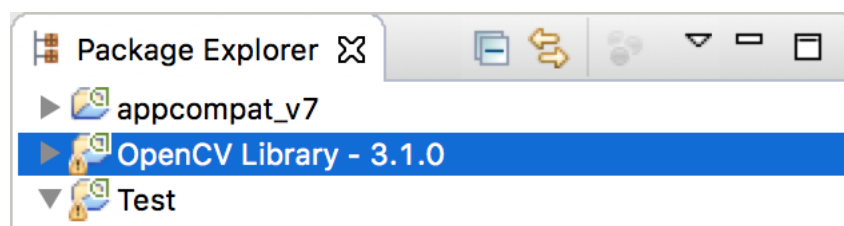
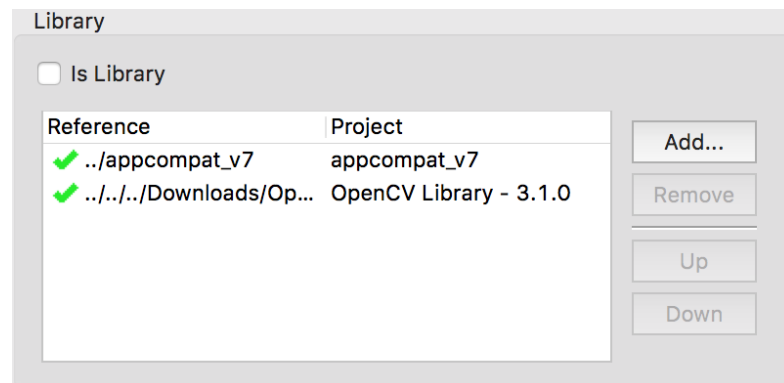


El último paso para añadir OpenCV al proyecto es dirigirse a las propiedades del proyecto. Para esto se presiona el botón secundario sobre el proyecto y seleccionando

“*Properties*” en el menú que aparece. En esta ventana hay que centrarse en la sección “*Library*”:



Presionando el botón “*Add...*” y seleccionando la librería OpenCV, esta se añadirá con éxito al proyecto.



3. Configurar el NDK y OpenCV nativo en Eclipse

El primer paso es descargarse el NDK de Android desde [4], al igual que con el SDK aparecerá unos términos que hay que aceptar para poder usar esta herramienta.

Download the Android NDK

Before installing the Android NDK, you must agree to the following terms and conditions.

Terms and Conditions

This is the Android Software Development Kit License Agreement

1. Introduction

1.1 The Android Software Development Kit (referred to in the License Agreement as the "SDK" and specifically including the Android system files, packaged APIs, and Google APIs add-ons) is licensed to you subject to the terms of the License Agreement. The License Agreement forms a legally binding contract between you and Google in relation to your use of the SDK.

1.2 "Android" means the Android software stack for devices, as made available under the Android Open Source Project, which is located at the following URL: <http://source.android.com/>, as updated from time to time.

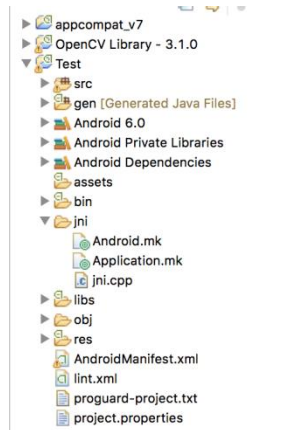
I have read and agree with the above terms and conditions

[DOWNLOAD ANDROID-NDK-R11C-DARWIN-X86_64.ZIP](#)

Una vez aceptado estos términos y descargado el archivo, habrá que guardar la carpeta creada en una ubicación accesible.

Nombre
▶ build
CHANGELOG.md
ndk-build
ndk-depends
ndk-gdb
ndk-stack
ndk-which
▶ platforms
▶ prebuilt
▶ python-packages
source.properties
▶ sources
▶ toolchains

Antes de empezar a construir la parte nativa, hay que crear la estructura de directorios correspondiente. Para ello hay que crear una carpeta llamada "jni" en la raíz del proyecto. Dentro hay que crear dos archivos, "Android.mk" y "Application.mk".



