Linköping University | Department of Computer and Information Science
Bachelor thesis, 16 ECTS | Computer Science
2017 | LIU-IDA/LITH-EX-G–17/059–SE

The Vuforia SDK and Unity3D Game Engine

– Evaluating Performance on Android Devices

Prestandautvärdering av Vuforias utvecklingskit för Unity3D på android-enheter

Ivar Grahn

Supervisor : Erik Berglund
Examiner : Anders Fröberg

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Abstract

This paper evaluates major features of PTC's augmented reality SDK Vuforia, available for Android, iOS and the Unity3D game engine. The performance of these features are evaluated in terms of frame rate and power consumption and the testing prototypes are developed using Unity3D. Augmented reality is a rapidly growing medium and the Vuforia SDK is very popular with over 325 000 registered developers and thousands of published applications. Despite being used to such extents, there are surprisingly few works evaluating different aspects of its performance. This paper provides an introduction to augmented reality and describes the technology used by the Vuforia SDK to deliver said features. This paper shows that Vuforia is capable of maintaining sufficient performance with interactive frame rates over 20 Hz in most cases. The power consumption of these features reduces the battery lifetime to acceptable levels, suitable for hand-held devices. In some cases, however, the performance in terms of frame rate reaches levels lower than recommended. These cases should be considered by developers looking to use Vuforia.

Acknowledgments

I would like to thank my supervisors at Cambio Healthcare Systems for their support and curiosity which has taught me to approach augmented reality from angles I did not expect. Their ideas and expectations of AR was a source of much inspiration. I would also like to thank my supervisor Erik Berglund at Linköping University and my thesis examiner Anders Fröberg. Despite their busy schedule, they have been tremendously forthcoming in helping me complete this thesis. Lastly I would like to express my gratitude to the opponent of this thesis, Christoffer Johansson, who during our coffee breaks would give me valuable feedback on ideas and thoughts for this document.

Contents

List of Figures

List of Tables

1 Introduction

Augmented Reality (AR) is currently undergoing a boom of active research [9] due to the recent advances of the mobile platform with devices such as smartphones and tablets. The majority of today's mobile devices come equipped with a camera, gyroscope, compass and accelerometer. This kind of hardware makes it possible to render virtual objects on top of the image provided by the device's camera in real time, and blend these virtual objects with the physical world in a realistic manner. This has formed the foundation of the development of AR applications on mobile platforms. According to Craig [9, p. xv] this medium shows great potential in the way information is conveyed and he states: "It is a medium that allows you to interact with digital data in a visual and spatial manner that is utterly seamless with your environment and everyday life."

This suggests that AR could potentially change the way we provide information and that many areas such as medicine, education, manufacturing, assembly and healthcare can benefit from the technology.

1.1 Motivation

Vuforia is currently the most established AR software development kit (SDK) on the market with over 325 000 registered developers and more than 35 000 apps currently utilizing their technology [25]. About 80% of these apps were developed using Vuforia's Unity plugin [19]. Vuforia was developed by Qualcomm and later bought by PTC in late 2015. Despite being such a widespread tool, however, there is not much documentation about performance on the Vuforia/Unity platform and how it performs on different devices such as Android or iOS. As the AR medium grows, and the use of mobile AR applications increases it is important to evaluate the performance of tools such as Vuforia to provide insight of how the software affects factors such as resource consumption and frame rate which have a significant impact on the user experience [10].

1.2 Aim

Developers looking to create AR applications must consider what kind of AR experience they will provide and by analyzing the strengths and weaknesses of different tools and solutions

they might come to a conclusion as to how they will proceed in development. In order to do that in the pre-development phase, however, there needs to be published works to analyze. This will be the purpose of this paper, to evaluate the performance of Vuforia's main features when running on Android smartphones, exported from Unity. Hopefully, the results and conclusions of this paper will aid future developers in selecting the tool most suited for their projects.

1.3 Research questions

- What is the performance of Vuforia SDK's image-, object- and text recognition as well as its video playback feature on Android devices when exported from the Unity3D game engine, in terms of frame rate and battery consumption?

1.4 Delimitations

Testing and evaluation of Vuforia will be conducted solely on apps ported from Unity3D to the Android platform. While Unity3D offers the possibility to export apps to most major platforms, the scope will be limited to Android due to time constraints. Also, Vuforia offers an Android native SDK which does not rely on Unity. However, as the Android native SDK is not the most used development platform for Vuforia, it will not be evaluated in this paper.

Testing will focus on Vuforia's major tracking features: image-, object- and text recognition as well as the video playback feature. Additional Vuforia features such as the *VuMark* 2D barcodes, *Virtual Buttons* and the *Smart Terrain* system will not be evaluated due to time limitations.

A moderate amount of study material will be fetched from Vuforia's documentation and developer forums as there is not much scientific material regarding Vuforia's inner workings available.

2 Theory

To understand how AR is used, a short description of its definition will be provided. A presentation of the Vuforia SDK will follow, with a description of the features to be evaluated in this paper. There will also be a short description of the Unity3D engine, with an emphasis on its multi-platform capabilities, especially how it exports applications written in C# to devices where C# is a non-native language.

2.1 Definition of Augmented Reality

In his paper, Azuma [5, p. 355] provides the following definition of AR : "Augmented reality (AR) is a variation of *virtual environments* (VE), or virtual reality as it is more commonly called. VE technologies completely immerse a user inside a synthetic environment. While immersed, the user cannot see the real world around him. In contrast, AR allows the user to see the real world, with virtual objects superimposed upon or composited with the real world."

Fast forward 20 years and the definition holds true still, what has changed is the conceived accessibility of AR. As the computational power of our mobile devices have increased they are now capable of rendering virtual environments on their own which has made accessing an augmented reality as easy as picking up your phone and starting the right application. Considering this and the rapidly increasing number of smart mobile devices, AR is available to more people than ever before.

2.2 Vuforia

Vuforia is the most popular SDK for developing AR-applications on a wide selection of devices. This chapter will explain the main features of Vuforia's Unity version, that will be evaluated. These include tracking, image recognition, object recognition, text recognition, video playback.

2.2.1 Tracking

It is not stated explicitly by Vuforia how their tracking algorithms function, but looking at their documentation [16] it is evident that they utilize some variation of natural feature track-

ing. Natural features in images are usually sharp details such as high contrast corners or the tips of a triangle (see figure 2.1).

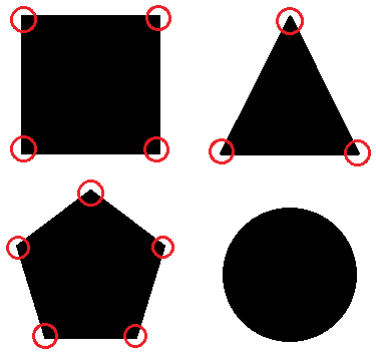


Figure 2.1: *Four shapes demonstrating what is registered as a feature by Vuforia. The features are marked with red circles. Take note of the circle, containing no features as there are no corners, sharp or pointy details.*

The technique was introduced in the early 1980s and revolves around splitting an image into several sections. Each section is shifted in several directions and the change in image intensity (RGB value) is measured. For instance, if a section of an image contains a straight line – or edge, as it is more commonly called – a shift along the edge will result in small overall changes in image intensity. However, a shift perpendicular to the edge will cause large changes. A corner, or spot will cause large changes when shifted in any direction. By considering these changes in image intensity, the feature tracking algorithm can determine what is a feature and map the positions of these features in the image. [11]

As a result, the algorithm creates a pattern of features. In Vuforia, this pattern of features is what define a target. When the target is detected by the application, the features are matched against pre-defined feature patterns and the pose of that target is estimated [26]. *Pose* is a common term used in the area of computer vision. It is the combination of position and orientation, often in relation to a 2D coordinate system. In Vuforia's case, the coordinate system is made up of the pattern created by the natural feature tracking.

2.2.2 Targets

Throughout this paper there will frequent mentions of different "targets", it is therefore important to have an understanding of what a target actually is. A target is a pattern of natural features pre-defined by the developer or user. In other words, if the Vuforia tracker is to find a certain image, or object, the software needs to know what patterns to look for. For this purpose, developers can use Vuforia's Target Manager. Vuforia offer developers a web service to create targets from 2D images, cuboids, cylinders and 3D objects (see figure 2.2).

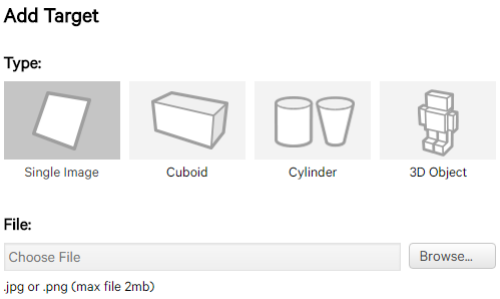


Figure 2.2: *The "Add Target" menu in Vuforia's Target Manager.*

2.2.3 Extended Tracking

Extended tracking is a feature that allows Vuforia to track targets using natural features surrounding it. With these natural features Vuforia is able build a map of features surrounding the target and maintaining detection even though it is not within the camera view, as demonstrated in Figure 2.3.

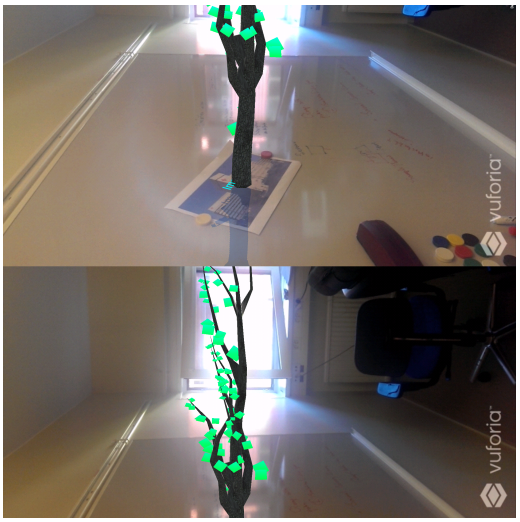


Figure 2.3: *With extended tracking enabled, Vuforia may track targets even when the associated target is out of view. In this image the designated target is the printed image fastened to the whiteboard.*

Creating and updating the map can be very computationally demanding.

