



Universidad  
de La Laguna

---

## **Análisis de rendimiento de aplicaciones Android**

*Performance analysis of Android applications*

Autor: Alberto Pérez Reyes

Directores: Francisco Almeida Rodríguez, Alejandro Acosta Díaz

Departamento de Informática

Escuela Técnica Superior de Ingeniería Informática

Trabajo de Fin de Grado

---

La Laguna, 10 de junio de 2014



D. **Francisco Almeida Rodríguez**, con N.I.F. 42.831.571-M Catedrático de Universidad y D. **Alejandro Acosta Díaz**, con N.I.F 78.852.786-T Ingeniero Informático, adscritos al Departamento de Informática de la Universidad de La Laguna.


## **C E R T I F I C A**

Que la presente memoria titulada:

*“Análisis de rendimiento de aplicaciones Android.”*

Ha sido realizada bajo su dirección por D. Alberto Pérez Reyes, con N.I.F. 42.220.176-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de Junio de 2012



Francisco Almeida



Alejandro Acosta



## **Agradecimientos**

En primer lugar quería agradecer a mis padres y al resto de mi familia por todo el ánimo y apoyo incondicional que me han estado prestando durante todos estos años. Aunque ha sido un camino largo y duro, ellos siempre han estado ahí, elevándome la moral en cada uno de mis fracasos, celebrando cada uno de mis éxitos y por supuesto llamando la atención cuando detectaban algún largo periodo de inactividad intelectual o algún desvío de la meta que me había propuesto. Gracias especiales a mi hermano Sergio, con el que he compartido muchos domingos de biblioteca, con la madrugada correspondiente para conseguir un sitio donde poder estar.

También quería agradecer a mi novia Haidée, sin la que estos últimos años no hubiese podido dedicarle tanto esfuerzo a terminar. Ella siempre me ha apoyado, animado a seguir y acompañado en tantos días de estudio. Sin ella hoy no estaría aquí.

Gracias a los muchos amigos que he hecho en esta carrera y sin algunos de ellos todavía estaría intentando sacar alguna práctica. De manera muy especial muchas gracias a Pedro, Alberto, Castor, Yauxi, Marcos, Alexandra e Iván por su inestimable y desinteresada ayuda y por compartir conmigo algo más que apuntes, prácticas y conocimientos.

Y por supuesto, agradecer a todos los profesores, incluidos los de 4.9, 4.8, 4.75... por su afán para que todos los alumnos consiguiéramos adquirir los conocimientos y habilidades necesarias para poder desenvolvemos dignamente en el mundo profesionales dándole, en algunos casos, más mérito al interés e ilusión que a jugarte toda una asignatura en un sólo examen. En particular me gustaría agradecer a Casiano, por la libertad que ha dado para desarrollar y usar proyectos personales en sus clases. A Candelaria, porque ha demostrado que la bondad, la ayuda y las buenas intenciones son el mejor camino para que el alumno aprenda sin miedo a equivocarse. A Dagoberto, porque me ha enseñado que un profesor y un alumno también pueden tomarse un café y hablar. A Roda, porque siempre tiene un hueco para ayudar, con cualquier problema. A Juan Carlos, por nunca cansarse de repetir

los conceptos hasta que todo el mundo los entendiera. En definitiva, grandes profesionales que le dan mucho valor a esta carrera y a esta Universidad.

Por último y no por ello menos importante, quiero agradecer a mi Director de proyecto, Francisco Almeida por sus consejos, ayuda, comprensión y guía durante todo este tiempo. Y de manera muy especial al codirector Alejandro Acosta, por la grandísima ayuda que me ha brindado y por tener siempre un hueco para ayudarme cada vez que me sentía perdido. Ante ambos disculparme por la lo pesado que he podido ser, pero les aseguro que sólo me movía el afán de que este proyecto saliera lo más digno posible.

En definitiva, y repito, muchas gracias a todos.

## **Resumen**

*El objetivo de este trabajo ha sido el desarrollo, y posterior análisis de rendimiento, de aplicaciones Android. Para ello se ha planteado como caso de estudio el algoritmo de procesamiento de imágenes conocido como la “Transformada de Hough”.*

*Tras una fase de análisis hemos optado finalmente por adaptar a la plataforma Android un código base disponible en [1] para la detección de líneas en imágenes digitales. Esta adaptación implicaría, necesariamente, familiarizarse con el entorno Android siguiendo las pautas descritas en la página oficial de desarrollo de la plataforma.*

*Para el análisis de rendimiento en Android, se siguieron las técnicas y herramientas descritas en la página oficial de desarrollo, que permitieron comprobar el tiempo de ejecución, las funciones computacionalmente más costosas y la memoria consumida.*

*Una vez familiarizado con la detección de líneas, se decidió ampliar el estudio a la detección de círculos en imágenes digitales, adaptando y mejorando, igual que en el procedimiento anterior, el código base disponible en [2] para su uso en Android.*

*Durante el estudio de las aplicaciones, se crearon nuevas versiones que mejoraban progresivamente los resultados obtenidos. Cuando se consiguió la versión más rápida usando código secuencial, se procedió a paralelizar la función o funciones computacionalmente más costosas usando Renderscript.*

*Por último, se ha hecho uso del framework Paralldroid, desarrollado en el grupo de Computación de Altas Prestaciones de la Universidad de La Laguna, como parte de la tesis doctoral de D. Alejandro Acosta. Paralldroid permite paralelizar código Android automáticamente mediante anotaciones en el código Java.*

*El proyecto finaliza con un análisis comparativo de los resultados obtenidos. También se muestran las conclusiones acerca de los aspectos más relevantes, al desarrollar en dispositivos Android, que afectan a la velocidad de procesamiento y a la memoria de la aplicación.*

## **Palabras clave**

Java, Android, Renderscript, Paralldroid, Paralelismo, Análisis de Rendimiento, Desarrollo en Móviles.



## **Abstract**

*The aim of this work has been the development and subsequent performance analysis of Android applications. It has been presented as a case of study an image processing algorithm known as the "Hough Transform".*

*After the analysis of several implementation options, we finally decided to adapt to the Android platform an available code base [1] to detect lines in digital images. This adaptation would necessarily imply being familiar with the Android environment, following the guidelines outlined in the official website of platform development.*

*For performance analysis in Android, techniques and tools described in the official development page were followed, which allowed to test the execution time, the most expensive computational functions and the memory consumed.*

*Once we were familiar with the detection of lines, we decided to extend the study to detect circles in digital images, adapting and improving the base code available in [2] to be used in Android, as we did for the procedure presented before.*

*During the study of the applications, new versions that progressively improved the results were created. When the fastest version was achieved using sequential code, we proceeded to parallelize the most expensive computationally function or functions using Renderscript.*

*Subsequently Paralldroid framework, developed in the group of High Performance Computing at the University of La Laguna, as part of the doctoral thesis of Alejandro Acosta was used. Paralldroid Android allows to parallelize code automatically via annotations in the Java code.*

*The project ends with a comparative analysis of the results obtained. The conclusions about the most important aspects are also shown, when developing on Android devices, affecting the processing speed and memory aspects of the application.*

**Keywords**

Java, Android, RenderScript, Paralldroid, Parallelism, Performance Analysis, Mobile Development.

## Índice general

Capítulo 1.....	17
Introducción.....	17
Capítulo 2.....	19
Introducción a Android.....	19
Modelos de desarrollo en Android.....	19
Android SDK .....	20
Android NDK.....	20
Renderscript .....	20
Paralldroid.....	20
Ciclo de creación de la app en Android.....	21
Gestión de memoria y recursos en Android.....	23
Capítulo 3.....	25
Herramientas para el análisis de rendimiento en Android .....	25
Monitor de sistema.....	28
Capítulo 4.....	31
Transformada de Hough .....	31
Historia .....	31
Detección de líneas rectas.....	31
Curvas paramétricas.....	33
Capítulo 5.....	35
Aplicaciones .....	35
Aplicaciones Java .....	35
Imágenes de prueba para las aplicaciones Java .....	35
Aplicaciones Renderscript .....	36
Aplicaciones OpenCV .....	37
Capítulo 6.....	45
Resultados.....	45
Conclusiones .....	67
Bibliografía.....	71



## Índice de Figuras

Figura 1: Fases de compilación en Android. ....	21
Figura 2: Ciclo extendido de ejecución en Android. ....	22
Figura 3: Zoom sobre panel Eclipse (Arriba) y secciones de trabajo en el DDMS (Abajo). ....	25
Figura 4: Zoom sobre la vista de dispositivos conectados. ....	26
Figura 5: Reporte creado después de realizar la traza de una ejecución. ....	27
Figura 6: Aspecto de la aplicación System Monitor. ....	29
Figura 7: Ejemplo de ejecución con Renderscript y System Monitor. ....	30
Figura 8: Imágenes de prueba para la detección de líneas y círculos. ....	36
Figura 9: Imagen de ejemplo de monedas (izda.) e imagen procesada (dcha.) ....	38
Figura 10: Imagen de ejemplo de edificio (izda.) e imagen procesada (dcha.) ....	39
Figura 11: Imágenes con células cancerígenas de Glioblastoma. ....	40
Figura 12: Imágenes procesadas con las células detectadas. ....	41
Figura 13: Detección de líneas en tiempo real. ....	42
Figura 14: Detección de círculos en tiempo real. ....	42
Figura 15: Paso de imagen de color a blanco y negro con Renderscript en tiempo real. ....	43
Figura 16: LG Optimus. ....	45
Figura 17: Nexus 5. ....	46
Figura 18: Gráfica con los resultados de rendimiento de LG Optimus. ....	47
Figura 19: Gráfica con los resultados de rendimiento de Nexus 5. ....	48
Figura 20: Gráfica comparando rendimiento de LG vs Nexus. Línea. ....	48
Figura 21: Gráfica con los resultados de rendimiento de Nexus 5 usando Array. ....	51
Figura 22: Bitmap vs Array en Nexus con RGB 565 y SD en línea. ....	51
Figura 23: Gráfica con resultados mostrando la imagen de la línea. ....	53
Figura 24: Guarda en SD vs mostrar por pantalla, problema línea. ....	53
Figura 25: Gráfica con resultados mostrando la imagen del círculo. ....	54
Figura 26: Funciones más costosas en Java. Detección de líneas. ....	55
Figura 27: Funciones más costosas portadas a Renderscript. Detección de líneas. ....	55
Figura 28: Nuevo código añadido al fichero Java para usar Renderscript. ....	56
Figura 29: Gráfica de resultados usando Renderscript en líneas. ....	57
Figura 30: Versión Java vs versión Renderscript. Línea. ....	57
Figura 31: Gráfica de resultados usando Renderscript en círculos. ....	58
Figura 32: Versión Java vs versión Renderscript. Círculo. ....	59
Figura 33: Gráfica de resultados usando Paralldroid en líneas. ....	60
Figura 34: Versión Renderscript vs versión Paralldroid. Línea. ....	60
Figura 35: Gráfica de resultados usando Paralldroid en círculos. ....	61
Figura 36: Versión Renderscript vs versión Paralldroid. ....	62
Figura 37: Java vs Renderscript vs Paralldroid. Línea. ....	63
Figura 38: Java vs Renderscript vs Paralldroid. Líneas cruzadas. ....	63
Figura 39: Java vs Renderscript vs Paralldroid. Círculo. ....	64
Figura 40: Java vs Renderscript vs Paralldroid. Círculos cruzados. ....	64
Figura 41: Rendimiento Renderscript vs OpenCV. ....	65
Figura 42: Versión Renderscript (sin tiempo de preprocesado) vs OpenCV. ....	66



## Índice de Tablas

Tabla 1: Características LG Optimus.....	45
Tabla 2: Características Nexus 5.....	46
Tabla 3: Resultados rendimiento LG Optimus.....	47
Tabla 4: Resultados rendimiento Nexus 5. ....	47
Tabla 5: Resultados rendimiento Nexus 5 usando Array.....	50
Tabla 6: Resultados de mostrar la imagen de la línea. ....	52
Tabla 7: Resultados Nexus 5 mostrando imagen círculo .....	54
Tabla 8: Resultados de rendimiento usando Renderscript en la detección de líneas. ....	56
Tabla 9: Resultados de rendimiento usando Renderscript en la detección de círculos.....	58
Tabla 10: Resultados de rendimiento usando Paralldroid en la detección de líneas.....	59
Tabla 11: Resultados de rendimiento usando Paralldroid en la detección de círculos.....	61
Tabla 12: Resultados Renderscript vs Opencv.....	65
Tabla 13: Renderscript vs OpenCV .....	66





# CAPÍTULO 1

## INTRODUCCIÓN

Los dispositivos móviles (Smartphones, Tablets, etc...) en estos momentos están siendo ampliamente utilizados, siguen una progresión evolutiva similar a los ordenadores domésticos actuales, haciendo uso de CPUs y GPUs de varios núcleos. Mientras que el hardware sigue un progreso similar a las arquitecturas de computadores, el software aún sigue un modelo de desarrollo lineal similar al de los procesadores de un solo núcleo. Es por ello que nos planteamos hacer uso de las ventajas que el hardware proporciona para mejorar el rendimiento y la eficiencia de las aplicaciones. Esto permitiría hacer un uso más eficiente de los dispositivos móviles, tanto desde el punto de vista de la velocidad de las aplicaciones como del consumo de energía. Es un hecho conocido que uno de los mayores hándicaps de los dispositivos móviles viene asociado a la duración de la batería y al consumo que han alcanzado ciertos elementos, como la pantalla y el procesador.

El desarrollo de software sobre dispositivos móviles bajo Android admite varios modelos de programación. Cada uno de ellos presenta características que proporcionan diferentes aproximaciones a la resolución de problemas. Una misma aproximación puede presentar distintos rendimientos sobre dispositivos diferentes. En este proyecto planteamos portar la *transformada de Hough* [3] a algunos de los distintos modelos de programación sobre Android. Esto permitirá evaluar el rendimiento de las distintas aproximaciones que se pueden realizar sobre el conjunto de dispositivos disponibles. De este modo, en la resolución de un problema, se podrá hacer uso del modelo más eficiente en el dispositivo disponible.



## CAPÍTULO 2

### **INTRODUCCIÓN A ANDROID**

Android es un sistema operativo basado en el kernel de Linux diseñado principalmente para dispositivos móviles con pantalla táctil como Smartphones o Tablets.

Inicialmente fue desarrollado por Android Inc. y posteriormente adquirido por Google en 2005. El primer móvil con el sistema operativo Android fue el HTC Dream y se vendió en octubre de 2008. La versión beta de Android fue lanzada el 5 de noviembre de 2007 y el SDK el 12 de noviembre de 2007. La primera versión comercial del software (Android 1.0) fue lanzada el 23 de septiembre de 2008. La última versión existente es la 4.4.3.

Android hace uso de una máquina virtual para ejecutar sus aplicaciones, Dalvik. La máquina virtual Dalvik (DVM) [4] permite ejecutar aplicaciones programadas en Java (a partir de Java 5). DVM sacrifica la portabilidad que caracteriza a Java, que puede ser ejecutado en cualquier sistema operativo, para poder crear aplicaciones sobre Android con un mejor rendimiento y un menor consumo de energía, características extremadamente importantes en dispositivos móviles, debido a la limitada capacidad de las baterías. DVM está optimizada para requerir poca memoria y está diseñada para permitir ejecutar varias instancias de la máquina virtual simultáneamente, delegando en el sistema operativo subyacente el soporte de aislamiento de procesos, gestión de memoria e hilos.

### **MODELOS DE DESARROLLO EN ANDROID**

La construcción de aplicaciones en Android admite varios modelos de desarrollo. Mediante el Android SDK (Software Development Kit) [5], usando el Android NDK (Native Development Kit) [6] o usando Renderscript [7]. Cada uno de ellos ofrece una serie de ventajas y desventajas que se explicarán a continuación. Así mismo es una propuesta del grupo de Computación de Altas Prestaciones de la Universidad de La Laguna el uso de Paralldroid como un framework de desarrollo que ayudaría a reducir algunas de las desventajas de los anteriores modelos.

## ANDROID SDK

El Android SDK (Software Development Kit) provee de la API (Application Programming Interface) de Android y de las herramientas necesarias para poder desarrollar, probar y depurar aplicaciones en Android. Es el modelo recomendado para cualquier desarrollador que desee iniciarse en aplicaciones en Android. En una versión previa incluía el IDE (Integrated Development Environment) Eclipse con el plugin ADT (Android Developer Tools), actualmente es posible desarrollar con el Android Studio, basado en IntelliJ IDEA que incorporará la última versión de la plataforma Android (4.4.3) y las imágenes de todas las versiones de Android para poder usarlas en un emulador y probar el desarrollo de la aplicación sobre diferentes versiones de Android. El lenguaje de programación para desarrollar con el SDK es Java.

## ANDROID NDK

El Android NDK (Native Development Kit) es un conjunto de herramientas que permiten implementar parte de las aplicaciones usando código nativo de lenguajes como C o C++. Para cierto tipo de aplicaciones esto puede ser muy útil ya que permite la reutilización de código y librerías que ya estuvieran escritas en esos lenguajes. La mayoría de las aplicaciones no necesitan usar el NDK. A diferencia de lo que suele parecer, usar código nativo no incrementa notablemente el rendimiento de la aplicación y por el contrario incrementa considerablemente la complejidad del código.

## RENDERSRIPT

RenderScript es un framework para la ejecución de tareas de cálculo de alto rendimiento en Android. Está orientado para usarlo en aplicaciones paralelas, aunque también puede ser beneficioso para computación secuencial con una alta carga de trabajo. En tiempo de ejecución, RenderScript paraleliza el trabajo entre todos los procesadores disponibles en un dispositivo, como por ejemplo CPUs de varios núcleos, GPU o DSPs. RenderScript es especialmente útil para aplicaciones que realizan procesamiento de imágenes, fotografía computacional, o visión por computador. Para desarrollar en RenderScript se usa versión derivada de C (estándar C99).

## PARALLDROID

Paralldroid es un framework de desarrollo que permite el desarrollo automático en código Nativo C, RenderScript, código secuencial y paralelo para aplicaciones en dispositivos móviles (Smartphones, Tablets,...).

