UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Wellingston Cataldo Roberti Junior

U-DiVE: Design and Evaluation of a Distributed Photorealistic Virtual Reality Environment

Juiz de Fora
2021

Wellingston Cataldo Roberti Junior

U-DiVE: Design and Evaluation of a Distributed Photorealistic Virtual Reality Environment

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Dr. Marcelo Ferreira Moreno

Coorientador: Dr. Rodrigo Luis de Souza da Silva

Juiz de Fora

2021

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Cataldo Roberti Junior, Wellingston.
U-DiVE : Design and Evaluation of a Distributed Photorealistic Virtual Reality Environment / Wellingston Cataldo Roberti Junior. – 2021.
51 f. : il.

Orientador: Marcelo Ferreira Moreno
Coorientador: Rodrigo Luis de Souza da Silva
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação, 2021.

1. Photorealism. 2. Virtual reality. 3. Streaming. I. Ferreira Moreno, Marcelo, orient. II. Luis de Souza da Silva, Rodrigo III. Título.

Wellingston Cataldo Roberti Junior

U-DiVE: Design and Evaluation of a Distributed Photorealistic Virtual Reality Environment

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do título de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação.

Aprovada em 21 de janeiro de 2022.

BANCA EXAMINADORA

Prof. Dr. Marcelo Ferreira Moreno - Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Rodrigo Luís de Souza da Silva - Coorientador
Universidade Federal de Juiz de Fora

Prof. Dr. Eduardo Barrére
Universidade Federal de Juiz de Fora

Prof. Dr. Alex Fernandes da Veiga Machado
Instituto Federal de Educação, Ciência e Tecnologia do Sudeste de Minas Gerais

Juiz de Fora, 17/01/2022.

Documento assinado eletronicamente por **Marcelo Ferreira Moreno**, **Professor(a)**, em 21/01/2022, às 16:16, conforme horário oficial de Brasília, com fundamento no § 3º

do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Rodrigo Luis de Souza da Silva**, **Professor(a)**, em 24/01/2022, às 13:50, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Alex Fernandes da Veiga Machado**, **Usuário Externo**, em 25/01/2022, às 14:27, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Eduardo Barrere**, **Professor(a)**, em 26/01/2022, às 09:23, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

A autenticidade deste documento pode ser conferida no Portal do SEI-Ufjf (www2.ufjf.br/SEI) através do ícone Conferência de Documentos, informando o código verificador **0647530** e o código CRC **F3FB91C7**.

Dedico este trabalho a quem colaborou diretamente comigo: meus orientadores e professores, sem os quais eu não teria concluído este projeto.

AGRADECIMENTOS

Agradeço aos meus orientadores, Marcelo e Rodrigo, pela paciência e ensinamentos durante todo o processo do mestrado.

Agradeço a minha família pelo sustento e amigos pelos tempos de descontração.

Agradeço a minha namorada que me ajudou a manter confiante e persistente durante todo o curso.

"Remember the two benefits of failure. First, if you do fail, you learn what doesn't work; and second, the failure gives you the opportunity to try a new approach." (Roger Von Oech)

RESUMO

Esta dissertação apresenta um *framework* que permite que dispositivos de baixo custo visualizem e interajam com cenas fotorrealísticas. Para realizar essa tarefa, o *framework* faz uso do pipeline de renderização de alta definição do Unity, que tem um algoritmo de rastreamento de raio proprietário, e o pacote de *streaming* do Unity, que permite o streaming de um aplicativo em seu editor. O *framework* permite a composição de uma cena realista usando um algoritmo de *Ray Tracing*, e uma câmera de realidade virtual com *shaders* de barril, para corrigir a distorção da lente necessária para usar um cardboard de baixo custo. Inclui também um método para coletar a orientação espacial do dispositivo móvel por meio de um navegador Web para controlar a visão do usuário, entregue via WebRTC. O *framework* proposto pode produzir ambientes de baixa latência, realistas e imersivos para serem acessados por meio de HMDs e dispositivos móveis de baixo custo. Para avaliar a estrutura, este trabalho considera a verificação da taxa de quadros alcançada pelo servidor e pelo dispositivo móvel, que deve ser superior a 30 FPS para uma experiência fluida. Além disso, discute se a qualidade geral da experiência é aceitável, ao avaliar o atraso da entrega das imagens desde o servidor até o dispositivo móvel, em face da movimentação do usuário. Nossos testes mostraram que o framework atinge uma latência média em torno dos 177 (ms) com equipamentos wi-fi de uso doméstico e uma variação máxima das latências igual a 77.9 (ms), entre as 8 cenas testadas.

Palavras-chave: Fotorealismo. Realidade virtual. Ray Tracing. Streaming. Dispositivos de baixo custo.

ABSTRACT

This dissertation presents a framework that allows low-cost devices to visualize and interact with photorealistic scenes. To accomplish this task, the framework makes use of Unity's high-definition rendering pipeline, which has a proprietary Ray Tracing algorithm, and Unity's streaming package, which allows an application to be streamed within its editor. The framework allows the composition of a realistic scene using a Ray Tracing algorithm, and a virtual reality camera with barrel shaders, to correct the lens distortion needed for the use on an inexpensive cardboard. It also includes a method to collect the mobile device's spatial orientation through a web browser to control the user's view, delivered via WebRTC. The proposed framework can produce low-latency, realistic and immersive environments to be accessed through low-cost HMDs and mobile devices. To evaluate the structure, this work includes the verification of the frame rate achieved by the server and mobile device, which should be higher than 30 FPS for a smooth experience. In addition, it discusses whether the overall quality of experience is acceptable by evaluating the delay of image delivery from the server up to the mobile device, in face of user's movement. Our tests showed that the framework reaches a mean latency around 177 (ms) with household Wi-Fi equipment and a maximum latency variation of 77.9 (ms), among the 8 scenes tested.

Keywords: Photorealism. Virtual reality. Ray tracing. Streaming. Low-cost devices.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AR	Augmented Reality
HMD	Head-mounted Display
FPS	Frames Per Second
CE	Consumer Electronics
QoE	Quality of Experience
IPTV	Internet Protocol Television
P2P	Peer to peer
RTP	Real-Time Transport Protocol
UDP	User Datagram Protocol
CDN	Content Delivery Network
HTTP	Hypertext Transfer Protocol
NAT	Network Address Translator
SDP	Session Description Protocol
ICE	Interactive Connectivity Establishment
HDRP	High Definition Render Pipeline
DXR	DirectX Ray Tracing
VR	Virtual Reality

SUMÁRIO

1 INTRODUCTION

Virtual reality is a branch of computer graphics defined by an experience in which a user is effectively immersed in a responsive virtual world (13). Virtual reality applications need to be stable so that the user does not lose the feeling of being in a virtual world. Four technologies are crucial for immersive virtual reality: an immersive display, rendering at least 20 frames per second (FPS), a tracking system that continuously reports the user's position and orientation, and a realistic environment (13).

