KTH Computer Science
and Communication

Graphics with limited resources

Ludvig Jonsson
ludjon@kth.se
Tim Johan Malmström
tjma@kth.se

Degree Project in Computer Science, First Level, DD143X
Supervisor: Mads Dam
Examiner: Mårten Björkman

Bachelor of Science Thesis Stockholm, Sweden 2012

Abstract

This paper aims to investigate what is being done to improve the use of graphics on mobile devices with focus on the Android platform and especially Android-based smartphones. Today most people use a smartphone in their everyday life and much of the intended use of traditional personal computers are moving towards mobile devices. Because of this, making the most of the limited system resources is a highly relevant issue.

The subject is divided into three parts. The first part investigates how graphics rendering is done on the Android platform. The second part is a comparison of the available rendering options on Android and their respective resource usage. The last part consists of a practical test of the different rendering methods and results analysis.

The conclusion that could be drawn from this exercise is that there is no method that is optimal for all situations, all available rendering options have different uses and advantages. The choice of method should be based on what kind of graphics is being rendered and how resource demanding the rendering instructions are.

Contents

Contents

Introduction **1**
- Preface . . . 1
- Purpose . . . 1
- Research focus . . . 2
- Problem statement . . . 2

Background **3**
- History . . . 3
- Resources . . . 3
  - Platform . . . 4
  - Central Processing Unit (CPU) . . . 4
  - Graphics Processing Unit (GPU) . . . 5
  - Random Access Memory (RAM) . . . 5
- Android graphics APIs and rendering . . . 5
  - Canvas API . . . 5
  - OpenGL . . . 6
  - Renderscript . . . 7
- Rendering optimization techniques . . . 7
  - Hardware and software rendering . . . 7
  - Android and hardware acceleration . . . 8

Discussion **9**
- Differences between the APIs . . . 9
- Hardware and software rendering . . . 10

Testing method **11**
- The testing application . . . 11
- The testing device . . . 12
- Benchmarking . . . 13
- Test cases . . . 13
- Expected results . . . 14

Results analysis **15**
Implementation using the different APIs 15
Test results 16

Conclusions **19**
Improvements being done to the graphics rendering 19
Android rendering methods 20

Abbreviations **21**

Test data **22**
Canvas with software rendering 22
Canvas with hardware accelerated rendering 22
OpenGL 23

Bibliography **24**

Introduction

Preface

This document is written for the course Degree Project in Computer Science, DD143X(dkand12) at the Royal Institute of Technology(KTH). It is written by Tim Malmström and Ludvig Jonsson who both at the time of writing this documents are third year students at the computer science program at the Royal Institute of Technology in Stockholm. The document is written with the purpose of investigating what is being done to improve the use of graphics within mobile devices with limited resources.

The supervisor for writing the document is Mads Dam.

The writing has been divided between the authors as follows:
Introduction - Both
Background - History - Ludvig
Background - Resources - Tim
Background - APIs and rendering - Ludvig
Background - Optimizations - Tim
Discussion - Differences - Ludvig
Discussion - Hardware and software - Tim
Testing method - Tim
Result analysis - Tim
Conclusions - Both
Layout and Design - Both
Implementations - Tim

Purpose

With personal computers being used everywhere in todays society the demand for mobile systems with the same funtionality has emerged. This has put pressure on manufacturers to create smaller and lighter products that have the same functionality as the bigger systems. To achieve this functionality, software developers need

to create software that use the more limited resources as efficient as possible.

When the smartphone and the tablet was introduced to the world the limited resources was one of the manufacturers biggest problems. To get people interested in change and trying something new, the product has to have clear advantages over the previous ones.

The use of this kind of limited system resources is the focus of this paper. In terms of graphics, what is being done and what can be done to use the resources as efficiently as possible?

Research focus

Since the subject "Graphics with limited resources" is quite wide, we decided to limit our focus to smartphones. We find the subject of smartphones interesting and highly relevant since most people in the modern world are using one in their everyday life.

We have also decided to focus on smartphones running the Android platform since the Android phones are dominating the smartphone market having 50,9 % of the total smartphone sales according to Gartner's analysis of global smartphone sales in Q4 2011.[8]

Problem statement

The problem statement is as follows:

- What recent changes have been made in Android to make the use of the limited resources more efficient?
- What are the differences between the different rendering methods used in Android and their resource usage?
- Evaluate the performance and resource usage of the rendering methods in Android.