2.2.4 Image Recognition

Using the Target Manager one can define an image target using any image of choice. Whether the image makes a good target is another question. The composition and amount of features are the deciding factors in how easily Vuforia can detect an image target. For instance, the target shown in Figure 2.4 demonstrates what Vuforia considers a "good" target.

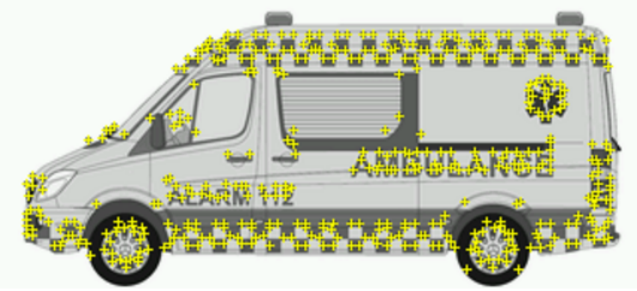


Figure 2.4: *An image target of an ambulance of with a high grade of detectability, indicating that it is easily trackable. Yellow crosses represent detected features.*

More specifically, good targets contain a higher number of features arranged as uniquely as possible. A picture of a dense 2D grid might contain many features, but as the pattern is very repetitive, every section of that image will be indistinguishable by Vuforia. This makes it impossible for the software to determine what is up, down, left or right thus making such images bad targets. [16]

Images of round shapes without sharp edges would also make bad targets, simply because there are no features for Vuforia to detect as shown in Figure 2.5.

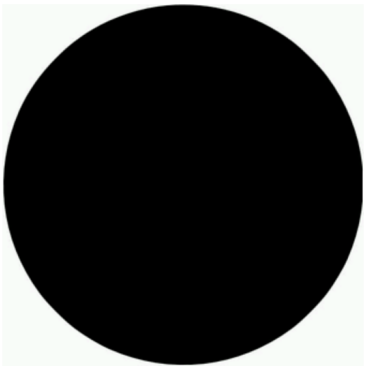


Figure 2.5: *Images of circular shapes makes for bad targets. Observe the complete lack of features (no yellow crosses).*

The image of the ambulance was deemed highly detectable by Vuforia, and as such should be used as a target. The opposite can be said about the image of the black circle. A simple augmentation test shown in Figure 2.6 demonstrate the image detection of these two targets.

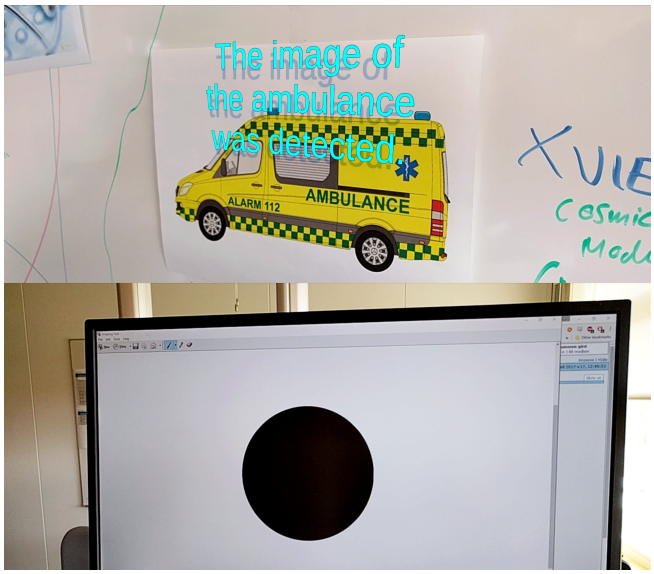


Figure 2.6: *Both targets are augmented with text informing the user if the image was detected. An augmentation will only appear if the target is detected. The ambulance target is detected immediately due to its high grade of detectability whereas the black circle is not detected at all.*

2.2.5 Object Recognition

While Vuforia's object recognition utilizes natural feature tracking at its core, the process of creating object targets is quite different compared to creating image targets. An image target maps features in 2D by analyzing the pixels of the image file provided to the target manager. A 3D object, however, has a third axis to consider. Object targets enable users to view an augmented object from several angles while the object maintains its augmentation. Prior to augmenting 3D objects, they need to be scanned somehow; Vuforia has developed a scanner app that is capable of scanning relatively small objects using a certain "scanning plane" upon which the object is placed (see Figure 2.7). This way, the scanner app can tell the orientation of every detected feature in 3D, as the scanner plane works as a spatial reference. In reality the "scanning plane" or Object Recognition Target is an image target that, when detected, provides the scanner with the XYZ-coordinates of the detected features on the 3D object. The Vuforia object scanner is shown in Figure 2.7.

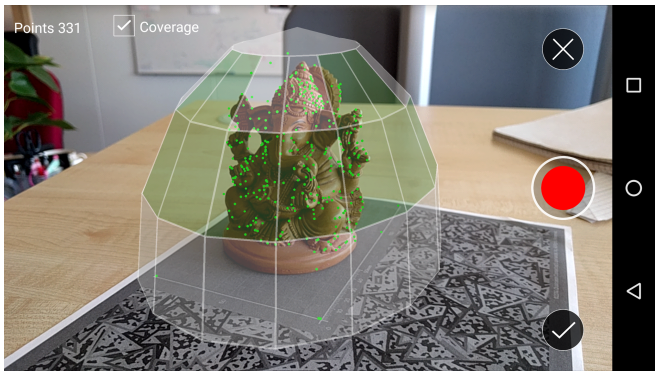


Figure 2.7: *Placing the to-be-scanned object upon the scanner image target, Vuforia's scanner can determine the XYZ-coordinate of a detected feature, represented by a green dot. The number of detected features can be viewed in the top left corner.*

When the object has been scanned it can be added to the target database and used for augmentation (see Figure 2.8). Vuforia allows for 20 object targets in the one device database [18].

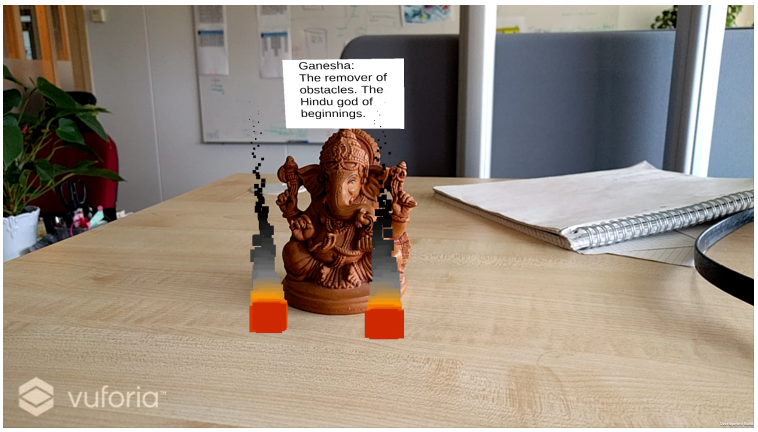


Figure 2.8: *A simple augmentation of the previously scanned object, now tracked without the aid of any additional image targets.*

Though Vuforia has not revealed exactly how they track 3D objects, looking at the provided scanner app and their focus on feature tracking, there is strong evidence of the usage of Interest Point Detection. Interest point detection works by having a set of template images, capturing the object appearance and pre-defined feature patterns. As features are detected by the tracker, they are compared to the features – interest points – of the template images and the orientation can be calculated. [14]

As seen in Figure 2.7, it would seem that the dome shaped grid with green and gray rectangles represents the coverage of template images and the features captured within each template image.

2.2.6 Text Recognition

Vuforia's text recognition detects 100 000 english words in the most common fonts by default. This number can be extended with the use of custom word lists, with support for 10 000 additional words. The text recognition relies on the UTF-8 character encoding standard and may only detect Aa-Zz characters. [23]

The text recognition component of Vuforia is made possible by something called the "text recognition engine" [23] and there are no available specifications of this "engine". What is known, however, is that text recognition on mobile devices is enabled by something called *Optical Character Recognition*, thus a short description of this will be provided.

Optical Character Recognition, OCR for short, is the process of detecting characters in images. There are many algorithms available for achieving this, but on the Android platform they all adhere to some necessary steps [8].

First, an image is provided as input. The image then undergoes some form of segmentation where the text is being differentiated from other graphical elements in the image. As the image was provided by a camera, there is bound to be noise in the picture. Pre-processing may reduce the noise to improve recognition. When the image has been processed, features associated with text are extracted. On Android devices this feature extraction is often done by Google's open source OCR SDK called Tesseract. With the features extracted the characters can be recognized by some algorithm, where these features are compared to known characters. [8]

It is not certain Vuforia uses Tesseract for its feature extraction on Android devices. What is certain though is that features detected in the image are compared to those of the predefined words in the standard and custom wordlists [23]. If the features match, the word is detected and identified as demonstrated in Figure 2.9.

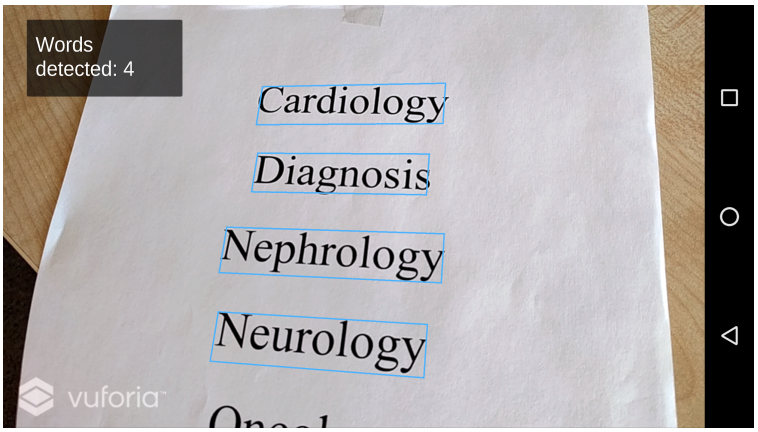


Figure 2.9: *Vuforia's Text Recognition engine can detect and track text as it would a regular image target. These words may then be augmented as the developer sees fit.*

When a word has been detected it is possible to augment this word to ones wishing. To demonstrate text augmentation, a simple translator app was developed and is shown in Figure 2.10.