There are several virtual reality displays on the market; these are known as HMD (Head-mounted display). Some technologies embedded in expensive HMDs, responsible for presenting and interacting with the virtual world, may be replaced by smartphones and take advantage of their sensors, creating an affordable option.

The main sensors used on smartphones to represent and interact with virtual worlds are accelerometer and gyroscope. The accelerometer is capable of measuring the acceleration of a body in relation to gravity, in other words, it is capable of measuring the acceleration exerted on an object in which it is embedded. The gyroscope does not have a different functionality, it is a sensor that uses the force of gravity to indicate the position of an object in space, which can identify whether the object rotates on its own axis or whether it is pointing towards up or down.

However, mobile devices still have a limited graphical processing capacity, which means that many applications cannot be rendered with a stable temporal resolution (frame rate) (3). For example, real-time photorealistic VR rendering (3) is currently unfeasible to be processed in mobile devices.

Photorealism is a realistic representation of computer images that, in addition to geometry, accurately simulate the physics of materials and light (14). These produced images may be indistinguishable from a photograph taken from the real world. To achieve photorealism, it is necessary to use some methods to render the scene like Ray Tracing and Path Tracing. These methods perform calculations based on camera position, object position, and light source, which may be direct, bounced, reflected and refracted. These methods use similar techniques, by shooting a ray from the camera's position, passing through the pixel mesh and hitting an object. When reaching a point on the object, a color calculation is performed considering the strength of the light exerted.

Streaming techniques can be combined with virtual reality to create a stable and immersive photorealistic environment for low-cost devices. Streaming is a form of continuous media delivery, commonly supported by well-known standardized protocols. Instead of a complete and previous transfer of the whole media content to the client application, streaming techniques usually deliver the content continuously, while it is being presented to the end user. Streaming may be even a requirement for some applications,

due to the amount of media data to be transferred or the real-time characteristics of the media itself (15). In this work we separate virtual reality processing and rendering from its visualization and interaction, using real-time streaming with interactive control of the virtual reality camera.

Unity is a game engine that supports 3D scene streaming, photorealism based on methods such as Ray Tracing and Path Tracing, and virtual reality environment. Unity provides a package, called Unity Render Streaming, still in alpha state, capable of transmitting, in real time, the image of a camera from inside the engine to a web browser. This package is built around Unity's new rendering pipeline. The new pipeline, called HDRP, has built-in Ray Tracing and Path Tracing algorithms with configurable parameters. Although the mobile virtual reality package no longer has support for the latest versions, it still has the features so that a virtual reality camera and its features can be built. With these tools it is possible to create a photorealistic scene that can be presented on a low-cost device and that makes Unity a great choice to use in this work.

1.1 PROBLEM DEFINITION

Photorealism algorithms were introduced in the previous century (16), but only in the last few years and with video cards evolution it was possible to create and execute complex and detailed scenes. To mobile devices, photorealism is still too far to execute with the required quality and fluidity.

Besides the lack of graphics processing power of mobile devices in general, another problem is the cost for building a computer with enough power to execute photorealism. If a powerful computer is on the server side, which can belong to a third party, we can reduce the cost to the client side.

To make photorealism applications more popular, while providing the desirable quality of experience, a solution may gather the best features of computers and mobile devices together. By decoupling the intensive processing tasks of photorealism from the scene presentation environment, a client-server approach becomes viable and capable of bringing more users with their affordable mobile devices.

Our work discusses the possibility of creating a framework capable of executing photorealistic virtual reality environments involving low-cost devices. To build this framework a streaming-based system can be used as an intermediary technology between a server with powerful hardware and a client with a low-cost mobile device. Thus, client and server may exchange information such as user orientation and video rendered scenes.

1.2 OBJECTIVES

The main objective of this work is to propose a framework, which allows low-cost devices to present complex scenes with a high level of realism, and a quantitative analysis to assess its latency. In this framework, we take advantage of streaming to make all the expensive processing on the server-side, and the mobile device executes the video stream player and sends the user's head orientation back to the server (Figure 1).

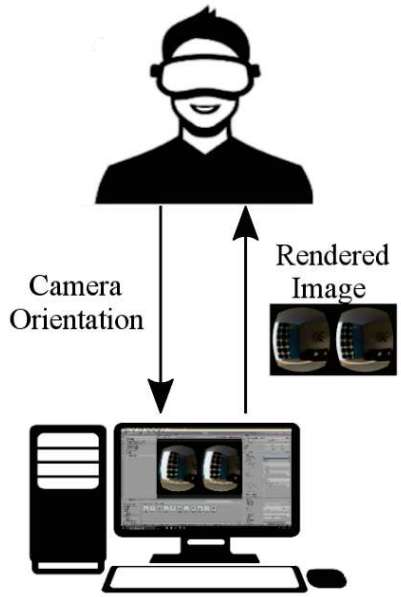


Figure 1 - Framework overview.

As a secondary objective, we intend to evaluate the framework. We divided the evaluation tests into two points of analysis: (i) checking if the frame rate reached by both server and mobile device is greater than 30 FPS for a fluid experience, and (ii) check the delay of images and user movement between the server and the mobile device.

1.3 OUTLINE

The reminder of this work is structured as follows. In Chapter 2, we present related work. Chapter 3 describes a theoretical background concerning photorealism and streaming. Chapter 4 presents the development of the framework. In Chapter 5, we show the analysis of the results, and discussions are detailed. Finally, Chapter 6 presents the conclusions and future work.

2 RELATED WORK

In this work, we present a novel framework that combines three main aspects: virtual reality content consumption through low-cost mobile devices, immersive media streaming from a server to the mobile device, and photorealistic VR rendering in real-time. This section describes some relevant works concerning these aspects in recent years.

Virtual Reality has become more accessible since the development of mobile-based Virtual Reality headsets similar to Google Cardboard (17). Despite their hardware limitations regarding frame rate and resolution, Papachristos et al. showed in (18) that these headsets can deliver a satisfactory immersive experience with a lower cost compared with traditional head-mounted displays.

Yoo and Kay presented in (1) an example of VR application for smartphones, the exergame VRun, as shown in Figure 2. They developed the game in Unity 5 game engine with the Google Cardboard SDK and tested it on the HTC One smartphone with low-cost cardboard. In the game, the player runs from a start zone to a finish line dodging from obstacles that are thrown in his way, like fire arrows and blobs. The player has a first-person view of the virtual world. To walk or run inside it, the player walks or runs in place in the real world. Through data capture of the smartphone's accelerometer sensor, the game detects walk-in-place and maps this to walk in the virtual world. The graphics are simple, in a cartoon style, without photorealistic effects.