With the first question the goal is to understand what techniques are used to make the resource usage more efficient and why. The second question is an analysis and comparison of the different rendering methods used in Android. The third question is answered in two different parts. First the rendering methods are compared from a theoretical point of view. Then the theoretical conclusions are tested using benchmarks to compare their performance. Following this the test results are analysed leading to conclusions that answer the problem statement.

Background

History

When comparing a modern mobile phone to one released fifteen, ten or even five years ago, it is clear that the development is progressing very fast. The devices still serve the same basic purposes, like being able to make phone calls and send text messages. With the increasing hardware performance and the phones being more advanced in general, the amount has grown from just a few to what seems like unlimited uses.

Some of the differences are obvious when comparing the looks of an older mobile phone and a modern smartphone. The older monochrome displays and physical buttons where replaced by much larger, touch-sensitive color displays. Since the older phones were mostly used to make phone calls and send text messages, the hardware was not and did not really need to be that powerful. With the new, larger screens supported by new, more powerful hardware, the differences in functionality between the smartphone and the standard personal computer have become virtually non-existent.

Earlier, the different functions of the mobile phones were integrated in the operating system and the different phone manufacturers had their own platforms. Nowadays almost all smartphones are based on an already established platform like Android or iOS. The big advantage with this is that when releasing an Android-based smartphone (for example), there already exists a couple of million applications which can be installed to extend the functionality of the device.

The more powerful hardware makes it possible for these applications to include advanced graphics, but since the smartphones are limited in aspects like physical size and battery life, it is still very important to use the hardware as efficiently as possible in the graphics rendering process to make the most of it.

Resources

The graphics rendering process is a process that requires the use of several different system resources. What resources are being used and how they are being used has varied a lot over time, from having distinct RAM and GPU that does not affect the performance of the rest of the system whatsoever, to the more advanced

cooperations between GPUs and the CPUs.

The graphics rendering in modern personal computers are developing in a direction that increases the separation between the graphics processing and other processes. This is mostly since the space required for cooling and for the processors is not a problem. Because of the limited space within a mobile device most of the newly developed devices contain a CPU with a integrated GPU which means that the graphics processing share the same resources as other system processes. This makes efficient graphics rendering an even more important part of the applications.

The most important system resources used by the graphics rendering is the CPU, GPU, RAM and the platform which is not really a resource per se, but more of a controller for the resources. Even though the CPU and GPU use the same resources they are separated since separating CPU requests from GPU requests makes the process handling more efficient.

Platform

A platform is both a sort of hardware architecture and a chipset containing a software framework.[10] In the case of a mobile device the platform usually contains all the hardware of the device since there rarely are any extension ports or possibilities to change the device's hardware. This is something that makes the system more stable and reliable than a personal computer since there is less that can go wrong when there is no possibility to switch hardware.

The software framework put on the chipset is in Android devices parts of the operating system and some basic resources like timers, boot instructions and interrupt controllers. The chipset also contains the instructions and controllers for the hardware components.

The reason why the platform is relevant in this case is because the platform controls the hardware resource usage and thereby how much and what resources can be used by the rendering processes.

The platform that we performed our tests on is the one used in the Android phone Samsung Galaxy Nexus. The platform is called OMAP4460 and is developed by Texas Instruments.[7] The Samsung Galaxy Nexus is a recently released Android smartphone running the Android 4.0 operating system.

Central Processing Unit (CPU)

The central processing unit is the part that executes instructions on a system. The CPU takes instructions in binary code and executes them. The biggest problem with the CPU within mobile devices is that it generates a lot of heat. This often forces the manufacturers to limit the processors capabilities to be able to use them in the smaller devices.

In our testing the device uses a 1,2 GHz dual core CPU. According to Texas Instruments the processor used in the OMAP4460 has the capacity to reach speeds

up to 1,5 GHz although because of the problem with cooling in a mobile device Google has chosen to lower the clock frequence.