El desarrollador rellena y anota, usando su lenguaje secuencial de alto nivel, las secciones de una plantilla que se ejecutará en lenguaje Nativo y Renderscript. Paralldroid usa la información proporcionada por esas anotaciones para generar un nuevo programa que incorpore las secciones de código que se ejecutarán a través de la CPU o GPU. Paralldroid es un framework que unifica los diferentes modelos de programación de Android.

## CICLO DE CREACIÓN DE LA APP EN ANDROID

La figura 1 describe el orden y las fases por las que pasa un proyecto Android [8] hasta convertirse en una aplicación. Durante la construcción de la aplicación, el proyecto Android es compilado y empaquetado en un fichero .apk, que contiene el binario de la aplicación. Este fichero contiene toda la información necesaria para poder lanzar la aplicación en un dispositivo o emulador.

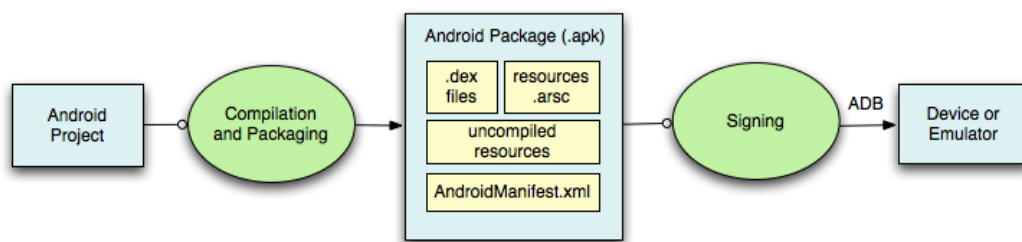


Figura 1: Fases de compilación en Android.

La figura 2 describe como es el proceso completo de compilación de una aplicación en Android. El Android Asset Packaging Tool (aapt) recibe como entrada los recursos de la aplicación, como las imágenes, el AndroidManifest.xml y los documentos XML que definen la interfaz de usuario y los compila, produciendo el fichero R.java que permite referenciar los recursos desde el código Java. El Android Asset Packaging Tool (aidl) convierte cualquier interfaz .aidl en una interfaz de java. Todo el código Java incluyendo los ficheros R.java y .aidl, es compilado por el compilador de Java y produce los ficheros .class. La herramienta dex convierte los ficheros .class en Dalvik byte code [9], un código intermedio para la máquina virtual Dalvik. Cualquier otra librería y fichero .class que esté incluido en el proyecto será convertido en un fichero .dex para que pueda ser empaquetado en el instalador de la aplicación .apk. Una vez que ha sido creado el fichero .apk, debe ser firmado con una llave de depuración o de lanzamiento antes de poder ser instalada en un dispositivo. Para terminar, si la aplicación ha sido firmada en modo de lanzamiento, se debe de usar el Zipalign para alinear el fichero .apk, para reducir el consumo de memoria cuando la aplicación se ejecuta en un dispositivo.

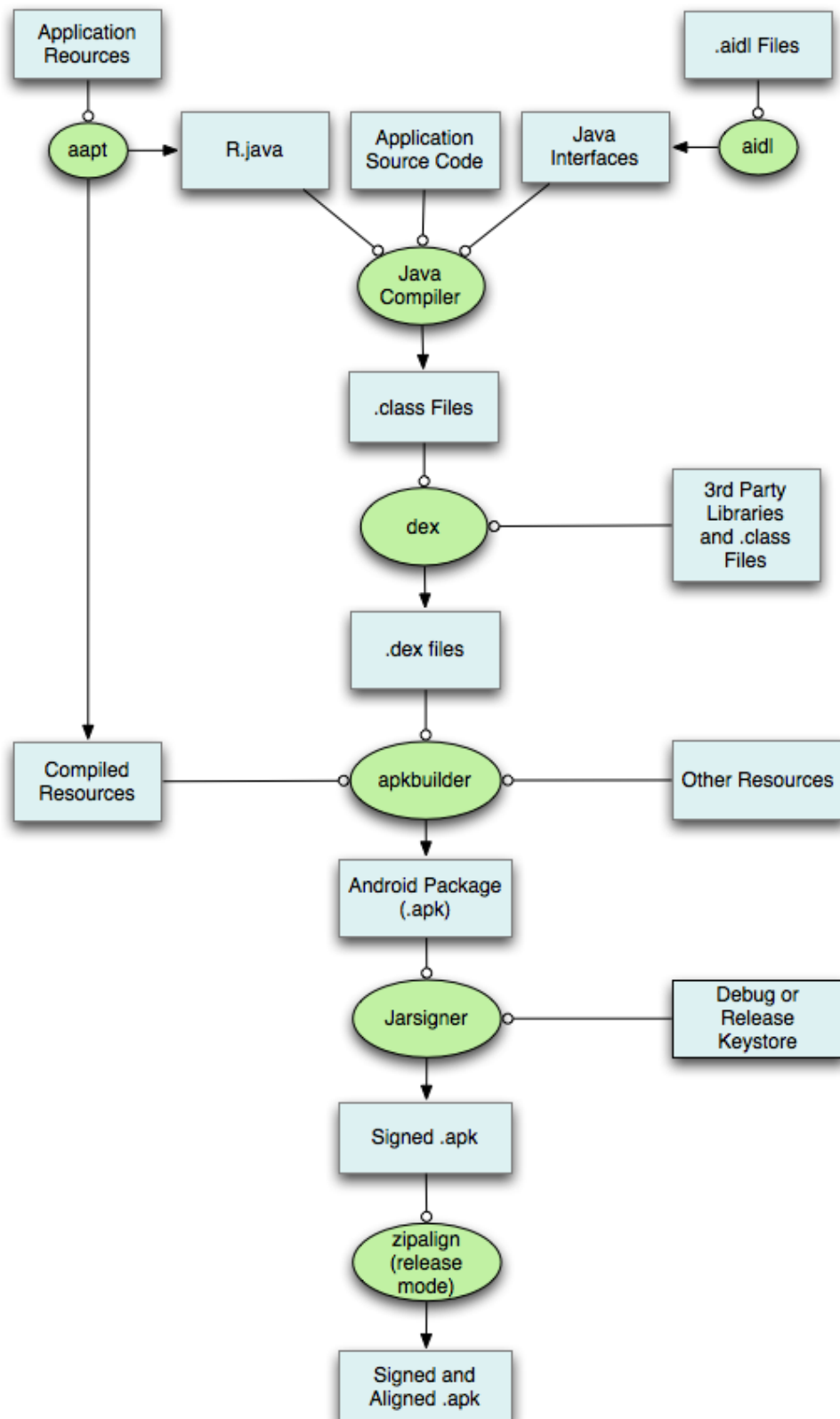


Figura 2: Ciclo extendido de ejecución en Android.

## GESTIÓN DE MEMORIA Y RECURSOS EN ANDROID

La memoria RAM suele ser considerada como un recurso valioso en cualquier sistema de cómputo. En los sistemas operativos móviles donde la memoria es reducida este recurso es aún más valioso. Aunque la máquina virtual Dalvik incluye un recolector de basura como el de Java, el desarrollador debe conocer dónde la aplicación consume y libera la memoria. Para que el recolector de basura pueda liberar la memoria de la aplicación, es necesario que todas las instancias y recursos que no vayan a ser usadas sean liberadas explícitamente. El recolector de basura se encarga del resto: el sistema reclama la memoria de la aplicación cuando el objeto correspondiente ha dejado el ámbito del hilo principal y no tiene ninguna referencia que lo apunte.

Android no tiene espacio de intercambio de memoria (swap space), por ello usa paginación y mapeado de memoria, para gestionar la memoria. Esto se traduce en que cualquier zona de memoria que sea modificada, ya sea mediante la creación de nuevos objetos u otros medios, permanece residente en memoria RAM y no puede ser paginada. Por esto, la única forma de liberar completamente la memoria de una aplicación es liberar la referencia al objeto que la está usando, para que la memoria esté disponible para el recolector de basura.

Con el fin de repartir la memoria RAM entre todas las aplicaciones que la necesitan, Android intenta compartir páginas de memoria RAM a través de los procesos. Puede hacerlo de las siguientes maneras:

- Cada proceso de cada aplicación realiza un fork de un proceso llamado Zygote. Zygote es lanzado cuando el sistema se inicia y es el encargado de cargar las librerías y los recursos. Para crear un nuevo proceso, el sistema realiza un fork del proceso Zygote y luego carga y ejecuta el código de la aplicación en un nuevo proceso. Esto permite que la mayoría de las páginas de la memoria asignadas a ese código y a sus recursos, puedan ser compartidas a través de todos los procesos de la aplicación.
- La mayoría de los datos estáticos son mapeados dentro del proceso. Esto no sólo permite que los mismos datos sean compartidos entre procesos sino también permite que sean paginados cuando sea necesario. Ejemplos de

datos estáticos: código Dalvik, recursos de aplicaciones, código nativo en los ficheros .so.

- Android comparte la misma memoria RAM entre procesos usando regiones de memoria de forma explícita (ya sea con ashmem o gralloc).



## CAPÍTULO 3

### HERRAMIENTAS PARA EL ANÁLISIS DE RENDIMIENTO EN ANDROID

Para medir el rendimiento de una aplicación en Android, el plugin ADT que instalamos con el SDK proporciona una herramienta denominada DDMS (Dalvik Debug Monitor Server) [10], que provee de multitud de servicios para obtener la máxima información de nuestra aplicación.

Como se puede observar en la figura 3, en el IDE Eclipse la pestaña del DDMS se encuentra al lado de la de Debug, al pinchar sobre el botón, se entraría en la interfaz de la herramienta.

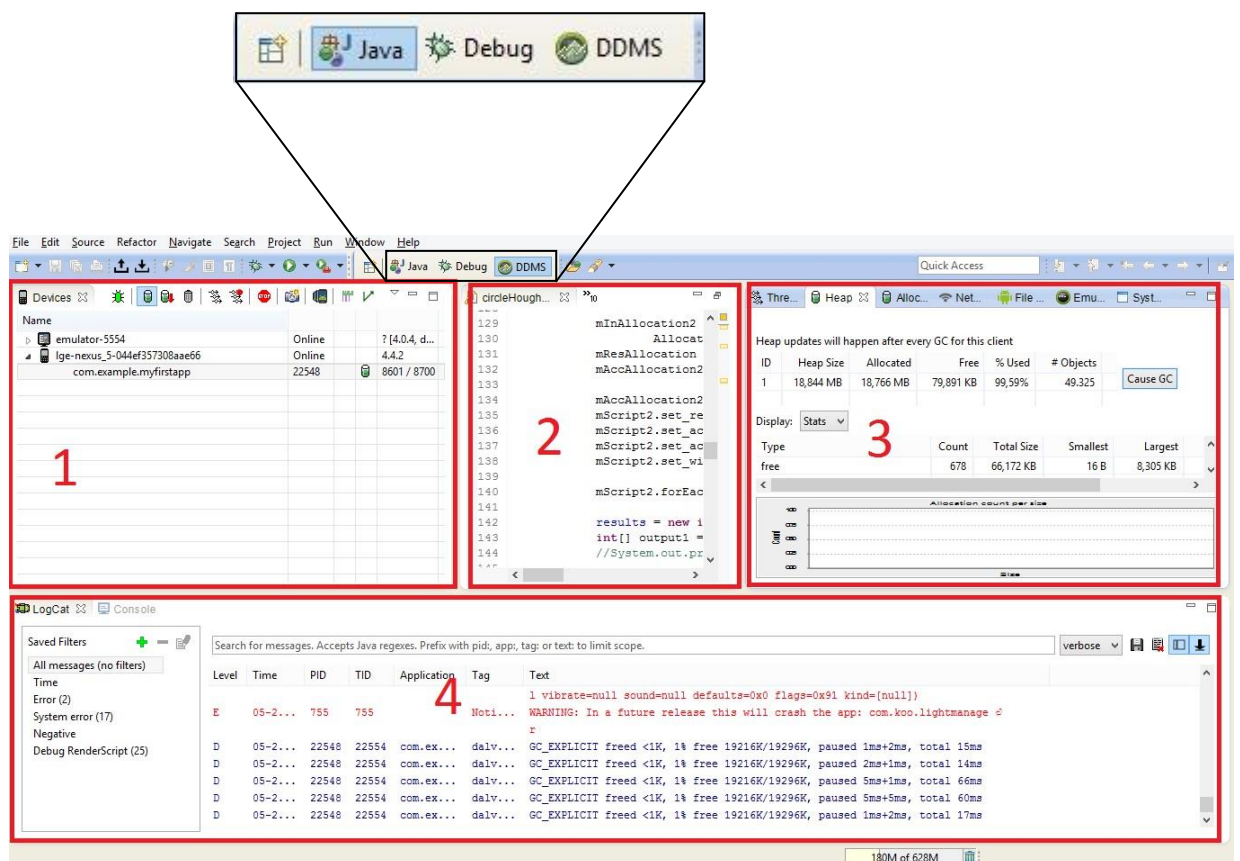


Figura 3: Zoom sobre panel Eclipse (Arriba) y secciones de trabajo en el DDMS (Abajo).

El panel etiquetado con el número 1, es la sección donde puede verse el dispositivo conectado y el proceso lanzado a ejecución. El panel con etiqueta número 2, muestra el resultado de realizar una traza a la ejecución de una aplicación. En el

panel número 3 se puede ver distinta información acerca de la aplicación, el tamaño de la pila del proceso, la memoria usada y libre y el número de objetos que son referenciados. Al pinchar sobre el botón “Cause GC”, se invoca al recolector de basura para liberar todos los objetos que no estén siendo referenciados. El número 4 es el LogCat, que permite visualizar los logs que se hayan anotado en el código, así como los errores o advertencias que surjan durante la ejecución.

En la figura 4 se puede observar cómo se haría la traza de ejecución de una aplicación, habría que ir a la parte de dispositivos y pinchar en el botón “Start Method Profiling”:

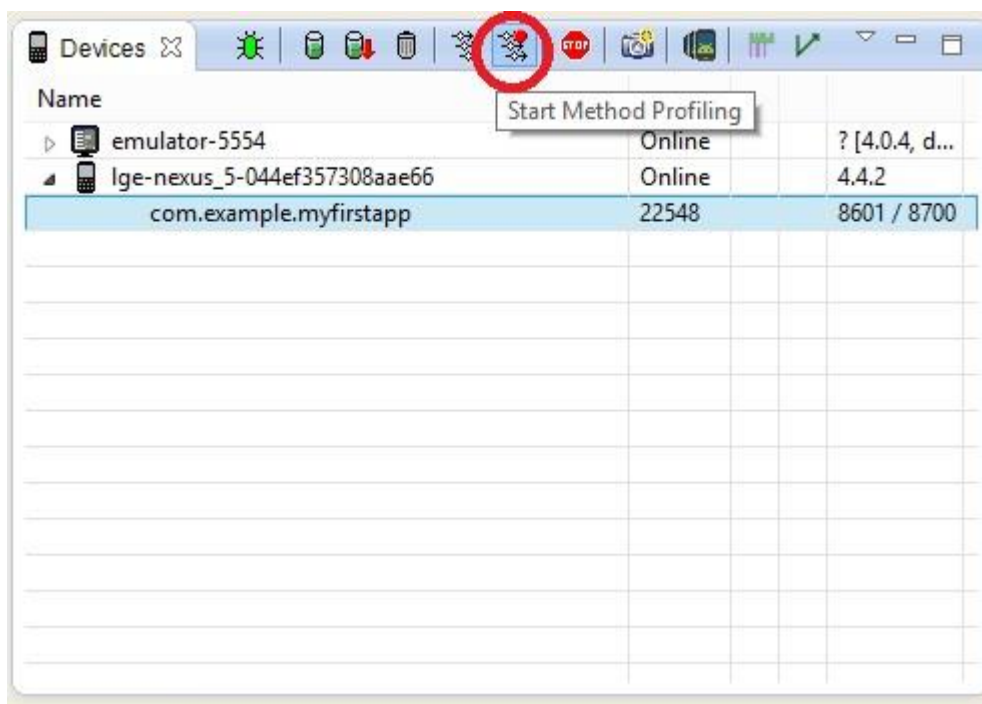


Figura 4: Zoom sobre la vista de dispositivos conectados.

Luego se ejecutaría la aplicación y cuando terminase se pincharía otra vez en el botón “Start Method Profiling”. Al lanzar la aplicación mientras se crea una traza de la ejecución, la aplicación se ejecuta considerablemente más lenta debido a que cada operación está siendo registrada.