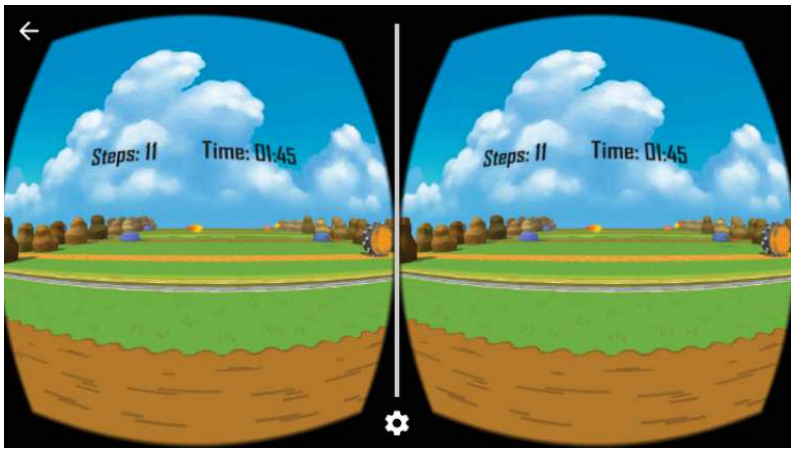


Figure 2 - Scene of the VRun App (1)

In (19), He et al. present the Rubiks, a framework for streaming 360° videos for smartphones that is suitable for streaming videos or VR content. It consists of a server running on a laptop and an Android app on a smartphone. The app captures the orientation of the smartphone from its sensors and sends it to the server. The server splits the video spatially into tiles and temporally into layers. Consequently, it is possible to encode and send video portions with a high probability of viewing, in high quality, while encoding and sending the remaining with low quality. The server uses orientation

data from the smartphone to calculate the field of view and determine the portion with a higher probability of viewing. In their proposal, it is necessary to decode the video on the smartphone. However, the authors encourage the streaming of all tiles to avoid missing parts on the screen, resulting in orientation estimating errors and high network bandwidth consumption.

One of the main techniques to achieve photorealism in computer graphics is the Ray Tracing algorithm. However, it has a high computational cost. In recent years, hardware evolution has enabled its use on VR applications. Lee et al. present in (2) a stereoscopic rendering technique based on mobile Ray Tracing GPU. Their proposal focuses on Samsung GPU based on Ray Tracing (SGRT), and its main contribution is the use of reprojection and tile-based Ray Tracing. Reprojection exploits the coherence between the left and right images, so it uses the ray-objects intersection from the left image to avoid a full Ray Tracing on the right image. The tile-based Ray Tracing allows storing the entire G-buffer into the on-chip memory, considerably reducing the number of accesses to the external DRAM during reprojection and reusing computations. As a result, their solution improves performance and energy efficiency, though being proposed for specific mobile hardware. Their stereoscopic rendering results are shown in Figure 3.

Figure 3 - Lee et al's (2) Stereoscopic rendering with test scenes: Teapot (15K triangles), Chess (42K), BMW (55K), Chemical Lab. (98K), Music box (106K), and Provence (600K). Yellow pixels indicate bad pixels. (2)

Authors in (3) propose an architecture that considers Ray Tracing algorithms being executed on future mobile devices. However, no real mobile device was used in their experiments. Although they achieved interesting results, they did not use sophisticated algorithms for their tests, showing that mobile devices still do not support the latest algorithms of Ray Tracing with stable frame rate. The tested scenes are shown in Figure 4.

The idea of remote computing to enable photorealism on mobile devices is not new. Godoy and Teixeira (4) proposes an architecture to promote the use of mobile devices, named as CE (Consumer Electronics), in interactions with remotely synthesized media.

Figure 4 - Lee et al's tested scenes for real-time ray tracing (3)

They build a proof of concept with two types of media, a virtual architectonic model as a more latency tolerant, and a game demo based on the Unreal Engine SDK as less latency tolerant, illustrated in Figure 5. They did not present details about rendering techniques and algorithms involved, neither its computational cost. The focus was more on transmission issues. In their work, it was possible to conclude that latency did not significantly spoil the interaction with neither of the media types.

Figure 5 - Scenes used in Godoy and Teixeira's experiments (4)

The authors in (20) presented methods for handling multiple data streams with different latency values associated with each other in a working Augmented Reality (AR) system. These methods are applied to an AR system for real-time ultrasound visualization and demonstrate improved registration and visualization. The first method is a technique to reduce relative latency, adjusting the moment of sampling the incoming data stream. A compromise that does not increase maximum latency to decrease relative latency is the just-in-time acquisition of the data while interleaving computation with the data acquisition. The second method is storing multiple readings and either interpolating or extrapolating these readings to simulate new readings. The work measured the latency differences, time-stamp on-host, adjust the moment of sampling, and interpolate or extrapolate data streams. The visual tests show that the real-world camera video is the lowest latency stream. For the external device trackers, it is determined relative latency by rendering a model in the tracker's coordinate system that should be aligned with the real-world object. The authors also measured the latency of the camera to the real world. To do this, a LED blinking at a rate of 5Hz is used as a trigger for an oscilloscope.

MmWave communication, edge computing, and proactive caching can be a solution to achieve an interconnected VR/AR determined by smooth and reliable service, minimal latency, and seamless support of different network deployments and application requirements. In (5) to generate a real-like view, a bit rate of up to 1 Gb/s is required, and this value is not reachable in 4G. To bring end-to-end latency down, it is needed to understand the various types of delays involved in calculating the pooled computing and communication latency budget. Delay contributions to the end-to-end wireless/mobile VR latency include sensor sampling delay, image processing or frame rendering computing delay, network delay, and display refresh delay. Also, lag spikes and dropouts needed to be kept to a minimum, or users will feel detached. This work proposes an optimization framework to maximize the successful high definition frame delivery subject to reliability and latency constraints. This case study demonstrates the performance gains and the underlying trade-offs inherent to wireless VR networks. Figure 6 shows the architecture of an interactive VR gaming arcade with mmWave APs and edge computing network.

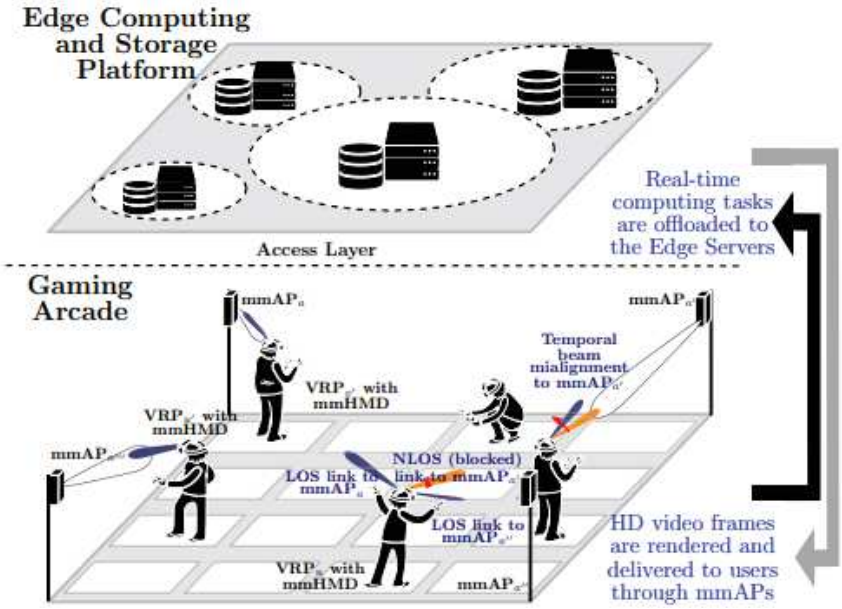


Figure 6 - Interactive VR gaming arcade with mmWave APs and edge computing network architecture (5)