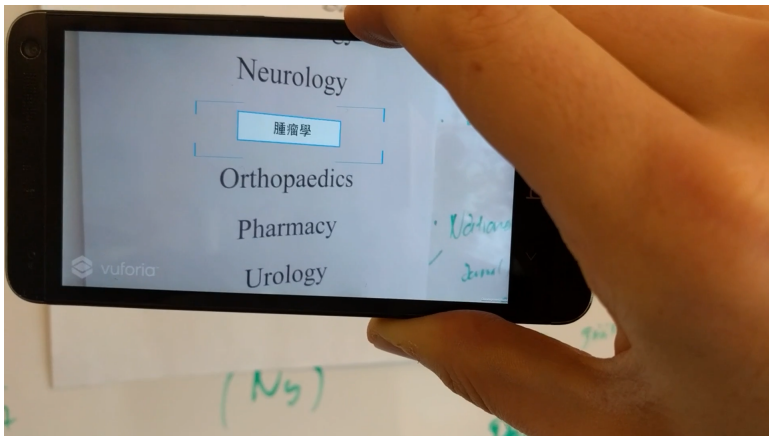


Figure 2.10: *The translator app detects english words and emplaces a chinese translation on top of the detected word for intuitive reading.*

In Figure 2.10 one can also see what is called a loupe region, represented by the blue rectangle. The loupe region defines the size of the screen area in which the tracker tracks text. The height and width of the loupe region is configured in percent, relative to the device's screen size. A loupe region with both height and width set to 50 would then be bigger on a device with a screen size of 2560x1440 pixels compared to a device with a screen size of 1920x1080 pixels. Something to be considered when testing and developing for multiple devices.

2.2.7 Video Playback

One useful feature of Vuforia is the ability to augment targets with a playing video. The video can be played in fullscreen or it can be rendered onto a texture. Rendering videos onto textures is supported by Android 4.0+ only (on Android devices). It is demonstrated in Figure 2.11.

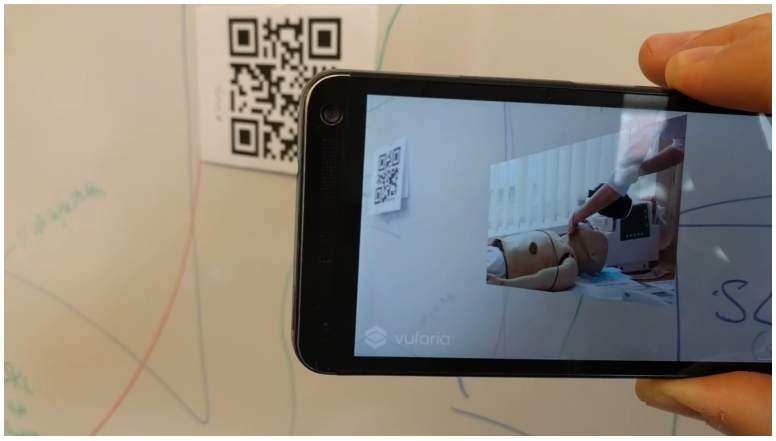


Figure 2.11: *A video rendered onto a 3D plane placed on top of the target.*

2.3 Unity

Unity is a complete 3D engine supporting C# scripting, 3D and 2D graphics, physics, sound and animation. It is one of the most popular engines used for mobile game applications and is progressing steadily in the AR/VR area [24]. The feature Unity is most acknowledged for is its rapid development and deployment capabilities. Developed applications are created once, and the process of porting it to different platforms is very simple thanks to Mono.

2.3.1 Mono

This paper will evaluate how AR applications written in C# performs on the Android platform, however, Android devices does not run C# natively, so it's important to understand how this process works in order analyze the performance properly.

It is all made possible with Mono. Mono is an ongoing, free and open-source project led by Xamarin with Microsoft as the holding company. It was created using C/C++ with the purpose of allowing developers to build cross-platform applications and is currently available to many different platforms[2].

So how does Android run C#? Android is commonly assumed to use the Java language, but it is also capable of running C/C++ using the Android NDK[4]. Mono, written in C/C++, may therefore run on Android systems, which enables compiling and running of C# code.

When an application is ported from Unity to an Android powered device, C# is compiled to CPU independent IL (Intermediate Language). This is because at this stage of the compilation, Mono does not yet know what platform the code will run on. It is then packaged with MonoVM (Mono Virtual Machine) and JIT (Just-in-time compiler). The result is an executable, packaged as an Android app that runs side-by-side with Java/ART (Android runtime) and utilizes native types via the Java Native Interface.

2.3.2 Unity Profiler

The Unity profiler is a tool for developers to monitor the time consumption of different components of the application (see Figure 2.12). This makes it possible to see how much of the time is spent on activities related to the CPU, GPU, rendering or the memory. There is also the option of "Deep Profiling" which records every function call made by the application. With this level of profiling it is possible to isolate the most time consuming functions and analyze their performance. The profiler may also connect to the mobile device to which the application was exported.

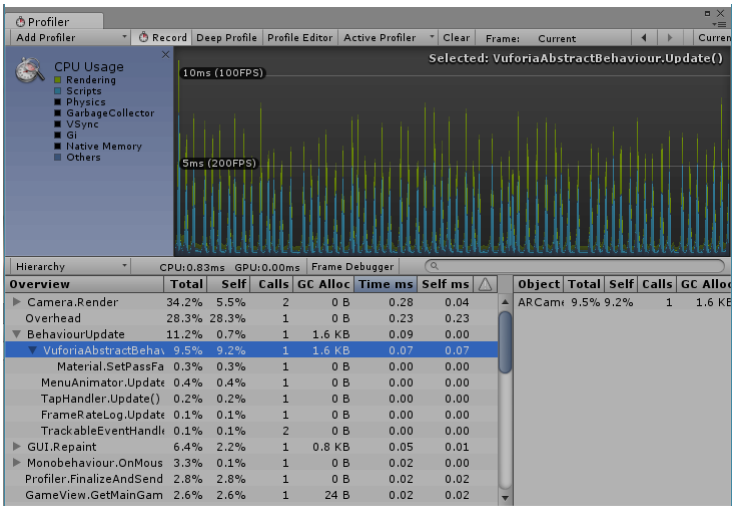


Figure 2.12: *Unity's profiler displays time consumption of various components used by the application. In this instance, we can see that VuforiaAbstractBehaviour makes up for 9.5% of the CPU time.*

2.4 Measurements

This chapter gives a brief overview of the measurements used in evaluating performance.

Frame Rate Frame rate is determined by the speed of which the device render frames onto the display. A frame rate of 30 Hz would imply 30 frames are being rendered every second. As stated in the introduction, the frame rate has a direct impact on the user experience in graphical applications. Barfield & Furness [6] explains also that frame rate measure virtual presence and frame rates too low breaks the illusion of the virtual elements "being there". This correlates with the research done by Airey et. al [3] who discovered that a frame rate of 6 Hz maintains the illusion while the interactivity (moving around the object, browsing from different angles etc. etc.) requires a frame rate above 20 Hz. As this research was conducted more than 20 years ago, it's reasonable to assume that today, from a user's perspective, the expected frame rates might be higher than 20 Hz but according to Li et. al [15], frame rates above 20 Hz are still considered fluent enough for user interaction, even in modern times. This coincides with Vuforia's tracker being capped at 30 Hz, as anything above 30 Hz might be considered redundant. The frame rate requirement of AR is therefore not as high as the frame rate requirement of Virtual Reality (VR), which is expected to reach rates above 100 Hz for the best experience.

While the frame rate is an important measure for the user experience, it also indicates how software taxes the hardware. If the algorithms and functions of a program are very demanding, the hardware will simply output fewer frames per time unit. By measuring the

frame rates of Vuforia features, one would hit two birds with one stone as it would reveal the computational performance while also giving insight about the user experience.

Power Consumption Power consumption is an increasingly important measurement of usability for todays's mobile devices. Critical even, as explained by Ra et. al [20]. The battery capacity of modern mobile devices is now considered their biggest bottleneck [22], so when evaluating the performance of Vuforia on mobile devices, limited by their batteries, this metric can not be neglected. For instance, Vuforia might offer a positive user experience with high frame rates, at the cost of a very high battery consumption. Minimizing battery consumption while also maintaining high framerates should therefore be a priority.

3 Method

This chapter aims to describe the methods used to answer the question:

"What is the performance of Vuforia SDK's image-, object- and text recognition as well as its video playback feature on Android devices when exported from the Unity3D game engine, in terms of frame rate, and battery consumption?"

3.1 Pre-study

The Vuforia SDK was studied in order to understand what features are available and what tools are offered to developers at the time of this study. When the knowledge of *what* features Vuforia offers had been established, the features were ranked according to their significance in terms of how frequently these features were used by developers. The process of determining the most frequently used features in Vuforia developed apps involved browsing appstores, developer forums and watching trailers of finished products online.

When the major features had been determined, the pre-study proceeded to the question of *how* these features were achieved. Answering how the features were achieved would hopefully give insight as to why the performance is the way it is. To understand the *how*, the official documentation and the official forums of Vuforia was studied.

Knowing what popular features Vuforia offers, and how they work, the pre-study approached the question of how to evaluate these features according to their frame rate and power consumption. This process revolved around looking at previous works and products to see which parameters should be altered during testing.

3.2 Test devices

The tests were performed on two different android devices: one HTC One from 2013 and one Huawei Nexus 6P from late 2016. Comparing the results of an old –and thoroughly used– device with a modern, brand new device may provide insight of Vuforia's performance on older versus newer devices.

Here, the specifications of the two devices used for testing are provided. The numbers are collected from [12] and [17].

HTC One M7(2013):

- **System on chip:** *Qualcomm Snapdragon 600*
- **CPU:** *1.7 GHz quad-core Krait 300*
- **GPU:** *57.6 GFLOPS Adreno 320*
- **Memory:** *2 GB LPDDR2 RAM*
- **Storage:** *32 GB*
- **Battery:** *2,300 mAh Li-Po, nominal 3.82 V*
- **Display:** *120 mm, 1920x1080 pixels, 16:9 aspect ratio*
- **Camera Resolution:** *4,0 MP*
- **OS:** *Android 5.0.2 "Lollipop" (HTC Sense)*

Huawei Nexus 6P (2016):

- **System on chip:** *Qualcomm Snapdragon 810 v2.1*
- **CPU:** *2.0 GHz octa-core ARMv8A*
- **GPU:** *420 GFLOPS Adreno 430*
- **Memory:** *3 GB LPDDR3 RAM*
- **Storage:** *64 GB*
- **Battery:** *3,450 mAh Li-Po, nominal 3.82 V*
- **Display:** *145 mm, 2560x1440 pixels, 16:9 aspect ratio*
- **Camera Resolution:** *12,3 MP*
- **OS:** *Android 7.1.2 "Nougat"*

3.3 Implementing prototypes and measuring performance of features

The features described in chapter 2 were analyzed individually according to their frame rate and power consumption in different scenarios. To test these features, a prototype was developed for the different features using Unity3D. The purpose of these prototypes was to test the performance of features provided by Vuforia. As such, the prototypes was primarily focusing on the different variations of tracking, and not so much of Unity's rendering features.

