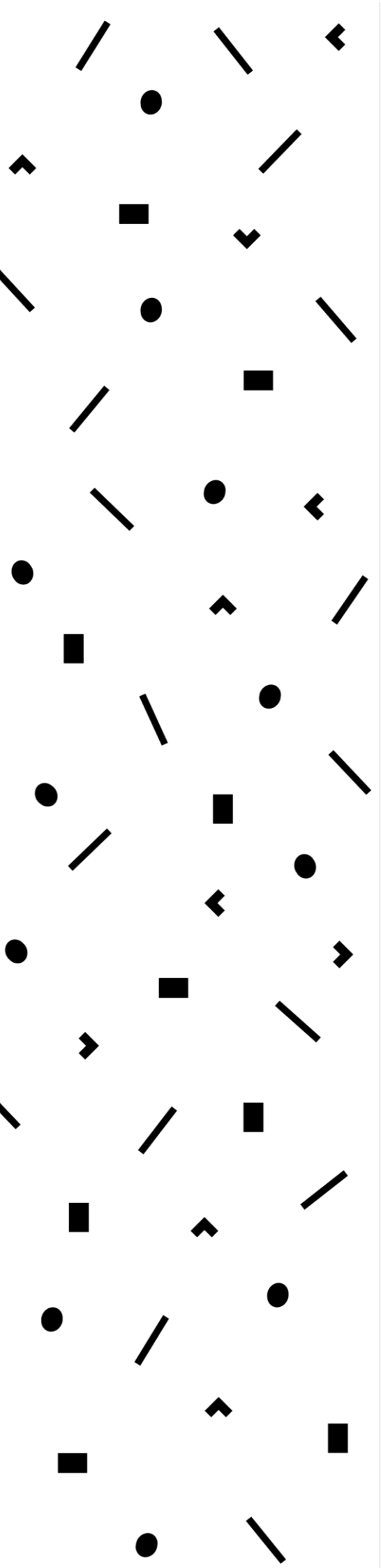

LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Remote Rendering for Virtual Reality on Mobile Devices

LAHTI UNIVERSITY OF APPLIED SCIENCES
Degree programme in Business IT
Bachelor's Thesis
29.11.2016
Aleksandr Menakhin

Lahti University of Applied Sciences
Degree Programme in Business Information Technology
Research Methods

Menakhin, Aleksandr: Remote rendering for virtual reality on mobile devices

Bachelor's Thesis 31 pages, 5 pages of references

Autumn 2016

ABSTRACT

Nowadays it is possible to launch complicated VR applications on mobile devices, using simple VR goggles, e.g. Google Cardboard. Nevertheless, this opportunity has not been introduced to the wide use yet. One of the reasons is the low processing power even of the hi-end devices. This is a massive obstacle for mobile VR technologies. One of the solutions is to render the high-quality 3D world on a remote server, streaming the video to the mobile device.

Key words: VR, virtual reality, remote rendering, Android, mobile, GamingAnywhere

ACKNOWLEDGEMENT

Before the actual thesis begins, I would like to express my gratitude to T-Systems Multimedia Solutions GmbH for introducing me to the topic of virtual reality and mobile development, for providing comfortable working conditions and technical support throughout the research process.

I am grateful to my thesis supervisor, Antti Salopuro, for the guidance and for meaningful rapid answers to every email.

Not less am I thankful to my supervisor from the company side, Frank Lamack for providing me with the topic, guidance and inspiration.

Dresden, Germany, November 2016

Alexander Menakhin

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND TERMS

AR	Augmented Reality
Bitrate or bit rate	Number of bits processed per second
CTO	Chief Technology Officer
D3D, Direct3D (9 or 11)	Graphic API for Windows (versions)
FPS	Frames Per Second
H.264, H.265	Video compression standards
IoT	Internet of Things
Jank (frame)	Lost, not displayed frame
NTP	Network Time Protocol
QoE	Quality of Experience
RAM	Random Access Memory
RTT	Round-Trip delay Time, ping
SDK	Software Development Kit
Unity3D or Unity	Cross-platform game engine
VP8	Video codec by On2Technologies
VR	Virtual Reality
VRAM	Video Random Access Memory

1 INTRODUCTION

The technology of virtual and augmented reality (VR & AR respectively) is at the stage of a rapid growth now. If recently it was taken mostly as a new way to play videogames, now more and more companies try to introduce it to a wider range of business spheres. AR has already become one of the important elements of the Internet of Things (IoT) concept (Kipper & Rampolla, 2013, 53). Previously thought to be unreachable quality of user experience makes both leading and arising brands investigate the technology and use it in marketing or to adapt their products, increasing their usability and user-friendliness.

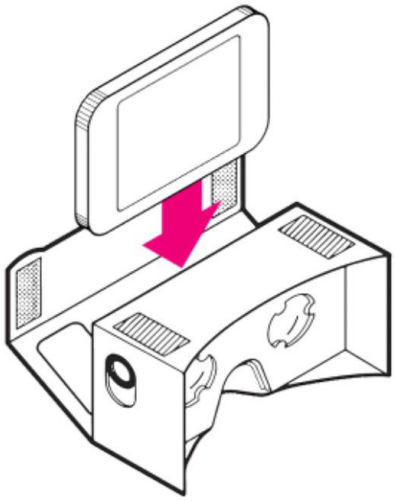

Figure 1. Google Cardboard scheme.

The introduction of VR to mobile devices made it even more attractive, especially after the release of Google Cardboard – a mobile VR platform that uses a low-cost viewer, made mostly of cardboard. It has no screen, camera or any computing capacity; for all of that you need your smartphone (figure 1). A Google-Cardboard-optimized app divides the smartphone's display in two parts (for left and right eye) and then the smartphone is inserted into the Google Cardboard. This has made VR affordable for every smartphone owner and Google VR software development kit (SDK) has provided developers with some powerful tools

that can be used with Unity, Android, iOS and Unreal Engine 4 development environments (Google VR, 2016).

Although the possibility of using a smartphone for a real VR experience has created a wide range of opportunities, it has also put some new boundaries: smartphone's display resolution and computing power are far from optimal for virtual reality.

The abovementioned limitations together with the demand of the high-quality 3D world in VR lead to a probable solution: rendering the high-latency 3D world on the remote server and streaming the video back to the smartphone.

The probability of the successful realization of both software and hardware parts of a solution for remote rendering for virtual reality on mobile devices, showing acceptable level of quality, is currently questionable and unclear for the author. Throughout the research process the comparable solutions will be found and studied, optimal technical requirements will be calculated and tested.

2 RESEARCH METHODOLOGY

2.1 Research Questions and Objectives

The purpose of the paper is to find an open source remote rendering platform that supports mobile devices and, using it as an example, to figure out if it is feasible to use the remote rendering approach for virtual reality on mobile devices now or in the nearest future.

1) Does a remote rendering system meet the minimal VR requirements?
2) What are the main challenges of using remote rendering with mobile VR?

2.2 Research Structure

The actual research can be generally divided into three parts: (1) theoretical part, serving to reach the understanding of the key concepts, the existing remote rendering systems, methods, approaches and classifications, to choose the software for testing and to define the minimal requirements the chosen software must meet; (2) experiment, including design, conduction and the measurement and data collection techniques that may vary, depending on the software, chosen after in the theoretical part; (3) data analysis, including the comparison of the resulted benchmarks to the minimal performance requirements, defined in the theoretical part and finding the key challenges. The actual thesis structure is divided into more parts for better understanding and readability.