Authors in (6) perform a rigorous analysis of 1300 VR head traces and propose a multicast DASH-based tiled streaming solution for mobile multicast environments. This paper weighs video tiles based on user's viewports, divides users into subgroups based on their channel conditions and tile weights, and determines each tile's bitrate in each subgroup. They compare the proposed solution against the literature's closest ones using simulated LTE networks and show that it substantially outperforms them. Three performance metrics are used to show their results: 1) average viewport bitrate, 2) the impact of viewport change during the scheduling window, and 3) spectral efficiency. The results show that it assigns 46% higher video bitrates for the video tiles, allowing them to

freely change their view directions while observing much less video quality degradation. The architecture of the work is showed in Figure 7.

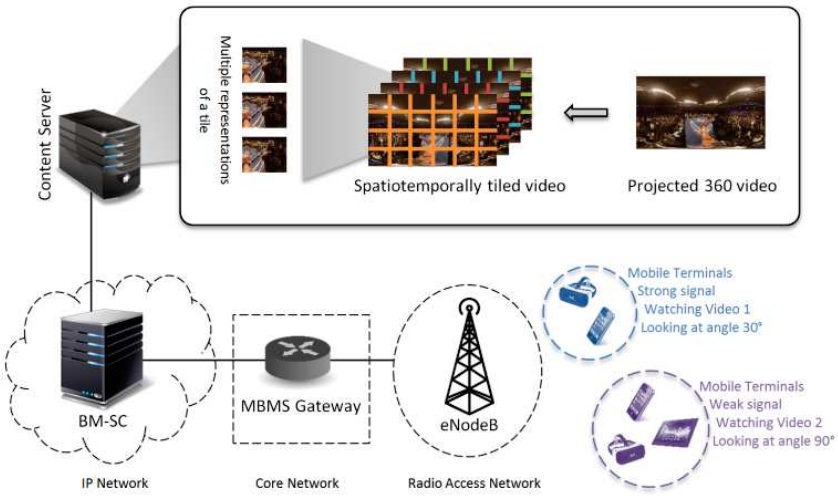


Figure 7 - Ahmadi et al's architecture of a mobile network for a VR tiled multicast streaming (6)

The next work covers different aspects of VR content representation, streaming, and quality assessment that will help establish the basic knowledge of building a VR streaming system. To (21), the VR streaming problem can be described as "the problem of panning around a high resolution-video using head movements". This study advanced over most recent studies dedicated to streaming VR content and present multiple models to assess the QoE (Quality of Experience) in a VR streaming system. The focus is on 360 videos, and there has not been much attention in the literature on evaluating the quality of such content. It is not apparent how to compare different projections of 360 videos at different bitrates with the original video.

In (7), the required throughput of VR applications is computed for an eye-like experience and a case study of running different VR applications is presented on an open source remote VR display characterizing their traffic statistics. This study calculates the number of bits per second to represent an image in the Oculus Quest display. In the case study, VR traffic between client and server is collected by periodically synchronizing the clock and rendering the frame on the edge server fragmenting into small MAC layer packet data units, traverse over the networks, and reassembling at the HMD, Figure 8. The video coding used in this paper is H.264 and H.265, and the performance is compared. This work shows that rendering VR applications can be offloaded to an edge server, reducing the energy consumption and production cost of the VR HMD.

In (8) a web-based framework for the creation and evaluation of social and shared VR experiences is presented. They elaborate on three multi-user VR cases: watching TV in VR, social collaboration in VR, and social VR conferencing in mixed reality. Node.js/Express, Angular.js, and A-Frame are the development base of the framework.

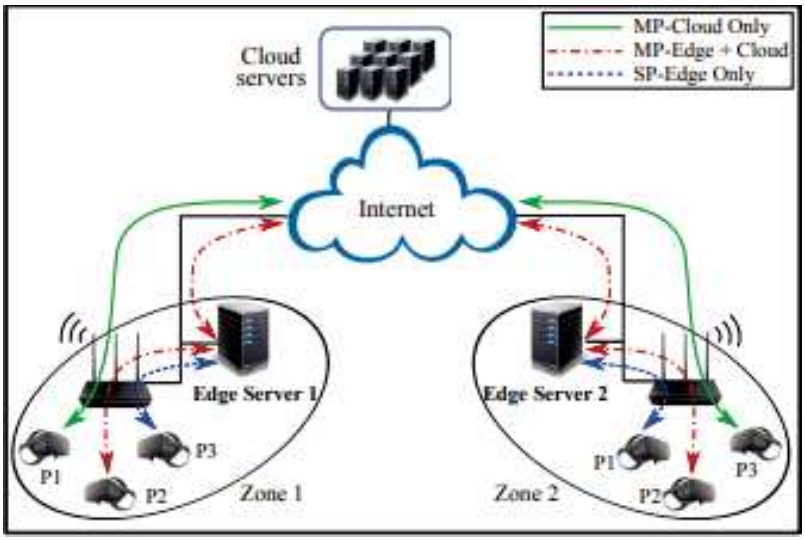


Figure 8 - VR communications with edge and cloud servers as proposed by Salehi et al. (7)

A-Frame is a web framework for building 3D/AR/VR experiences that allows participants to be placed in a photo-realistic 360-degree panoramic room. TogetherVR, as the authors named the framework, consists of a frontend component, which runs in the web browser, and a backend component, which hosts the VR applications, and facilitates inter-user communication and media playback control. The authors manage to create three scenarios: Social VR TV experience, which allows two VR participants to watch TV together remotely; Interactive Social VR experience, which supports shared interaction with objects in the VR space next to conversing with each other, Figure 9; Social VR conferencing in a mixed reality setting, which explore to what extent a VR can be used to bring a remote user into a meeting room. They collected feedback from the participants through a short questionnaire evaluating the quality of the experience and the sense of presence. On a 5-point Likert scale, participants appreciated the overall quality (77% scored four or higher) of the experience. They also felt involved in the virtual environment experience (72% scored four or higher).

Figure 9 - Embedding non-browser applications via WebRTC screen share functionality in TogetherVR (8)

Authors in (9) develop a photorealistic algorithm to run directly on mobile devices. In this paper, they present a novel distributed illumination approach for AR with consistent illumination direct light, indirect light, and shadows of primary and strong secondary lights. They split the illumination into two parts: capture the existing radiance values by HDR video cameras placed at different locations in the scene, and display augmentations with consistent illumination at interactive frame rate on the mobile device, Figure 10. This acquisition process reduces the amount of transferred data between a stationary PC and the participating mobile device. Their goal is to achieve a consistent illumination of virtual objects on mobile devices in a real environment and interaction of the real world with photorealistic augmentations with multiple users.