Although they were tested individually, the testing followed a general theme of certain parameters being increased progressively. Increasing these parameters would supposedly demand an increasing amount of CPU time. Image recognition and object recognition also included a test where Extended Tracking was enabled.

The frame rate was measured using the Unity profiler and a small C# script that calculated and stored frame rate values for later analysis. The power consumption was monitored with an application called PowerTutor. PowerTutor was developed by the University of Michigan in collaboration with Google and the National Science Foundation [1]. It measures the rate of energy transfer in Watts (W).

The performance of each parameter configuration was measured for 60 seconds. The average frame rate and battery consumption were extracted from the collected data and plotted against the parameter value. When running the tests, no other application on the device may

run. GPS, WiFi, Bluetooth and mobile data are all disabled. The screen brightness was set to its highest setting, and any battery saving mode was disabled. This configuration was used on both the HTC One, and Nexus 6P.

Application development in Unity involves creating empty *objects* and attaching appropriate C# scripts to said objects. When creating AR applications with Vuforia, a fundamental requirement is the camera object to which Vuforia's *AR_Camera* script is attached. This script contains the code that activates the device's camera and tracks natural features in the camera view. Additionally, the application also requires pre-defined target objects with the proper target type script attached to them. A more in-depth presentation of Vuforia's code and classes is provided in the earlier work of Simonetti Ibañez and Paredes Figueras [21].

3.3.1 Image recognition

The detection and tracking of image targets was one of the first features available when Vuforia released. Vuforia allows tracking of five simultaneous image targets, and this was the progressively increasing parameter when testing image recognition performance. The performance was measured with one up to five image targets being tracked simultaneously, first with extended tracking disabled and then with extended tracking enabled. Five different image targets were created using Unity's hierarchy functionality seen in Figure 3.1.

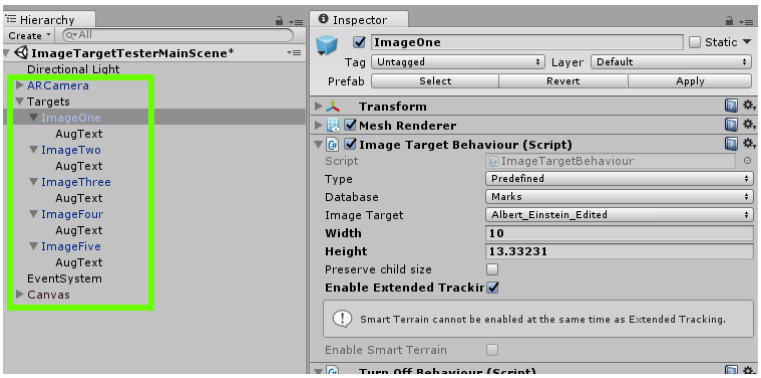


Figure 3.1: *The image recognition prototype as seen in the Unity hierarchy view.*

The images used for these five targets were highly detectable by Vuforia's target manager and testing was performed in bright, neutral lightning for optimal conditions.

3.3.2 Object recognition

Testing the performance of object recognition followed a similar procedure to the testing of image recognition. Vuforia may only track up to two object targets at one time. The performance was therefore measured from one up to two object targets being tracked simultaneously, with and without extended tracking enabled. The prototype structure can be seen in Figure 3.2.

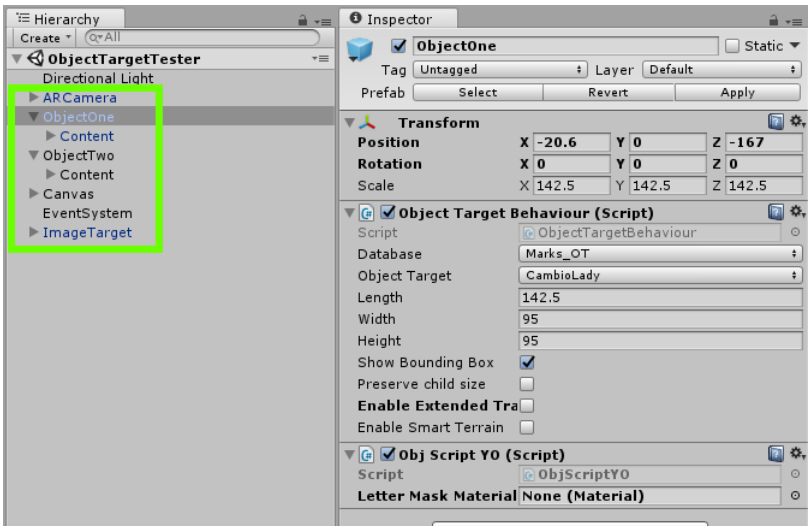


Figure 3.2: *The object recognition prototype as seen in the Unity hierarchy view.*

3.3.3 Text recognition

Measuring the performance of text recognition involved increasing the number of simultaneous words detected. Exactly how many words Vuforia can track simultaneously has not been explicitly stated, but Vuforia's documentation provide some guidelines when designing text recognition applications. The size of the loupe region should for example not be *too big*, as this might increase detection times and needlessly reduce performance. When testing text recognition, the loupe region was set to 50% of the screen height and 30% of the screen width. The prototype application was forced to operate only in landscape-orientation, so that testing was not interrupted by sudden screen rotations. This means that on the HTC One, the loupe region covered an area of 540x576 pixels which is equal to 15% of total screen size. The loupe region size of the Nexus 6P model would then be 720x768, also 15% of the total screen size. The text recognition prototype consisted of an AR Camera object but no specific target object, instead a *TextRecognition* object with the necessary scripts attached to it was included in the Unity hierarchy (see Figure 3.3).

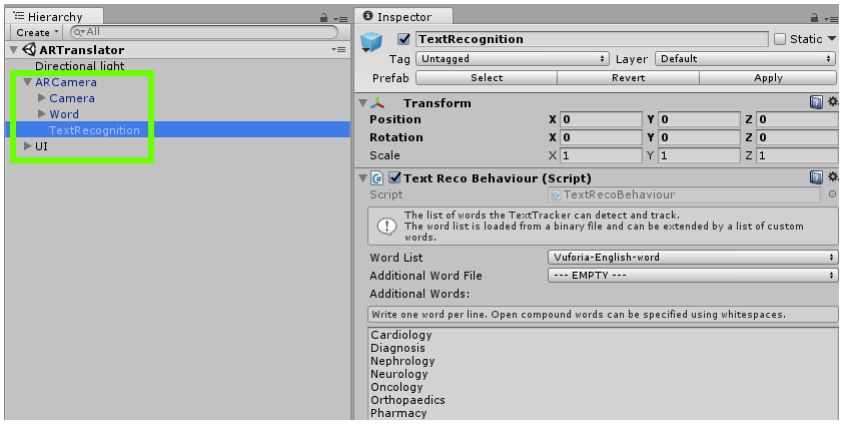


Figure 3.3: *The text recognition prototype as seen in the Unity hierarchy view.*

3.3.4 Video playback

The parameters to be adjusted when testing video playback were the number of simultaneous videos playing, while augmenting five different image targets. It is important to note that video playback is applied to regular image targets, and the performance of image recognition may correlate with the performance of video playback. Every video instance are of the same resolution of 1280x720 pixels. The prototype consisted of five image targets, each augmented with a *Video* object, pre-fabricated by Vuforia (see Figure 3.4). The videos attached to the Video objects were 15 seconds long.

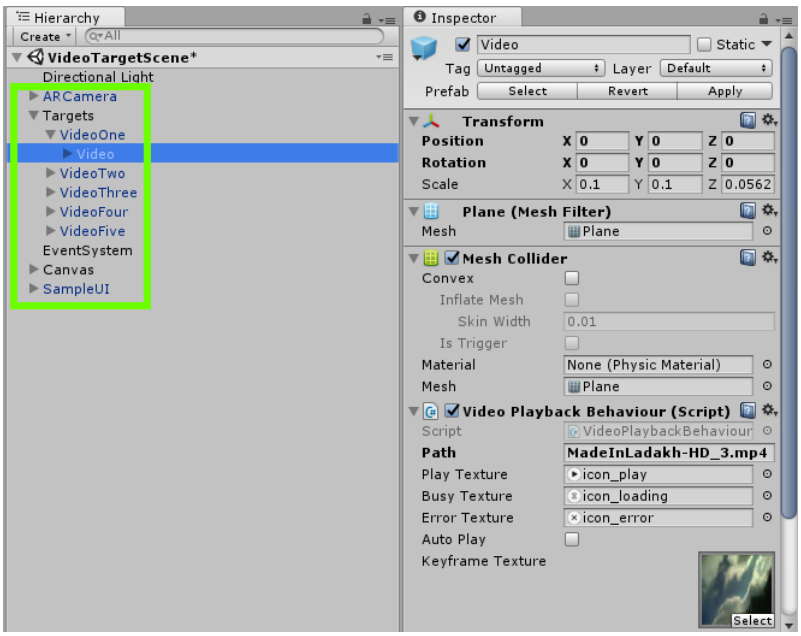


Figure 3.4: *The video playback prototype as seen in the Unity hierarchy view.*

4 Results

This chapter presents the results acquired from the method described in chapter 3.

4.1 Pre-study

The features of Vuforia The top features discovered during the pre-study were Vuforia's *image recognition*, *object recognition* and *text recognition* that enables developers to augment said recognized items. Additionally, Vuforia offers video playback as a form of augmentation that will be evaluated.

Other features not to be evaluated were Vuforia's *Smart Terrain*, *Virtual Buttons* and Vuforia's own 2D barcode *VuMark*. These features were lowly prioritized as their function simply extends image- and object recognition.