Graphics Processing Unit (GPU)

A GPU is an electronic circuit designed to be able to quickly alter memory and perform instructions rendering images to a frame buffer that is intended to be shown on a display. As mentioned earlier there are several different kinds of GPU types. The ideal type performance-wise is the one where the GPU runs a separate processor with the sole purpose of executing graphics instructions. This gives the graphics processor better image rendering speed than the more general CPU. The kind of GPU that is used in our testing is an integrated GPU.

The integrated GPU does not have a processor of its own but is instead more of an optimizing unit that optimizes the instructions for calculating graphics and then relies on the CPU for executing them. The integrated GPU provides enough performance to calculate graphics in most situations. There are many advantages with using integrated GPUs - it is more cost efficient, space efficient and consumes less power. The only downside is the performance.

Random Access Memory (RAM)

The random access memory (RAM) is a memory which is made for being able to read and write from all memory positions quickly. RAM is therefore used as a temporary data storage for active processes, any data that needs to be accessed quickly. For more long term storage, devices use a storage memory that is not as quick but has room for much more data.

In mobile devices the RAM is usually divided into two parts, the part that is accessed and used by the operating system and the part reserved for other processes. This prevents less important processes from having any affect on the performance of the operating system.

The RAM is a very important resource for the graphics processing since the already existing bitmaps used for the rendering as well as the draw buffer are stored there for fast access.

Android graphics APIs and rendering

Rendering is the process of generating the graphical components of a computer program. The purpose of this section is to go through the different rendering methods, their advantages and disadvantages.

Canvas API

The Canvas API is a very simple tool for rendering graphics on Android. To be able to understand how the Canvas API works, the first thing to understand is what a

View object is and how it is used in the Android development process. The View object is the basic building block for user interface components when developing a standard SDK application for Android. It handles the graphics rendering and user input.

A Canvas is drawn by calling the onDraw method of the View with the Canvas object as a parameter. By calling methods of the Canvas class such as drawText, drawCircle and drawPicture the developer can decide what graphical components should be rendered and where they should be positioned on the screen.

The Canvas API is the standard way of rendering graphical components of a GUI application on Android. It is a great tool for simple graphics, but there are other methods better suited for including more advanced graphics in an application.

OpenGL

OpenGL (short for Open Graphics Library) is a cross-platform graphics language for producing 2D and 3D graphics. It is developed by Silicon Graphics and is the most widely used graphics API of the computer industry.[12]

Android uses a subset of OpenGL called OpenGL ES (OpenGL for Embedded Systems). It is specifically designed for embedded systems like video game consoles and mobile phones, and is basically a stripped down version of OpenGL. [5]

Android supports OpenGL in two different ways - through the standard Android framework API and through something called Android NDK. NDK is short for Native Development Kit and it is used to build applications (or parts of applications) in native code outside the Android Java VM(Virtual Machine) as opposed to building applications in Java on the SDK level.

When building applications with the NDK, there is no access to the usual GUI toolkit (View objects, events) of the Android platform, but with full access to OpenGL the NDK is ideal for a few specific purposes (like porting applications written in OpenGL for another platform).

Using native code does not automatically increase the performance of the application, but it provides a more complex view of the application and more customizability. The native code also gives developers the possibility of embedding corresponding native libraries into the application package file and thus the possibility to embed OpenGL.

The standard option is to use something called OpenGL wrapper functions. A wrapper function is basically a function whose main purpose is to call a second function in another framework or library. In this case, the wrapper functions are using something called JNI (Java Native Interface) to make calls down to the native OpenGL library. OpenGL wrapper functions makes it possible for the developer to use the OpenGL API at the SDK level. This means an application can be written using the standard SDK APIs and still include parts of OpenGL.

Renderscript

Renderscript is another graphics rendering API for Android, developed with the goal of bringing a lower level, high performance graphics API to the Android platform.[6] An application using Renderscript is a standard SDK application running in the Android VM but with some external parts operating at native level. The external parts are written in Renderscript code (in the C99 language).

The application can be divided into two parts, the Android framework and the Renderscript runtime. The lower level Renderscript runtime is controlled by the higher level Android system that runs in the Android VM. The memory allocation is handled by the Android VM which binds the memory to the Renderscript runtime so the Renderscript code can read from and write to it.

By compiling the Renderscript native code at runtime, the application maintains the portability of the application and the benefits of native code.