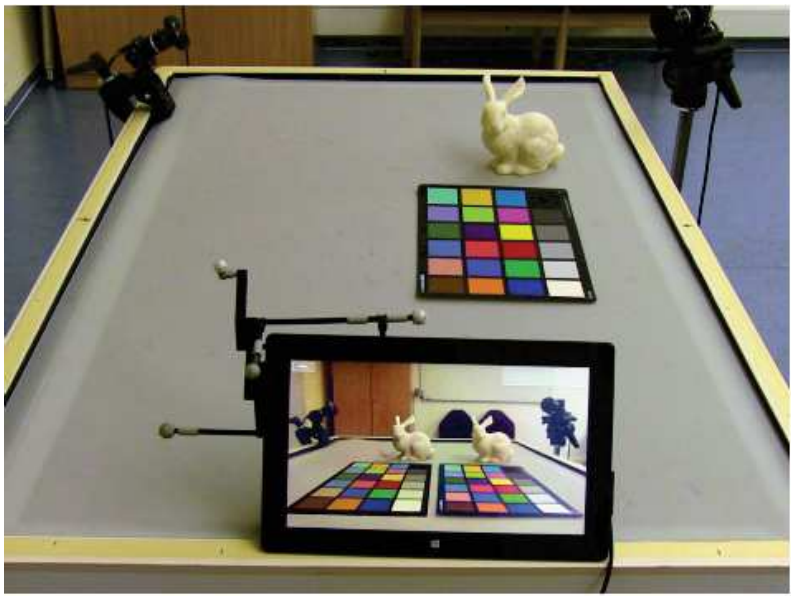


Figure 10 - The tablet with augmented virtual objects with consistent illumination as proposed by Rohmer et al. (9)

Authors in (10) present VRComm, a web-based social VR framework for enabling remote communications via video conferencing, Figure 11. Authors have three main contributions: a new VR communication framework, a novel approach for real-time depth data transmitting as a 2D grayscale for 3D user representation, including a central multi-point control unit approach for this new format, and a technical evaluation of the system with respect to processing delay, CPU and GPU usage. Their evaluation shows that the proposed capture and depth to grayscale conversion is suitable for real-time video transmission but other solutions might result in better visual quality if the bitrate is below 1.5 Mbit or above 3 Mbit, as well as pre-encoded content.

2.1 RELATED FRAMEWORKS

To find an equivalent framework to compare with, we used the main search engines (IEEE Xplore, Scopus, Science Direct, ACM, and Web of Science) and found 378 papers.

Figure 11 - 16 user streams in a VRComm Virtual Experience (left) and RGBD user transmission of RGB-part (middle) and depth-part (right) (10)

After the removal of duplicated and not related works, focusing on papers regarding photorealism and mobile devices as display we end up with 14 papers. In these 14 papers, most discuss augmented reality.

In (11), the main goal is to create highly realistic graphics for Augmented Reality on mobile phones. Their approach is based on Ray Tracing, perform distributed rendering to address the limited mobile GPU capabilities, and use image-based lighting from a pre-captured panorama to incorporate real-world lighting. They use one marker for object tracking and one for registering the panorama. This work still needs to be validated in human subject studies, especially regarding the trade-off between latency of remote rendering and visual quality. The prototypes' results are shown in Figure 12.

Figure 12 - Prototypes' results for global illumination effects for Augmented Reality on mobile phones, proposed by Csongei et al. (11)

GLEAM is a framework that provides robust illumination estimation in real-time by integrating physical light probe estimation with current mobile AR systems (12). The main contribution of this work is a solution for integrating traditional image-based lighting estimation on mobile systems, situation-driven system trade-offs to optimize for specific quality factors, and a user study to evaluate the effect of illumination estimation methods on human perception and situational quality preference. GLEAM can network multiple devices to sense illumination from different viewpoints to enhance realism and fidelity. The difference between ARKit and the result of the GLEAM framework is shown in Figure 13

The work of (22) presents a middleware streaming engine that can implement existing OpenGL-based 3D network games onto heterogeneous platforms. The engine consists

Figure 13 - AR scene illuminated with ARKit (left) and GLEAM (right) illumination estimation, along with reflected environment in light probe (inset) (12)

of capturing OpenGL command stream, scene graph reconstruction, data simplification, and compression and transmission. The development of the system is on WLAN and consists of a game server and game clients on a PC, and clients on heterogeneous devices. As (22) describe the overview, a client on a mobile device is connected to invocation manager on the PC, the manager executes the 3D game server, the game server and client makes a connection, and every mobile client is connected to a game client through the invocation manager. By appending a middleware the system can extend the PC-based 3D games onto mobile platforms without modification of the source code. In their results, 3D streaming is possible in 4 5 frames/second in spite of a dynamic game environment under software rendering.

We did not find a work condensing all elements presented in our proposal. Our work differs from those exactly for the combination of multiple techniques in a more versatile way, allowing us to reach a vast number of hardware configurations while keeping the low cost of the solution. Table 1 shows the comparison of requirements for the works presented in this section and the related works section.

Tabela 1 – Comparison of our requirements with related work.

Work	**Requirements**					
	Photorealism	**Mobile Device**	**Virtual Reality**	**Augmented Reality**	**Low Cost**	**Streaming**
(1), (8)	NO	YES	YES	NO	YES	NO
(19), (10)	NO	YES	YES	NO	YES	YES
(2)	YES	YES	YES	NO	NO	NO
(3)	YES	YES	NO	NO	NO	NO
(4)	YES	YES	NO	NO	YES	NO
(20)	NO	YES	NO	YES	NO	YES
(5)	NO	YES	YES	YES	YES	YES
(6)	NO	YES	YES	NO	NO	YES
(21), (7)	NO	NO	YES	NO	NO	YES
(9), (11), (12)	YES	YES	NO	YES	YES	NO
(22)	NO	YES	NO	NO	YES	YES
This work	YES	YES	YES	NO	YES	YES

3 THEORETICAL FOUNDATION

This section presents a theoretical foundation over concepts applied in our framework.

3.1 PHOTOREALISM

There are a few techniques capable of promoting photorealism in image rendering. Unity has two of the main photorealism algorithms in its rendering pipeline: Ray Tracing, and Path Tracing. In this section, we will present the Ray Tracing technique used in our project and a brief explanation about Path Tracing.