How they work The recognition features all utilize something called *natural feature tracking*, a technique that sees pointy details in images. Video playback utilize the devices' inherent video player to read video files and display the video data according to the orientation of the augmentation.

How they ought to be tested Vuforia developed applications mostly involve augmenting one or several targets with simpler augmentations, as the augmentations of these targets seldom competed with Vuforia's tracking computationally, it was decided that the testing would revolve around increasing the number of simultaneous targets being detected. As for this reason, it was decided that the augmentations indeed were to be kept to a none compute intensive level.

4.2 Measurements

In this section the measurement data is presented using charts and tables. For each feature, the frame rate will be presented first, followed by the power consumption. For reference, when the screen was turned on, the idle power consumption of the HTC One and the Nexus 6P model were 975 mW and 650 mW respectively.

4.2.1 Image Recognition

The frame rate of the image recognition prototype is presented in figure 4.1. A maximum of five targets were being tracked simultaneously during well lit conditions. All five image targets were of grade 5 detectability in the target manager.

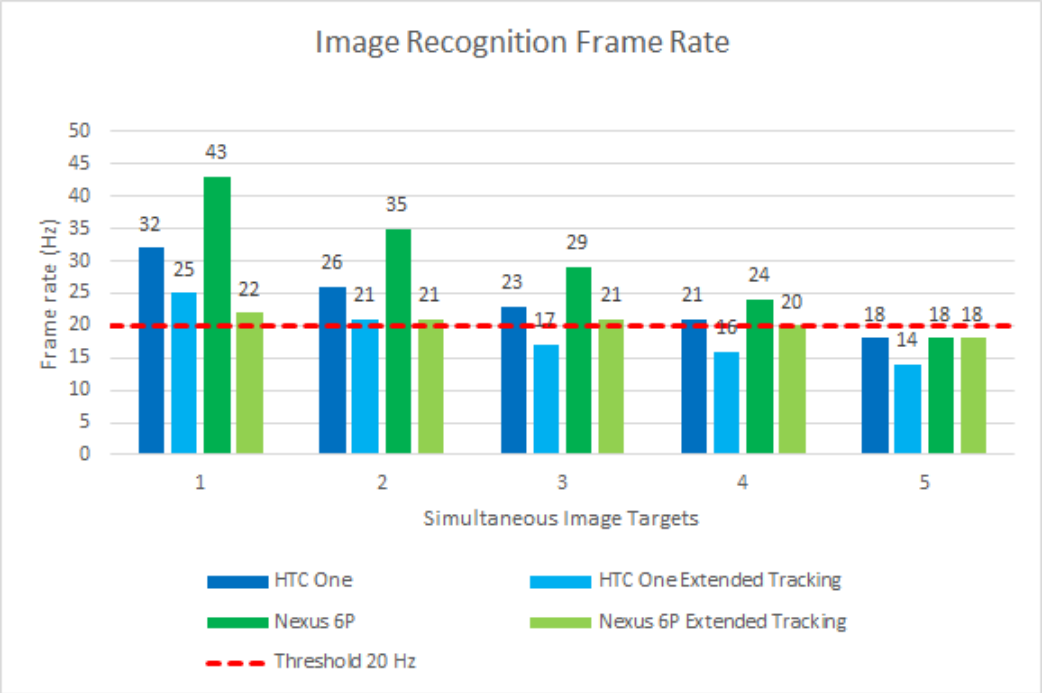


Figure 4.1: *Image recognition frame rate for an increasing number of simultaneous tracked images.*

For each added image target the frame rate decreased by a few Hz, for the most part the frame rate is kept above 20 Hz, but as 5 targets are being tracked simultaneously the frame rate goes below this value for both the HTC One and the Nexus 6P. In table 4.1 one can observe the average decrease in frame rate per added target.

Table 4.1: Image recognition frame rate chart overview

	HTC One M7	Huawei Nexus 6P
	Extended Tracking Disabled	
Initial frame rate	32 Hz	43 Hz
Final frame rate	18 Hz	18 Hz
Average decrease/image	-3.5 Hz	-6.25 Hz
	Extended tracking Enabled	
Initial frame rate	25 Hz	22 Hz
Final frame rate	14 Hz	18 Hz
Average decrease/image	-2.75 Hz	-1 Hz

The power consumption measured in mW (milliwatts) for image recognition can be seen in figure 4.2.

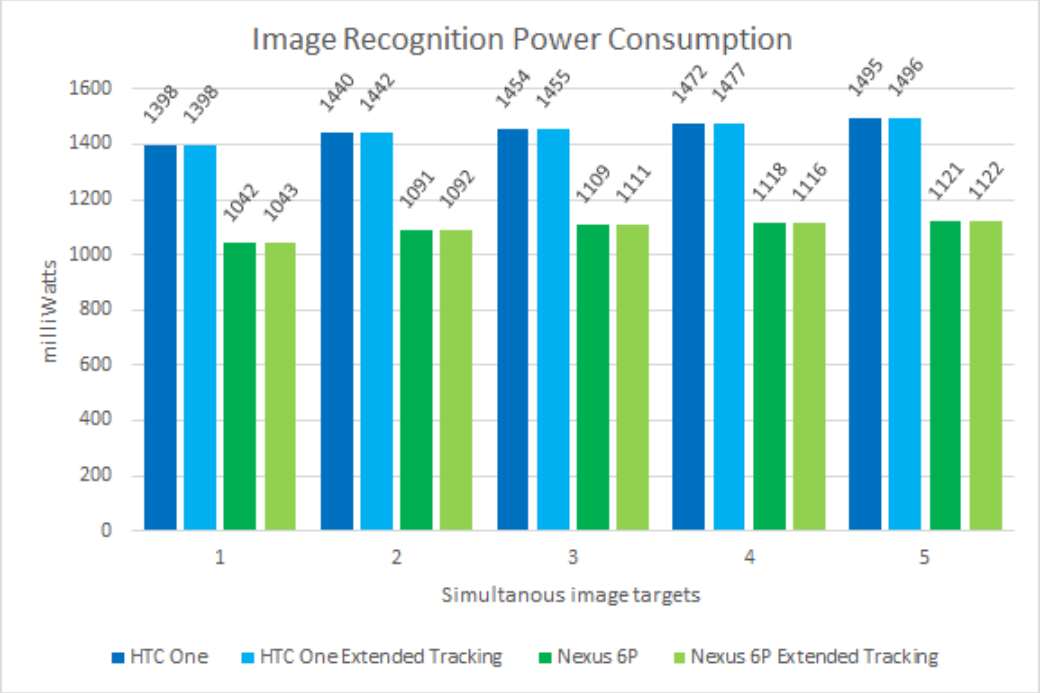


Figure 4.2: *Image recognition power consumption for an increasing number of simultaneously tracked images.*

For added image targets the power consumption increases slightly. Even though extended tracking seems to have a moderate effect on frame rate, power consumption seems less affected. The average increase in power consumption per added image is displayed in table 4.2.

Table 4.2: Image recognition power consumption chart overview

	HTC One M7	Huawei Nexus 6P
	Extended Tracking Disabled	
Initial power consumption	1398 mW	1042 mW
Final power consumption	1495 mW	1121 mW
Average increase/image	24.25 mW	19.75 mW
	Extended tracking Enabled	
Initial power consumption	1398 mW	1043 mW
Final power consumption	1496 mW	1122 mW
Average increase/image	24.25 mW	19.75 mW

4.2.2 Object Recognition

The different frame rates of the object recognition prototype is presented in figure 4.3. Only two objects can be detected simultaneously. The objects used were not much bigger than a fist. Both were moderately detailed.

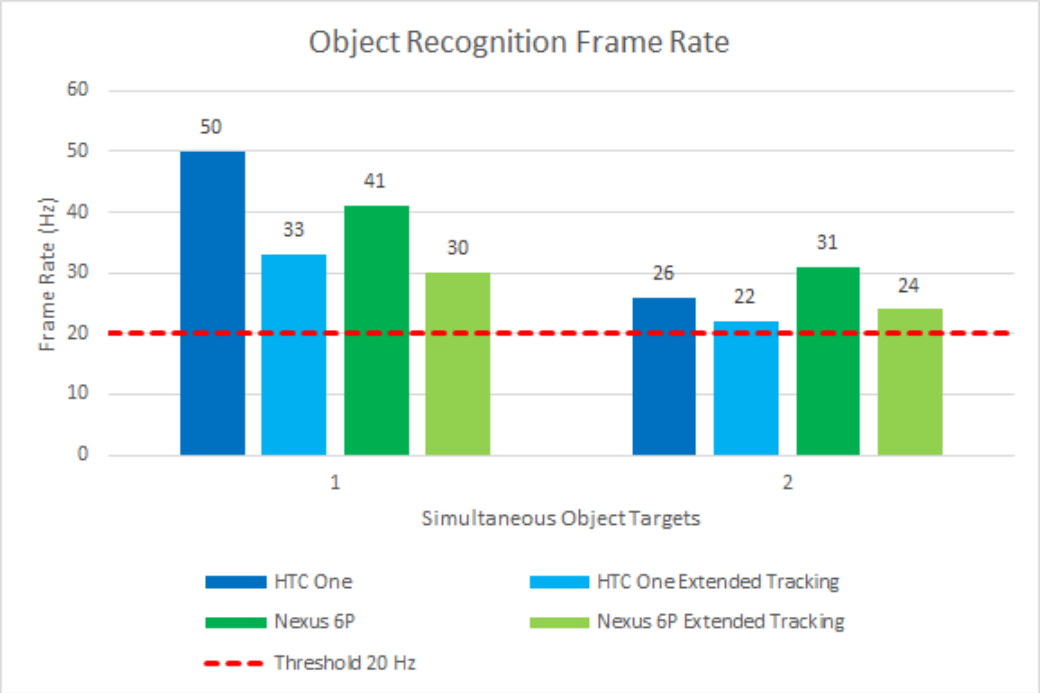


Figure 4.3: *Object recognition frame rate for an increasing number of simultaneously tracked objects.*

A decrease in frame rate going from one detected object target to two detected objects were observed (see table 4.3). However, the frame rate does not dip below 20 Hz for either the HTC One and Nexus 6P. Extended tracking lowers the frame rate further for both devices.