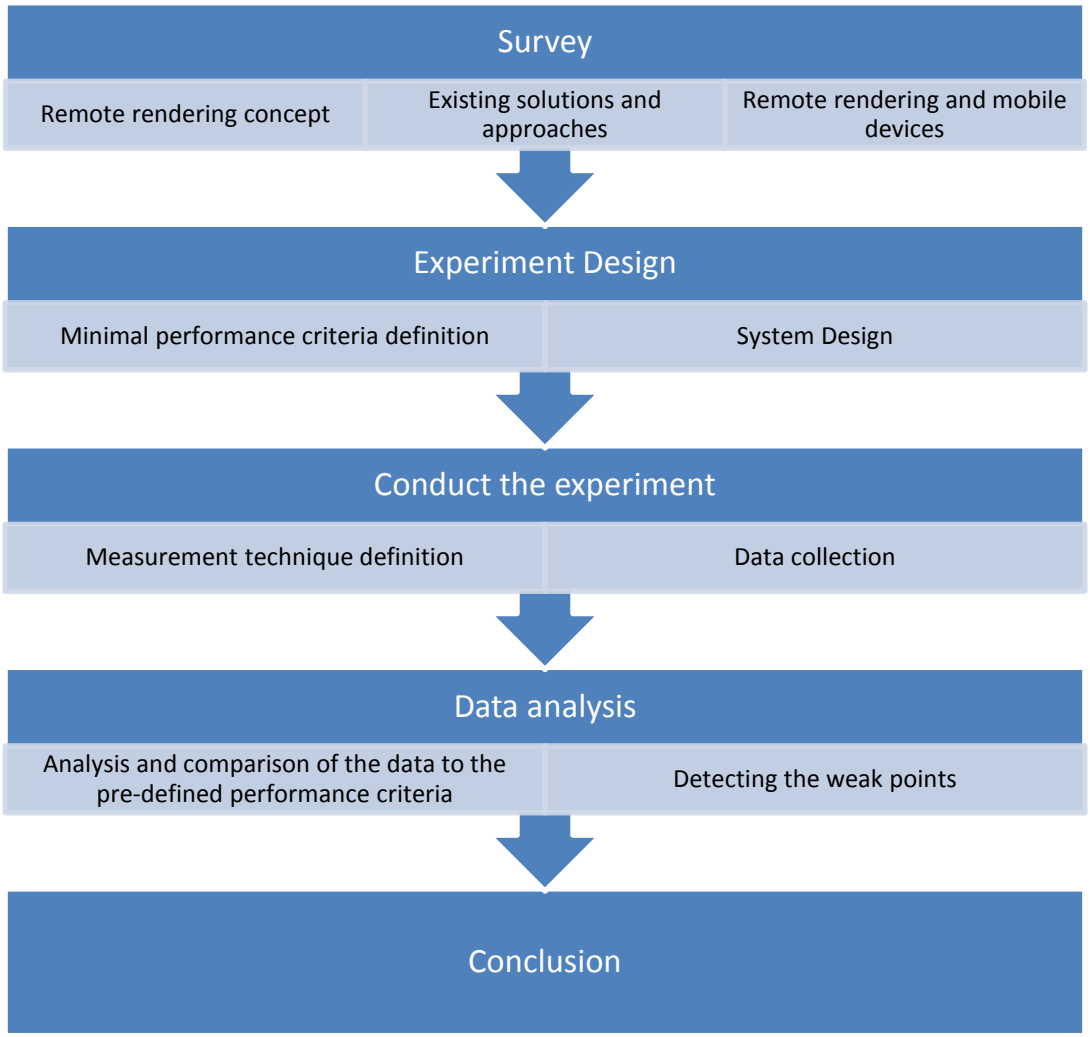

Figure 2. Research structure

Before the experiment a simple VR application for Windows will be implemented and built with the help of Unity3D game engine. The app will be run on the PC, and streamed to a smartphone in the local network with the help of a chosen remote rendering platform.

2.3 Research Method

The data to be collected during the experiment is needed to evaluate the quality of experience that is achieved. According to the survey, made by Huang et al. (2014), delay and video quality affect the QoE the most in cloud gaming.

Video quality can be evaluated by bitrate, resolution and frame rate. *Bitrate* is a numeric measure which tells the number of bits that can be processed within one second. *Frame rate* is another criteria of the overall video quality, it is measured in frames per second (FPS) and shows the frequency at which a device displays frames. Low frame rate causes the "slideshow" effect in games and streaming, high frame rate is essential for the smoothness of the video stream, especially in VR.

Delay is the time span between the moment, when a frame is rendered on the server and the moment, when it is displayed on the device's screen. It consists of the *network delay*, which is often referred as the network *round-trip time (RTT)* or *ping*, and represents the time needed to deliver a player's command to the server and return a game screen to the client and the *processing delay* on the client side – a period of time, needed to decode and display the screen (Chen et al. 2011).

Hence, after the experiment sessions, held with different configurations, described in further chapters, the measurement data, collected from different measurement points (configuration combinations), for every or n^{th} frame will be the revealed. This brings us to the point, where it is possible to state: the data analysis part will be **quantitative**.

2.4 Limitations and Boundaries

Before the start of the survey the author is aware of the lack of popular and widely used remote rendering systems with open code, especially of those supporting Android and/or iOS. Also the author expects the existing platform(-s) not to support gyroscope interaction with the server, which is the core feature of mobile VR headmounted devices (e.g. Google Cardboard). Nevertheless those cases will be reviewed and evaluated by other aspects of quality of performance, since the additional ways of interaction are possible to add to the mobile clients in the future.

The list of technical limitations is amplified by: (1) *remote rendering* is meant to happen in the local network (LAN/WLAN), (2) *mobile VR* is limited to Google Cardboard, (3) *Samsung Galaxy S5 Plus* is used as a client device.

3 REMOTE RENDERING

Remote rendering in the context of this paper is the process, when rendering of 3D models and showing the output graphical data take place on separate devices: a rendering server and a client device respectively.

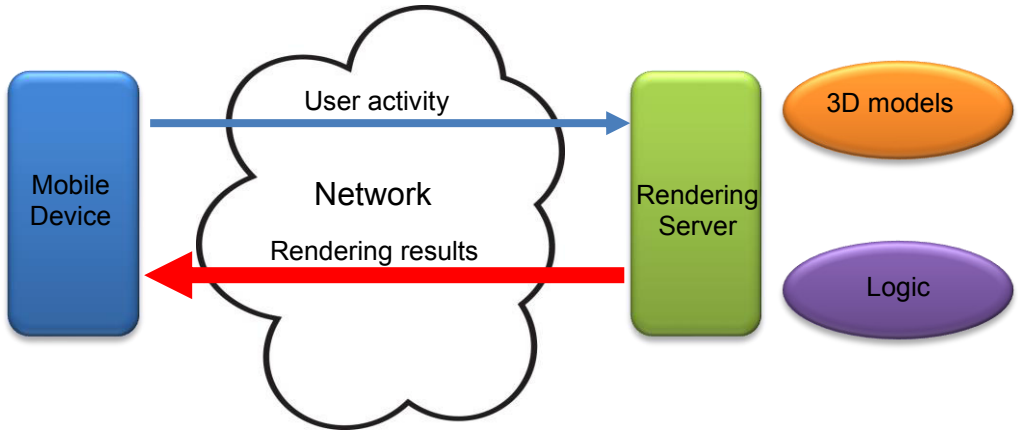

Figure 3. Basic architecture overview

A client device serves as an input device for user interaction and as an output device for graphical data, received from the server in response. Such an approach helps to use low-capacity devices for tasks requiring highly detailed 3D data, e.g. virtual reality applications.

However, it creates a demand on high quality broadband connection and puts a strict limit on the reaction time that may be problematic to achieve if working with the server situated elsewhere than the local network.

3.1 Model- and Image-Based Classification Approaches

Shi et al. (2012) proposed classification, according to which, there exist two major approaches to remote rendering: model based and image based. In the following sub-clauses they will be analyzed and evaluated by the bandwidth and client device capacity requirements.