Ray Tracing is a technique for image synthesis by creating a 2D picture of a 3D world (23). The concept behind Ray Tracing algorithm was introduced by Albrecht Dürer in 1525, proposing the creation of an image using a grid (24). In May 1968, Arthur Appel published the first use of ray-casting for visualization (25). Arthur Appel projected light at a 3D computer model and displayed the results on a plotter using a form of tone mapping to create dark and light areas (26). In 1980, Turner Whitted created the Ray Tracing algorithm following the path of a ray beyond the initial surface it hits. After a ray hits an object, it produces three new rays, a reflection, refraction, and shadow, which can also improve realism when cast. It is necessary to know the amount of light of each pixel to contribute to global illumination information. This information is stored in a "tree of rays", starting from the viewer to the first surface encountered, bouncing in other surfaces and light sources (27).

Representing materials from nature as diffuse, specular, reflective, or emissive is a difficult task, as these materials have complex behavior. A material may additionally manifest transparency causing light refraction, absorption, and scattering. These effects can be described with a bi-directional reflectance distribution function (BRDF) that produces reflection and transmission dependent on both incident and outgoing light direction (26).

Determining radiation transfer in thermal analysis problems and stray light analysis, was one of the first uses of Ray Tracing. To solve these problems, the Monte Carlo method for the propagation of light from one surface to another in a physical system was implemented. The Monte Carlo method refers to any calculation that handles the problem as a stochastic process and uses random numbers to generate sample points (28).

As shown in Figure 14, the primary rays are cast from the camera and pass through the pixel until it hits a 3D object or a defined limit. Once the intersecting object is defined for a given ray, additional rays are cast from that point depending on the type of surface. There are two types of surface: reflexive and refractive. If the object is reflexive, the rays continue their path in the reflective direction, and if the object is refractive, it maintains its way in the refracted direction (27). From each hit point, shadow rays are cast toward

the light source, and if these rays do not reach the light source, it creates shadows at that point.

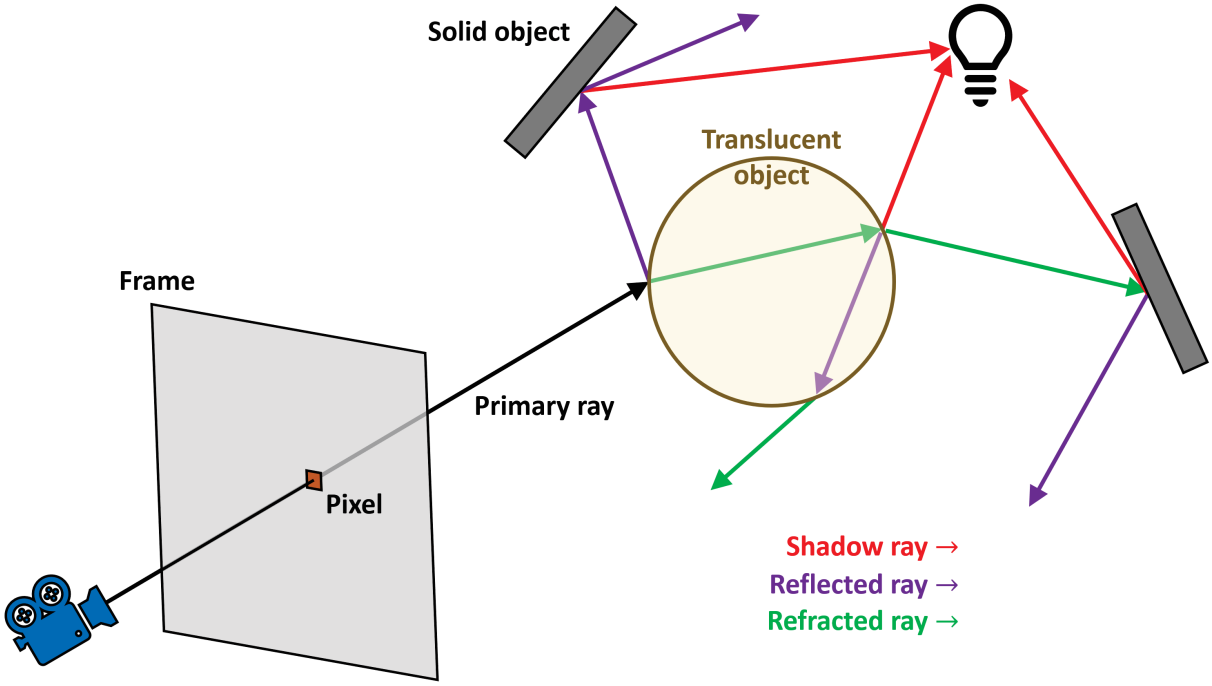


Figure 14 - Visual representation of Ray Tracing[1]

Rays are not only white, but they can also take on many colors, and their colors influence the color of a pixel. To define the color of a pixel, we also need to know the ray color that hits the pixel. Colors from multiple light rays can mix and generate a new color. For example, if a green and red ray intersects themselves at the same point, the colors combine and form a more yellowish pixel. The more the light travels through the environment, the weaker the strength of its color when it hits a surface (23).

Path Tracing was introduced as an algorithm to find a numerical solution to the integral of the rendering equation in (29). In the real world, objects and surfaces are visible because they are reflecting light and this reflected light illuminates other objects around. In Path Tracing, all objects must contribute illumination to every other object. Path Tracing simulates various effects that have to be additionally created in the Ray Tracing algorithm, such as soft shadows, depth of field, motion blur, caustics, ambient occlusion, and indirect lighting.

3.2 MEDIA STREAMING

Internet audio/video delivery may be implemented over push- or pull-mode protocols, some of them specifically designed for media streaming. Historically, many applications such as IPTV (Internet Protocol Television) and VoIP (Voice over Internet Protocol)

[1] `https://cutt.ly/ap8WIvD`; Online, accessed in 28/06/2020

have used protocols such as RTP (Real-Time Transport Protocol) (30), which is primarily a push-mode application-level protocol over UDP (User Datagram Protocol) (31). An RTP packet transfers a given media sample (some milliseconds of audio and/or video) accompanied with relevant information like the media type and a timestamp for an easier decoding process.

However, due to UDP traffic's difficulties on traversing firewalls and with the rapid growth of CDNs (content delivery networks) for Web applications, the robust HTTP (Hypertext Transfer Protocol) has been used in an adapted way for most media streaming traffic on the Internet. HTTP-based schemes like HLS (HTTP Live Streaming) (31) and MPEG-DASH (Dynamic Adaptive Streaming over HTTP) (32) allow for adapting the media delivery bit rate according to the always-changing congestion conditions of the best-effort Internet. Such schemes do not require modifications on the HTTP servers. They operate in pull mode and thus require the client to continuously request the appropriate media segments, one by one, to the server, while monitoring its buffer status for possible bit rate adaptations. Each media segment is delivered as a downloaded file.

In the photorealistic virtual reality environment for low-cost mobile devices, we have the need for using the available APIs and possibly converge to a web-based solution. However, HTTP streaming introduces a noticeable latency (to generate a downloadable file with a media segment in real time) and client processing (to request each media segment) that would result in a poor interactive experience. If we reduce the segment size, the number of HTTP requests from the client increases, together with upstream traffic and power consumption. If we want to reduce the number of HTTP requests, the media segments must be increased, resulting in longer latency.