Table 4.3: Object recognition frame rate chart overview

	HTC One M7	Huawei Nexus 6P
	Extended Tracking Disabled	
Initial frame rate	50 Hz	41 Hz
Final frame rate	26 Hz	31 Hz
Average decrease/object	-24 Hz	-10 Hz
	Extended tracking Enabled	
Initial frame rate	33 Hz	30 Hz
Final frame rate	22 Hz	24 Hz
Average decrease/object	-11 Hz	-6 Hz

The power consumption measured in mW for the object recognition prototype can be seen in figure 4.4. For the HTC One, tracking of one object target kept the power consumption moderately higher compared to other types of targets. The power consumption of the Nexus 6P was comparable, although slightly higher.

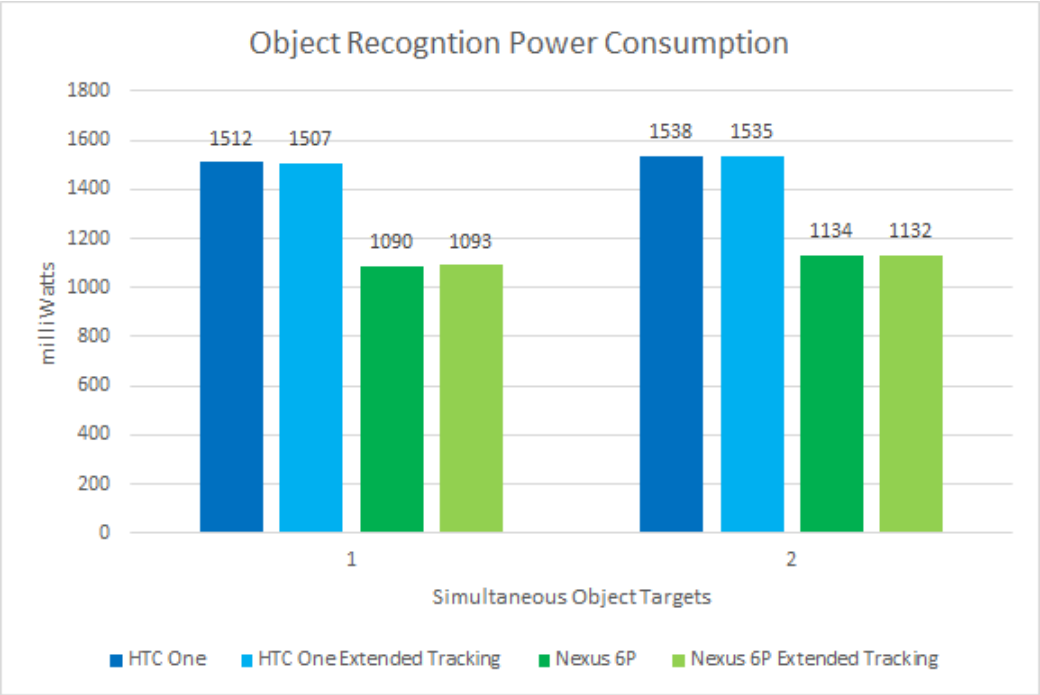


Figure 4.4: *Object recognition power consumption for an increasing number of simultaneously tracked objects.*

No major difference in power consumption could be observed despite the larger differences in frame rate. Table 4.4 displays the increase in power consumption going from one to two detected object targets.

Table 4.4: Object recognition power consumption chart overview

	HTC One M7	Huawei Nexus 6P
	Extended Tracking Disabled	
Initial power consumption	1512 mW	1090 mW
Final power consumption	1538 mW	1134 mW
Average increase/object	26 mW	44 mW
	Extended tracking Enabled	
Initial power consumption	1507 mW	1093 mW
Final power consumption	1535 mW	1132 mW
Average increase/object	28 mW	39 mW

4.2.3 Text Recognition

The frame rate of the text recognition prototype is presented in figure 4.5. One up to six words were tracked simultaneously within a loupe region covering 15% of the screen size. The word length varied from 4 characters up to 11 characters.

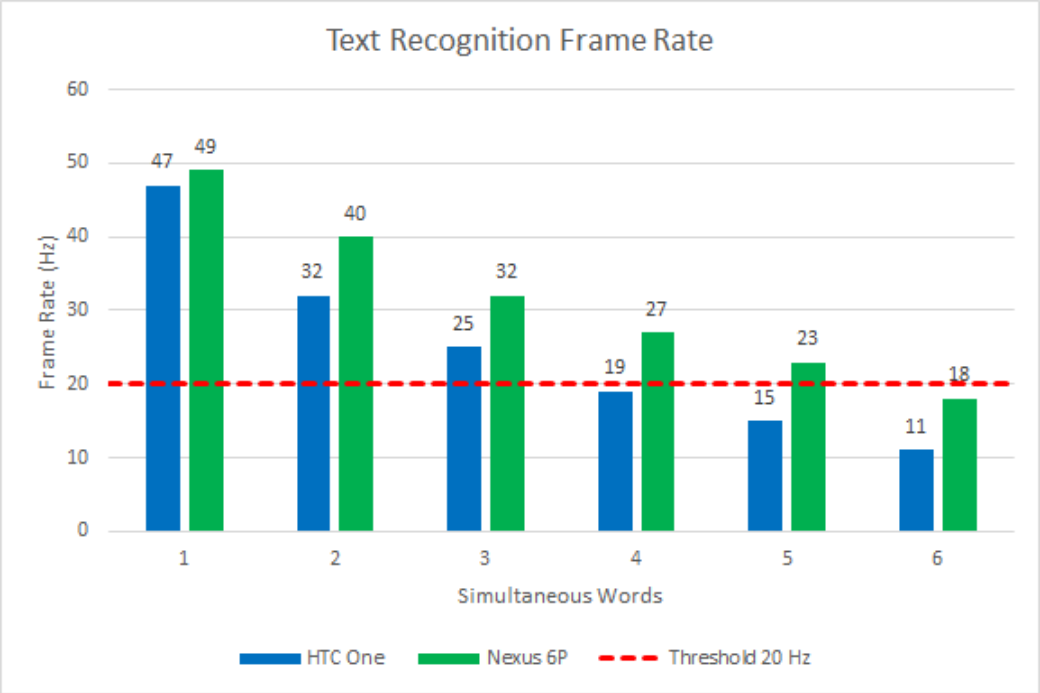


Figure 4.5: *Text recognition frame rate for an increasing number of simultaneously tracked words.*

Both the HTC One and Nexus 6P follow the same trend as more words are added (see table 4.5) with the exception of the Nexus 6P maintaining a slightly higher frame rate. At four words, however, the HTC One dips below 20 Hz while the Nexus 6P stays at 27 Hz. At six words the Nexus 6P goes below 20 Hz, reaching a frame rate of 18 Hz.

Table 4.5: Text recognition frame rate chart overview

	HTC One M7	Huawei Nexus 6P
Initial frame rate	47 Hz	49 Hz
Final frame rate	11 Hz	18 Hz
Average decrease/word	-7.2 Hz	-6.2 Hz

The power consumption measured in mW for the text recognition prototype can be seen in figure 4.6.

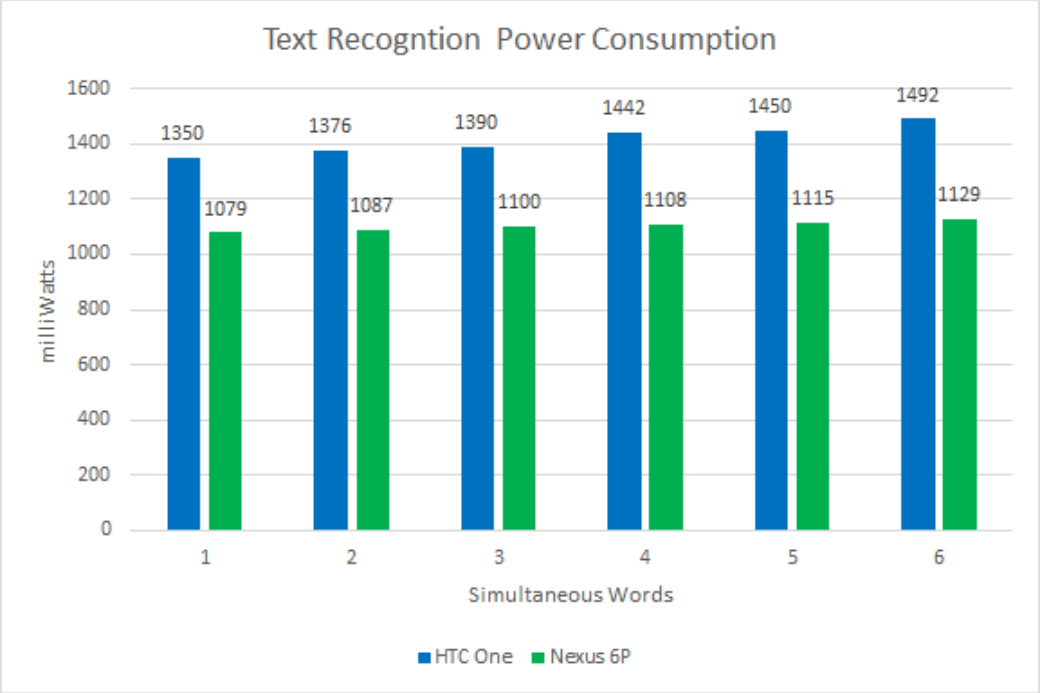


Figure 4.6: *Text recognition power consumption for an increasing number of simultaneously tracked words.*

The power consumption increase was fairly unchanged for the Nexus 6P, a larger increase could be observed for the HTC One, with an average increase per words almost three times as big compared to the Nexus 6P (see table 4.6).

Table 4.6: Text recognition power consumption chart overview

	HTC One M7	Huawei Nexus 6P
Initial power consumption	1350 mW	1079 mW
Final power consumption	1492 mW	1129 mW
Average increase/word	28.50 mW	10 mW

4.2.4 Video Playback

The frame rates of the video playback prototype are presented in figure 4.7. In this test, the older HTC One model managed to maintain similar frame rates to the newer Nexus 6P model.