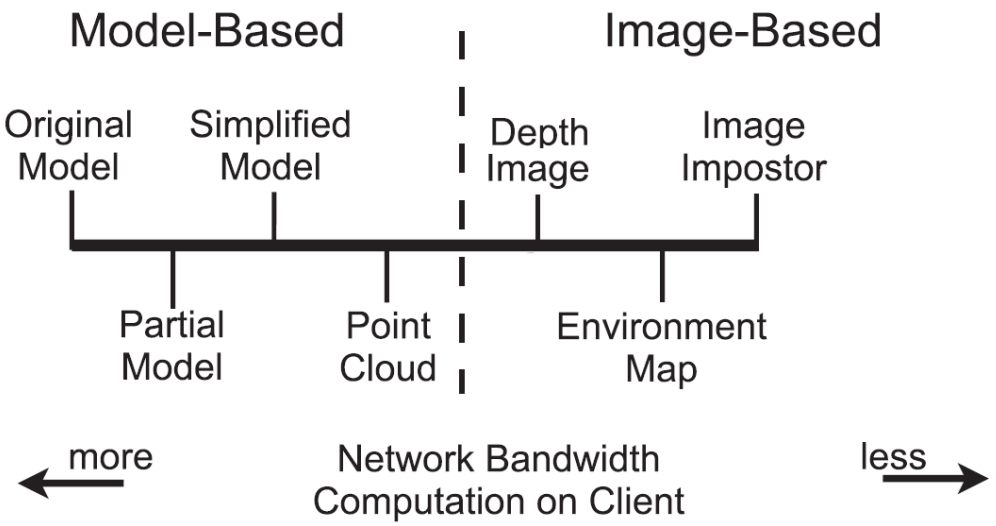

Figure 4. Classify remote rendering and remote visualization systems based on data types (Shi et al., 2012, 6).

3.1.1 Model-Based Approach

A group of remote rendering techniques that send the full-size or anyhow simplified 3D models to the client are called model-based. There exist a number of variations inside this group, differing by the level of involvement of the client device to the rendering process.

Original Model. In this approach the whole 3D models are stored or generated and then streamed to the client that fully handles the rendering part. It is feasible to use only in cases, where the major difficulty is to generate, rather than to render the models.

Partial Model. The original model approach creates a lot of difficulties, when the environment to be streamed and rendered is large and of high resolution or includes complex texture and geometry, then it will most possibly take excessive network bandwidth and a considerable amount of time. In the partial model the 3D objects are sorted by their importance, according to the viewpoint's position. The objects that are closer to the user's position are sent in their original resolution, while objects, situated

farther and therefore in less-important zones, can be streamed at the reduced quality and resolution.

Simplified Model. This approach was introduced by Levoy (1995) and is based on sending a simplified model together with an image of the rendering difference between the simplified and the original models to the client. The client may have a lightweight rendering engine, since it has to render a simplified model and then to add the difference image to the rendering result. Such an approach loads the network connection much less than the abovementioned ones, uses fewer client device computing resources and maintains the same rendering quality. However, the highly loaded server, which has to render both original and simplified models and then compare them, must be mentioned among the disadvantages.

Point Cloud. This approach is related to a simplified model, described above, with the difference that a point cloud, instead of a polygon mesh is streamed to the client. The idea originates from the fact that the mobile devices are incapable not only to render, but also to display the high-polygon models. (Duguet & Drettakis, 2004). However, the technical capabilities of the smartphones have changed since 2004 and nowadays bad quality of the models is the same level problem as the difficulty of heavy models rendering.

3.1.2 Image-Based Approach

An image-based systems implements all rendering operation on the server, streaming the result images to the client. Therefore there is no need in graphical hardware for 3D rendering on the client device, but on the other hand it creates the demand on a stable network connection and increases the dependency of the performance quality on the common reaction time, including both package transactions and server processing.

Image Impostor. This approach is the most popular by far (Ohazama, 1999; Lamberti & Sanna, 2007; Noimark & Cohen-Or, 2003). All the 3D

models are rendered on the server and the client receives only 2D images that it needs to display and then send the user interactions back. The required network bandwidth becomes dependent only on the target device's screen resolution and the complexity of the 3D models affects only the server-side rendering time. (Shi et al., 2012).

Environment Map. Environment map has found its niche and has been widely used in 3D game development to simplify the rendering of faraway background objects. Chen (1995) and Boukerche & Pazzi (2006) claim that it can be effectively applied to reduce the iteration latency. The environmental map is actually a 360 degrees image of the environment, rendered by the server according to the viewpoint position given by the client. The major obvious advantage is that tilting and turning the virtual camera around will not require any further server requests and the results will be shown with no delay. The problems might appear when it comes to moving the viewpoint position and the server must render the whole environment again and the streamed results are much heavier than in image impostor approach.

Depth Image. In depth image remote rendering systems the server sends back an image containing both colour and depth map, rendered according to the user's viewpoint position. The client device can display the received colour map immediately, like in the image impostor approach, until the viewpoint changes. After the user changes the viewpoint, the client implements 3D image warping according to the depth map. This approach can be considered as a simple version of *Point Cloud* because every pixel of the depth image represents a 3D point in the space, with the difference in the amount of computation: 3D warping is a much more lightweight operation, than rendering a point cloud with 3D graphics pipeline (Shi et al., 2012).

3.2 Current Situation in Remote Rendering

The first attempts of remote rendering research appeared already in the 1990s, caused by the inability of an average PC to deal with 3D graphics at all (Ohazama, 1999). Later, when the development of high-speed connections and cloud computing, Beermann & Humphreys (2003) proposed that 3D rendering will become a remote service and it came true with introduction of cloud gaming (Ross, 2009). Current solutions in remote rendering can be divided into 2 main categories (Shi & Hsu, 2015).

Thin clients and remote sharing. Thin client systems, like SLIM (Schmidt et al., 1999) or THiNC (Baratto et al., 2005) as well as remote desktop sharing systems, like RDP (Cumberland et al., 1999) or VNC (Richardson et al., 1998) allow users to access applications remotely and share computing resources. Through the thin client a user can access and interact with the app, launched on the server. Practically no computing happens on the client side that is why such an approach is called “thin”, or in other words lightweight.

Discussing on thin clients, the following details must be considered: those systems initially appeared without any support for 3D graphics and were used only for remote desktop sharing. Only several recent implementations include 3D rendering support (e.g. THiNC and TurboVNC). What is more, the major aim of the protocols, designed for 2D graphics sharing, was to efficiently update regional changes on the screen, since 2D rendering is relatively lightweight and can happen either on a server or on a client (Shi & Hsu, 2015). Therefore, further in this research we will focus only on those thin clients that support 3D graphics remote rendering.

Distributed Graphics and Cluster Rendering. Talking about remote rendering solutions, distributed graphics and cluster rendering systems must be considered. This approach is used, when one server is not enough to render complex 3D graphics, or when the outcome should be a

huge image (e.g. a wall with multiple screens (Staadt et al., 2003)). Although distributed rendering stands close to the issue of remote rendering, there are key differences in the aim of research: distributed rendering focuses more on dividing graphics rendering operations between several servers, while remote rendering in its pure sense aims to optimise the process of client-server interactions, including data compression, image processing and transferring, latency reduction.

Among the examples of existing distributed graphics and cluster rendering systems WireGL (Humphreys et al., 2001), Chromium (Humphreys et al., 2002), OpenGL Multipipe SDK (Bhaniramka et al., 2005), ParaView (Cedilnik et al., 2006) and Equalizer (Eilemann et al., 2009) can be mentioned.

4 REMOTE RENDERING ON MOBILE DEVICES

The situation on the mobile remote rendering systems market is not that delightful though. Shi et al. (2012) introduced a multi-depth image-based mobile remote rendering system with user interaction prediction algorithm, but common logic and calculations in Shi et al. (2012, 2009) and Shi & Hsu (2015) showed that such an approach is optimal for scientific visualisations or any other case with static models and restricted interaction, preferably if the number or viewpoint positions and allowed virtual camera movement directions is also limited.

2D remote rendering, as well as remote desktop sharing platforms with no support for 3D remote rendering together with systems, based on command streaming, will not be reviewed and considered further in this paper, since either they do not meet the requirements for mobile VR QoE requirements or they do not help to visualize high-quality 3D graphics on mobile devices.