With Renderscript, the performance of your application can be optimized by running graphics operations or other advanced computations on the native level (when they simply take too much time to run at the SDK level). Renderscript also has the possibility to, at runtime, determine the best way to run a particular operation to achieve the highest possible performance. Renderscript may decide to send the graphic operations to the GPU or decide that it is better to split up the computations and send them to the multi-core CPU.

All this can be done without losing any of the functionality of the SDK APIs.

Rendering optimization techniques

After chosing an API to work with there are also some techniques to optimize the rendering performance further. One of the biggest changes recently to optimize rendering performance is Hardware acceleration in the Canvas API. This section explains the difference between hardware and software rendering and how hardware acceleration was introduced in Android development.

Hardware and software rendering

The main difference between hardware and software rendering is how rendering processes are executed. When rendering using software rendering all instructions are sent directly to the CPU without using any form of graphics hardware. As opposed to software rendering the hardware rendering runs all its rendering instructions through the graphics hardware. There are advantages and disadvantages of both methods.

The biggest advantage of using software rendering is that it is simple. Since the instructions are sent directly to the general purpose CPU they do not have to be restricted by the more limited capabilities of the GPU. The limitations of the GPU is a big disadvantage with using hardware accelerated rendering, they make the code more complex. Another disadvantage of using hardware rendering is that

the more complex instructions require use of more memory. Since the instructions are more specific in the hardware rendering the processor can in most cases execute them faster than in the case with software rendering.

So choosing between hardware and software rendering is a question of the balance between the use of RAM and CPU. Since they both have their advantages and disadvantages the choice must be made for each case.

Android and hardware acceleration

Android has always used some kind of hardware acceleration[3] although before the Android 3.0 release it was limited to the use of system graphics such as menus and the top scrollbar. Since the graphics in Android devices has always been divided into different components called windows, the handling of those windows was done using hardware accelerated rendering while the content of those windows was software accelerated.

Before Android 3.0 the limtations of hardware use provided additional stability and security since no third party applications had full access to the hardware resources of the device. In Android 3.0 Google added the

<application android:hardwareAccelerated="true" ... >

attribute to applications. This was mostly made as a test to see if the system would run stable using the hardware acceleration and was therefore disabled as default. After seeing the improvement made by this change Google decided to have the hardware acceleration enabled as default in Android 4.0 even though it was not yet usable for all drawing operations.

Discussion

In this chapter the rendering methods are compared leading to conclusions which are tested in the following chapter.

Differences between the APIs

In the previous chapter, four rendering methods were mentioned:

- Canvas API
- Renderscript
- OpenGL Wrappers
- NDK OpenGL

These rendering methods were of course not created with the exact same purposes in mind. The Canvas API is meant to be a simple tool for rendering the GUI components (like buttons and menus) and other simple graphics of a standard Android application. Using the more advanced options for this kind of graphics would just make things more complicated rather than improve the performance of the application.

The purpose of the Canvas API is to be able to render simple graphical components in a simple and straightforward way. The other options are meant to be used for more advanced graphics like 3D graphics, something that the Canvas API simply does not support. OpenGL and Renderscript were created for the same main purpose - to make it possible for the Android developers to go beyond what the Canvas API has to offer, both in terms of functionality and performance.

Since Android 3.0, when GPU acceleration was added to third party applications, the overall performance of the Canvas API has increased. When hardware acceleration is enabled, all rendering done at the SDK level is processed as OpenGL calls to the GPU. This increases the overall performance when rendering advanced graphical components.

Both the OpenGL options come with some disadvantages.

Since the wrapper option needs to call from SDK to native level for every OpenGL operation the NDK option will achieve better performance rendering advanced graphics. When using the NDK option with native access to OpenGL, the

application loses its portability since the NDK compiles the application for the specific chip architecture that it is running on. The NDK option also does not contain the GUI toolkit found in the SDK APIs, like the View objects and the events.

Renderscript is supposed to solve the problem of having to choose between performance and portability. Renderscript offers performance similar to the performance of OpenGL with the NDK. This is accomplished by executing performance critical parts of the application on a native level as opposed to executing them in the Android VM.[2]

By compiling the Renderscript code at runtime, it solves the problem with native code and portability. This is different from the OpenGL options where the whole application either is written in native code (with the NDK) or the whole application runs in the Android VM. Renderscript is a mix between these two in the sense that it offers both the portability of the OpenGL wrappers option (and the rest of the tools and APIs of the SDK) and performance equal to OpenGL in the NDK.