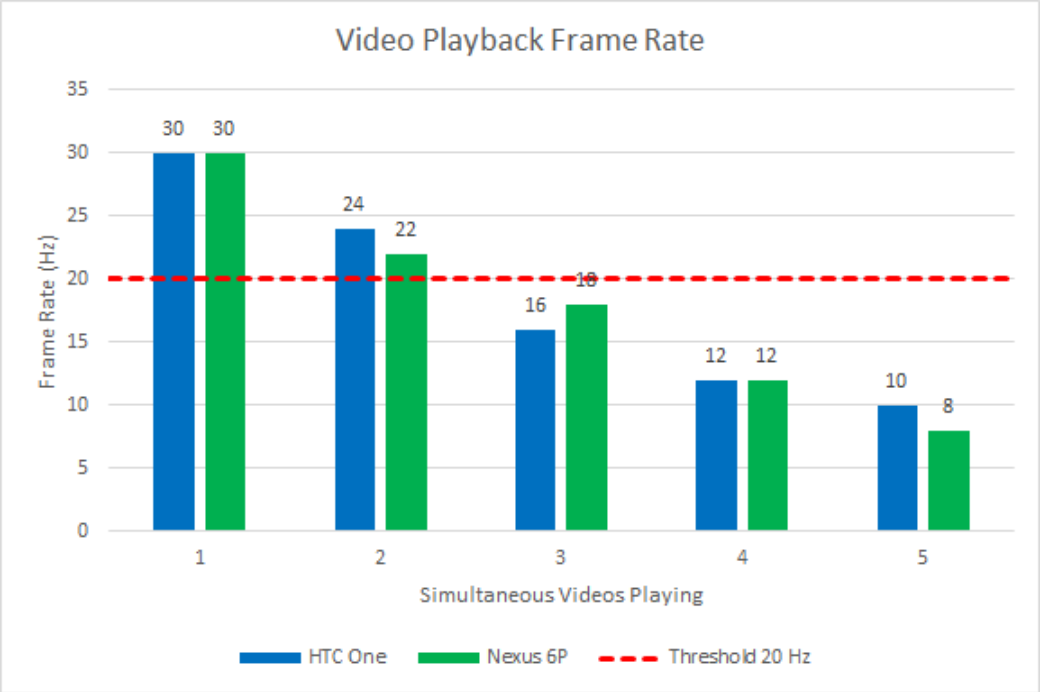


Figure 4.7: *Video Playback frame rate for an increasing number of simultaneously tracked targets.*

However, the HTC One wasn't capable of playing more than 3 videos simultaneously, as the videos went blank and stopped functioning. The Nexus 6P, on the other hand, managed to play all five videos but the frame rate went as low as 8 Hz. The average decrease in frame rate is displayed in table 4.7.

Table 4.7: Video playback frame rate chart overview

	HTC One M7	Huawei Nexus 6P
Initial frame rate	47 Hz	49 Hz
Final frame rate	10 Hz	8 Hz
Average decrease/word	-7.2 Hz	-6.2 Hz

The power consumption measured in mW for the video playback prototype is displayed in figure 4.8.

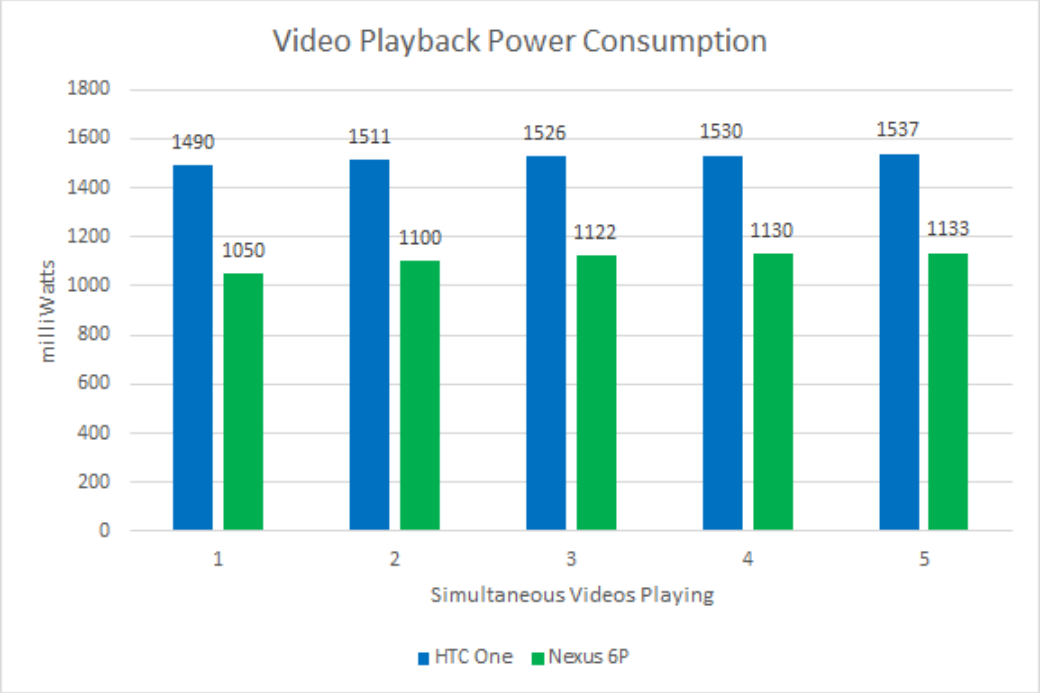


Figure 4.8: *Video playback frame rate for an increasing number of simultaneously tracked targets.*

The power consumption of the HTC One was higher initially than the previous tests of image, object and text recognition, starting at 1490 mW. The initial power consumption of the Nexus 6P was similar to these tests. The increase in power consumption per added video behaved similarly to the other tests where small increases can be observed(see table 4.8).

Table 4.8: Video playback power consumption chart overview

	HTC One M7	Huawei Nexus 6P
Initial power consumption	1490 mW	1050 mW
Final power consumption	1537 mW	1133 mW
Average increase/video	11.75 mW	20.75 mW

5 Discussion

This chapter will discuss and analyse varying aspects of the results, the method as well as present the work in relation to a wider context.

5.1 Results

The results and observed peculiarities are discussed and analysed in this section.

5.1.1 Frame rate

The general behaviour of the frame rate was as expected: as the tests progressed the load on the hardware increased progressively and the frame rate decreased in a linear fashion. What's interesting is the difference between the almost four year old HTC One model and the new Nexus 6P model. The frame rate of the HTC One was never significantly lower compared to the Nexus 6P and sometimes the frame rates were equal despite the differences in hardware seen in chapter 3.2. This is most probably caused by the difference in camera resolution. The HTC One comes equipped with a 4 MP camera, whereas the Nexus 6P model comes equipped with a 12.3 MP camera. In terms of pixels, the Nexus 6P exceeds the HTC One more than three times over. Vuforia tracks targets using natural feature tracking, a technique that is basically splitting the image into smaller sections to look for certain details. As the camera of the Nexus 6P provides the software with images three times larger compared to the camera of the HTC One, it is not surprising to see the HTC One maintaining just a slightly lower frame rate; the Nexus 6P pulls a significantly heavier load.

It is not possible to set the camera resolution of Vuforia's camera module to exact resolutions, but Vuforia offers an option called "Camera Device Mode" with three different settings:

- *MODE_OPTIMIZE_SPEED*
- *MODE_DEFAULT*
- *MODE_OPTIMIZE_QUALITY*

According to the developer forums, these settings alter the camera resolution and the resolution is device independent. All the tests were performed using *MODE_DEFAULT* on both

the HTC One and the Nexus 6P, but when searching the documentation and forums we could not find exactly which resolutions these modes translates to. However, changing the *Camera Device Mode* to *MODE_OPTIMIZE_SPEED* seems to increase the frame rate by approximately 18% on both devices when tracking image targets. *MODE_OPTIMIZE_QUALITY*, on the other hand, decreases the frame rate by approximately 15% on the Nexus 6P and 19% on the HTC One when tracking image targets. The drawback of using *MODE_OPTIMIZE_SPEED* being the loss of image quality, which was indeed noticeable on the Nexus 6P.

Looking at the results in general, one can observe that the frame rate goes below 20 Hz only in a few cases, and this is good. 20 Hz is the lower limit for visual interaction as stated by Airey et. al [3] and Li et. al[15]. It is therefore preferable to stay above 20 Hz, but when we are tracking more than 4 image targets, or more than 5 words for instance, the frame rate goes below this threshold. These frame rates are most probably caused by the CPU intensive task of estimating the pose of several targets simultaneously and this relates firmly to the research of Wagner et. al [26]. While the frame rates were acceptable overall, one has to remember that the tests did not make use of any extensive augmentations. So when developing AR applications for smartphones, and the plan is to use complex graphics on top of Vuforia's tracking, a developer should avoid tracking too many targets at once on similar hardware.

The video playback prototype also showed that there are different playback capacities on different devices. The HTC One model could not play any more than 3 videos simultaneously, while the Nexus 6P could play all five at a considerably low frame rate. One must remember that the video playback feature utilizes device dependent video players to render the video on a texture placed in the camera view. Different device's video players may or may not be capable of too many videos playing at once, which should be considered when using this feature.

5.1.2 Power Consumption

Looking at the power consumption one can observe that the Nexus 6P model consumes consistently less power compared to the HTC One, the main reason being that the Nexus 6P comes equipped with more modern hardware. For instance, the CPU of the HTC One is built using 28 nm semiconductor fabrication compared to the 14 nm of the Nexus 6P. The smaller the circuits are the more energy efficient the device is, so this is not surprising.

However, looking at the results we can see only minor differences in power consumption as more targets are added. The overhead of running the tracker seems to be the major factor here. The idle power consumption (with the screen turned on) of the HTC One and Nexus 6P was 975 mW and 650 mW respectively, with battery capacities of 2300 mAh and 3450 mAh. This translates to a battery lifetime of roughly 9 hours for the HTC One and 20 hours for the Nexus 6P in the idle case. We have to remember that natural feature tracking remains a very complex computational operation, and while the AR_Camera module is running, it *will* look for natural features in search of targets.

Simply running the prototypes with the AR_Camera active, without any detected targets or augmentations, the average power consumption of the HTC One and Nexus 6P was 1440 mW and 1060 mW respectively. These values should be considered as the minimum power consumption. Altering the *Camera Device Mode* does not seem to have any noticeable effect on these values as it did with the frame rate. The minimum power consumption results in battery lifetimes of 6.1 hours for the HTC One and 12.5 hours for the Nexus 6P. The average *maximum* power consumption when tracking multiple targets, playing video and so on was 1515 mW for the HTC One and 1129 mW for the Nexus 6P. With max power consumption the battery lifetimes decreased to 5.8 hours for the HTC One and 11.7 hours for the Nexus 6P.