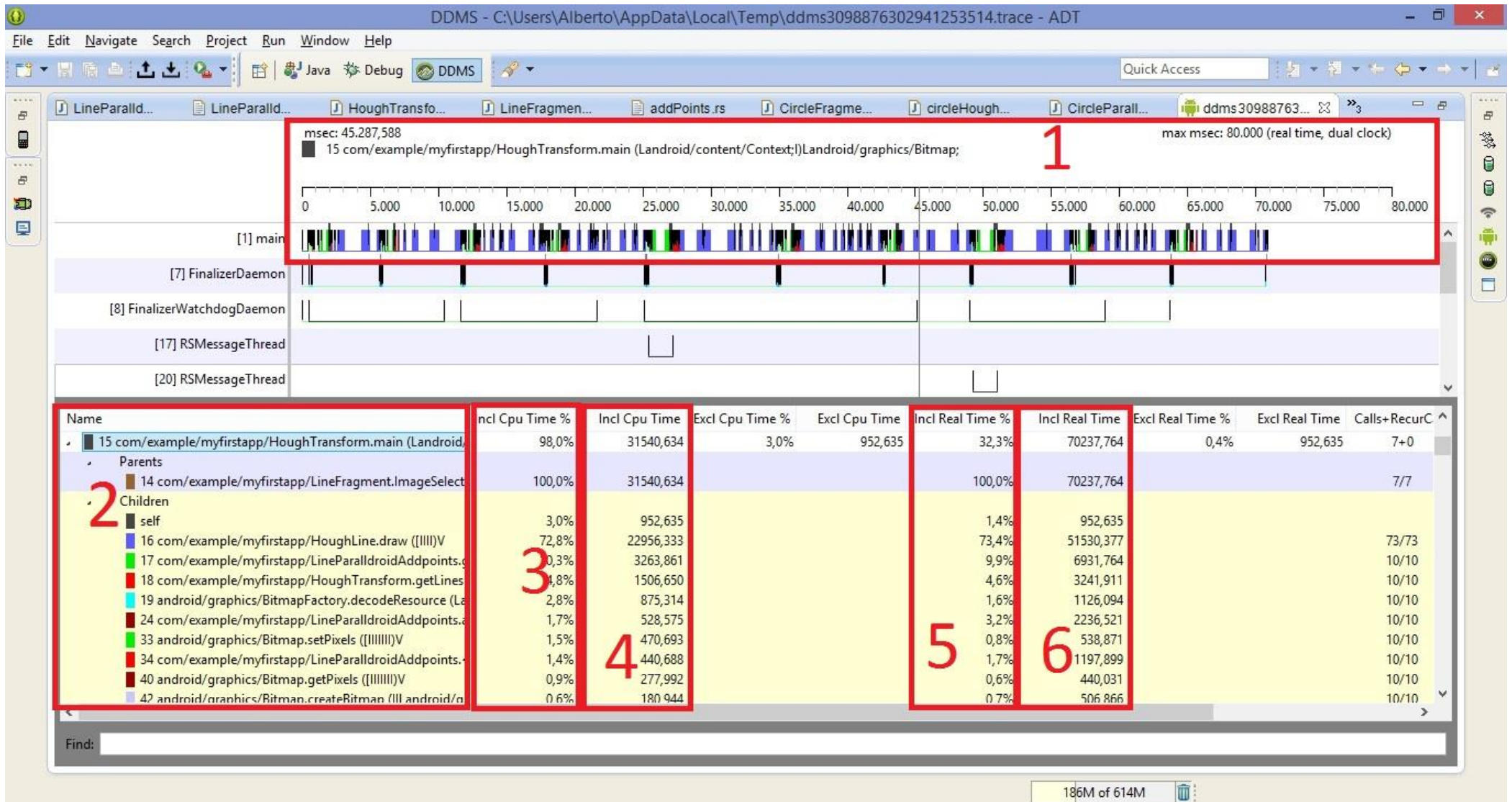


Figura 5: Reporte creado después de realizar la traza de una ejecución.

Cuando termine la ejecución aparecerá un reporte con el análisis de la aplicación como se puede ver en la figura 5 para un ejemplo de uso de la *Transformada de Hough*. En la sección etiquetada con el número 1, se puede ver gráficamente como ha sido la ejecución, el tiempo total que duró y el momento exacto en el que se ejecutó cada función. En la sección etiquetada con el número 2, se puede observar qué funciones, de qué objetos se han ejecutado y también se puede apreciar quien es el padre y quien, en caso de haberlo, el hijo de cada objeto. En la sección etiquetada con el número 3, se puede ver el porcentaje de tiempo con respecto al total que cada función ocupó la CPU, por ejemplo en este caso la función que más consume es `HoughLine.Draw()` que es hija de `HoughTransform.main()`, con un 72.8% de tiempo de CPU. En la sección 4, se ve el tiempo en milisegundos que cada función ocupó la CPU. En la sección 5, el porcentaje real del tiempo con respecto al total que duró la ejecución de cada función. En la sección 6, se muestra el tiempo real en milisegundos que duró la ejecución de cada función.

En el caso de que la ejecución sea muy rápida, puede resultar complicado de analizar, por esa razón, lo más cómodo es introducir un bucle que haga la ejecución 5, 10 o incluso 100 veces, dependiendo del caso. De esta forma se puede apreciar más claramente qué funciones son las que consumen más tiempo de ejecución en la CPU y por tanto cuales habría que optimizar para mejorar el rendimiento de la aplicación.

## **MONITOR DE SISTEMA**

Para comprobar el uso del procesador dentro del propio sistema operativo Android, se usó una aplicación que se conoce como System Monitor [11], dispone de una versión gratuita y de una versión de pago. Se usó la versión de pago debido a que permite superponer una gráfica de uso de procesador sobre la aplicación que se está utilizando. Esta aplicación permite conocer, desde el propio Android, el uso del procesador o procesadores, el consumo de memoria RAM, el uso de la memoria interna, el uso de la red, las aplicaciones que más consumen recursos, las frecuencias que ha tenido la CPU y durante cuánto tiempo, además de estadísticas de uso de batería, de almacenamiento y temperatura del dispositivo.



Figura 6: Aspecto de la aplicación System Monitor

Como se puede ver en la figura 6, la aplicación System Monitor está en la pestaña de CPU donde se puede observar el número de núcleos que tiene el procesador, el porcentaje de carga, la frecuencia de cada núcleo y si el núcleo está activo o inactivo.

Esta aplicación se usó para comprobar que Renderscript efectivamente usaba todos los procesadores disponibles. Como se puede ver en la figura 7, una ejecución que hace uso de Renderscript, se puede apreciar como hay picos en los que se usan los cuatro núcleos disponibles, estos picos corresponden a la función que se portó de Java a Renderscript para mejorar su rendimiento.

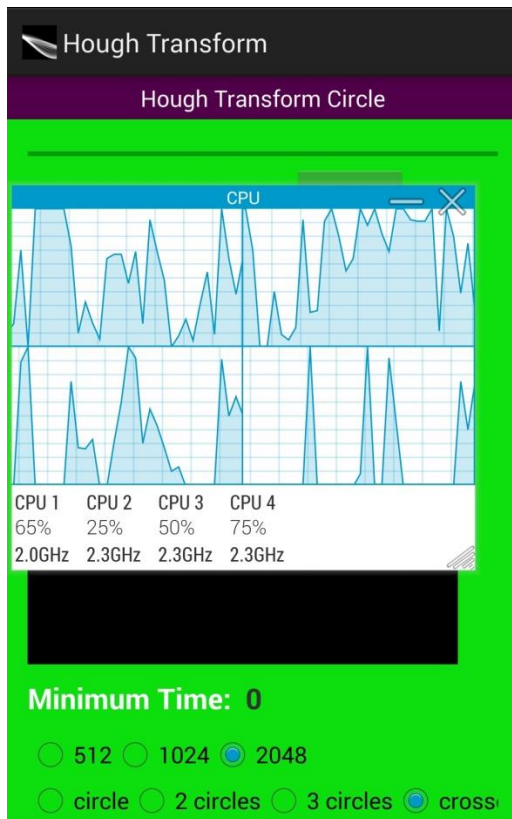


Figura 7: Ejemplo de ejecución con Renderscript y System Monitor

## Capítulo 4

### **TRANSFORMADA DE HOUGH**

En este capítulo se explicará qué es la *Transformada de Hough* [3], para que se usa, quién la propuso por primera vez y los fundamentos matemáticos que están detrás de la detección de líneas y de círculos.

La Transformada de Hough es un algoritmo para la detección de figuras en imágenes digitales. Este algoritmo se utiliza principalmente en el contexto de la Visión por Ordenador. Con la transformada de Hough es posible encontrar, en una imagen, figuras que puedan ser expresadas matemáticamente, tales como rectas, circunferencias o elipses. Actualmente es muy usada para reconocimiento de objetos en vídeos de vigilancia, rayos x, detección de cara fatigada durante la conducción, etc. En este proyecto se usa como medio para estresar el dispositivo, estudiar su rendimiento e intentar mejorarlo en lo posible. No lo usamos en este caso como utilidad eficiente en la detección de figuras.

### **HISTORIA**

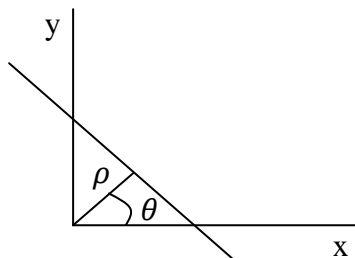
La transformada de Hough fue propuesta y patentada en 1962 por Paul Hough [3.1]. Inicialmente esta técnica solo se aplicaba a la detección de rectas en una imagen, posteriormente se extendió para identificar cualquier figura que se pudiera describir con un conjunto de parámetros; más comúnmente circunferencias y elipses. La transformada de Hough, como se usa actualmente, fue propuesta por Richard Duda y Peter Hart en 1972, quienes la denominaron "Transformada de Hough Generalizada" [3.2]. Dana H. Ballard popularizó este método en la comunidad de Visión por Ordenador en un artículo publicado en 1981, llamado Generalizando la transformada de Hough para detectar figuras arbitrarias [3.3].

### **DETECCIÓN DE LÍNEAS RECTAS**

El caso más simple para la transformada de Hough es la transformación lineal [12][13]. En el espacio de la imagen, la recta se puede representar con la ecuación:

$$y = m * x + n$$

Donde  $m$  es la pendiente de la recta y  $n$  es el punto donde se intercepta la recta con el eje  $Y$ . Se puede representar gráficamente para cada par  $(x_0, y_0)$  de la imagen. En la transformada de Hough, la idea principal es considerar las características de una recta en término de sus parámetros  $(m, n)$ , y no como puntos de la imagen  $(x_1, y_1), \dots, (x_n, y_n)$ . Basándose en lo anterior, la recta  $y = m * x + n$  se puede representar como un punto  $(m, n)$  en el espacio de parámetros. Sin embargo, cuando se tienen rectas verticales, los parámetros de la recta  $(m, n)$  quedan sin definir. Por esta razón es mejor usar los parámetros que describen una recta en coordenadas polares, denotadas  $(\rho, \theta)$ . El parámetro  $\rho$  representa la distancia entre el origen de coordenadas y el punto  $(x, y)$ , mientras que  $\theta$  es el ángulo del vector director de la recta perpendicular a la recta original y que pasa por el origen de coordenadas.



Usando esta parametrización, la ecuación de una recta se puede escribir de la siguiente forma:

$$y = \left(-\frac{\cos \theta}{\sin \theta}\right) * x + \left(\frac{\rho}{\sin \theta}\right)$$

Que se puede reescribir como:

$$\rho = x * \cos \theta + y * \sin \theta.$$

Entonces, es posible asociar a cada recta un par  $(\rho, \theta)$ , que es único si  $\theta \in [0, \pi]$  y  $\rho \in R$  o  $\theta \in [0, 2 * \pi]$  y  $\rho \geq 0$ . El espacio  $(\rho, \theta)$  se denomina espacio de Hough para el conjunto de rectas en dos dimensiones.

Para un punto arbitrario en la imagen con coordenadas  $(x_0, y_0)$ , las rectas que pasan por ese punto son los pares  $(\rho, \theta)$  con  $r = x * \cos \theta + y * \sin \theta$  donde  $\rho$  (la distancia entre la línea y el origen) está determinada por  $\theta$ . Esto corresponde a una curva sinusoidal en el espacio  $(\rho, \theta)$ , que es única para ese punto. Si las curvas correspondientes a dos puntos se intersectan, el punto de intersección en el espacio de Hough corresponde a una línea en el espacio de la imagen que pasa por estos dos puntos. Generalizando, un conjunto de puntos que forman una recta, producirán sinusoides que se intersectan en los parámetros de esa línea. Por tanto, el problema de detectar puntos que pertenezcan a la misma línea se puede convertir en un problema de buscar curvas que confluyen en un mismo punto.



## CURVAS PARAMÉTRICAS

La Transformada de Hough no se restringe solamente a la detección de rectas, a pesar de que se usa comúnmente con este propósito. Otras figuras geométricas que se puedan describir con varios parámetros se pueden detectar también con esta técnica.

Por ejemplo, si se quieren detectar circunferencias [14] con la ecuación:

$$(x - a)^2 + (y - b)^2 = r^2$$

Para describir una circunferencia son necesarios tres parámetros:

- Centro de la circunferencia  $(a, b)$
- Radio  $(r)$

Para encontrar circunferencias usando la transformada de Hough, se necesita un acumulador con tres dimensiones  $(a, b, r)$ . Después cada punto en la imagen vota por las circunferencias en los que pudiera estar. Una vez terminado este procedimiento se buscan los picos en el acumulador y con esto se obtienen el radio y el centro de la circunferencia. Si se conociera el radio de antemano, solo se necesitaría un acumulador de dos dimensiones.



## CAPÍTULO 5

### **APLICACIONES**

En este capítulo se mostrarán las aplicaciones creadas, las imágenes usadas para las pruebas, la tecnología usada y los principales inconvenientes encontrados.

### **APLICACIONES JAVA**

Durante el desarrollo de este proyecto, se crearon varias aplicaciones para el análisis y estudio. La primera de ellas, fue una aplicación que reconocía líneas en imágenes digitales en blanco y negro utilizando la *Transformada de Hough*. Con esta aplicación se iniciaron las primeras pruebas, en las que se midió el tiempo de ejecución, el consumo de memoria y las funciones computacionalmente más costosas. Para ello se parte del código Java disponible en [1] y se realiza una adaptación para su uso en Android y su análisis de rendimiento. Fruto de ese análisis se crearon cinco versiones de esta aplicación, mejorando progresivamente tanto los tiempos de ejecución como la memoria consumida. Este primer análisis permitió entender que factores afectaban a la velocidad de procesamiento y que factores afectaban al consumo de memoria.

### **IMÁGENES DE PRUEBA PARA LAS APLICACIONES JAVA**

Para el análisis del rendimiento de este algoritmo se usaron diferentes imágenes en blanco y negro. Se usaron imágenes de 512x512, de 1024x1024 y de 2040x2048 pixeles. Además, se usaron imágenes donde aparecían: una figura, dos figuras, tres figuras y cuatro figuras que se intersectan. En la figura 8 se puede apreciar las imágenes usadas tanto para la línea como para el círculo:

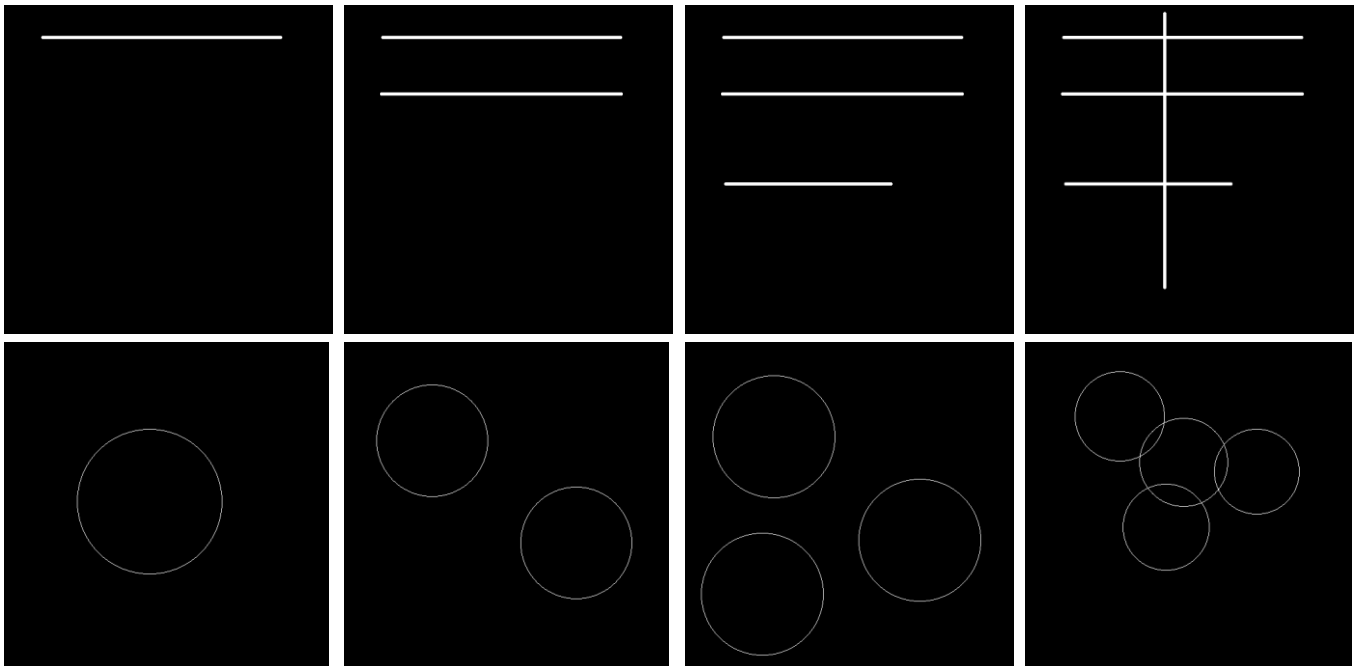


Figura 8: Imágenes de prueba para la detección de líneas y círculos

Después de las optimizaciones realizadas al código Java, se pasó a crear la segunda aplicación que reconocía círculos en imágenes en blanco y negro. Se partió del código base en Java disponible en [2] y se realizó la correspondiente modificación para su uso en Android. Se optimizó usando los conocimientos adquiridos durante la creación de la primera versión. Estas dos aplicaciones han sido unificadas en una única aplicación mediante la construcción de una UI (interfaz de usuario) para facilitar las pruebas.

Cuando se optimizó el código Java de las dos aplicaciones, se procedió a investigar otras soluciones de visión por computador.

## **APLICACIONES RENDERSRIPT**

RenderScript es una API para computación intensiva usando procesadores heterogéneos, es decir, un mismo código sirve para varios tipos de procesadores. Permite al desarrollador incrementar el rendimiento de su aplicación con el coste que conlleva crear ese código, más complejo. Provee a los desarrolladores de dos herramientas principales: Una API de computación de propósito general para uso con diferente tipo de hardware y un lenguaje “familiar” como el C99. En versiones

anteriores de la API se le añadió renderizado 3D experimental, pero en las siguientes versiones se quitó.

Con Rendscript se trabaja con Allocation, que es como se llama al tipo de objeto que se crea en Java para pasarle al código escrito en Rendscript. Los Allocation permiten que la memoria RAM donde reside ese objeto pueda ser compartida por todos los núcleos del procesador. Al crear el Allocation se debe definir el tipo de dato y el tamaño que va a tener.

El principal problema que existe con Rendscript, radica en la falta de documentación, la fiabilidad del plugin de Eclipse y de ejemplos en los que poder fijarte. Hoy en día no se sabe si Rendscript seguirá o desaparecerá en el futuro y debido a esta incertidumbre unida a la falta de documentación, provoca que desarrollar sea muy complicado y no haya demasiados desarrolladores apostando por este modelo de programación.

Rendscript se usó en la aplicación Android para la detección de líneas y círculos. Cuando con Java no se pudo mejorar más el rendimiento, se migro el código computacionalmente más costoso a Rendscript.