The first open-source cloud gaming system, called GamingAnywhere (Huang et al., 2014 (1, 2)) was chosen as a case study for this research. Cloud gaming and VR systems are similar in terms of QoE requirements, especially latency. Later in this research, a VR app will be launched on a remote server, running GamingAnywhere server module, it will be accessed from a GamingAnywhere thin Android client, the benchmarking will be made and analysed.

GamingAnywhere server can be of two types: periodic and event-driven (*ga-server-periodic* and *ga-server-event-driven*). The first one periodically captures the desktop or a window and the second one hooks directly into the game executables to capture a game screen every time right after the game updates the screen. (Huang et al., 2013 (2)). To achieve the best performance an event-driven server should be used.

5 QUALITY REQUIREMENTS

Virtual Reality puts strict quality requirements, which can be explained by two factors: (1) wearing a head-mounted device means the close proximity of the display to the user's eyes, so that he/she can see all the details and any image irregularities, like low sharpness or blur, result in awful user-experience, (2) limited eyeshot makes your brain to accept the VR world as real, so long reaction time between the movement of the user's head (or any other interaction) and the resulted image, shown on the screen(s), may lead to vestibular system confusions, causing symptoms, like general discomfort, headache, stomach awareness, nausea, vomiting, pallor, sweating, fatigue, drowsiness, disorientation, and apathy. (Kolasinski, 2014).

The CTO of Oculus VR John Carmack has said:

> *A total system latency of 50 milliseconds will feel responsive, but still noticeable laggy (Oculus Rift Blog, 2014).*

Nevertheless, considering the fact that we are working with remote rendering and mobile devices, as well as the fact that the OnLive and StreamMyGame cloud gaming platforms perform 250ms on average (Huang et al., 2013 (2)), and based on empirical evidence, **100ms** are assumed to be tolerable and will be further referred as *maximum delay value*.

The criteria, responsible for the video quality may be adjusted on the server side and will result in a higher or a lower delay, since higher FPS, bitrate and resolution considerably increase the bandwidth requirements and the overall amount of data, streamed within a network. Therefore, the positive answer on the question "Is it feasible to use a remote rendering system, based on GamingAnywhere platform for virtual reality on mobile devices?" will be given in case at least one configuration variant that

meets the VR video quality requirements will be able to perform a tolerable delay value in milliseconds during an experiment.

Both senior staff engineer of Sony Chris Norden and the founder of Oculus VR Palmer Luckey have established the minimum frame rate requirement for VR developers of 60 frames per seconds (LinusTechTips, 2014; Hall, 2016). Nevertheless, in this research the topic of mobile VR is reviewed and here the situation is a bit different: mobile devices, running iOS and Android, cannot render more than 60FPS due to vertical synchronization, and if the app runs at less than 60FPS, it will drop to 30FPS (Purcell, 2016). Therefore client side frame rate of **30FPS** will be considered acceptable in the experiment and only two frame rate configurations will be tested: 30FPS and 60FPS.

The minimum resolution configuration will be 1280x720p (HD), 1920x1080p (FullHD) will also be tested. Default and optimal for QoE on mobile client bitrate of GamingAnywhere is 3 Mbps. YouTube proposes **1.5Mbps** to be the minimal bitrate for an HD (720p) video streaming, higher values will not be taken, since they strongly increase the delay time. (Huang et al., 2014). Thus, 1.5Mbps and 3Mbps will be taken as test cases.

6 EXPERIMENT SETUP

6.1 Test Cases

As it was mentioned in section 2.2, the aim of the experiment is to figure out, if GamingAnywhere is able to perform a tolerable delay value, delivering a video quality level that meets the VR requirements. According to the minimal VR quality requirements, given in the fifth chapter, 8 test cases were designed.

Table 1. Test cases.

Frame rate	Resolution	Video bitrate
30FPS	1280x720	1.5Mbps
30FPS	1280x720	3Mbps
30FPS	1920x1080	1.5Mbps
30FPS	1920x1080	3Mbps
60FPS	1280x720	1.5Mbps
60FPS	1280x720	3Mbps
60FPS	1920x1080	1.5Mbps
60FPS	1920x1080	3Mbps

All 8 configuration combinations, shown in table 1, are assumed to be acceptable for virtual reality. During the experiment, we will use GamingAnywhere with the VR application from a thin client, running on the smartphone in the course of 5 minutes for each of those configuration combinations. Here it is important to mention that both frame rate and bitrate, set up on the server side, may and most probably will vary from the ones, resulted on the client side, so only if *at least one of the configuration variants* is able to perform a *delay value less or equal to 100ms*, as well as a *client side frame rate value not less, than 30FPS and the bitrate, not less than 1.5Mbps*, the positive answer to the first research question will be given as a conclusion.

6.2 Configuration

GamingAnywhere (GA) provides several specially adapted games together with the configuration files for them on their website, but will not be reviewed or used during the research, since none of them is available in VR mode. A simple Windows standalone app, consisting of one 3D scene and two virtual cameras for both eyes, was made in Unity 5.4.0f (64 bit). A user can look around an assembly hall, get information about its features and components by staring at the infopoints (magenta circles) and move between three viewpoints by staring at the blue spheres on each of the viewpoints.

Figure 5. Screenshot of the app.

Figure 6. Screenshot of the app: InfoPoint.

Empirically, Direct3D9 was chosen as a graphical API for Windows, since Direct3D11, as well as SDL, could not be hooked by GA event-driven server. Thus, any standalone Windows app, built by Unity and using Direct3D9 can be used for mobile gaming with GA.

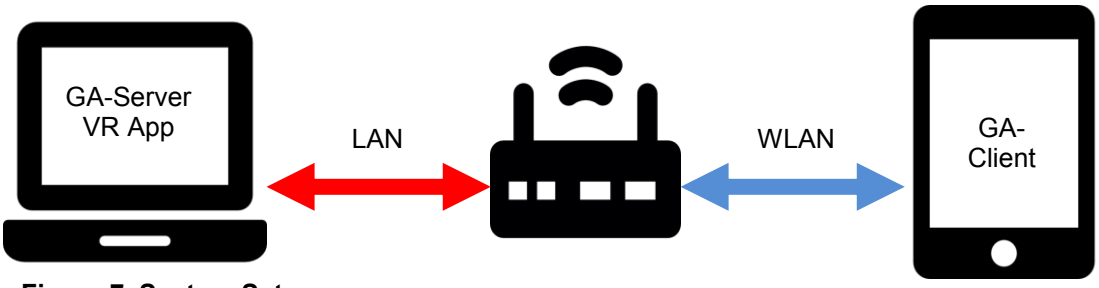

Figure 7. System Setup.

The architecture of the system was rather simple: a laptop (Intel Core i5 3230M 2,6Ghz, 6GB RAM, NVIDIA GeForce GT 730M with 2GB VRAM, Windows 8.1) was connected to a router (802.11n) with an Ethernet cable, and the smartphone (Samsung Galaxy S5 plus) used Wi-Fi to connect to the network. The event-driven binary of GA was running on the laptop, capturing the frames from the VR app and streaming them through RTP to a GA client instance on the smartphone.