El primero de estos es el encargado de compilar código fuente C++ mientras que el segundo suele ser opcional, pero aporta ciertas funcionalidades bastante útiles. A continuación se muestra un ejemplo muy simple de cada archivo:

Android.mk

```
1 LOCAL_PATH := $(call my-dir)
2
3 include $(CLEAR_VARS)
4
5 LOCAL_MODULE := test
6 LOCAL_SRC_FILES := jni.cpp
7 LOCAL_LDLIBS += -llog -ldl
8
9 include $(BUILD_SHARED_LIBRARY)
```

Application.mk

```
1 APP_STL := gnu STL static
2 APP_CPPFLAGS := -frtti -fexceptions
3 APP_ABI := armeabi-v7a
```

Las dos primeras líneas del archivo *Android.mk* son obligatorias. La línea cinco indica el nombre que tendrá el módulo, lo cual es necesario saber para poder llamarlo desde la parte Java. Línea seis indica todos los archivos que tienen que ser compilados para este módulo. Finalmente con la línea nueve se crea una librería conteniendo el módulo especificado anteriormente.

La parte más importante de *Application.mk* es la línea 3, donde se especifica la arquitectura para la cual se quiera compilar. Lo más recomendable es poner esta opción a “*all*” para aumentar la compatibilidad de la aplicación.

Para que el NDK funcione y se pueda establecer una comunicación entre Java y C++, hay que llamar al módulo compilado mediante el *Android.mk* y definir la función nativa en la parte de Java:

```
static {  
    System.loadLibrary("test");  
}  
  
public native int testNDK();
```





En la parte C++, hay que declarar el origen de donde es llamada de la siguiente forma:

```
extern "C" {  
  
    JNIEXPORT jint JNICALL Java_com_example_test_MainActivity_testNDK(JNIEnv *env, jobject obj);  
  
    JNIEXPORT jint JNICALL Java_com_example_test_MainActivity_testNDK(JNIEnv *env, jobject obj) {  
  
        int conv;  
        jint retVal;  
  
        conv = 1;  
        retVal = (jint)conv;  
  
        return retVal;  
    }  
}
```

En este ejemplo, lo único que se hace es devolver un número entero a la parte Java. Como se puede observar, Eclipse indica que hay muchos errores de sintaxis aunque realmente este bien. Para remediar esto hace falta configuración adicional, por lo que hay que dirigirse a:

Properties > C/C++ General > Paths and Symbols

e incluir lo siguiente:

Languages	Include directories
Assembly	 <code>\${NDKROOT}/platforms/android-9/arch-arm/usr/include</code>
GNU C	 <code>\${NDKROOT}/sources/cxx-stl/gnu-libstdc++/4.9/include</code>
GNU C++	 <code>\${NDKROOT}/sources/cxx-stl/gnu-libstdc++/4.9/libs/armeabi-v7a/include</code>
	 <code>/Users/rubenuoff/Downloads/OpenCV-android-sdk/sdk/native/jni/include</code>

Cabe destacar que dependiendo de la versión de NDK que se utilice, cambian las versiones a incluir, por lo que es recomendable asegurarse antes de que versión se disponga. Una vez hecho esto, los errores de sintaxis habrán desaparecido.