## **APLICACIONES OPENCV**

OpenCV [15] es una librería libre de Visión por Ordenador originalmente desarrollada por Intel. Desde que apareció su primera versión alfa, en el mes de enero de 1999, se ha utilizado en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicativos de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se proporciona bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

OpenCV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows. Contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados.

Se desarrollaron varias aplicaciones utilizando OpenCV. Las tres primeras se unieron a la interfaz principal donde también se encuentran la detección de líneas y círculos, comentada en los párrafos anteriores.

- La primera de ellas, recibe como entrada una imagen en color con monedas y es capaz de detectarlas. Como se puede observar en la figura 9.



Figura 9: Imagen de ejemplo de monedas (izda.) e imagen procesada (dcha.)

- La segunda aplicación recibe como entrada una imagen en color de una construcción y detecta el contorno de la construcción. Como se puede apreciar en la figura 10.

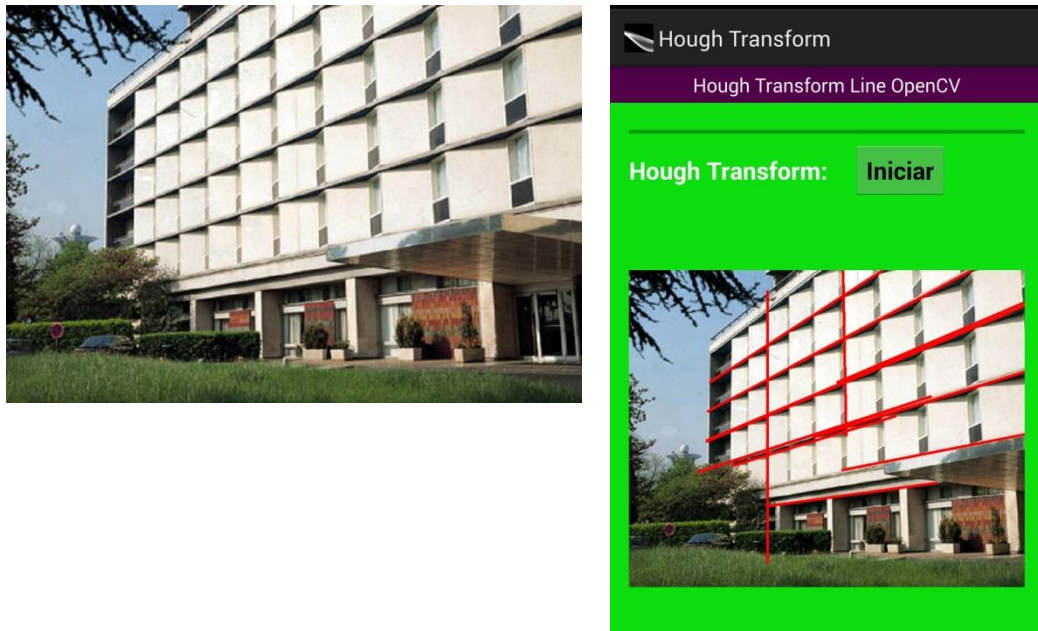
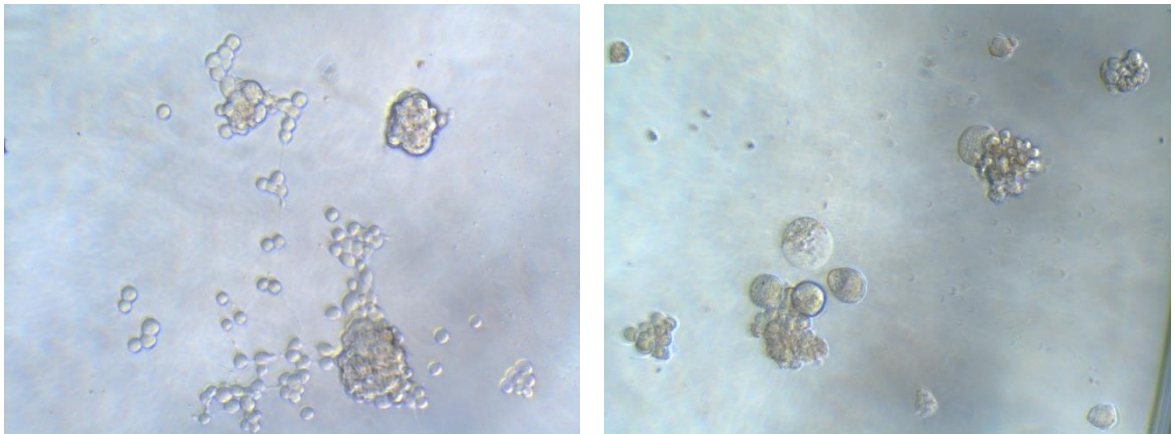


Figura 10: Imagen de ejemplo de edificio (izda) e imagen procesada (dcha.).

- La tercera aplicación permite diferenciar las células cancerígenas de las células sanas en imágenes en color proporcionadas de un microscopio. Las imágenes usadas son las que se muestran en la figura 11. En la figura 12 se puede comprobar el resultado después de procesarlas.



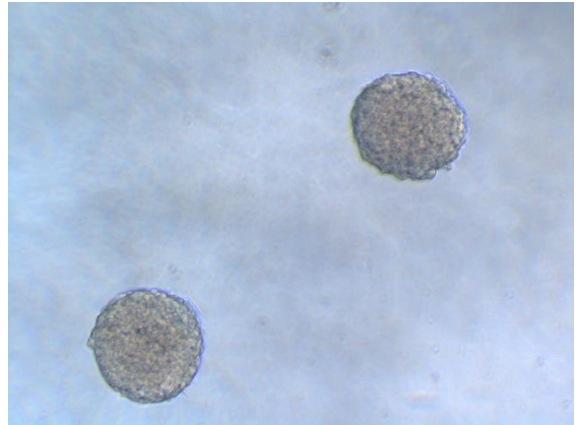
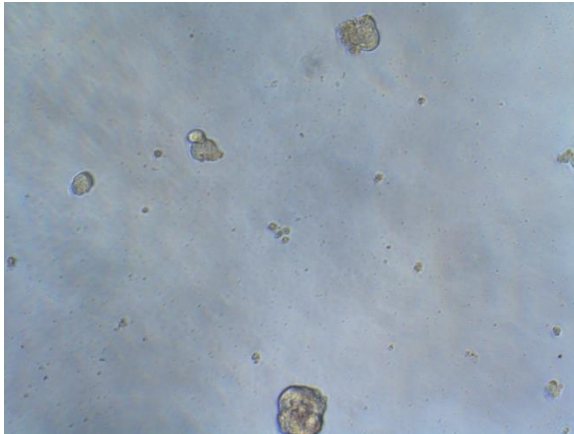


Figura 11: Imágenes con células cancerígenas de Glioblastoma.







Figura 12: Imágenes procesadas con las células detectadas.

- Se creó una aplicación independiente de la anterior, que hace uso de la cámara del dispositivo Android. Esta nueva aplicación permite detectar líneas como se puede observar en la figura 13, y círculos en tiempo real como se aprecia en la figura 14. Además también se implementó dos versiones para transformar la imagen de color a blanco y negro. La primera versión hace uso de OpenCV y la segunda versión hace uso de Renderscript como se puede ver en la figura 15, además aparece también una instancia del programa System Monitor para comprobar el uso de todos los núcleos.



Figura 13: Detección de líneas en tiempo real.

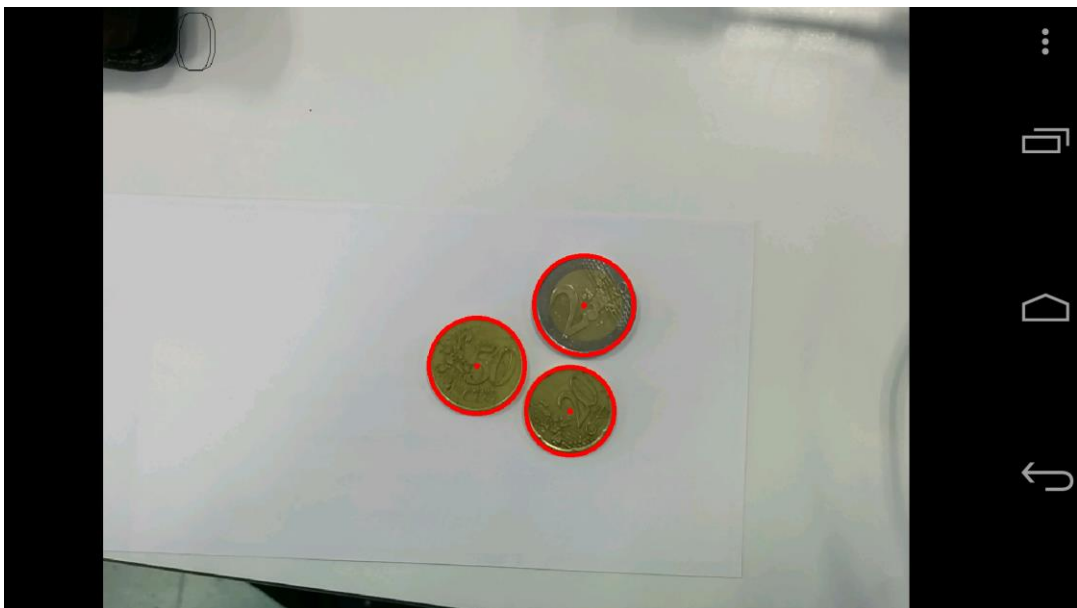


Figura 14: Detección de círculos en tiempo real.

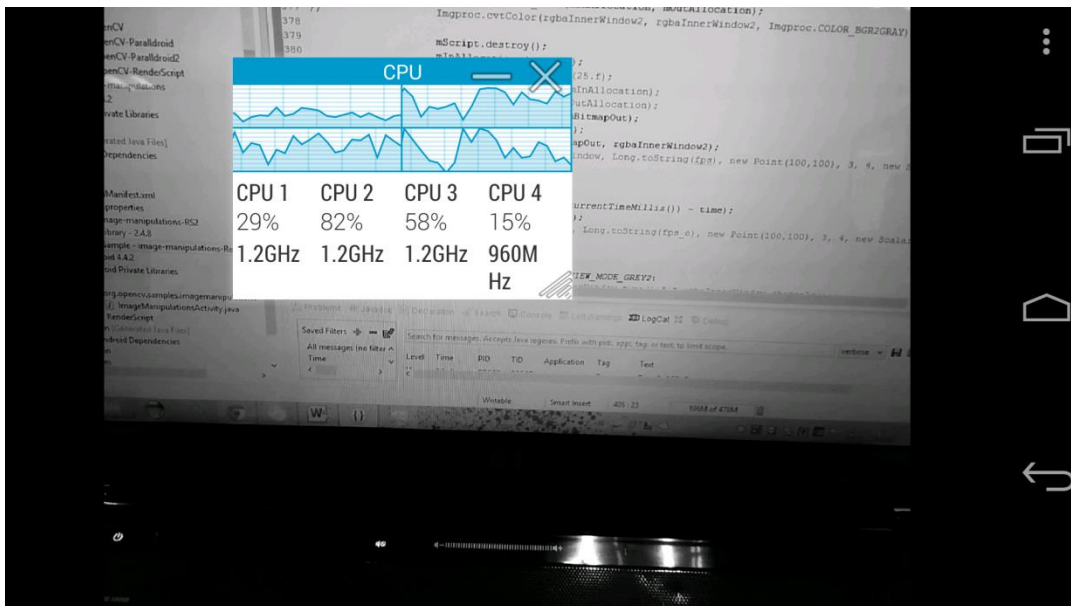


Figura 15: Paso de imagen de color a blanco y negro con Renderscript en tiempo real.

Un inconveniente observado en el uso de OpenCV sobre Android es que está escrita en código nativo, ya que suponemos fue migrada desde la versión de C. Por esta razón presenta algunas restricciones en referencia a las características de la imagen a utilizar, como por ejemplo: el tipo de pixel de la imagen, el número de canales y el formato.

Si se desea utilizar determinadas funciones, hay que tratar previamente la imagen, lo que conlleva un sobrecoste computacional. La librería no está optimizada para uso en dispositivos Android multinúcleo y resulta imposible usar Renderscript en las funciones de OpenCV.



## CAPÍTULO 6

### RESULTADOS

En este capítulo se mostrarán los resultados obtenidos. Durante el desarrollo del proyecto, se han obtenido varias versiones para poder conocer mejor como afecta cada parámetro a la velocidad de procesamiento y a la memoria.

El proyecto se inició con una versión muy ineficiente, que se estudió y se fue mejorando paulatinamente. Inicialmente, se disponía de un dispositivo móvil de aproximadamente tres años de antigüedad, el LG Optimus Black P970, en la tabla 1 se describen sus características y en la figura 16 se puede ver su aspecto. Posteriormente, se migró a un terminal actual, el Google Nexus 5, en la tabla 2 se describe sus características y en la figura 17 se puede ver su aspecto.

Modelo	LG Optimus Black P970.
Chipset	TI OMAP 3630
Procesador	ARM Cortex-A8. Single Core - 1Ghz 1000Mhz 32bits.
GPU	PowerVR SGX530
RAM	512 MB.
Pantalla	4" 480 x 800 píxeles.
Versión Android	4.1



Figura 16: LG Optimus

Tabla 1: Características LG Optimus.

Modelo	Google LG Nexus 5
Chipset	Qualcomm Snapdragon 800 MSM8974
Procesador	Quad core, 2260 MHz, Krait 400.
GPU	Adreno 330
RAM	2058 MB.
Pantalla	5" 1080 x 1920 pixeles.
Versión Android	4.4.2



Figura 17: Nexus 5

Tabla 2: Características Nexus 5.

Se comenzará analizando las diferencias entre los dos dispositivos y en que afectan al rendimiento de una aplicación.

El Nexus tiene cuatro veces más memoria RAM, su procesador tiene más del doble de frecuencia y además, la versión del Android es superior a la del Optimus, con las mejoras en gestión de recursos que ello conlleva y, por tanto, el tiempo de ejecución de las aplicaciones se reduce.

La primera versión de la aplicación se probó en los dos dispositivos para determinar el impacto en la mejora de dispositivo y la actualización del sistema operativo, una aplicación solo por usar un dispositivo más moderno y con un sistema operativo más actualizado. En la tabla 3 y 4 aparecen los datos del LG Optimus y del Nexus 5 respectivamente. En las figuras 18 y 19 aparecen las gráficas con esos datos.

Para medir el tiempo de ejecución, se realizaron diez ejecuciones, registrando el tiempo por iteración así como el tiempo medio. Para medir la memoria consumida y la función o funciones más costosas se usó DDMS.

En la figura 20 se puede ver más cómodamente los resultados mostrados en las gráficas 18 y 19. Las primeras barras de cada color corresponden con el LG y las segundas con el Nexus.

		<b>Tiempo de Ejecución</b>			
<b>Tamaño</b>	<b>Lg Optimus</b>	<b>Línea</b>	<b>Dos líneas</b>	<b>Tres líneas</b>	<b>Cuatro líneas</b>
	<b>512 x 512</b>	873 ms	979 ms	1233 ms	1739 ms
	<b>1024 x 1024</b>	2954 ms	3176 ms	3405 ms	5558 ms
	<b>2048 x 2048</b>	9595 ms	9646 ms	10543 ms	15616 ms

Tabla 3: Resultados rendimiento LG Optimus.

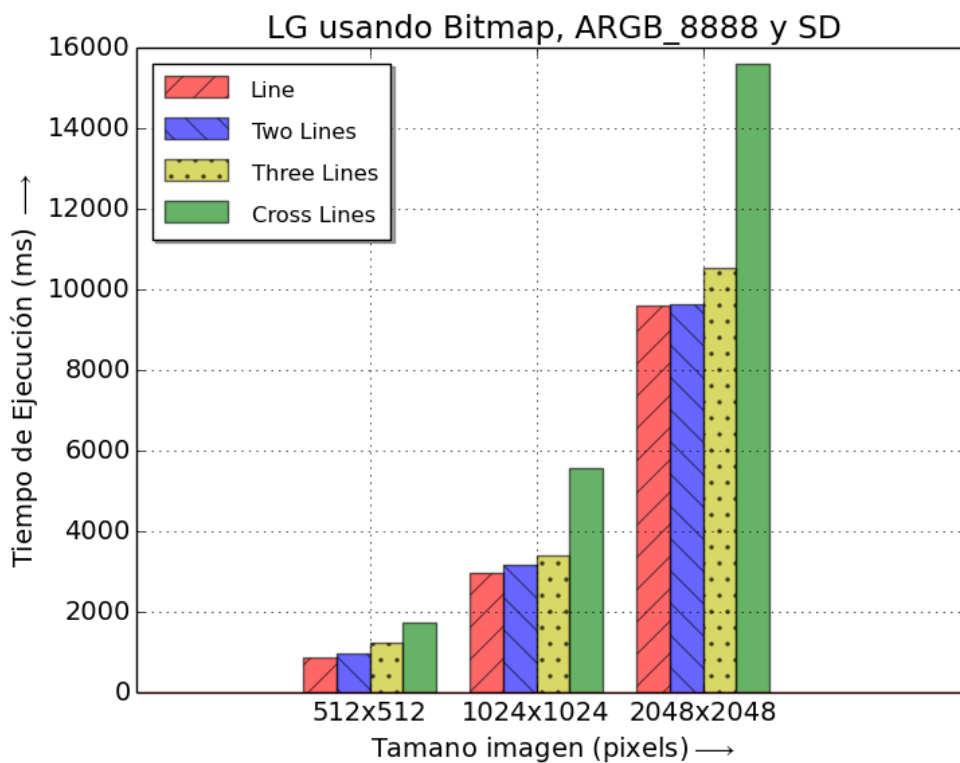


Figura 18: Gráfica con los resultados de rendimiento de LG Optimus.

		<b>Tiempo de Ejecución</b>			
<b>Tamaño</b>	<b>Nexus 5</b>	<b>Línea</b>	<b>Dos líneas</b>	<b>Tres líneas</b>	<b>Cuatro líneas</b>
	<b>512 x 512</b>	293 ms	318 ms	370 ms	495 ms
	<b>1024 x 1024</b>	1193 ms	1255 ms	1392 ms	1968 ms
	<b>2048 x 2048</b>	4641 ms	4709 ms	4771 ms	6851 ms

Tabla 4: Resultados rendimiento Nexus 5.