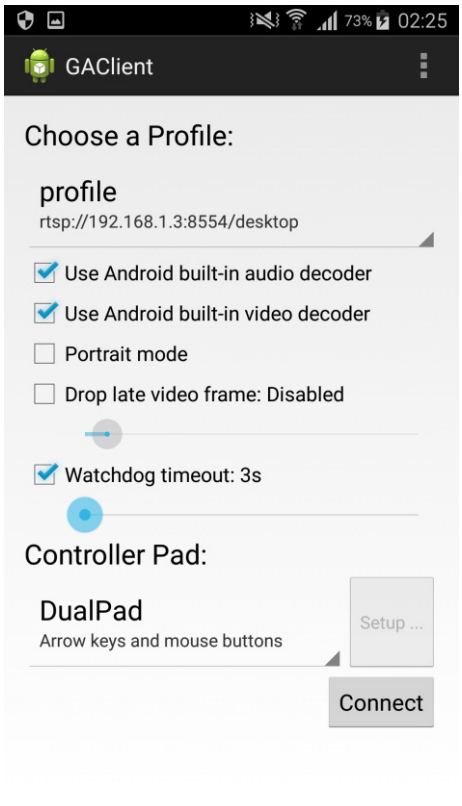

Figure 8. GA client: Home screen

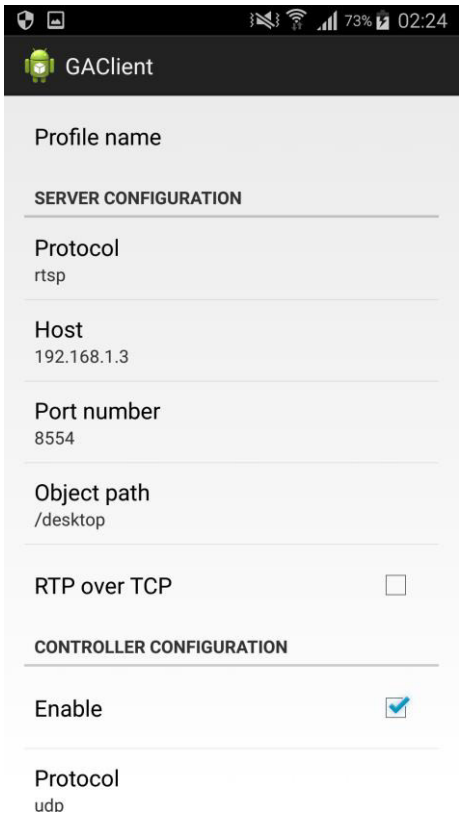

Figure 9. GA client: Profile configuration

Two types of codecs can be used by GA mobile client to decode the incoming video stream: software codecs and Android built-in codecs (figure 8). The software codecs are the same, as the ones, used by GA-server, while the built-in codecs are provided by the Android *MediaCodec* framework. Obviously, the built-in hardware-accelerated codecs perform better in terms of decoding speed. (Huang et al., 2014 (2)). However, the usage of the built-in ones results in totally corrupted colours, with no difference, whether H.264, H.265 or VP8 are used for video encoding on the server side. All the experiment sessions (table 1) are held with H.264 encoders on the server side and the client is configured to use software decoders.

Moreover, there was one issue: the Android version of GA client, available in alpha, does not support gyroscope control, which is essential for head-mounted VR devices. Theoretically this functionality can be added in the future and the lack of it will not affect the research results. Arrow buttons control, emulated by the client, is used for lookaround instead.

6.3 Measurement Technique

6.3.1 Applications and tools used for measurements

GameBench app is installed on the client device to count frame rate. The choice was made empirically and also based on Zhu & Shen (2016). Among calculating median FPS and showing overall performance statistics, including CPU, battery and GPU usage, the app takes screenshots every second during the benchmarking session, which is indeed helpful.

ClockSync app is used to measure the difference between Android system time, and the time on Windows NTP server (time.windows.com).

Time and Memo app is used to display the overlaying widget with Android system time with milliseconds above.

GamingAnywhere provide log files, containing the information about the resulted bitrate and RTT.

6.3.2 The process

As described in section 5.1, 8 configuration combinations are used to evaluate the feasibility of using a remote rendering system, based on GamingAnywhere cloud gaming platform with virtual reality applications on mobile devices. All the configurations (resolution, bitrate and frame rate) are set up on the server side. The delay and resulted frame rate values measurements are taken on the client side.

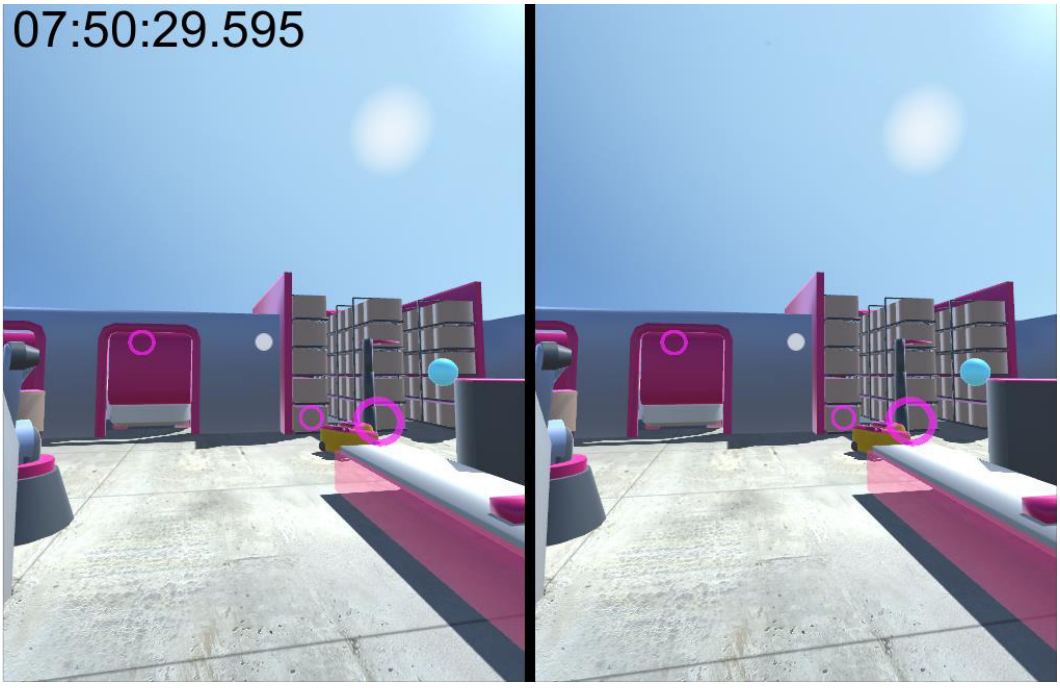

Figure 10. Visual timestamp.

In order to measure the common delay, a method, similar to the one, described in Chen et al. (2011), is applied. A text field with system time with milliseconds updates each frame after all the calculations are performed (Unity LateUpdate() method), so we have a visual timestamp for every frame rendered, as it is shown in figure 10.

On the client side Time and Memo app is installed to display Android system time with milliseconds on top of all open application screens, so every time GA client shows a frame, information about both when the frame has been rendered on the server and when it has been displayed on the client can be seen. The time is updated on both server and client, but there still exists a difference from -330 to -300 milliseconds. This fact is considered and kept in mind.

The time difference values were elicited using ClockSync after updating the time 10 times within one minute.

Figure 11. The view during benchmarking on the client side.

After each five-minutes-long testing session, 300 snapshots are made by GameBench (e.g. figure 11), every 30th screenshot is manually inspected, both server and client timestamps are collected and put into an Excel list, then server-client time difference is added.

If an average value, considering the measurement uncertainty of 30ms, appears to be less or equal to 100ms, and the average frame rate, measured by GameBench is more or equals to 30FPS, the configuration meets the requirements of VR QoE.

7 RESULTS

As it was mentioned in 6.1, 8 testing sessions were made with different configurations.

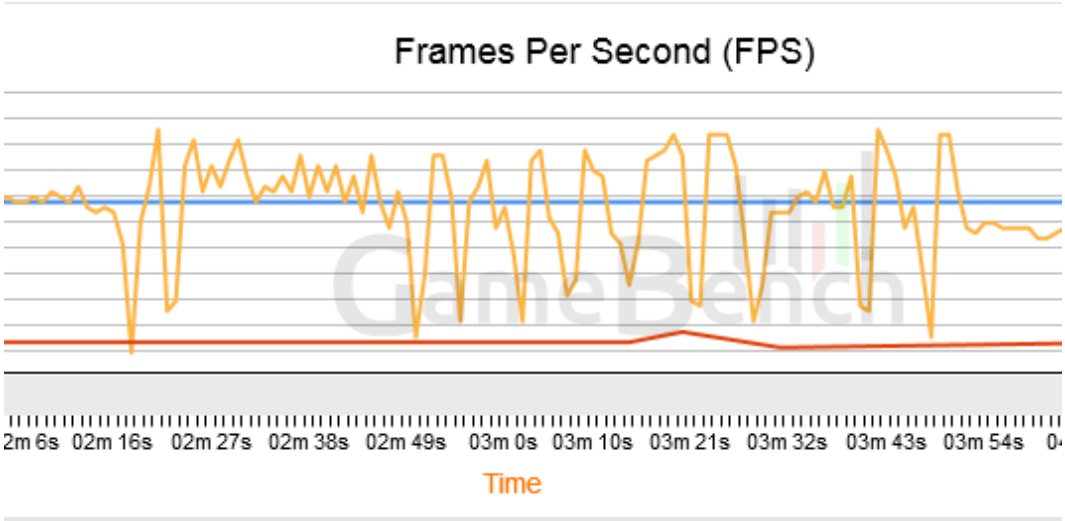

Figure 12. GameBench metrics interface

The median frame rate is counted by GameBench (figure 12, blue line), the average frame rate is calculated by exporting the frame list as CSV (timestamp, fps) and applying a simple C# script to read it and count the amount and the sum of values. The information about average RTT and real bitrate is provided by GamingAnywhere log files.