In our tests we employed WebRTC (Web Real-Time Communication) API (33). This API allows us to use RTP, with its much lower overhead in terms of latency, processing and traffic, in a local setup or over the Internet.

With WebRTC, one can enable real-time communication capabilities combining open standards, protocols, and JavaScript APIs (34). This technology is available on all modern browsers, like Chrome, Safari, Opera, Edge, and their mobile versions. The functionality of WebRTC enables a peer to peer (P2P), browser to browser communication. However, it may need client-server communication to exchange metadata to coordinate communication and to cope with network address translators (NATs) and firewalls. For two endpoints to start talking to each other, some information exchange must be done:

- Session control information used to initialize, close, modify communications, and report error messages;
- Media metadata in common between callers such as codecs and codec settings, bandwidth, and media types;

- Network data, such as a host's IP address and port.

Interestingly, the Unity Engine Version 2019.2 introduced experimental support for WebRTC. It remains experimental in its current 2020.2 Alpha version.

3.3 LATENCY MEASUREMENT

Latency in interactive real-time graphics simulations comes from various sources. The authors in (35) present these sources in:

- sensor reading and computation
- sensor data communication
- application computation
- rendering computation
- display refresh

In streaming, other latency sources are generated, such as package delivery and distance between computers on the network.

Measuring the latency between the movement of an object in the real world and the correspondent action in a virtual world may be challenging. (36) proposes latency estimation with a regular video camera, and the estimation is automatic once the video is captured. The method uses a tracked pendulum and a small light attached to it and then record the pendulum and a screen behind it which shows a simulated image whose position is driven by the tracking information.

The authors in (37) deduce the latency by recording tracking data via a video image at 60Hz. A tracker is attached to a moving pendulum, and when the pendulum passes the vertical axis, they can count frames. This method requires reconfiguration of the tracker space, which is impractical in some situations. The authors in (38) manage to determine the frame offset of a motion automatically using a motion detection algorithm, but the latency is detected in multiples of the frame rate.

In (39) different methods were used, such as Sine-Fitting Method and Di Luca's, to measure the latency of several interactive systems that may be of interest to the virtual environments engineer, with a significant level of confidence. They develop a new latency measurement technique called Automated Frame Counting to assess latency using high-speed video. This technique uses image processing techniques to extract the tracked object's position, resulting in a set of samples that characterizes the motion of the object. The algorithm guides selecting the threshold for binarising the frames and identifying salient object's locations to track. Once tracking is complete, the user selects the feature

scale, then extracts the features and subtracts the locations providing the number of frames. The average of these frames is returned as the latency estimated for that capture.

Like other authors, (40) describes an end-to-end latency measurement method for virtual environments. In this method, a video camera records a physical controller and the corresponding virtual cursor simultaneously and analyzes the playback to determine the lag between wand motion and the motion of the virtual image of the wand.

4 U-DiVE FRAMEWORK

Unity already has Ray tracing technology and a package to develop streaming applications, so this engine is the best choice for expediting the instantiation of U-DiVE framework. The first version of U-DiVE framework was developed using the Unity Engine 2019.3.12f version with the HDRP (High Definition Render Pipeline) that contains the Ray Tracing algorithm. Based on DXR (DirectX Ray Tracing), this Ray Tracing algorithm is a feature of Microsoft's DirectX capable of producing real-time scenes with Ray Tracing. In short, U-DiVE's main task is to stream the output color buffer to the mobile device and present it to the user, while capturing orientation data from the user's mobile device and sending it back to the server to control the camera. U-DiVE sets up one virtual reality camera (composed of two cameras, one for each eye) and a scene, in which we apply the Ray Tracing algorithm. The rendered images are then captured and distortion shaders are applied. These two images are merged to form a single stereo image, which is streamed to the mobile device to be displayed to the user. Finally, U-DiVE extracts the device orientation and sends it back to the Unity's camera, forming a loop shown in Figure 15.

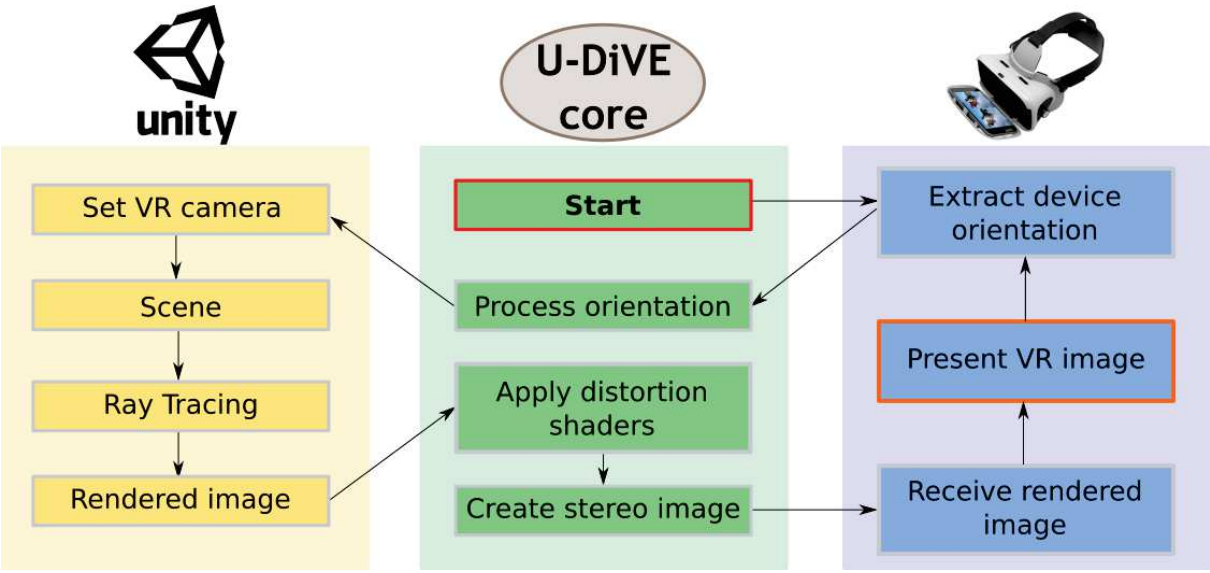


Figure 15 - U-DiVE's pipeline components.

U-DiVE's pipeline components are described as follows:

Start - To start U-DiVE, firstly a Node.js server is initiated. Unity is configured to connect to the address provided by the server. From that server, Unity is started. Then the user may open the web browser on his mobile device and type the same server address to be accessed. Thus, the browser connects to Unity via the Node.js scripts and the exchange of information is now enabled to occur.

Extract device orientation - After U-DiVE starting process, the next step comes from the mobile web browser. The client-side script delivered to the browser periodically

extracts the mobile device's orientation in a quaternion format [1]. The relative orientation is extracted at a frequency of 60 times per second and referenced to the screen frame. Each quaternion value occupies 4 bytes of memory and an input event flag occupies 1 byte totaling a 17-byte buffer array. This buffer array is sent to the server to be processed internally by Unity.