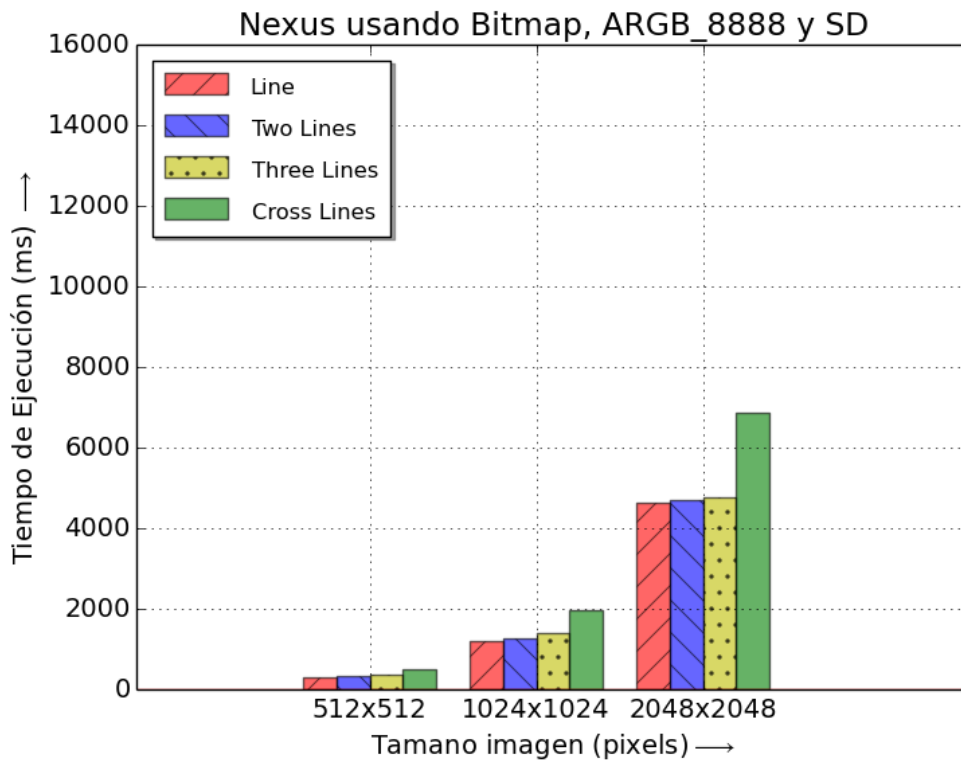


Figura 19: Gráfica con los resultados de rendimiento de Nexus 5.

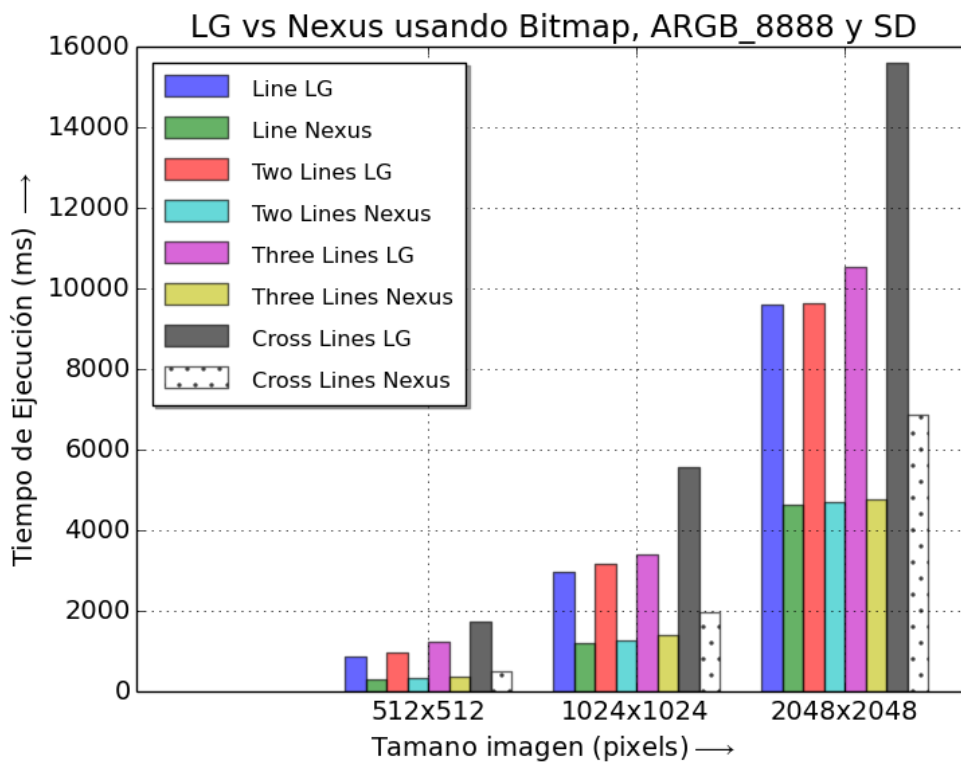


Figura 20: Gráfica comparando rendimiento de LG vs Nexus. Línea.



Como se puede observar, la diferencia de mejora es superior al 50%. Esto es debido a la diferencia en el hardware y el sistema operativo. Un procesador el doble de rápido y un sistema operativo que gestiona más eficientemente los recursos, hacen que el tiempo se reduzca en más de la mitad en el peor de los casos. Después de terminar las pruebas de hardware, se decide usar el Nexus 5 para el resto de pruebas.

Con el fin de mejorar el rendimiento hacemos un análisis de las estructuras de datos analizadas para almacenar las imágenes. Podemos encontrar tres tipos de estructuras: ARGB\_4444, ARGB\_8888 y RGB\_565. El tipo ARGB\_4444 está en desuso desde la API 13 (Android 3.2), aun así se usó en las pruebas. En este tipo ARGB\_4444, cada pixel se almacena en 4 bytes, en el tipo ARGB\_8888, cada pixel se almacena en 4 bytes y en el tipo RGB\_565 cada pixel se almacena en 2 bytes. El primer cambio que se realiza para mejorar el rendimiento, es del de cambiar el tipo estructura de cada pixel en la imagen [16]. El análisis de las primeras pruebas, nos permite concluir que la velocidad del procesado no se ve afectada por el tipo en el que se guarde cada pixel y que, para imágenes de tamaño pequeño, no influye, pero para imágenes mayores si hay diferencia en el consumo de memoria de la aplicación y en las imágenes almacenadas. Por ejemplo, una imagen de 512 x 512 pixeles, si cada pixel se guarda en 4 bytes, tendríamos  $512 * 512 = 262144$  bytes \* 4 bytes = 1048576 bytes / 1024 = 1024 KB / 1024 = 1 MB, cada imagen almacenada ocuparía 1 MB. Si para esa misma imagen almacenamos cada pixel en 2 bytes, la imagen almacenada ocuparía la mitad de espacio, 512 KB. Por todo lo anterior, y ya que ARGB\_4444 estaba en desuso, se optó por usar RGB\_565. De esta forma, se redujo el consumo de memoria y el tamaño de la imagen guardada en la tarjeta SD.

Las conclusiones hasta este momento eran:

- Respecto al tamaño de la pila, esta depende de las dimensiones de la imagen (resolución), cuanto mayor es la imagen más memoria necesita.
- En cuanto a la velocidad, esta depende de la complejidad de la imagen, cuanto más compleja es (en este caso, más figuras tiene) más tiempo necesita para procesarla.

Desde el inicio, para trabajar con las imágenes, se usó la clase Bitmap de Android, ya que era la forma que se encontró de leer las imágenes y poder trabajar con ellas. Después de analizar su rendimiento, se siguió usando la clase Bitmap para leer las

imágenes, pero en lugar de trabajar con esta clase, se pasó del Bitmap a un Array de java, aumentando considerablemente la velocidad del procesado. El único inconveniente de hacerlo de esta forma es que la aplicación en general consume más memoria, porque en lugar de tener un Bitmap en memoria, hay un momento en el que al pasar de Bitmap a Array de Java, el Bitmap y el Array se encuentran cargados simultáneamente en memoria. Por ello, fue necesario añadir en el fichero Manifest del proyecto, la directiva `android:largeHeap="true"`, para incrementar el tamaño de la pila para imágenes de 2048 x 2048 pixeles. Los resultados obtenidos se pueden apreciar en la tabla 5 y gráficamente en la figura 21. Todos los resultados que se obtuvieron con la detección de la línea, se usaron para mejorar también la detección de círculos.

En la figura 22 se puede observar la diferencia de rendimiento entre la versión con Bitmap frente a la versión con Array.

		<b>Tiempo de Ejecución</b>				
		<b>Nexus 5</b>	<b>Línea</b>	<b>Dos líneas</b>	<b>Tres líneas</b>	<b>Cuatro líneas</b>
<b>Tamaño</b>	<b>512 x 512</b>	155 ms	178 ms	207 ms	274 ms	
	<b>1024 x 1024</b>	412 ms	510 ms	592 ms	870 ms	
	<b>2048 x 2048</b>	1459 ms	1589 ms	1713 ms	2619 ms	

Tabla 5: Resultados rendimiento Nexus 5 usando Array

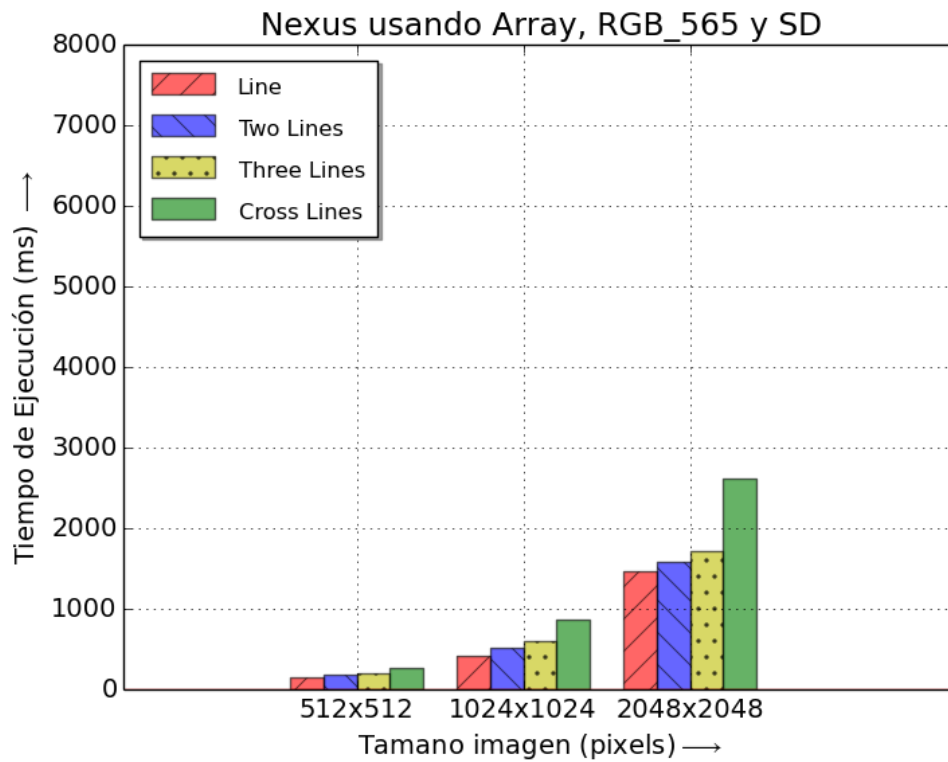


Figura 21: Gráfica con los resultados de rendimiento de Nexus 5 usando Array.

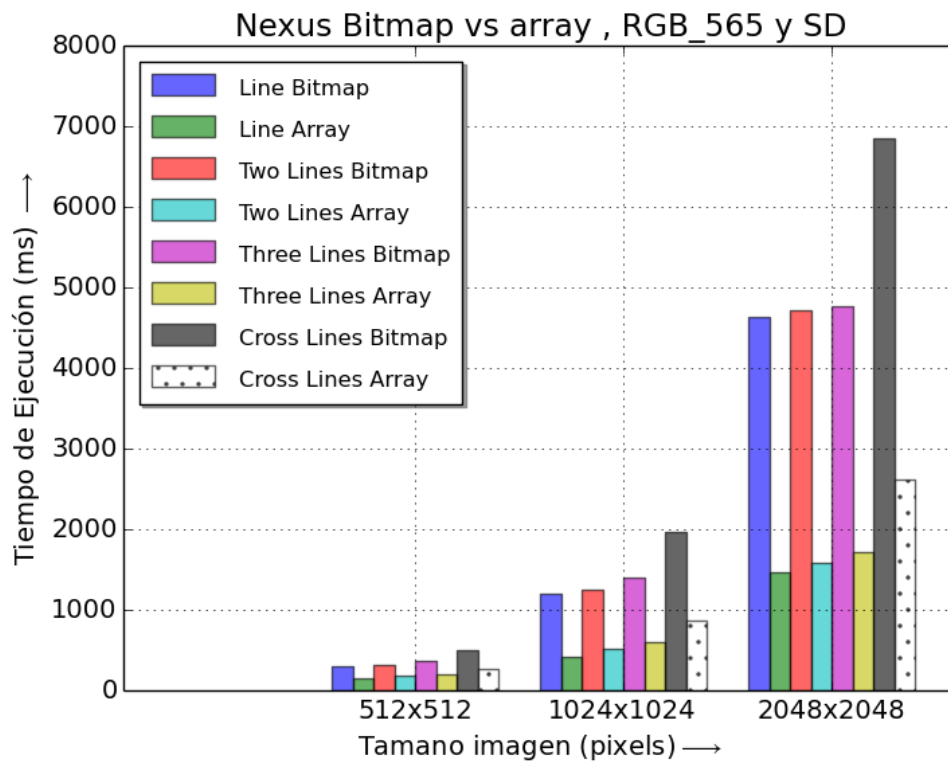


Figura 22: Bitmap vs Array en Nexus con RGB 565 y SD en línea.

Para intentar reducir el consumo de memoria, se probó a usar el `Bitmap.Recycle()` y a liberar todas las variables asignándolas a `Null` cuando ya no eran necesarias, con el fin de que el recolector de basura pudiera liberar memoria. No obstante no se notó diferencia apreciable, tanto el consumo de memoria como la velocidad se mantuvieron prácticamente iguales. El `Bitmap.Recycle()` se usaba hasta la API 10 (Android 2.3.3), en la API 11 (Android 3.0) se introdujo el `BitmapFactory.Options.inBitmap()` que reutiliza un `Bitmap` usado previamente, hasta antes de la API 19 (Android 4.4), solo se aceptaban `Bitmaps` del mismo tamaño. En la API 19 se incrementó su funcionalidad para poder utilizarlos con `Bitmaps` de igual o menor tamaño. Tampoco hubo diferencia de rendimiento usando `inBitmap()`. Una conclusión acerca de estas funciones, es que seguramente, se obtenga diferencia de rendimientos cuando se disponga de un número considerable de `Bitmaps` creados y se trabaje con varios a la vez o con una listas de ellos, como por ejemplo, una galería de imágenes.

Hasta ahora, las imágenes procesadas se almacenaban en la tarjeta SD (en el Nexus 5 es simulada), por lo que se optó por sólo mostrar la imagen procesada en lugar de guardarla, esto hizo aumentar el consumo de memoria, y además se incrementó considerablemente la velocidad del procesado. Este echo se puede observar para el caso de la línea en la tabla 6 y gráficamente en la figura 23, y para el caso del círculo en la tabla 7 y gráficamente en la figura 24. En la figura 25 se puede ver en una sola gráfica la diferencia entre los tiempos de guardar en la tarjeta SD o mostrar la imagen por pantalla.

<b>Tiempo de Ejecución</b>					
	<b>Nexus 5</b>	<b>Línea</b>	<b>Dos líneas</b>	<b>Tres líneas</b>	<b>Cuatro líneas</b>
<b>Tamaño</b>	<b>512 x 512</b>	72 ms	94 ms	119 ms	154 ms
	<b>1024 x 1024</b>	153 ms	219 ms	288 ms	394 ms
	<b>2048 x 2048</b>	383 ms	502 ms	632 ms	919 ms

Tabla 6: Resultados de mostrar la imagen de la línea.

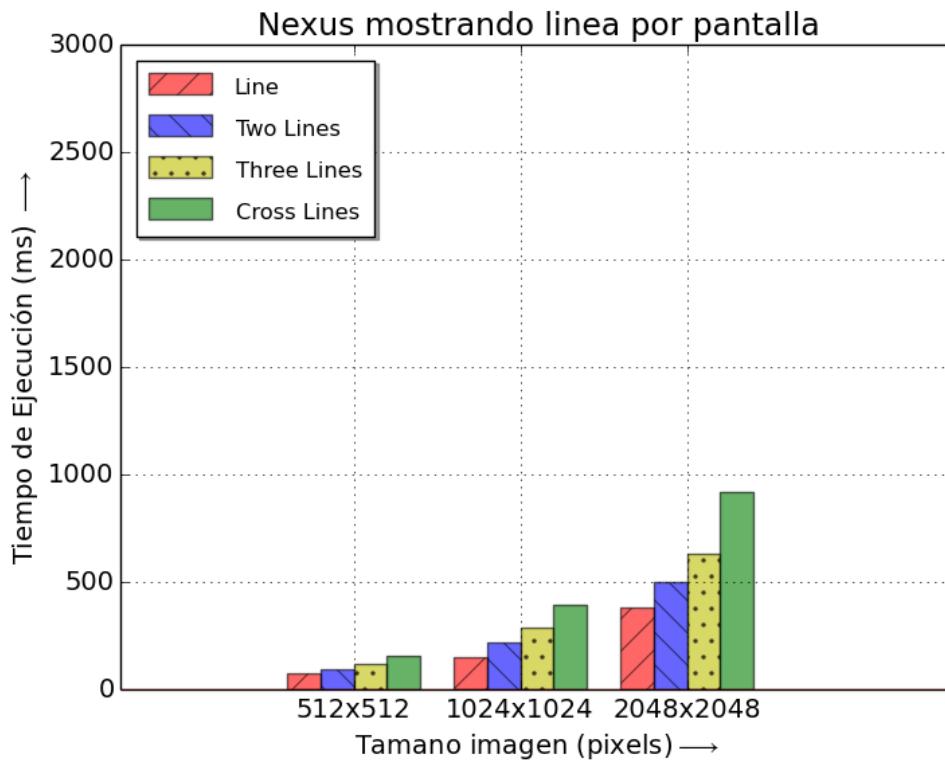


Figura 23: Gráfica con resultados mostrando la imagen de la línea.