Hardware and software rendering

Since hardware rendering is considered to be an optimization of the Canvas APIs rendering and therefore also enabled by default in the latest Android release, assumptions could be made that hardware rendering is simply faster and better than software rendering. This, however, is not always the case.

Since the hardware rendering uses a different model for the graphics rendering the instructions go through more stages of processing than they do in the software rendering model before actually being executed. This means that for simple rendering instructions where the graphics processing can not improve the execution time of the instructions, hardware acceleration just delays the execution of the instructions.

So when the rendering instructions runs hardware accelerated they are being processed more before being executed. This also provides the effect of hardware accelerated rendering needing to allocate more memory than the software rendering would need to execute the same instructions.

For most advanced rendering instructions the hardware rendering can improve the instructions execution time more than the actual processing takes which improves the overall performance. But if the instructions are simple the rendering actually benefits from just running the instructions by software rendering. Hardware accelerated rendering needs to allocate more memory than the software accelerated rendering for both simple and advanced instructions.

Testing method

As discussed in the previous chapter there are big differences between the APIs regarding performance within applications and quite specific cases for where to use which API. Another important aspect of the choice is the complexity of the code used for the rendering in the different cases.

The testing application

The test case that is going to be run in the test is a simple graphics rendering application. Since the goal of the application is not to push the hardware to the maximum performance, but instead to show differences between the rendering methods in a concrete test.

The testing application will show ovals and squares of different colors and sizes as fast as possible. To keep the randomization of the shapes from stealing system resources from the rendering the first part of the testing is a program generating test data. It creates 1000 randomized shapes and stores them in a file read by the testing programs. The predefined shapes also makes sure that the APIs face the same level of difficulty in the rendering since they all render exactly the same frames during the testing.

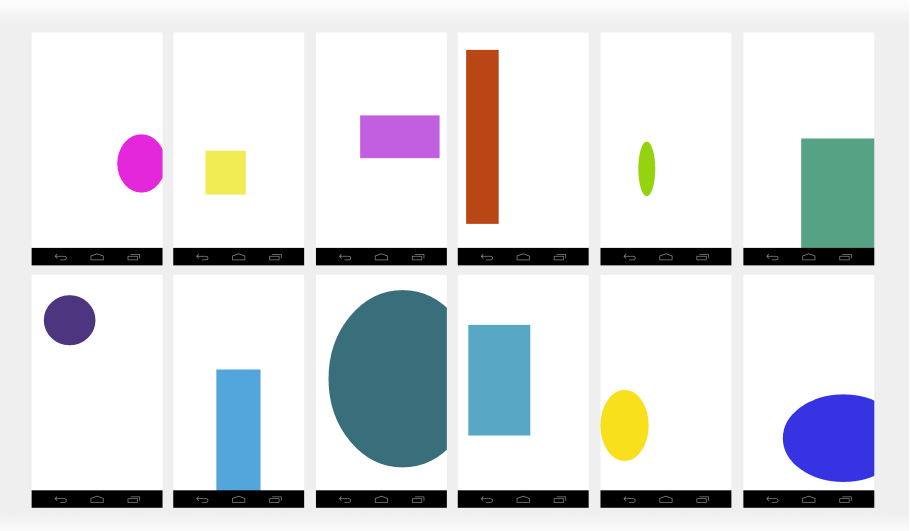

Figure 1. Some screenshots from the testing application

The testing device

The testing device is as mentioned earlier a smartphone using the Android 4.0 operating system. The model chosen to work with is a Samsung Galaxy Nexus which was released in November 2011.

Figure 2. Samsung Galaxy Nexus

Benchmarking

Benchmarking is the process of comparing performance metrics. The dimensions compared are often in terms of time, cost and efficiency. Benchmarking is used to compare the different methods in the test. The units that are benchmarked are FPS (Frames Per Second) and memory usage.