The observed battery lifetimes are overall acceptable, there is seldom need for AR applications on smartphones to run for several hours straight. It is also not desirable by the user as the handling of hand-held smart devices during long sessions may cause physical discomfort [13].

5.2 Method

When measuring the frame rate and power consumption of each device, the study relied on the devices' own sensors which can affect the reliability of the results as the condition and output of these sensors may vary. For example, when measuring the frame rate, the delta time between each frame is measured by the device's own CPU; without any external alternative it is hard to say if the device provides accurate estimations, but we consider the inaccuracies to be minimal judging by the consistency observed during testing, and by using Unity's profiler we're confident they are sufficiently accurate.

The same can not be said about the tool used for measuring power consumption. PowerTutor is one of the more accurate power loggers available and one of few alternatives that actually measures power consumption in Watts and not just a percentage of the total battery pool. Despite this, PowerTutor is only capable of estimating the power consumption based on previously defined energy models [1]. These energy models were created for different phones than the ones used in this study, and this may have an impact on the validity of the results. While the values were consistent during testing, we have no way of confirming the degree of accuracy without measuring the physical electrical output from the battery. As a potential improvement, we propose using physical measuring equipment as described by Carroll and Heiser [7].

Also, the study used a very small selection of devices, with little variation in capacity. Both phones used during testing were classified as high-end devices at the time of their release. The hardware of the Nexus 6P is significantly more powerful compared to the HTC One M7, but the HTC One M7 model is still equipped with a quad-core CPU and 2 GBs of high speed RAM. It would have been interesting to evaluate the performance on mid-end to low-end devices to get a notion of performance across a wider spectrum of hardware. We have already seen that the camera resolution can affect the frame rate significantly, but what about core count, GPU and RAM? This study also isolated testing to smartphones, but the study would also benefit from analysing the performance of tablets as well, to see if Vuforia performs suitably on such devices.

We've focused on the Android operating system, but by using Unity it is possible to export our testing prototypes to a multitude of other platforms such as Windows Phones and iOS. With more time we would have liked to export our prototypes to the iOS platform and compare these results with the Android platform, as this is a major development platform - not to be neglected.

In addition to evaluation of performance on more devices, we would also have liked to develop testing prototypes for *every* Vuforia feature, not just the most major ones. This would likely give a more complete overview of Vuforia's perfomance and provide AR developers with a valuable reference when selecting their tools.

Source criticism Many of the sources regarding visual interaction, frame rate, augmented reality and its concepts were published in the late 80s and the late 90s. There are not many recent works on these subjects. Seeing how this was over 25 years ago one may question how valid these sources are today. However, looking at more recent scientific publications referenced in this paper one can also see that they too reference material published in the late 20th century. So perhaps recent works have been referencing outdated sources, or we can assume that the work done in the 80s and 90s is still valid. When covering the technology behind Vuforia's features, we mostly referenced the official documentation, but as the source code was not available for browsing, it cannot be said for certain that what is stated in the documentation is actually true. Some things are only briefly mentioned, like their natural feature tracking and object recognition for instance. Attempts were made to contact people working with Vuforia, but these attempts bore no fruits. Because of this we were forced to reference alternative sources when explaining these technologies. These alternative sources does a great

job of giving a general foundation, but nothing implementation-specific to Vuforia which is regrettable as it hurts the credibility of this paper.

5.3 The work in a wider context

This thesis achieved what it was set out to do: to provide a clear picture of Vuforia's actual performance on Android devices. However, there are many alternatives to the Vuforia SDK. The value of the results and conclusions of this paper are only fully realised when compared to the results and conclusions of other works; works that evaluate the performance of the alternatives to Vuforia. That is when developers can truly benefit and select the tool most suited for them. Naturally, the work can be extended further to include more devices, graded by hardware, and platforms to cover the versatility of Vuforia to a higher degree.

However, Vuforia is a commercial product. Developers are required to spend money if they wish to publish any app using the SDK. As such, the results presented in this paper may or may not impact Vuforia's reputation negatively. For example, we could see that the frame rate reached undesirably low levels when tracking more than 4 image targets. This fact could discourage a developer and potential Vuforia-customer from using Vuforia and instead turn to other alternatives. This implies also that other parties can use the results of this paper to portray Vuforia in a positive or negative manner compared to other products and solutions. While we realise these consequences, the goal was never to promote or demote any product, but to educate.

6 Conclusion

The purpose of this paper was to evaluate the performance of major features of Vuforia. Specifically image-, object- and text recognition as well as the video playback feature. The aim was to provide a reference for developers looking to develop augmented reality applications as AR is a rapidly growing medium. The aim has been achieved to the extent allowed by the time constraints. We ask:

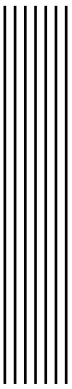
What is the performance of Vuforia SDK's image-, object- and text recognition as well as its video playback feature on Android devices when exported from the Unity3D game engine, in terms of frame rate, and battery consumption?

The answer to this question can be summarized as:

The performance of image-, object-, text recognition and video playback is acceptable in all except a few cases. The results reveal that Vuforia is capable of maintaining interactive frame rates above 20 Hz and levels of power consumption resulting in acceptable battery lifetimes for hand-held devices. In some cases, however, the frame rate reached levels lower than 20 Hz. These cases should be studied by developers, to aid them in designing their AR applications.

6.1 Future work

We have touched briefly on potential extensions to this paper when discussing the method used in this thesis. These extensions were to include testing of every feature Vuforia has to offer, not just the major ones. This would also imply development of testing prototypes suitable for these features. Additionally, the testing prototypes could easily be exported to other platforms using Unity's multi-platform capabilities, allowing evaluation of different operating systems such as iOS and Windows Phone. Future work will also include a wider selection of testing devices to observe the performance of low- to high-end devices.



Bibliography

[1] *A Power Monitor for Android-Based Mobile Platforms*. http://ziyang.eecs.umich.edu/projects/powertutor/. Accessed May 2017.

[2] *About Mono*. http://www.mono-project.com/docs/about-mono/. Accessed April 2017.

[3] John M Airey, John H Rohlf, and Frederick P Brooks Jr. "Towards image realism with interactive update rates in complex virtual building environments". In: *ACM SIGGRAPH Computer Graphics*. Vol. 24. 2. ACM. 1990, pp. 41–50.

[4] *Android NDK*. https://developer.android.com/ndk/index.html. Accessed April 2017.

[5] Ronald T Azuma. "A survey of augmented reality". In: *Presence: Teleoperators and virtual environments* 6.4 (1997), pp. 355–385.

[6] Woodrow Barfield and Thomas A Furness. *Virtual environments and advanced interface design*. Vol. 55. Oxford University Press on Demand, 1995.

[7] Aaron Carroll, Gernot Heiser, et al. "An Analysis of Power Consumption in a Smartphone." In: *USENIX annual technical conference*. Vol. 14. Boston, MA. 2010, pp. 21–21.

[8] Sravan Ch, Shivanku Mahna, and Nirbhay Kashyap. "Optical Character Recognition on Handheld Devices". In: *International Journal of Computer Applications* 115.22 (2015).

[9] Alan B Craig. *Understanding augmented reality: Concepts and applications*. Newnes, 2013.

[10] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. "Understanding the impact of video quality on user engagement". In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 362–373.

[11] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[12] *HTC One*. http://www.htc.com/us/smartphones/htc-one-m7/. Accessed May 2017.

[13] Gun A Lee, Ungyeon Yang, Yongwan Kim, Dongsik Jo, Ki-Hong Kim, Jae Ha Kim, and Jin Sung Choi. "Freeze-Set-Go interaction method for handheld mobile augmented reality environments". In: *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*. ACM. 2009, pp. 143–146.

[14] Vincent Lepetit, Pascal Fua, et al. "Monocular model-based 3d tracking of rigid objects: A survey". In: *Foundations and Trends® in Computer Graphics and Vision* 1.1 (2005), pp. 1–89.

[15] Ming Li, Katrin Arning, Luisa Vervier, Martina Ziefle, and Leif Kobbelt. "Influence of temporal delay and display update rate in an augmented reality application scenario". In: *Proceedings of the 14th International Conference on Mobile and Ubiquitous Multimedia*. ACM. 2015, pp. 278–286.

[16] *Natural Features and Image Ratings*. https://library.vuforia.com/articles/Solution/Optimizing-Target-Detection-and-Tracking-Stability#Natural-Features-and-Image-Ratings. Accessed April 2017.

[17] *Nexus 6P*. http://consumer.huawei.com/en/mobile-phones/nexus6p/specifications.htm. Accessed May 2017.

[18] *Object Recognition*. https://library.vuforia.com/articles/Training/Object-Recognition. Accessed April 2017.

[19] *PTC and Unity Announce Strategic Collaboration to Accelerate Augmented Reality Development with Vuforia*. https://www.ptc.com/news/2016/ptc-and-unity-announce-strategic-collaboration-to-accelerate-ar-development-with-vuforia. Accessed April 2017.

[20] Moo-Ryong Ra, Jeongyeup Paek, Abhishek B Sharma, Ramesh Govindan, Martin H Krieger, and Michael J Neely. "Energy-delay tradeoffs in smartphone applications". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM. 2010, pp. 255–270.

[21] Alexandro Simonetti Ibañez and Josep Paredes Figueras. "Vuforia v1. 5 SDK: Analysis and evaluation of capabilities". MA thesis. Universitat Politècnica de Catalunya, 2013.

[22] Tuan Ta, John S Baras, and Chenxi Zhu. "Improving smartphone battery life utilizing device-to-device cooperative relays underlaying LTE networks". In: *Communications (ICC), 2014 IEEE International Conference on*. IEEE. 2014, pp. 5263–5268.

[23] *Text Recognition*. https://library.vuforia.com/articles/Training/Text-Recognition-Guide. Accessed April 2017.

[24] *Unity - Public Relations*. https://unity3d.com/public-relations. Accessed April 2017.

[25] *Vuforia*. https://www.vuforia.com/. Accessed April 2017.

[26] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. "Pose tracking from natural features on mobile phones". In: *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society. 2008, pp. 125–134.