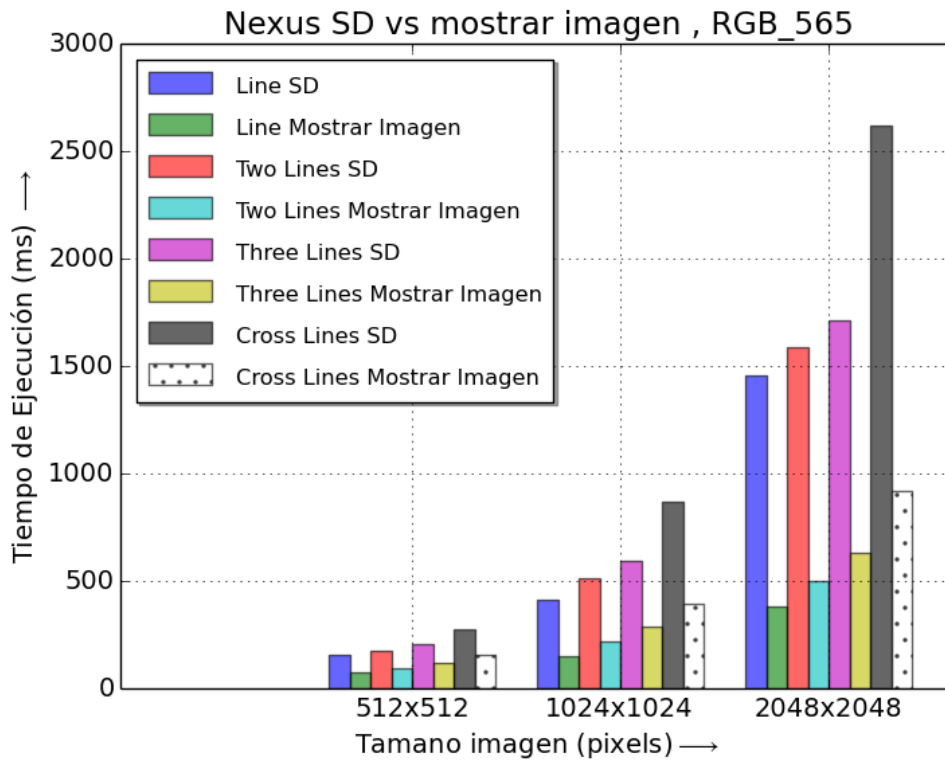


Figura 24: Guarda en SD vs mostrar por pantalla, problema línea.

		Complejidad			
Tamaño	Nexus 5	Círculo	Dos Círculos	Tres Círculos	Cuatro Círculos
	512 x 512	384 ms	533 ms	658 ms	783 ms
	1024 x 1024	842 ms	1086 ms	1277 ms	1681 ms
	2048 x 2048	2599 ms	3748 ms	4519 ms	5075 ms

Tabla 7: Resultados Nexus 5 mostrando imagen círculo

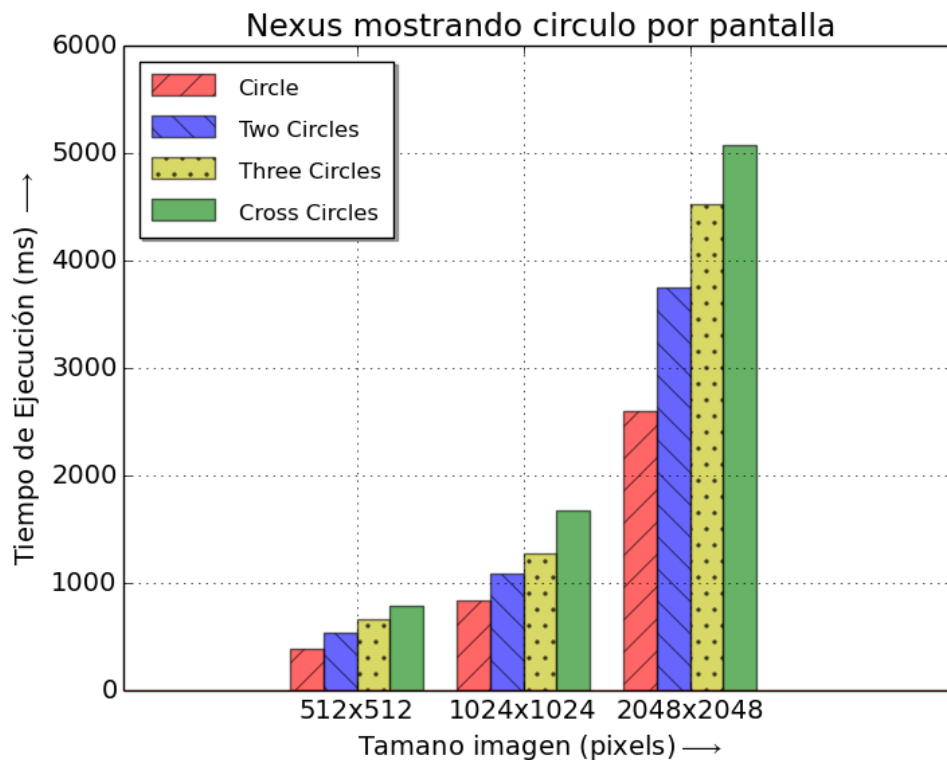


Figura 25: Gráfica con resultados mostrando la imagen del círculo.

Después de analizar y optimizar el código Java se pasó a usar RenderScript. Para ello se usó el DDMS con el fin de conocer que función era la más costosa computacionalmente. Como ejemplo, se usará el caso de la línea, sus dos funciones más costosas son: `addPoint` y `addPoints`. Por ello, se tradujeron esas dos funciones de Java a C99 para usarlas en Renderscript. Al trabajar con Renderscript, hubo que realizar una transformación de `RGB_565` a `ARGB_8888` porque es el único formato compatible. Se puede apreciar en las figuras 26 y 27 como es la traducción del código Java al código Renderscript.

```

public void addPoints(int[] image, int width, int height) {
    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            if ((image[(x*width)+y] & 0x000000ff) != 0) {
                addPoint(x, y);
            }
        }
    }
}

public void addPoint(int x, int y) {
    for (int t = 0; t < maxTheta; t++) {
        int r = (int) ((x - centerX) * cosCache[t] + ((y - centerY) * sinCache[t]));
        r += houghHeight;
        if (r < 0 || r >= doubleHeight) continue;
        houghArray[t][r]++;
    }
    numPoints++;
}

```

Figura 26: Funciones más costosas en Java. Detección de líneas.

```

static void addPoint(uint32_t x, uint32_t y){
    int acc;
    for (int i = 0; i < maxTheta; i++){
        int r = (int) ((x - centerX) * cosCache[i] + ((y - centerY) * sinCache[i]));
        r += houghHeight;
        if ( (r < 0) || (r >= doubleHeight)) continue;
        acc = rsGetElementAt_int(rsHough, (doubleHeight*i)+r);
        acc += 1;
        rsSetElementAt_int(rsHough, acc, (doubleHeight*i)+r);
    }
}

void root(const uchar4 *in, uint32_t x, uint32_t y) {
    float4 pixel = convert_float4(rsGetElementAt_uchar4(rsBitmap, x, y));
    if ((pixel.r != 0) & (pixel.g != 0) & (pixel.b != 0)){
        addPoint(y, x); // RS read the images up to down from array.
        rsSetElementAt_int(np, 1, (y*height)+x);
    }
    else{
        rsSetElementAt_int(np, 0, (y*height)+x);
    }
}
}

```

Figura 27: Funciones más costosas portadas a Rendscript. Detección de líneas.

En la figura 28, se puede observar el código adicional que habría que añadir a Java para poder usar las funciones en Rendscript. Se crea un script con el que se invoca a la función Rendscript, con los métodos set se pasan los datos de entradas necesarios para la ejecución. Con el método `invoke_nombreFunción`, se ejecuta una función de Rendscript que no sea la función principal y con el método `forEach_root`, se ejecuta la función paralela de Rendscript que se denomina como `root`, los valores  $(x,y)$  que aparecen en la cabecera no es necesario pasarlos, los usa Rendscript automáticamente para recorrer el allocation como una matriz.

```

mRS = RenderScript.create(mcontext);
mInAllocation = Allocation.createFromBitmap(mRS, imag, Allocation.MipmapControl.MIPMAP_NONE,
Allocation.USAGE_SCRIPT);
numPointsAllocation = Allocation.createSized(mRS, Element.I32(mRS), (width*height));
mHoughAllocation = Allocation.createSized(mRS, Element.I32(mRS), (maxTheta*doubleHeight));
mScript = new ScriptC_addPoints(mRS,mcontext.getResources(), R.raw.addpoints);
mScript.set_rsBitmap(mInAllocation);
mScript.set_np(numPointsAllocation);
mScript.set_rsHough(mHoughAllocation);
mScript.set_PI(Math.PI);
mScript.set_height(height);
mScript.set_width(width);
mScript.set_centerX(centerX);
mScript.set_centerY(centerY);
mScript.set_doubleHeight(doubleHeight);
mScript.set_maxTheta(maxTheta);
mScript.set_thetaStep(Math.PI / maxTheta);
mScript.set_houghHeight(houghHeight);
mScript.invoke_cache();
mScript.forEach_root(mInAllocation);

```

Figura 28: Nuevo código añadido al fichero Java para usar Renderscript.

A continuación, se muestra en la tabla 8 y en la figura 29 los resultados de usar Renderscript para la detección de líneas. Como se puede apreciar la mejora de rendimiento se hace más notable cuando más costo es el problema, haciéndose casi inapreciable para los casos menos costosos. La aceleración obtenida para el caso de la línea es 1.454. En la figura 30 se puede apreciar la diferencia de tiempos entre la versión de Java y la versión de Renderscript.

Tiempo de Ejecución					
	Nexus 5	Línea	Dos líneas	Tres líneas	Cuatro líneas
Tamaño	512 x 512	70 ms	81 ms	94 ms	110 ms
	1024 x 1024	149 ms	208 ms	231 ms	254 ms
	2048 x 2048	375 ms	544 ms	583 ms	632 ms

Tabla 8: Resultados de rendimiento usando Renderscript en la detección de líneas.



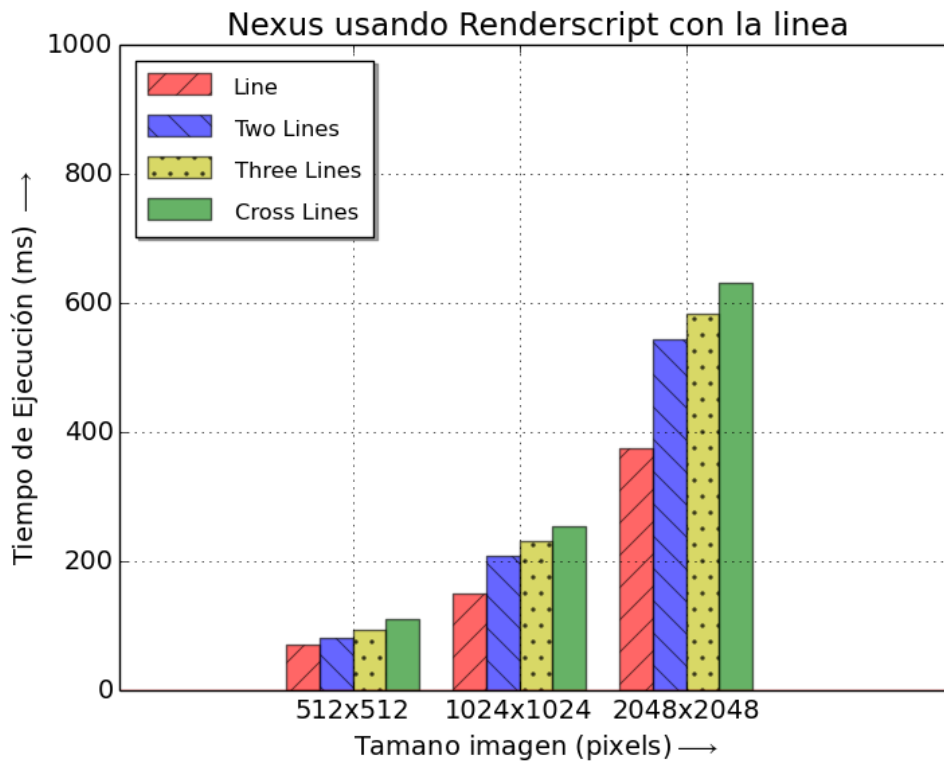


Figura 29: Gráfica de resultados usando Renderscript en líneas.

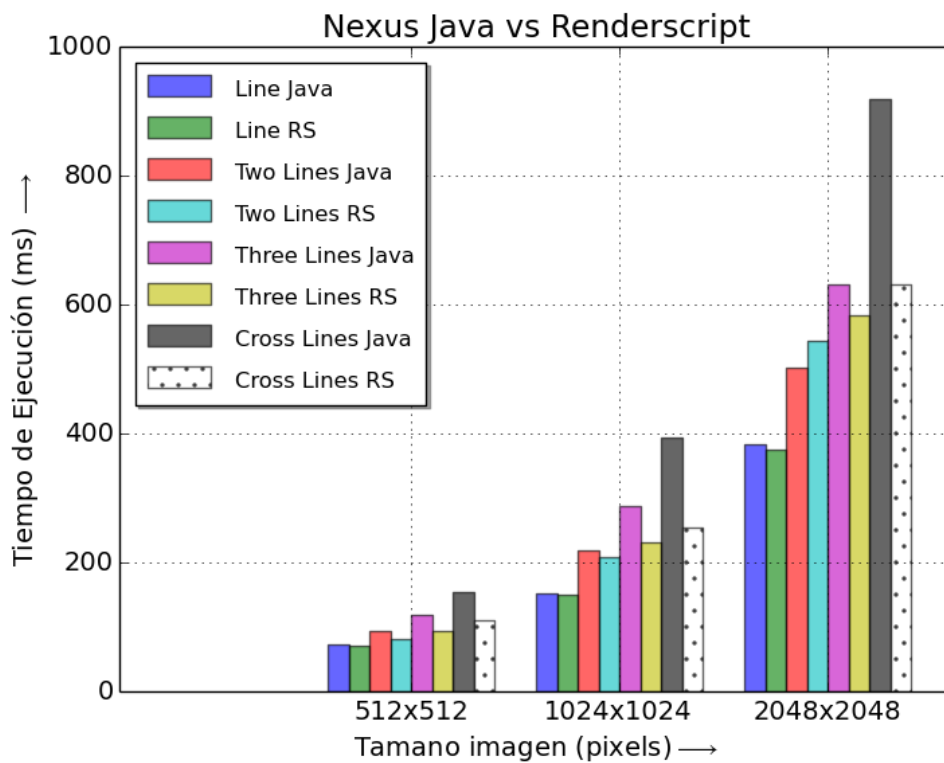


Figura 30: Versión Java vs versión Renderscript. Línea.

En la tabla 9 y la figura 31 se pueden ver los datos del uso de Renderscript en la detección de círculos. Como se puede observar el aumento de rendimiento es más marcado que en el caso de la línea, esto es debido a que en la detección de círculos se tradujeron dos funciones a Renderscript: process y findMaxima. La aceleración obtenida para el caso del círculo es de 2.057. En la figura 32 se puede ver la diferencia de tiempos entre la versión de Java y la versión de Renderscript para el caso del círculo.

		Tiempo de Ejecución			
Tamaño	Nexus 5	Círculo	Dos Círculos	Tres Círculos	Cuatro Círculos
	512 x 512	168 ms	190 ms	203 ms	215 ms
	1024 x 1024	181 ms	196 ms	577 ms	589 ms
	2048 x 2048	2027 ms	2056 ms	2250 ms	2466 ms

Tabla 9: Resultados de rendimiento usando Renderscript en la detección de círculos.

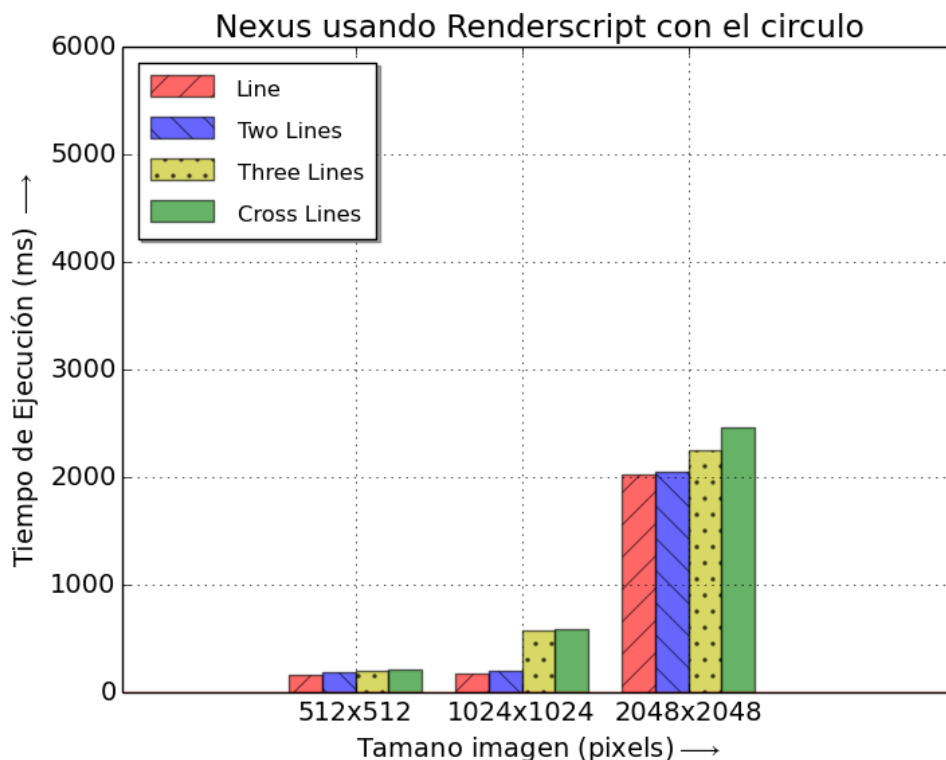


Figura 31: Gráfica de resultados usando Renderscript en círculos.