FPS is probably the most interesting unit since it is the main performance unit within rendering, and user-experienced problems with graphics often concern having a low framerate. The memory usage is very interesting when looking at the improvements of the rendering techniques. As stated earlier the decision between hardware or software accelerated rendering is often a compromise between memory usage and performance.

Test cases

To get results that seem relevant to the discussion we chose to run the test using Canvas with both hardware accelerated rendering and software rendering. We also chose to run the test using OpenGL to get a comparison between the higher level Canvas API and one operating at a lower level.

Expected results

The rendering used in the test cases will not be too advanced since there will only be one object rendered at a time but the application will still push the APIs to give as high FPS as they can. We expected OpenGL to be the API that can render the highest amount of FPS since it runs at lower level and is made for rendering animated graphics. Other results that are expected is that the hardware accelerated rendering should need to allocate more memory than the software rendering and should show a different amount of FPS. The FPS difference between the software and hardware rendering will probably be in favor of the software rendering because of the simplicity of the rendering instructions used.

Results analysis

Implementing and running tests using some of the APIs gives an even deeper understanding about their differences. This chapter contains information about what results the testing gave and discussion about the results.

Implementation using the different APIs

For a developer it is very important how the different APIs are implemented. If the application is not meant to be using advanced graphics rendering the developer wants simple functions for drawing graphics. Meanwhile a developer that writes an application with the advanced animations and rendering will want more freedom in customizing the graphics and therefore will not mind the more complex code.

The differences in difficulty is something that became obvious directly when implementing the test code. As the canvas API is not made for the more advanced rendering it has simple functions to draw the basic shapes. The functions such as drawRect() and drawOval() are in the API to enable the simple drawing of rectangles and ovals. These functions made the implementation of the canvas API very easy. Where a canvas is created, the shape is drawn, the graphics buffer is invalidated to provoke the redrawing that gives the next shape and so on.

When implementing the OpenGL test code it was obvious that the whole rendering process would be much more advanced, but at the same time that the possibilities of rendering more advanced graphics was there. OpenGL builds its graphics on triples of vertices that are interpreted as triangles so in the testing example when drawing a square, the square is written as 4 vertexes connected together by 5 edges.[9] The ovals were drawn by first calculating the coordinates of the edge of the oval using polar coordinates, and then rendered using triangles originating from a vertex in the center of the circle to the calculated coordinates.

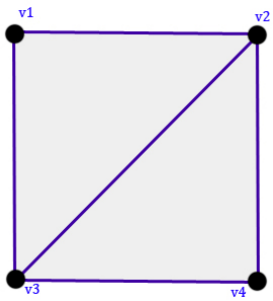

Figure 3. A square rendered using 4 vertexes and 5 edges

Test results

The results from the benchmarks taken while running the tests on the device gave the following results:

	Canvas with software rendering	Canvas with hardware accelerated rendering	OpenGL
Average FPS	19	15	90
Maximum FPS	22	34	96
Minimum FPS	19	7	62
Memory usage(bytes)	3,991,872	4,407,840	4,155,632

As predicted in the discussion OpenGL as the API made for animations shows to be by far the fastest of the three in rendering graphics. Between the hardware accelerated rendering and the software rendering the difference is not as significant. The software rendering is a bit faster since as mentioned earlier the instructions for rendering in the test are very simple which benefits the software rendering.