Table 2. Sessions' results

	Session 1	Session 2	Session 3	Session 4	Session 5	Session 6	Session 7	Session 8
FPS	30 FPS	30 FPS	30 FPS	30 FPS	60 FPS	60 FPS	60 FPS	60 FPS
Resolution	HD	HD	FullHD	FullHD	HD	HD	FullHD	FullHD
Bitrate	1,5 Mbps	3,0 Mbps	1,5 Mbps	3,0 Mbps	1,5 Mbps	3,0 Mbps	1,5 Mbps	3,0 Mbps
Median FPS	29 FPS	26 FPS	12 FPS	11 FPS	28 FPS	23 FPS	16 FPS	19 FPS
Average FPS	25 FPS	21 FPS	10 FPS	8 FPS	26 FPS	23 FPS	13 FPS	15 FPS
Real bitrate	1,6 Mbps	3,5 Mbps	1,2 Mbps	1,8 Mbps	0,9 Mbps	1,6 Mbps	0,6 Mbps	1,2 Mbps
Average delay	550 ms	2086 ms	1106 ms	731 ms	1266 ms	866 ms	1862 ms	1948 ms
Median delay	489 ms	1936 ms	1128 ms	411 ms	1091 ms	488 ms	1397 ms	1773 ms
RTT	23 ms	55 ms	36 ms	59 ms	23 ms	16 ms	66 ms	43 ms

The cell colouring in table 2 marks the acceptable (green), tolerable (yellow) and inacceptable (red) benchmarks. As we can see, none of the configurations performed well enough to meet the VR, described in chapter 5, although the results of session 1 are relatively close: median frame rate is 29, which one FPS less, than the target value of 30, and real

bitrate is even a bit higher, than was required. Nevertheless, even with the minimal configuration, both average and median delay time is around 5 times bigger, than the required 100ms, which makes it totally impossible to use with virtual reality.

8 DATA ANALYSIS

Although, preliminary conclusions could be made, according to the mean and median values, deeper data analysis is mandatory to achieve the maximum objectiveness. Especially it concerns the data about delay, due to the physical inability to count the value for each of 300 frames in each of 8 testing sessions.

Table 3. Delay analysis

		Delay, ms							
		Sessions							
		1	2	3	4	5	6	7	8
Frames	1	45 ms	705 ms	292 ms	1031 ms	301 ms	177 ms	992 ms	2795 ms
	2	8 ms	1370 ms	1238 ms	1405 ms	667 ms	117 ms	641 ms	1075 ms
	3	638 ms	2804 ms	2719 ms	1054 ms	2025 ms	1620 ms	1756 ms	187 ms
	4	967 ms	2142 ms	1037 ms	1831 ms	1118 ms	2576 ms	6779 ms	1470 ms
	5	1014 ms	1963 ms	1197 ms	357 ms	1064 ms	677 ms	1858 ms	1797 ms
	6	1108 ms	1910 ms	719 ms	399 ms	165 ms	1277 ms	1951 ms	1750 ms
	7	108 ms	1515 ms	443 ms	238 ms	170 ms	1390 ms	876 ms	369 ms
	8	1053 ms	3764 ms	1152 ms	424 ms	2374 ms	250 ms	1037 ms	1987 ms
	9	341 ms	3379 ms	1161 ms	280 ms	3055 ms	298 ms	922 ms	3001 ms
	10	220 ms	1313 ms	1104 ms	287 ms	1717 ms	274 ms	1808 ms	5048 ms
Mean		550	2087	1106	731	1266	866	1862	1948
Standard Deviation		454	966	657	562	999	821	1795	1418
Confidence Interval (N)		282	599	407	349	619	509	1112	879
Lowest border (N)		**269**	**1488**	**699**	**382**	**647**	**357**	**750**	**1069**
Highest border (N)		832	2685	1514	1079	1884	1374	2974	2826
Confidence Interval (t)		325	691	470	402	714	587	1284	1014
Lowest border (t)		**225**	**1396**	**636**	**328**	**551**	**279**	**578**	**934**
Highest border (t)		875	2777	1576	1133	1980	1453	3146	2962

Risk	0,05

In table 3 the delay values in milliseconds are input for every 30th frame of each session. The “mean” row contains the same values, is table 2 (average delay). The standard deviation is counted with the use of Excel STDEV.S function that estimates population standard deviation based on a sample (Microsoft Office Support, 2016 (2)), afterwards the confidence interval is calculated with CONFIDENCE Excel function (Microsoft Office Support, 2016 (1)), which estimates the range of values, where the population mean will be situated, with the given risk of 5%, or in other words with 95% level of certainty. The confidence interval is counted with both normal (marked with “N”) and Student’s (marked with “t”) distribution. One of the reasons to use Student’s distribution is the small sample size. Knowing the confidence interval and the sample mean, it is easy to

calculate the lowest and the highest borders of predicted population mean values. Nevertheless, even the lowest possible value of the population mean delay with normal distribution in every session is at least 2.69 times as high, as the maximal value, acceptable in VR, and the one with Student's distribution is still more than twice as high.

The same procedure is done with another average value: the resulted frame rate. Here it is impossible to provide the similar table, since the FPS data is available for every frame, and the resulted table contains over 300 rows.

Table 4. Frame rate analysis

	Sessions							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Mean	25,6	21,5	11,1	8,9	26,1	23,6	13,6	15,8
St. Deviation	12,39	14,37	8,34	5,85	12,83	13,45	9,29	9,20
Confidence	1,40	1,63	0,94	0,66	1,45	1,52	1,05	1,04
Lowest border	**24,2**	**19,9**	**10,1**	**8,3**	**24,6**	**22,1**	**12,5**	**14,7**
Highest border	**27,0**	**23,1**	**12,0**	**9,6**	**27,6**	**25,2**	**14,6**	**16,8**

Risk	0,05

Table 4 shows much higher level of certainty. With a sample of 300 values for each session, the difference between confidence interval with normal and Student's distribution is less, than 1/100, so it was decided not to include the ones with Student's distribution to the table.

Answering the second research question, the data from the second table was analysed, in order to figure out, what exactly is the main challenge for using remote rendering in general and GamingAnywhere particularly in VR solutions for mobile devices.

Table 5. Processing delay

Average delay	550 ms	2086 ms	1106 ms	731 ms	1266 ms	866 ms	1862 ms	1948 ms
RTT	23 ms	55 ms	36 ms	59 ms	23 ms	16 ms	66 ms	43 ms
Processing delay	527 ms	2031 ms	1070 ms	672 ms	1243 ms	850 ms	1796 ms	1905 ms
PD / Delay	95,82%	97,36%	96,75%	91,93%	98,18%	98,15%	96,46%	97,79%

Another interesting question is which part of overall delay time is the largest. As it can be seen from table 5, the processing delay (PD = average delay – round-trip delay time) makes from 91.93% to 98.18% of the overall delay. It is important to keep in mind that the experiment was held within the local network, if the server and the client had been situated in the remote networks, the round-trip delay time would have been considerably bigger. However, it is possible to conclude that the time, needed for a mobile device to decode and display the image, is unacceptably long. Thus, the decoding process is mostly responsible for such a long delay, which can be partially explained by the fact that software decoders were used during the experiment, as it was stated in section 6.2.