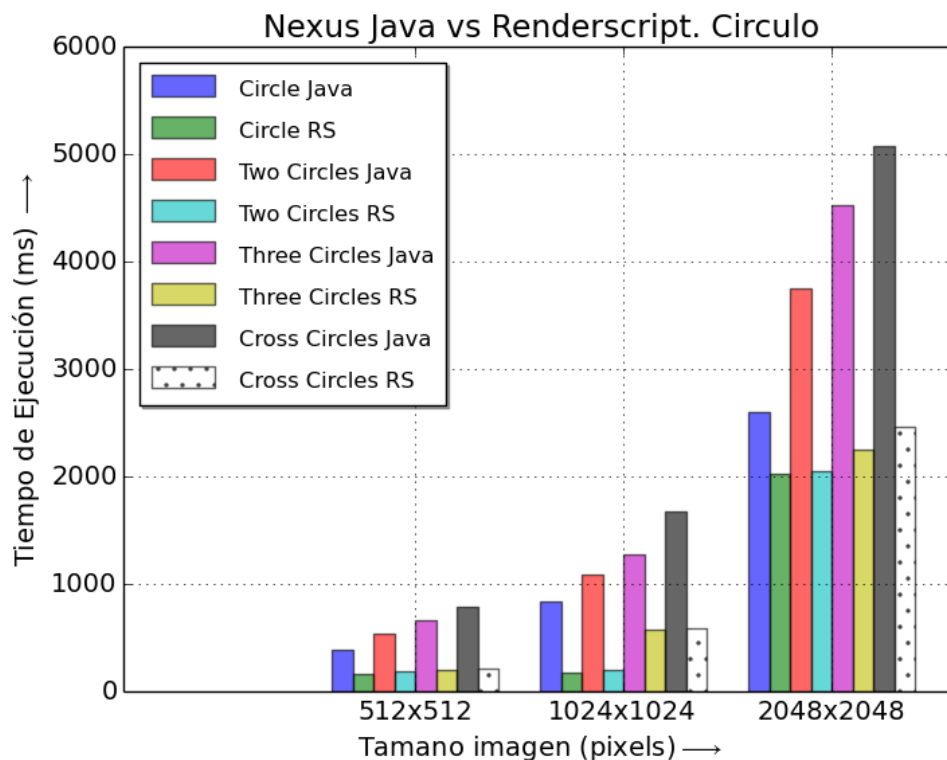


Figura 32: Versión Java vs versión Renderscript. Círculo.

Para terminar el proyecto, se propuso usar el framework Paralldroid, desarrollado en el grupo de Computación de Altas Prestaciones de la Universidad de La Laguna, que automáticamente crea el Renderscript según unas anotaciones que es necesario poner en el código Java. Esto se hizo en las mismas funciones que se tradujeron en las pruebas anteriores, tanto para la línea como para el círculo. Como se puede observar en la tabla 10 y la figura 33, para el caso de la línea los tiempos son similares, siendo en algunos casos más rápido y en otros más lento con respecto a la versión de Renderscript manual. En la figura 34 se puede observar la diferencia de tiempos entre la versión de Renderscript y la versión de Pararlldroid para el caso de la línea.

		Tiempo de Ejecución				
		Nexus 5	Línea	Dos líneas	Tres líneas	Cuatro líneas
Tamaño	512 x 512		75 ms	93 ms	102 ms	129 ms
	1024 x 1024		188 ms	198 ms	220 ms	310 ms
	2048 x 2048		461 ms	514 ms	533 ms	771 ms

Tabla 10: Resultados de rendimiento usando Paralldroid en la detección de líneas.

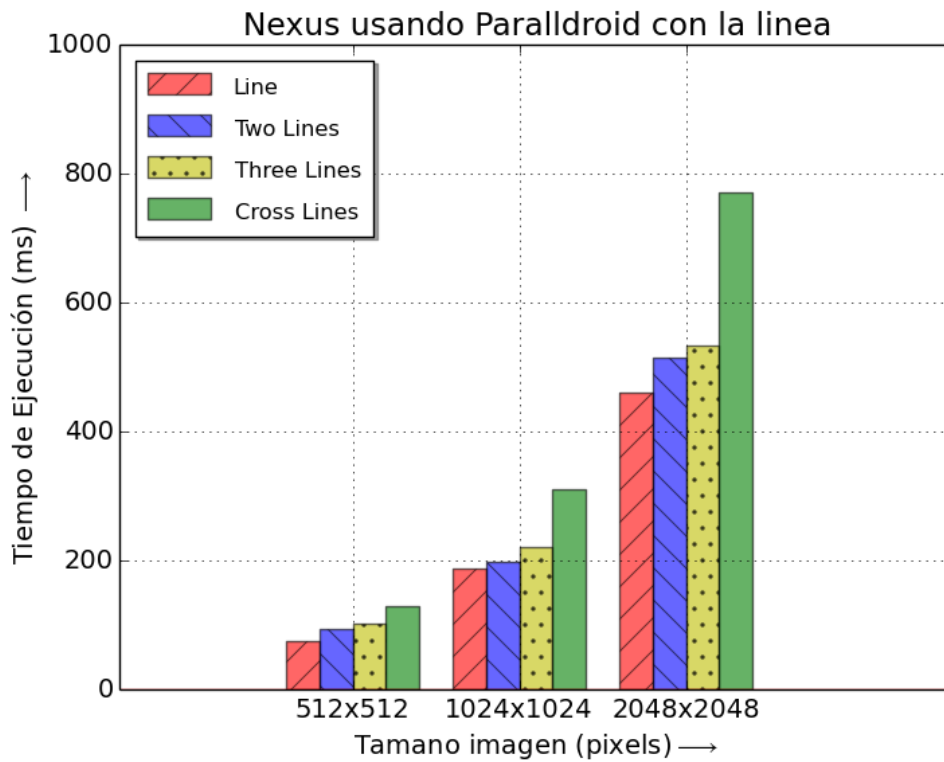


Figura 33: Gráfica de resultados usando Paralldroid en líneas.

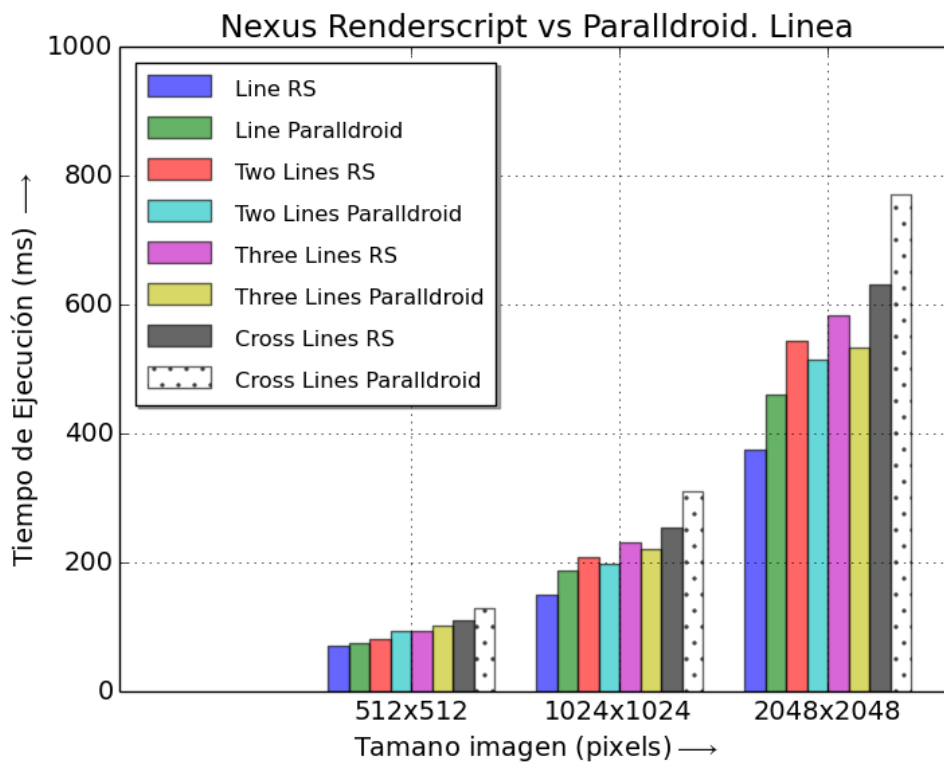


Figura 34: Versión Renderscript vs versión Paralldroid. Línea.

Y para el caso del círculo se puede observar en la tabla 11 y la figura 35, como ocurre lo mismo, para algunos casos es más rápido y para otros es más lento que la versión de Renderscript manual. En la figura 36 se puede apreciar la diferencia de tiempos entre la versión de Renderscript y la versión de Pararldroid para el caso del círculo.

Tiempo de Ejecución					
	Nexus 5	Círculo	Dos Círculos	Tres Círculos	Cuatro Círculos
<b>Tamaño</b>	<b>512 x 512</b>	157 ms	175 ms	192 ms	201 ms
	<b>1024 x 1024</b>	207 ms	213 ms	503 ms	529 ms
	<b>2048 x 2048</b>	1907 ms	1910 ms	2443 ms	2486 ms

Tabla 11: Resultados de rendimiento usando Paralldroid en la detección de círculos.

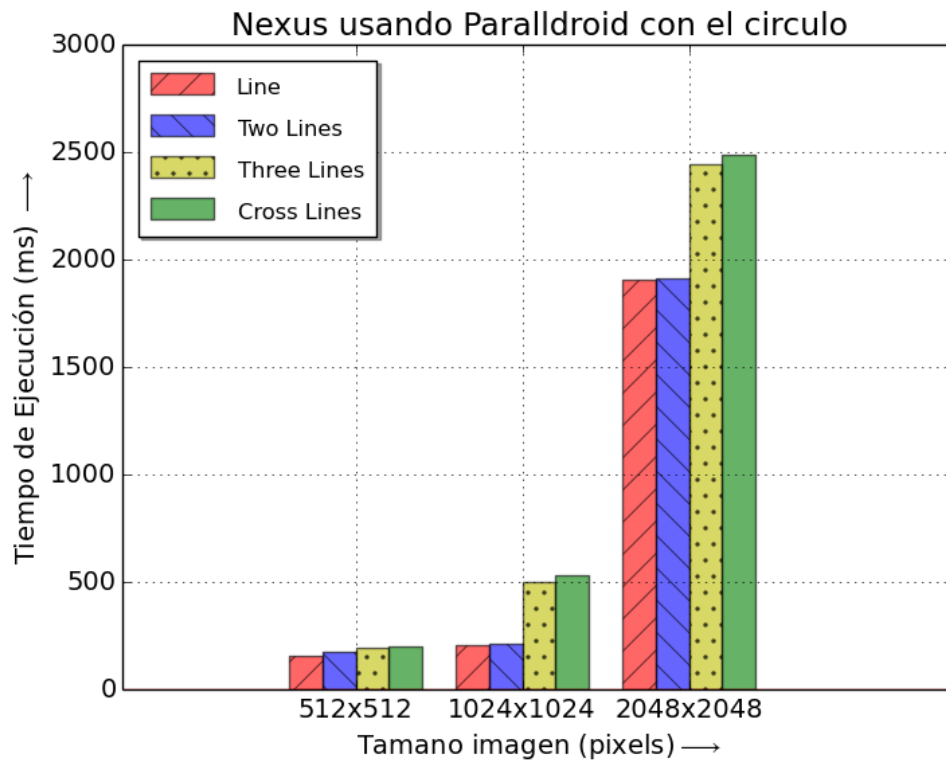


Figura 35: Gráfica de resultados usando Paralldroid en círculos.

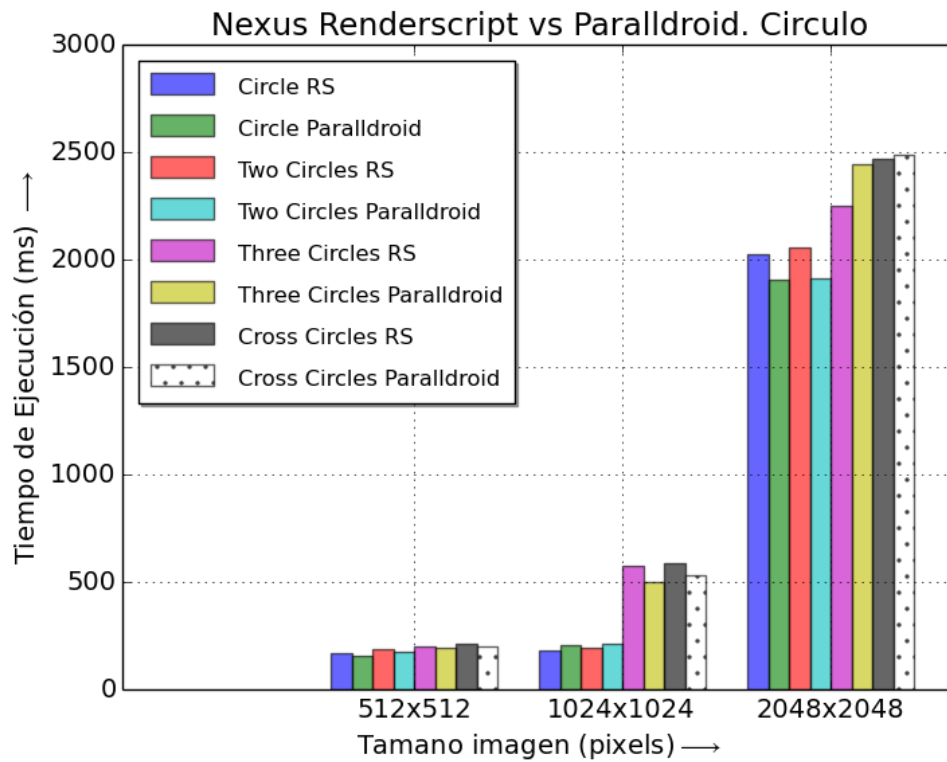


Figura 36: Versión Renderscript vs versión Paralldroid

En la figura 37 y 38, se puede apreciar la diferencia de tiempos entre las versiones de Java, Renderscript y Paralldroid, para el caso más sencillo y más complejo de la línea respectivamente. En las figuras 39 y 40, se puede apreciar de igual manera para el caso del círculo.

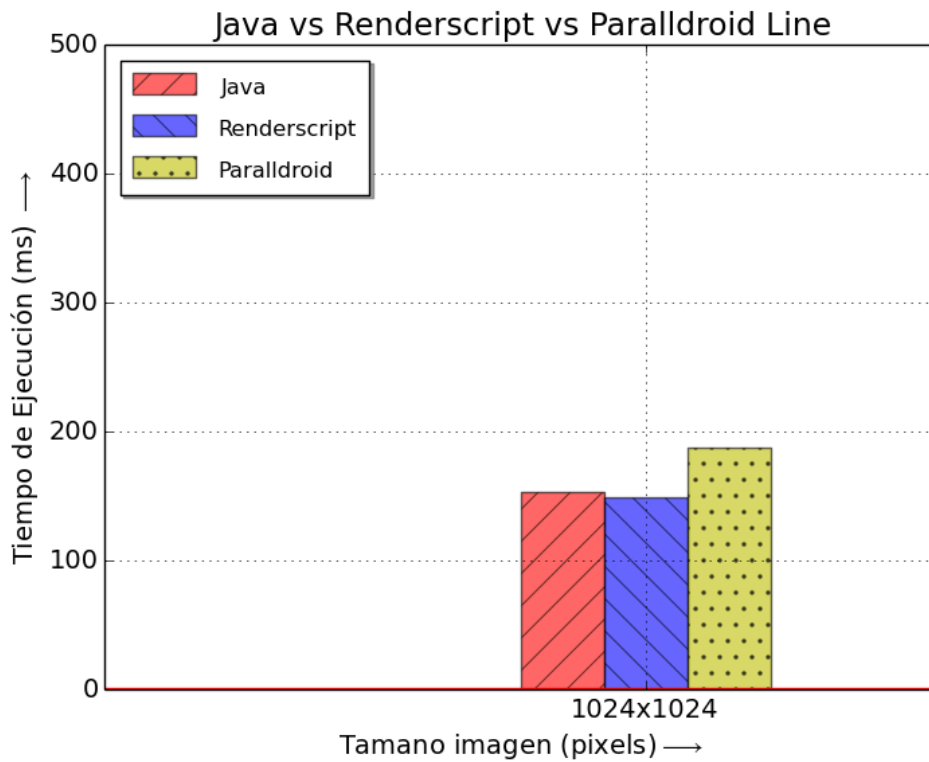


Figura 37: Java vs Renderscript vs Paralldroid. Línea.

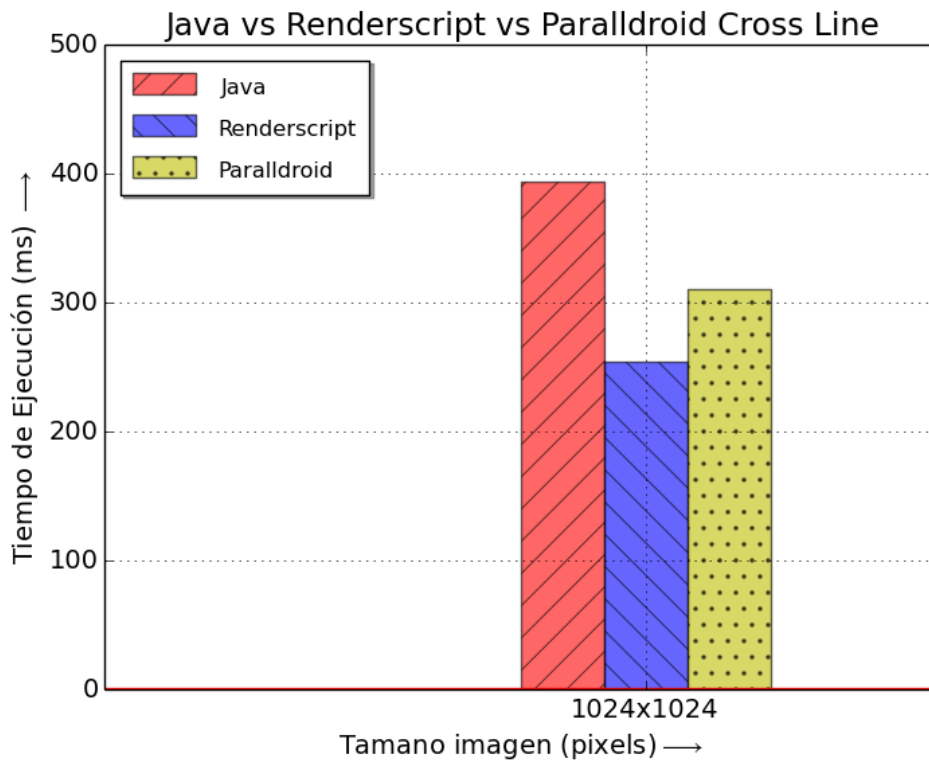


Figura 38: Java vs Renderscript vs Paralldroid. Líneas cruzadas.