Para poder utilizar la librería OpenCV nativa, basta con incluir el archivo *OpenCV.mk* en el archivo *Android.mk*:


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

include $(CLEAR_VARS)
include /Users/rubenruoff/Downloads/OpenCV-android-sdk/sdk/native/jni/OpenCV.mk

LOCAL_MODULE     := test
LOCAL_SRC_FILES  := jni.cpp
LOCAL_LDLIBS +=  -llog -ldl

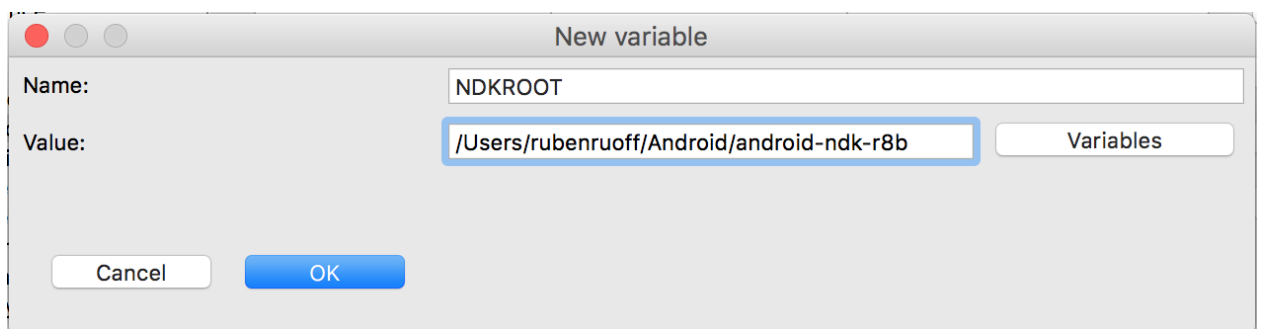
include $(BUILD_SHARED_LIBRARY)
```

Para construir la parte nativa, lo más recomendado es utilizar el *CDT Builder* integrado en Eclipse con el *ADT Plugin*. Antes que nada, hay que definir la variable "NDKROOT" como variable de entorno. En OSX esto se realiza añadiendo la siguiente línea al archivo "~/.bash_profile" :

```
export NDKROOT=/ubicación/del/NDK
```

También es posible establecer esta variable dentro del de Eclipse, pero será necesario realizar para cada nuevo "workspace". Para ello hay que dirigirse a:

```
Eclipse > Preferences > C/C++ > Build > Environment > Add...
```



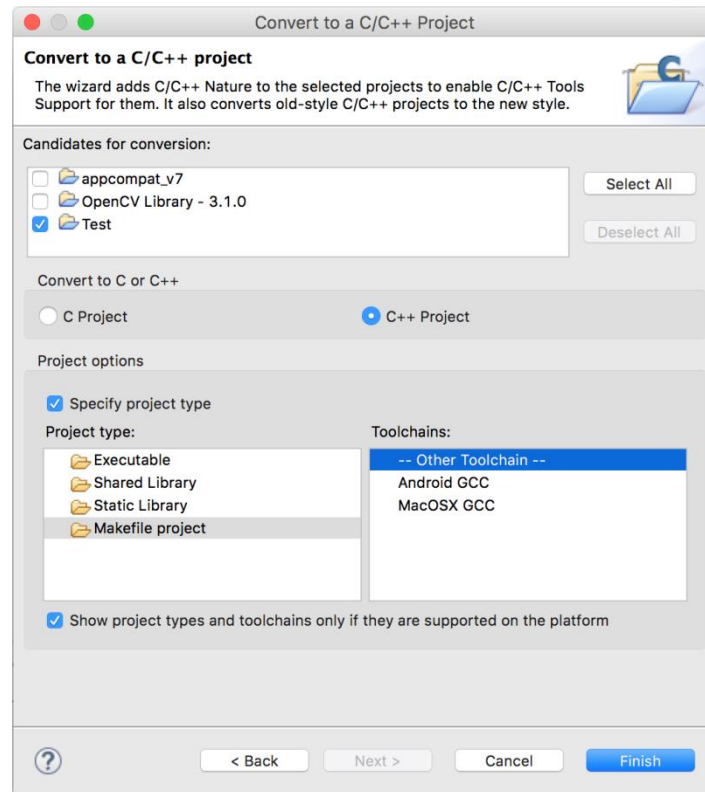
Después de realizar esto, es necesario reiniciar Eclipse para que los cambios sean aplicados.

El siguiente paso es aplicar naturaleza C/C++ al proyecto. Esto se consigue dirigiéndose a:

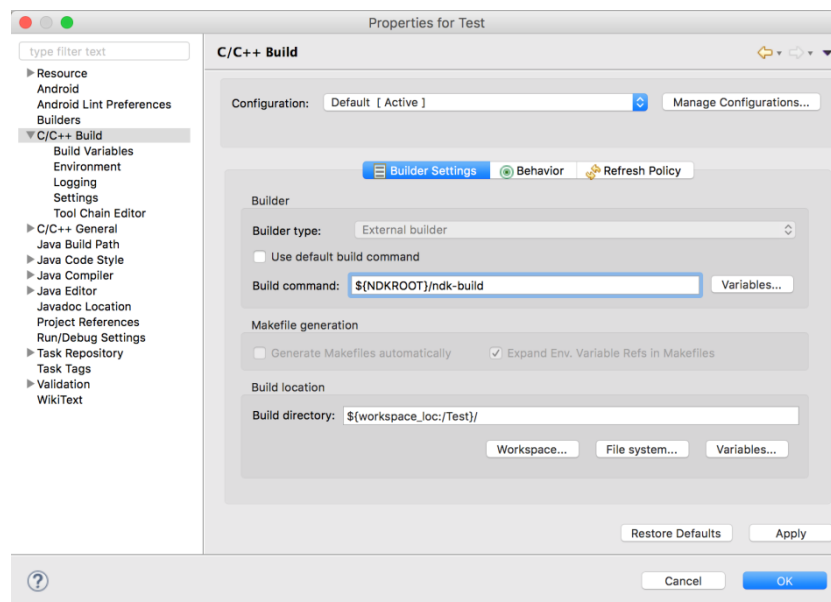
```
New > Other > C/C++ > Convert to a C/C++ Project
```

La ventana que aparece hay que configurarla como sigue:

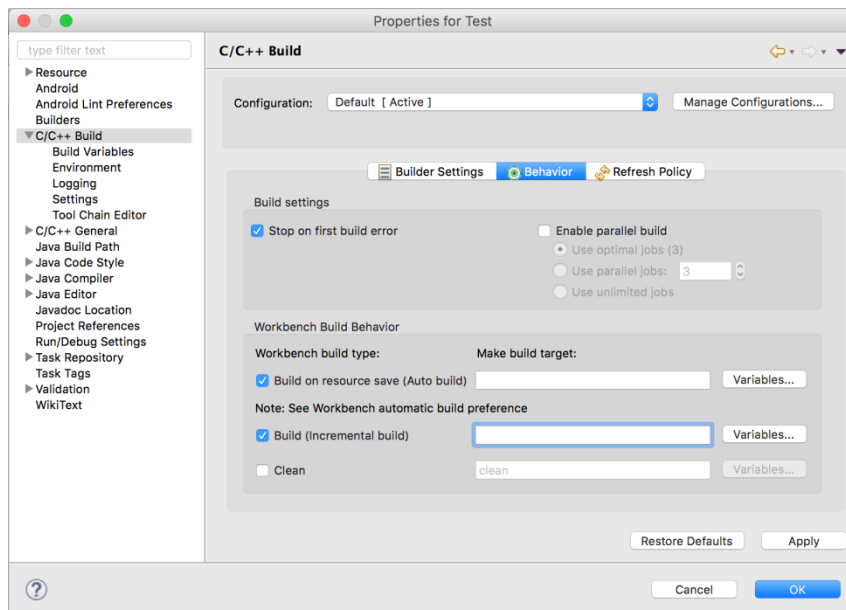
Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android



Una vez haya terminado el proceso, hay que dirigirse a las propiedades del proyecto y bajo “C/C++ Build” hay que deseleccionar la opción “Use default build command” y cambiar el comando de construcción de “make” a “`${NDKROOT}/ndk-build`”, como se muestra a continuación:



Bajo la pestaña “Behavior” en la sección “Workbench Build Behavior” hay que modificar los parámetros como sigue:



Al presionar el botón “OK” el “*ndk-build*” será invocado y construirá el proyecto, indicando un mensaje como el siguiente si todo ha salido según lo previsto:

```
CDT Build Console [Test]
13:55:04 **** Auto Build of configuration Default for project Test ****
/Users/rubenruoff/Android/android-ndk-r8b/ndk-build
Compile++ thumb : test <= jni.cpp
In file included from jni/jni.cpp:1:0:
/Users/rubenruoff/Android/android-ndk-r8b/platforms/android-14/arch-arm/usr/include/jni.h:592:13
SharedLibrary : libtest.so
Install : libtest.so => libs/armeabi-v7a/libtest.so

13:55:04 Build Finished (took 279ms)
```

4. Compilar y añadir la librería de PCL

Para la compilación de esta librería se ha empleado un script obtenido de [6]. El primer paso es declarar la variable de entorno `ANDROID_NDK` indicando la ubicación del NDK de Android. Una vez hecho esto, hay que crear una carpeta donde se quiera guardar la librería, navegar hacia ella con la terminal de OSX e introducir lo siguiente:

```
git clone https://github.com/hirotakaster/pcl-superbuild.git
```

Esto descargará todo lo necesario para proceder con la compilación de la librería. A continuación hay que dirigirse a la carpeta “*pcl-superbuild*” y crear una nueva carpeta llamada “*build*”:

```
cd pcl-superbuild && mkdir build && cd build
```

Ahora ya se puede proceder a preparar los archivos para su posterior compilación con:

```
cmake ../
```

Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

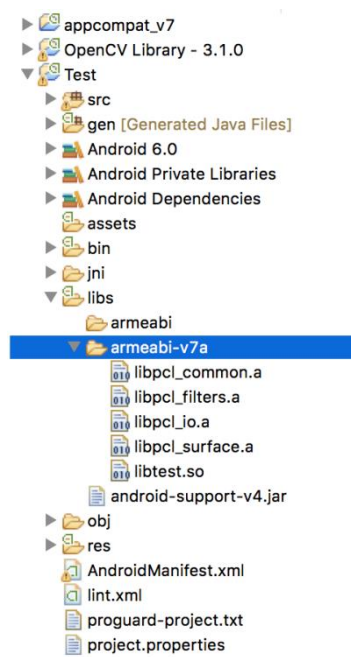
El último comando se encargará de descargar y compilar todas las dependencias de la librería:

```
make -j4
```

Una vez terminado esto, ya estarán compilados los archivos necesarios para incluir PCL en Android. Para añadir estos archivos al entorno de desarrollo hay que dirigirse primeramente desde la misma carpeta “*build*” a:

```
CMakeExternals > Install > pcl-android > lib
```

Aquí hay que seleccionar todos los módulos de la librería que sean necesarios, copiándolos a la carpeta “*libs*” en el proyecto de Eclipse.



Para indicarle a Eclipse que incluya estas librerías en la aplicación hay que añadir un par de líneas al archivo *Android.mk*. La siguiente ilustración muestra cómo se añade el módulo común de PCL localizado en la carpeta “*libs/pcl*” :

```
include $(CLEAR_VARS)
LOCAL_MODULE := libpclcommon
LOCAL_SRC_FILES := ../libs/pcl/libpcl_common.a
include $(PREBUILT_STATIC_LIBRARY)
```

Al crear la librería compartida que incluye a todas las librerías estáticas y el código nativo, hay que indicar dónde se encuentran las cabeceras de estas librerías:

```
LOCAL_C_INCLUDES += /Users/rubenruoff/Android/pcl-superbuild/build/CMakeExternals/Install/pcl-android/include/pcl-1.6/
LOCAL_C_INCLUDES += /Users/rubenruoff/Android/pcl-superbuild/build/CMakeExternals/Install/eigen/
LOCAL_C_INCLUDES += /Users/rubenruoff/Android/pcl-superbuild/build/CMakeExternals/Install/boost-android/include
LOCAL_C_INCLUDES += /Users/rubenruoff/Android/pcl-superbuild/build/CMakeExternals/Install/flann-android/include
```

También hay que indicar las librerías estáticas que se desean incluir:

```
LOCAL_STATIC_LIBRARIES := libpclcommon libpclfilter libpclsearch libpclkdtree libpclsurface libpclfeatures libpclio
```

Con esto ya se puede utilizar la librería en Android, para minimizar el tamaño del ejecutable solamente se han empleado los módulos necesarios, por lo que si se desea ampliar la aplicación e implementar una funcionalidad que no esté disponible en los módulos seleccionados habrá que añadir el/los módulo/s necesario/s.

Bibliografía y recursos

[1] Android SDK

<https://developer.android.com/studio/index.html#downloads>

[Febrero 2015]

[2] Eclipse IDE

<https://www.eclipse.org/downloads/>

[Febrero 2015]

[3] OpenCV for Android

<http://opencv.org/downloads.html>

[Febrero 2015]

[4] Android NDK

<http://developer.android.com/intl/es/ndk/downloads/index.html>

[Febrero 2015]

[5] How to compile PCL for Android

<https://www.hirotakaster.com/weblog/how-to-build-pcl-for-android-memo/>

[Enero 2016]

[6] PCL compilation script

<https://github.com/hirotakaster/pcl-superbuild>

[Enero 2016]

[7] How to install ADT plug-in for Eclipse

<http://stackoverflow.com/questions/34301997/how-to-install-last-version-of-adt-plugin-for-eclipse>

[Febrero 2015]

[8] Configure native OpenCV with Android NDK for Eclipse

http://docs.opencv.org/2.4/doc/tutorials/introduction/android_binary_package/android_dev_intro.html

[Marzo 2015]

[9] SfM with OpenCV

<http://www.morethantechnical.com/2012/02/07/structure-from-motion-and-3d-reconstruction-on-the-easy-in-opencv-2-3-w-code/>

[Febrero 2015]

[10] Learn OpenGL ES for Android

<http://www.learnopengles.com/>

[Marzo 2016]

[11] No resource found that matches given name solution

<http://stackoverflow.com/questions/32075498/error-retrieving-parent-for-item-no-resource-found-that-matches-the-given-name>

[Marzo 2015]

[12] Hartley, Zisserman – Multiple View Geometry in Computer Vision

[Diciembre 2015]

[13] Roy Shilkrot – Mastering OpenCV with Practical Computer Vision Projects

[Diciembre 2015]

[14] Kevin Brothaler – OpenGL ES 2 for Android

[Febrero 2016]



Aplicación para la obtención de un mapa 3D de una escena con un dispositivo Android

[15] Joseph Howse – Android Application Programmig with OpenCV

[Febrero 2015]

[16] Compile SSBA-3.0 for Android

<http://stackoverflow.com/questions/35462857/compile-ssba-3-0-for-android>

[Marzo 2016]

[17] Ceres-solver

<http://ceres-solver.org/>

[Marzo 2016]

[18] Removing outliers using a StatisticalOutlierRemoval filter

http://pointclouds.org/documentation/tutorials/statistical_outlier.php

[Febrero 2016]

[19] Surface reconstruction with PCL using the Poisson algorithm

<http://www.pointclouds.org/assets/icra2012/surface.pdf>

[Febrero 2016]

[20] 123D Catch

<https://play.google.com/store/apps/details?id=com.autodesk.Catch&hl=es>

[Febrero 2015]

[21] ArgoUML

<http://argouml.tigris.org/>

[Enero 2016]

[22] Triangulation from Hartley and Sturm

<https://users.cecs.anu.edu.au/~hartley/Papers/triangulation/triangulation.pdf>



[Febrero 2016]

[23] Parche error libstdc++3

<https://svn.boost.org/trac/boost/attachment/ticket/6165/libstdc++3.hpp.patch>

[Marzo 2016]