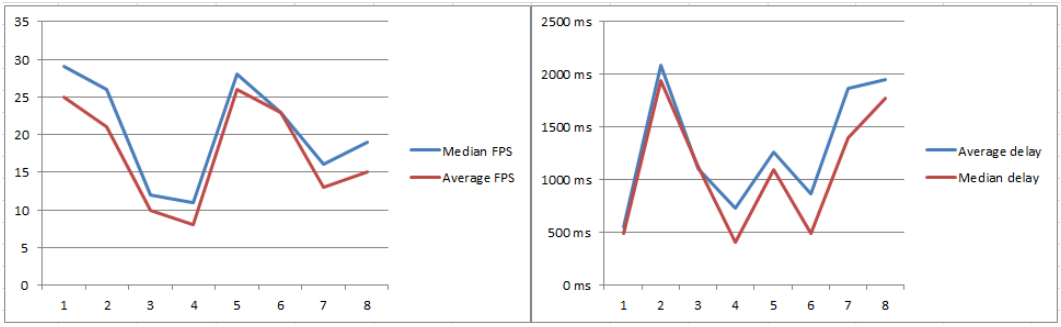

Figure 13. Median and average values correlation charts.

What is more, the average real frame rate is always lower, than the median, and the average delay is always higher, than the median (figure 13), which means that the video was relatively instable and several times performed minimal FPS and high delay values, resulting in frame losses and janks, which is totally inacceptable for VR QoE.

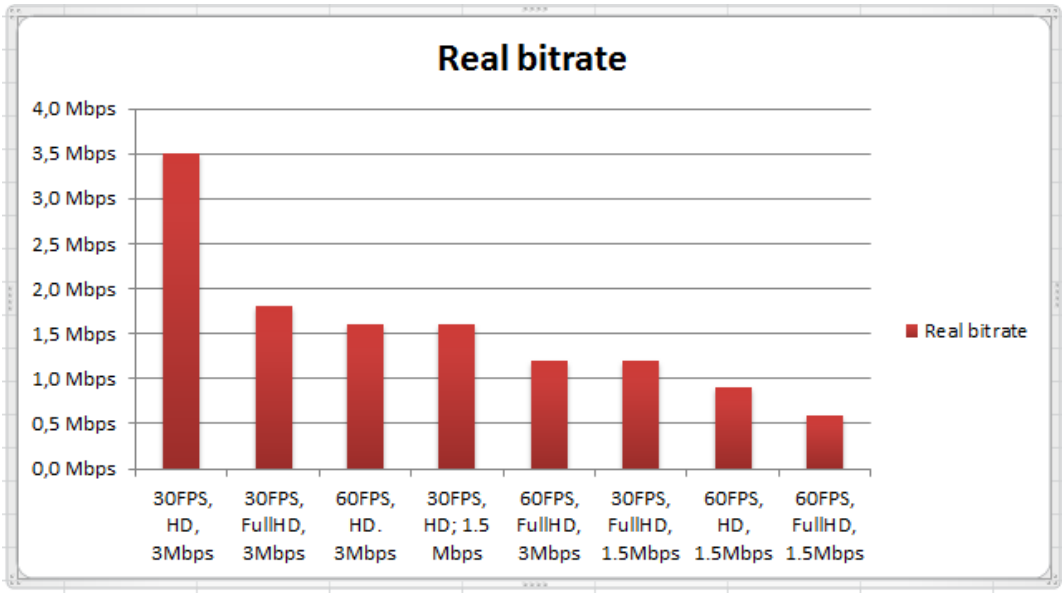

Figure 14. Real bitrate

GamingAnywhere could perform the target bitrate values only in two cases (first and fourth configuration in figure 13), both configured on the minimal frame rate (30FPS) and resolution (1280x720, HD). Two more cases (second and third in figure 14) could perform the acceptable values (more or equal to 1.5Mbps), although the resulted bitrate was nearly half as low, as configured. As it can be seen, heightening at least one the configurations: frame rate or resolution, results in GamingAnywhere's inability to maintain the target bitrate and drops it nearly to half.

9 CONCLUSION

In this chapter a brief overview of thesis outcomes, in the form of answering the two main research questions, defined in 2.1, is delivered. The further research suggestions are given in the last subchapter.

9.1 Answers to the Research Questions

1) Does a remote rendering system meet the minimal VR requirements?

In chapters 3 and 4 the survey of the existing remote rendering approaches was implemented and the only platform, which supports hooking into the executable of your custom application and provides the mobile client (although in alpha), was found and appeared to be a research open source project: GamingAnywhere.

An experiment was conducted to figure out, if a remote rendering system, based on GamingAnywhere cloud gaming platform, meets the minimal VR QoE requirements, defined in chapter 5. The results of the experiment have shown that using GamingAnywhere as a remote rendering platform for virtual reality on mobile devices is **untimely yet**, though the results were better, than the author expected in the beginning.

2) What are the main challenges of using remote rendering with mobile VR?

The **incapability** of the tested device, together with the GA client software, **to decode and display** high-resolution video frames even with minimal tolerable frequency **is the key obstacle**, while the network delay appeared to have a much lower impact on the overall performance, as the author supposed before. Nevertheless, there exist no objective reasons, why this obstacle cannot be overcome in the very nearest future, especially considering the fact that the part of the problem lies in the

software codecs, used by GA client application, which is currently on its alpha testing stage.

9.2 Reliability and Validity

The reliability and validity of the current research have been ensured by the usage of a considerable number of reliable sources. The test cases have been designed with the configurations that follow the recognised standards and during the measurement phase, commonly used and reliable tools have been applied. The correlations between the resulted performance benchmarks, done with the use of different software, match the ones, found in scientific articles, common logic and mathematical laws. Moreover, the quantitative data analysis has been implemented with the usage of common methods, based on evaluation of sample mean, sample median and predicted of population mean values, considering both normal and Student's distribution.

9.3 Further Research Suggestions

As mentioned in 2.3, the research has been held with strict limitations on technical resources, therefore conducting similar experiments with more test devices and/or devices with different computing capacity is among the suggestions for further research. In addition to that, a deeper research, including the modification of GamingAnywhere source code, or building a new client application, e.g. in order to add the gyroscope lookaround control and/or to improve the video decoding functionality, can significantly heighten the outcomes' objectiveness in the context of the feasibility to use the remote rendering approach for virtual reality on mobile devices in general, and not affected by the imperfectness of software.

As a separate point, it is possible to mention that the delay measurement technique may be improved further. It is shows objective results in the current paper, but if the values, less than 100ms have to be identified and

evaluated, it will be more effective to make an injection into the client and count the difference between sending the command and displaying the resulted frame, in order to eliminate the measurement uncertainty, caused by a slight difference between the system time of different devices.

Moreover, if further researchers succeed in achieving higher performance, it would be relevant to put a standard deviation limit on the delay and frame rate, e.g. 500ms and 50FPS, because such instability and the possibility of janks and losses is inacceptable even if the population mean is rather high.

10 SUMMARY

To sum up, the objectives of the research has been achieved. The survey of the remote rendering approaches and techniques has been made, resulting in the increased level of understanding the process of rendering, encoding, transferring, receiving and decoding the graphical data, with its pros and cons.

Based on this survey, a cloud gaming platform has been found and tested during the experiment, together with the VR application, developed and built especially for testing purpose. After measuring and analysis, it is clear that on the current stage it is impractical to use real-time remote rendering for mobile VR. Nevertheless, there is plenty of space for further research and improvements and much better results will definitely be achieved in the future, making it possible to use most ubiquitous mobile devices for exploring the virtual worlds of nearly unlimited complexity, size and quality of graphics.

The results of the current research can be used by companies and/or researchers as theoretical base, helping understand the key problems and issues to work on in the future and evaluate the current abilities of certain software and hardware solutions in the fields of video streaming and remote rendering.