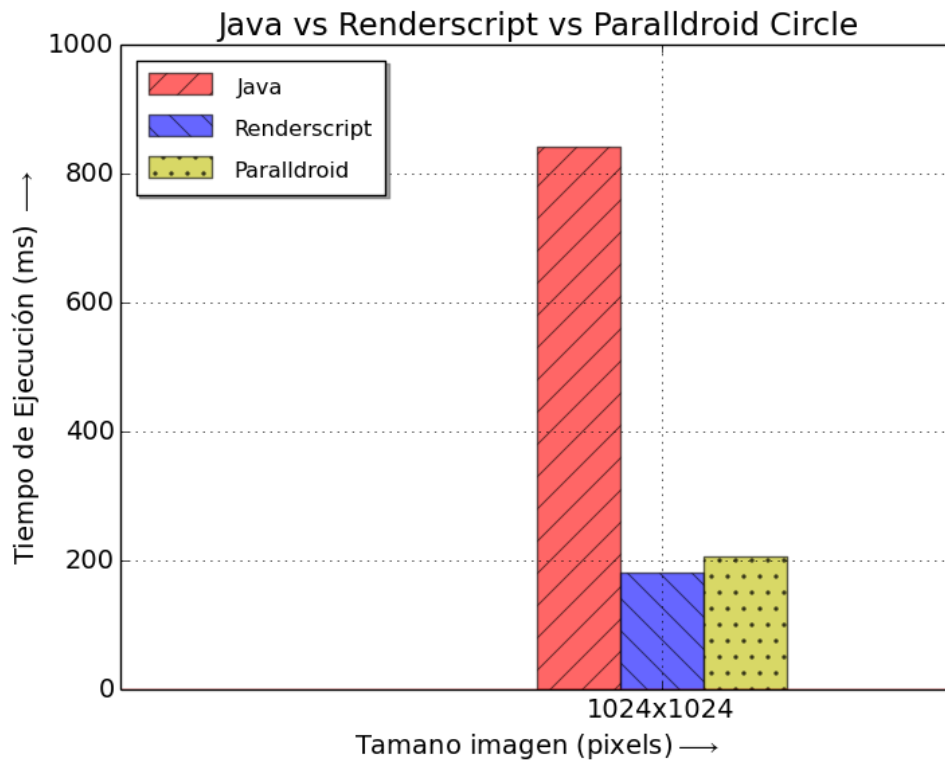


Figura 39: Java vs Renderscript vs Paralldroid. Círculo.

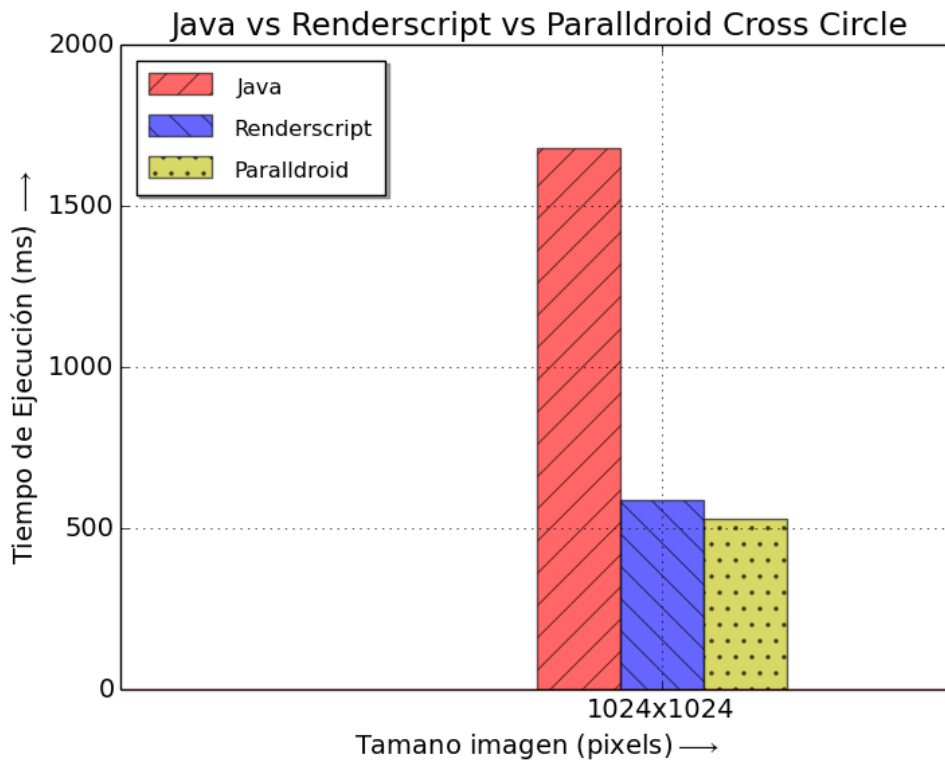


Figura 40: Java vs Renderscript vs Paralldroid. Círculos cruzados.



Hemos implementado versiones de las aplicaciones anteriores haciendo uso de la librería OpenCV. El objetivo es el de realizar un análisis comparativo de esta librería respecto de las versiones implementadas con RenderScript. Con este fin se ha hecho uso de las funciones de OpenCV que permiten acceder a la cámara del dispositivo. Como problema de partida se considera la transformación de una imagen de color a blanco y negro realizando la ejecución durante diez segundos. Se ha registrado el número de imágenes procesadas en cada segundo y el tiempo medio en procesar una imagen tanto con Rendscript como con OpenCV. Como se puede observar en la tabla 12 y gráficamente en la figura 41, el rendimiento de OpenCV es superior al de Rendscript. Esto se debe a que OpenCV carga directamente desde la cámara la imagen en blanco y negro. Debido a esto es del orden de diez veces más rápido que Rendscript, aunque se haga uso de los cuatro núcleos del dispositivo.

	Renderscript	OpenCV
Imágenes por segundo	9.2	86
Tiempo medio por imagen	108.6 ms	11.6 ms

Tabla 12: Resultados Renderscript vs Opencv.

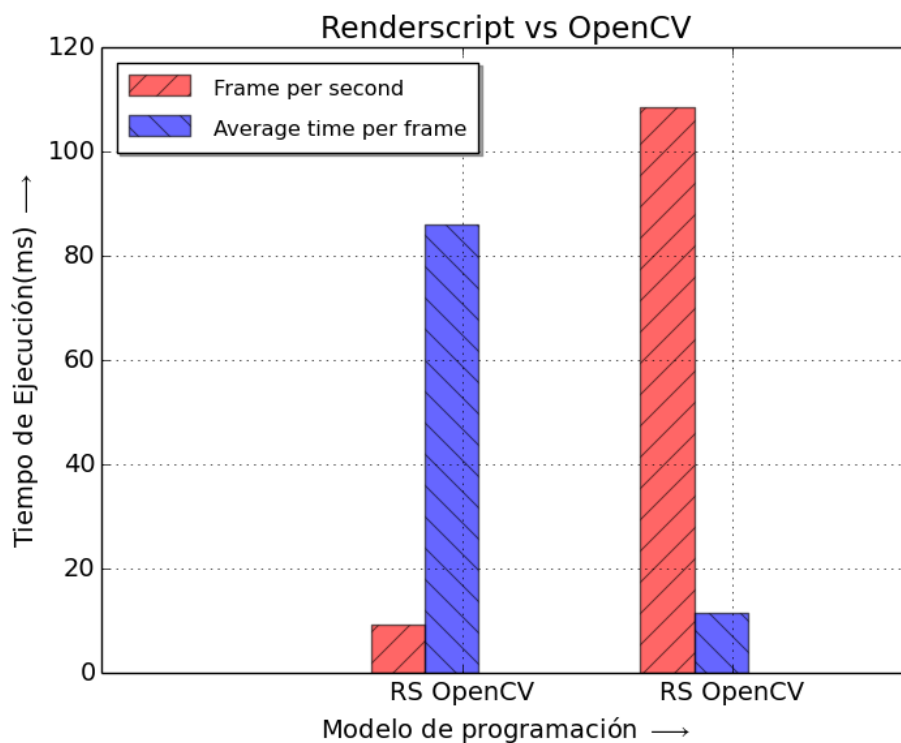


Figura 41: Rendimiento Renderscript vs OpenCV

Para que la comparativa entre Renderscript y OpenCV fuese un poco más justa, se realizaron medidas únicamente del tiempo invertido por Renderscript en procesar cada frame de la cámara, obviando el tiempo de creación y preprocesado de objetos. Los resultados se muestran en la tabla 13 y en la figura 42. Como se puede apreciar, el rendimiento es muy cercano al que ofrece OpenCV.

	Renderscript	OpenCV
Imágenes por segundo	75.8	86
Tiempo medio por imagen	13.2 ms	11.6 ms

Tabla 13: Renderscript vs OpenCV

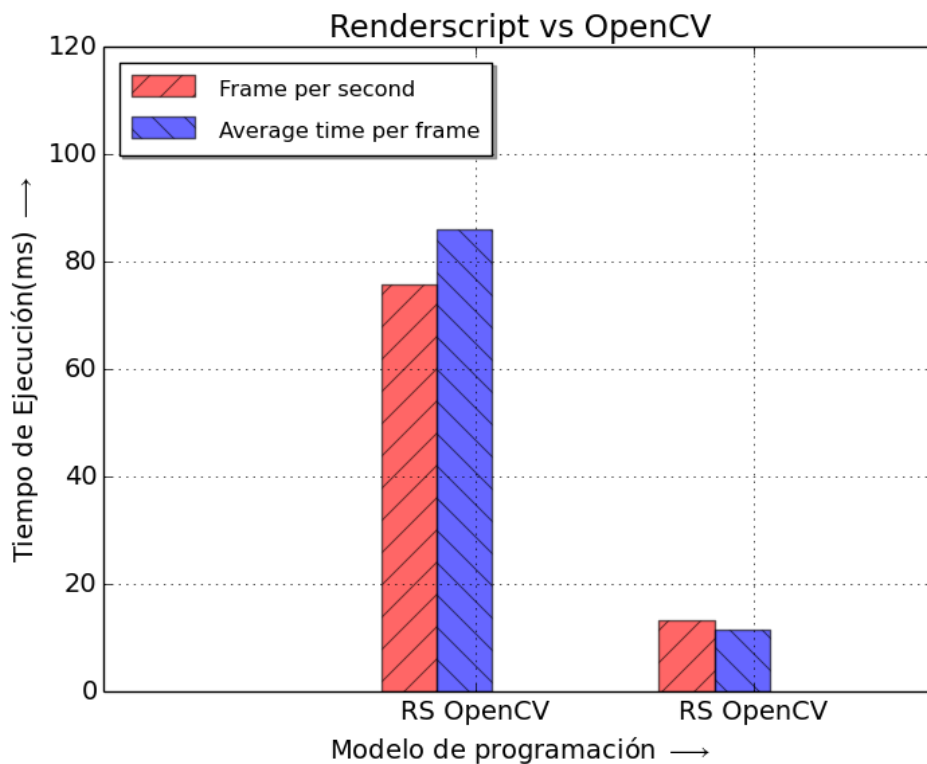


Figura 42: Versión Renderscript (sin tiempo de preprocesado) vs OpenCV

## CONCLUSIONES

A modo de conclusión, intentaremos matizar aspectos que deben ser considerados al desarrollar aplicaciones en la plataforma Android que pueden tener impacto en el rendimiento.

Un aspecto importante está relacionado con la memoria, es importante liberar los recursos cuando ya no sean necesarios, se usará la función `.recycle` o `.destroy`, o en caso de que no disponga, se igualará la referencia a `Null`. Con esto se evitará que los procesos iterativos hagan crecer la pila hasta quedarse sin memoria y producir una excepción.

Cuando se desarrollan aplicaciones que tratan imágenes hay que considerar un conjunto de factores que influyen en la memoria consumida y en la velocidad de procesado. Estos factores vienen determinados por el uso que se pretende dar a la imagen, es necesario almacenarla o sólo mostrarla, calidad requerida en la imagen, necesidad de escalado, mantener la resolución original. Si se desea realizar modificar la imagen, es recomendable convertirla en un `Array` de `Int` para su proceso y luego realizar la conversión a `Bitmap`.

Respecto a `OpenCV` hay que destacar las ventajas de usar una librería de Visión por Ordenador, que aporta todas las funciones necesarias para que resulte realmente fácil la creación de cualquier aplicación. Presenta la desventaja de las limitaciones en las imágenes que hay que usar, y el procesado previo requerido por algunas funciones. También cabe destacar la incompatibilidad con `RenderScript` que permitiría mejorar el rendimiento de esta librería.

Si la aplicación necesita más velocidad de la que puede ofrecer Java, se hace necesario usar `RenderScript`. Desarrollar en `RenderScript` no es una tarea fácil, debido a la falta de documentación, a los pocos ejemplos de código que existen, a la poca información que aporta el compilador cuando el código contiene un error y a lo limitada que está la depuración. Presenta una curva de aprendizaje bastante pronunciada. Una vez que se ha conseguido un cierto conocimiento del paradigma, el aumento de rendimiento es bastante notable. En estos momentos no está claro el nivel de compromiso de Google con este proyecto, se hace necesario incluir un incremento de funcionalidades y una mejora de la documentación, los códigos de ejemplo, la gestión de errores en el compilador y la depuración de las aplicaciones.

Paralldroid, intenta resolver, parcialmente, algunos de los problemas derivados de usar Renderscript. Se encarga de la creación automática de este código a partir de las anotaciones realizadas en Java. Como se ha apreciado en los resultados experimentales el rendimiento es similar al de código realizado por un desarrollador. Se trata de una herramienta interesante dada la funcionalidad que aporta evitando la complejidad inherente al Renderscript.

## CONCLUSIONS

In conclusion, we will try to refine aspects that must be considered when developing applications on the Android platform that can have an impact on the performance.

An important aspect is related to memory. It is important to free resources when they are not longer needed. The function `.recycle` or `.destroy` will be used. When not available, the reference sets equal to `Null`. This will prevent the iterative processes from making the stack grow up running out of memory, causing an exception.

When applications deal with images, it must be considered a number of factors that influence the consumed memory and the processing speed developed. These factors are determined by the intended use of the picture. You have to store it or just display it, image quality required, need scaling, maintaining the original resolution. If you want to change the image, you should make it using a `Int Array` to process it and then convert it to `Bitmap`.

Regarding `OpenCV` we must highlight the advantages of using a computer vision library, which provides all the necessary functions to make it really easy to create any application. It has the disadvantage of the limitations on the images to be used, and pre-processing required for some features. It is also noteworthy the inconsistency with `Renderscript` that would improve the performance of this library.

If the application needs more speed than Java can offer, it is necessary to use `Renderscript`. Developing `Renderscript` is not an easy task due to the lack of documentation, a few code examples that exist, the limited information provided by the compiler when the code contains an error which is already limited debugging. It has a fairly steep learning curve. Once you have achieved a certain knowledge of the paradigm, the performance increase is quite remarkable. At present it is unclear the level of commitment to Google with this project, it is necessary to include increased functionality and improved documentation, sample code, error handling in the compiler and debugging applications.

Paralldroid tries to partially resolve some of the problems of using Renderscript. It is responsible for the automatic creation of this code from the Java annotations. As it has been seen in the experimental results the performance is similar to the code created by a developer. This is an interesting tool that provides functionality, avoiding the complexity inherent in Renderscript.

## Bibliografía

- [1] VASE Laboratory. School of Computer Science and Electronic Engineering. University of Essex. Dr Adrian F Clark.  
<http://vase.essex.ac.uk/software/HoughTransform/>
- [2] Computer Vision Demonstration Website. Electronics and Computer Science. University of Southampton.  
[http://users.ecs.soton.ac.uk/msn/book/new\\_demo/houghCircles/](http://users.ecs.soton.ac.uk/msn/book/new_demo/houghCircles/)
- [3] [https://en.wikipedia.org/wiki/Hough\\_transform](https://en.wikipedia.org/wiki/Hough_transform)
- [3.1] <https://www.google.com/patents/US3069654>
- [3.2] <http://dl.acm.org/citation.cfm?id=361242>
- [3.3] <http://www.cs.utexas.edu/~dana/HoughT.pdf>
- [4] <https://es.wikipedia.org/wiki/Dalvik>
- [5] <https://developer.android.com/sdk/index.html>
- [6] <https://developer.android.com/tools/sdk/ndk/index.html>
- [7] <https://developer.android.com/guide/topics/renderscript/index.html>
- [8] <https://developer.android.com/tools/building/index.html>
- [9] [https://en.wikipedia.org/wiki/Dalvik\\_\(software\)](https://en.wikipedia.org/wiki/Dalvik_(software))
- [10] <https://developer.android.com/tools/debugging/ddms.html>
- [11] <https://play.google.com/store/apps/details?id=com.cgollner.systemmonitor>
- [12] <http://www.aishack.in/2010/03/the-hough-transform/>
- [13] <http://www.aishack.in/2010/03/the-hough-transform/2/>
- [14] <http://www.aishack.in/2010/03/circle-hough-transform/>
- [15] <http://opencv.org/platforms/android.html>
- [16] <https://developer.android.com/reference/android/graphics/Bitmap.Config.html>