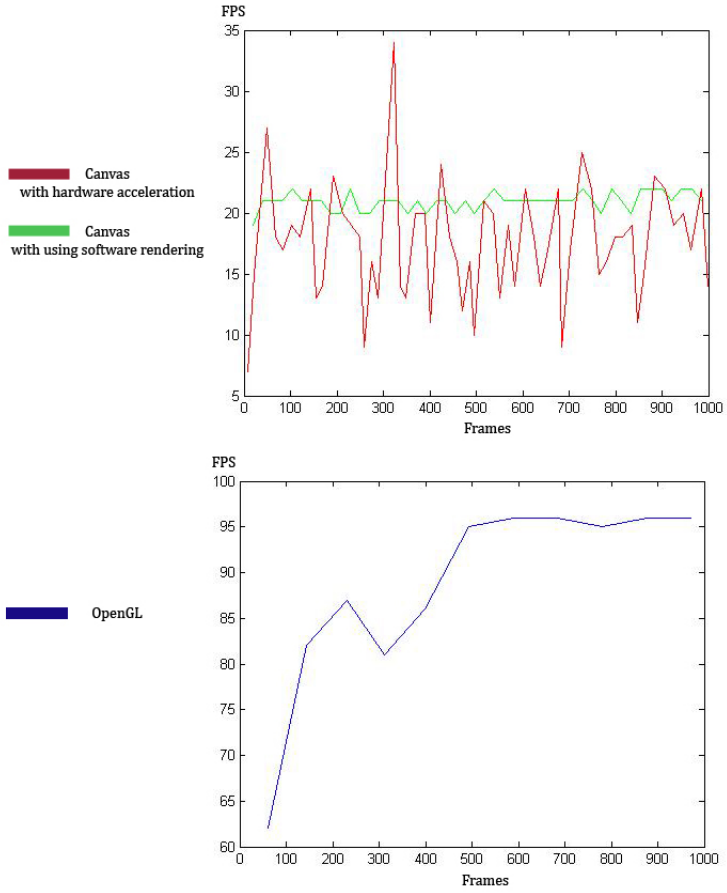

Figure 4. Graphs over FPS rendering performance

The really interesting fact about the FPS output from the different APIs is how stable the different APIs are. With the software accelerated rendering only having a fluctuation of 3 FPS both the Canvas with hardware acceleration and the OpenGL vary much more in stability.

In the graphs we can also see that the minimum FPS rendered is in the beginning of the test in the OpenGL and Canvas using hardware rendering cases. The reason for this is probably because of the GPUs drawing cache which in the beginning loads the instructions needed for future rendering.

The big difference between the hardware accelerated Canvas and OpenGL is that OpenGL has clear cache instructions and therefore keeps the rendering speed after the initial cache loading while the Canvas tries to make more improvements to the rendering and by doing this has to rewrite the GPU cache which creates the frame drops that can be seen in the graph above.

The software rendering also has its minimum in the beginning loading instructions to the CPU cache. After this the software rendering keeps a steady rate since

the rendering instructions are run without any severe changes in the process.

The memory allocation of the different APIs showed exactly as expected that the hardware accelerated Canvas needed to allocate the most memory of the three during the test. OpenGL needed a bit less memory because of not trying to make unnecessary improvements for the rendering. Also as expected the software rendered Canvas needed the least memory since the instructions are run directly on the CPU and not needing to allocate any memory for the GPU improvements.

Conclusions

The problem statement is as follows:

- What recent changes have been made in Android to make the use of the limited resources more efficient?
- What are the differences between the different rendering methods used in Android and their resource usage?
- Evaluate the performance and resource usage of the rendering methods in Android.

The theoretical discussion combined with the test results lead to the conclusions in this chapter.

Improvements being done to the graphics rendering

The main improvement within the graphics rendering of Android devices lately has been the support for hardware acceleration. The hardware acceleration allows the code written in the higher level to be processed and improved at the lower level before actually being executed by the CPU. For advanced rendering instructions this improves the rendering performance. However since the tests run where using such simple instructions for rendering the hardware acceleration proved to be unnecessary and actually decreasing the performance, which proved the doubts that Google had when introducing the option. But in general it is an improvement and since there are downsides as well Google has allowed the developers having more knowledge the choice of enabling or disabling the hardware acceleration for separate levels of the program.

In general many of the improvements being done are about getting more processing to run at a lower level which can be seen by the constant development of new lower level APIs that are being released to the Android system. However with the hardware development and the focus on graphics increasing there will always be new ways of improving the rendering performance.

Android rendering methods

Even though the Android rendering methods overlap each other in their functionality, the purposes for which they were created differ a lot. There is not a rendering method that is optimal in all situations. The choise of rendering method depends on what needs to be accomplished.

The main difference that affects the performance of the application is what parts of the graphic instructions are being run at what level. With the Canvas API, there is the option to enable hardware acceleration. When developing standard GUI applications with no additional graphical effects, the application will probably not benefit from enabling hardware acceleration. However, when the application being developed contains custom graphics and the graphical computations are more advanced, hardware acceleration will increase the performance.