REFERENCES

Published sources

Baratto, R. A., Kim, L. N., Nieh, J. 2005. Thinc: a virtual display architecture for thin-client computing. In Proceedings of the twentieth ACM symposium on Operating systems principles. SOSP ’05. ACM, New York, NY, USA, 277–290.

Beermann, D. & Humphreys, G. 2003. Visual computing in the future: Computer graphics as a remote service. University of Virginia, Computer Science Department, University of Virginia Technical Report CS-2003-16 25.

Bhaniramka, P., Robert, P., Eilemann, S. 2005. OpenGL multipipe sdk: a toolkit for scalable parallel rendering. In Visualization, 2005. VIS 05. IEEE. 119–126.

Boukerche, A. & Pazzi, R. W. N. 2006. Remote rendering and streaming of progressive panoramas for mobile devices. In Proceedings of the 14th Annual ACM International Conference on Multimedia (Multimedia’06). ACM, New York, 691–694.

Cedilnik, A., Geveci, B., Moreland, K., Ahrens, J., & Favre, J. 2006. Remote large data visualization in the Paraview framework. In Proceedings of the 6th Eurographics Conference on Parallel Graphics and Visualization. EG PGV’06. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 163–170.

Chen, K., Chang, Y., Tseng, P., Huang, C. & Lei, C. 2011. Measuring the latency of cloud gaming systems. Proceedings of ACM Multimedia 2011. Taipei, Taiwan.

Chen, S. E. 1995. Quicktime VR: An image-based approach to virtual environment navigation. In Proceedings of the 22nd Annual Conference

on Computer Graphics and Interactive Techniques (SIGGRAPH'95). ACM, New York, 29–38.

Cumberland, B. C., Carius, G., Muir, A. 1999. Microsoft windows NT server 4.0 terminal server edition technical reference. Microsoft Press.

Duguet, F. & Drettakis, G. 2004. Flexible point-based rendering on mobile devices. IEEE Trans. Comput. Graph. Appl.

Eilemann, S., Makhinya, M., Pajarola, R. 2009. Equalizer: A scalable parallel rendering framework. IEEE Transactions on Visualization and Computer Graphics 15, 3, 436–452.

Huang, C., Chen, D., Hsu, C. & Chen, K. 2013. GamingAnywhere: An Open-Source Cloud Gaming Testbed. Proceedings of ACM Multimedia 2013 (Open Source Software Competition Track).

Huang, C., Hsu, C., Chang, Y. & Chen, K. 2014. GamingAnywhere: The First Open Source Cloud Gaming System. ACM Transactions on Multimedia Computing, Communications and Applications, Vol 10, No 1s.

Huang, C., Hsu, C., Chen, D. & Chen, K. 2014. Quantifying user satisfaction in mobile cloud games. Proceedings of ACM MoVid 2014. Taipei, Taiwan.

Humphreys, G., Eldridge, M., Buck, I., Stoll, G., Everett, M., Hanrahan, P. 2001. WireGL: a scalable graphics system for clusters. In International Conference on Computer Graphics and Interactive Techniques: Proceedings of the 28th annual conference on Computer graphics and interactive techniques. Vol. 2001. 129–140.

Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., Klosowski, J. T. 2002. Chromium: A stream-processing framework for interactive rendering on clusters. ACM Trans. Graph. 21, 3, 693–702.

Kipper, G. & Rampolla, J. 2013. Augmented Reality: An Emerging Technologies Guide to AR. Elsevier, Inc. Waltham, MA.

Lamberti, F. & Sanna, A. 2007. A streaming-based solution for remote visualization of 3D graphics on mobile devices. IEEE Trans. Vis. Comput. Graph. 13, 2, 247–260.

Levoy, M. 1995. Polygon-assisted jpeg and mpeg compression of synthetic images. In Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH ’95). ACM, New York, 21–28.

Noimark, Y. & Cohen-Or, D. 2003. Streaming scenes to mpeg-4 video-enabled devices. IEEE Comput. Graph. Appl. 23, 58–64.

Ohazama, C. 1999. OpenGL VizServer. White paper, Silicon Graphics, Inc.

Richardson, T., Stafford-Fraser, Q., Wood, K. R., & Hopper, A. 1998. Virtual network computing. IEEE Internet Computing 2, 33–38.

Ross, P. 2009. Cloud computing’s killer app: Gaming. IEEE Spectrum 46, 3, 14.

Schmidt, B. K., Lam, M. S., Northcutt, J. D. 1999. The interactive performance of slim: a stateless, thin-client architecture. In Proceedings of the seventeenth ACM symposium on Operating systems principles. SOSP ’99. ACM, New York, NY, USA, 32–47.

Shi, S. & Hsu, C. 2015. A Survey of Interactive Remote Rendering Systems. ACM Computing Surveys 47(4):1-29.

Shi, S., Jeon, W., Nahrstedt, K. & Campbell, R. 2009. Real-time remote rendering of 3D video for mobile devices. Urbana, IL: University of Illinois at Urbana-Champaign.

Shi, S., Nahrstedt, K. & Campbell, R. 2012. A Real-Time Remote Rendering System for Interactive Mobile Graphics. Urbana, IL: University of Illinois at Urbana-Champaign.

Staadt, O. G., Walker, J., Nuber, C., Hamann, B. 2003. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In Proceedings of the workshop on Virtual environments 2003. ACM, 261–270.

U. Lampe, R. Hans & R. Steinmetz. 2013. Will mobile cloud gaming work? Findings on latency, energy, and cost. In Proc. of IEEE International Conference on Mobile Services (MS'13), pages 960-961, Santa Clara, CA.

Zhu, M. & Shen, K. 2016. Energy Discounted Computing on Multicore Smartphones. Proceedings of 2016 USENIX Annual Technical Conference. Denver, CO.

Electronic sources

Google VR. 2016. Developer Overview. [referenced on 29.11.2016]. Available on https://developers.google.com/vr/cardboard/overview.

Hall, C. 2016. Sony to devs: If you drop below 60 fps in VR we will not certify your game. Polygon. [referenced on 25.11.2016]. Available on http://www.polygon.com/2016/3/17/11256142/sony-framerate-60fps-vr-certification.

Kolasinski, E. M. 2014. Simulator sickness in virtual environments (ARI 1027). U.S. Army Research Institute for the Behavioral and Social Sciences. [referenced on 29.11.16]. Available on http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA295861.

LinusTechTips. 2014. 30FPS vs High FPS discussion with Palmer Luckey. YouTube. [referenced on 25.11.2016]. Available on https://youtu.be/BQoPvZGjYvQ.

Microsoft Office Support. CONFIDENCE Function. [referenced on 29.11.2016]. https://support.office.com/en-us/article/CONFIDENCE-function-75ccc007-f77c-4343-bc14-673642091ad6.

Microsoft Office Support. STDEV.S Function. [referenced on 29.11.2016]. Available on https://support.office.com/en-us/article/STDEV-S-function-7d69cf97-0c1f-4acf-be27-f3e83904cc23?ui=en-US&rs=en-US&ad=US.

Oculus Rift Blog. 2014. John Carmack's Delivers Some Home Truths On Latency. [referenced on 25.11.2016]. Available on http://oculusrift-blog.com/john-carmacks-message-of-latency/682/.

Purcell, P. 2016. Mobile game flips between running at 30FPS and 60FPS. Unity3D Support. [referenced on 25.11.2016]. Available on https://support.unity3d.com/hc/en-us/articles/205824295-Mobile-game-flips-between-running-at-30FPS-and-60FPS.

YouTube. Live encoder settings, bitrates, and resolutions. [referenced on 25.11.2016]. Available on https://support.google.com/youtube/answer/2853702?hl=en.

Applications and tools

ClockSync. https://play.google.com/store/apps/details?id=ru.org.amip.ClockSync&hl=en.

GameBench. Official website: https://www.gamebench.net/, GameBench on Google Play Store: https://play.google.com/store/apps/details?id=com.gamebench.metricscollector&hl=en.