The other rendering options (OpenGL, Renderscript) are not meant to compete with the Canvas API, but rather be extensions where the Canvas API simply can not do what you want. This applies to both performance and functionality. The main difference performance-wise is that with OpenGL, much of the computation is being done at native level. When the application contains advanced graphics with resource demanding computation that simply takes too much time to run in the Android VM, OpenGL will be a better choice since the application performance will benefit from running these computations at native level.

Abbreviations

CPU	Central Processing Unit
GPU	Graphics Processing Unit
RAM	Random Access Memory
SDK	Software Development Kit
NDK	Native Development Kit
API	Application Programming Interface
FPS	Frames Per Second

Test data

Canvas with software rendering

Benchmarked FPS at given frame number

FPS	19	21	21	21	22	21	21	21	20	20	22	20
Frame	19	40	61	82	104	125	146	167	187	207	229	249

FPS	20	21	21	21	20	21	20	21	21	20	21	20
Frame	269	290	311	332	352	373	393	414	435	455	476	496

FPS	21	22	21	21	21	21	21	21	21	21	22	21
Frame	517	539	560	581	602	623	644	665	686	707	729	750

FPS	20	22	21	20	22	22	22	21	22	22	21
Frame	770	792	813	833	855	877	899	920	942	964	985

Average FPS calculated by number of frames rendered / runtime
19 FPS
Total amount of memory allocated by the application
3,991,872 bytes

Canvas with hardware accelerated rendering

Benchmarked FPS at given frame number

FPS	7	15	27	18	17	19	18	22	13	14	23	20
Frame	7	22	49	67	84	103	121	143	156	170	193	213

FPS	19	18	9	16	13	34	14	13	20	20	11	24
Frame	232	250	259	275	288	322	336	349	369	389	400	424

FPS	18	16	12	16	10	21	20	13	19	14	22	18
Frame	442	458	470	486	496	517	537	550	569	583	605	623

FPS	14	17	22	9	17	25	22	15	16	18	18	19
Frame	637	654	676	685	702	727	749	764	780	798	816	835

TEST DATA

FPS	11	15	23	22	19	20	17	22	14
Frame	846	861	884	906	925	945	962	984	998

Average FPS calculated by number of frames rendered / runtime
15 FPS
Total amount of memory allocated by the application
4,407,840 bytes

OpenGL

Benchmarked FPS at given frame number

FPS	62	82	87	81	86	95	96	96	95	96	96
Frame	62	144	231	312	398	493	589	685	780	876	972

Average FPS calculated by number of frames rendered / runtime
90 FPS
Total amount of memory allocated by the application
4,155,632 bytes

Bibliography

[1] blog post Google Engineer, Chet Haase. Android rendering options. http://graphics-geek.blogspot.se/2011/06/android-rendering-options.html.

[2] blog post Google Engineer, R. Jason Sams. Introducing renderscript. http://android-developers.blogspot.se/2011/02/introducing-renderscript.html.

[3] Google+ post Google Engineer, Dianne Hackborn. How about some android graphics true facts? https://plus.google.com/105051985738280261832/posts/2FXDCz8x93s.

[4] Google Inc. Graphics on android. http://developer.android.com/guide/topics/graphics/index.html.

[5] Google Inc. Opengl on android. http://developer.android.com/guide/topics/graphics/opengl.html.

[6] Google Inc. Renderscript. http://developer.android.com/guide/topics/renderscript/index.html.

[7] Texas Instruments. Omap 4 mobile applications platform. http://www.ti.com/lit/ml/swpt034b/swpt034b.pdf, 2011.

[8] Feb 2012" "Mark Brownlow. Smartphone statistics and market share. http://www.email-marketing-reports.com/wireless-mobile/smartphone-statistics.htm#smartphones.

[9] Consultant at Jayway Per-Erik Bergman. Opengl es tutorial part ii - building a polygon. http://blog.jayway.com/2009/12/04/opengl-es-tutorial-for-android-

[10] Wikipedia: Computing platform. simple definition of a platform. http://en.wikipedia.org/wiki/Computing_platform.

[11] Adreno Graphics processing Units. Snapdragon processors and adreno gpu integration within them. https://developer.qualcomm.com/discover/chipsets-and-modems/adreno, 2012.

[12] Wikipedia. Opengl in general. http://sv.wikipedia.org/wiki/OpenGL.