Process orientation - The internal processing of the orientation quaternion depends on the flag that was sent in the first position of the buffer array. The signal can take the value of a click, a key press, device sensors, etc. Unity recognizes the data type being received from this information. Values are stored in Unity by the input system in global variables and can be used in any object in the environment.

Set VR camera - The virtual reality camera object is constructed with an empty parent object. Below this object two cameras are placed, one for each eye. Thus, the coordinate received by Unity's input system is applied to that parent object. With this operation, the internal cameras will move accordingly, whenever this information is updated.

Scene, Ray Tracing, Rendered image - When starting Unity, the scene is rendered by HDRP using its native Ray Tracing algorithm. The Ray Tracing works like post-processing and is applied as global post-processing, that is, it will be applied to all the existing cameras in the scene (right and left eyes).

Apply distortion shaders - In addition to Ray Tracing, image distortion shaders are applied to bring out the virtual reality view. This distortion is based on the Brown–Conrady barrel distortion model. This distortion is applied after the image has been generated using Ray Tracing and then stored to be sent to the mobile device's browser.

Create stereo image - Unity has a rendered texture object that can be used to store camera images. Since the cameras were created occupying half the resolution at the X coordinate, when applying the two cameras to the same render texture, the image fits perfectly. That object then generates a single, stereo image with both cameras, producing a virtual reality view.

Receive rendered image, Present VR image - Once the image is rendered and stored in the texture, it is streamed to the client-side script running on the mobile browser. The client-side script receives the image and applies it to its instantiated video player. The flow continues, with the periodic extraction of new orientation of the mobile device.

[1] Quaternions are defined by four components x, y, z, w and used to represent rotations.

4.1 DEVELOPMENT

We have three major steps to build the U-DiVE framework: (i) WebRTC connection for streaming; (ii) scene processing for virtual reality; and (iii) management of the orientation captured by the mobile devices.

(i) WebRTC connection for streaming

Our approach for WebRTC streaming is to use Unity's Render Streaming, which provides Unity's high definition rendering abilities via a browser. This streaming technology takes advantage of WebRTC and makes it possible to send and receive data from the client and the server.

The Render Streaming system consists of 3 components: Unity (Editor or Application), Web server, and Web browser (Figure 16). In Render Streaming, a P2P communication is created between Unity and the Web browser, exchanging data via UDP/IP. The Web server enables a signaling communication between the Web browser and Unity. Signaling is the exchange of information between involved points in the network. In Unity, signaling follows eight steps: Web browser sends Session Description Protocol (SDP) Offer to the Web server; Unity checks the Web server for unprocessed SDP Offers and receives any found; Unity sends SDP Answer to the Web server; web browser checks the Web server for unprocessed SDP Answers and receives any found; web browser sends Interactive Connectivity Establishment (ICE) Candidate to the Web server; Unity checks the Web server for unprocessed ICE Candidates and receives any found; Unity sends ICE Candidate to the Web server; and Web browser checks the Web server for unprocessed ICE Candidate and receives any found.

To provide high-fidelity graphics and a solid streaming frame rate for high-quality user experience, the Render Streaming framework tackle two problems: performance and latency. This framework broadcasts applications to the browser using the NVIDIA Video Codec SDK to perform GPU hardware encoding on the frame buffer, reducing latency. The image sent to the client is fixed to a 1280x720 pixels resolution and video bit rate starts at 16.000kbps and tops at 160.000kbps.

(ii) Scene processing for virtual reality

To explain the scene processing for virtual virtual reality, let us consider a sample scene with three graphical features: reflection, refraction, and shadows. The items added to the scene are a mirror (reflection), an aquarium (refraction), and some random objects generating shadow.

In Unity, Ray Tracing parameters for the test scene are configured as follows (see Figure 17): Ray Length with a value of 25; Clamp Value with a value of 1; performance mode; active Denoise with active Half Resolution Denoiser; and Second Denoiser Pass.

[2] `https://cutt.ly/cp8Wlq0`; Online, accessed in 28/06/2020

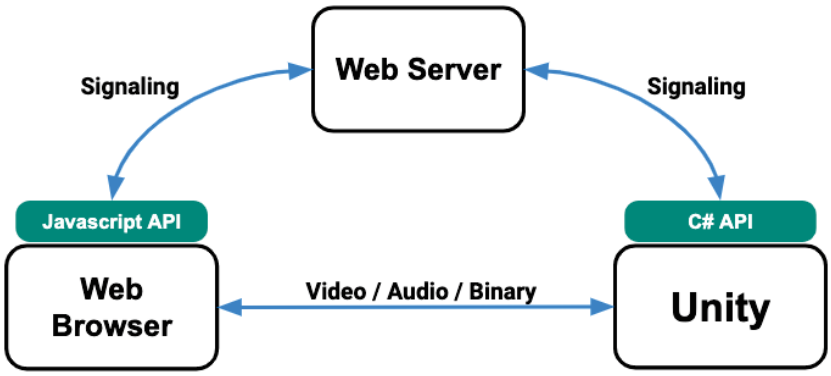


Figure 16 - Render Streaming overview[2]

After finished, the scene is ready to be sent to our virtual camera.

Figure 17 - Global Illumination configuration inside Unity

In the global illumination configuration, when Ray Tracing option is active, HDRP uses Ray Tracing to evaluate indirect diffuse lighting. LayerMask defines the layers that HDRP processes this ray-traced effect. Ray Length is the maximum distance a ray can travel. Clamp Value controls the threshold that HDRP uses to clamp the pre-exposed value and makes the global illumination more stable to denoise, but reduces quality. The choice for HDRP to evaluate the effect between performance and quality is set in the Mode property. In performance mode, used in this work, the options of Upscale Radius and Full Resolution are available, responsible for controlling the radius of the up-scaler that HDRP uses to build the global lighting and increasing the budget radius to one radius per pixel, per frame. The Denoise parameter enables the Spatio-temporal filter that HDRP uses to remove noise from the Ray-Traced Global Illumination. Half Resolution Denoiser decreases the resource intensity of denoising, but reduces quality. Denoiser Radius set the radius of the Spatio-temporal filter. Second Denoiser Pass option helps to remove noise from the effect. The second Denoiser Radius set the radius of the Spatio-temporal filter

for the second denoiser pass[3].

As shown in Figure 18, lenses like the ones used in Google Cardboard distort a planar image when trying to zoom. The lines of the pixel grid bend inward to the center of the image resulting in a pincushion distortion. The Google Cardboard SDK for Unity[4] was discontinued, so we had to rebuild the mobile virtual reality system from scratch.

To compensate for this distortion, an inverse pre-distortion must be applied to the image. The inverse distortion for a pincushion distortion is the barrel distortion. Figure 19 shows the process of inversely distorting an image with an already applied pincushion distortion. Usually, this type of pre-distortion is performed with the help of shaders in the frame buffer after image generation (41).

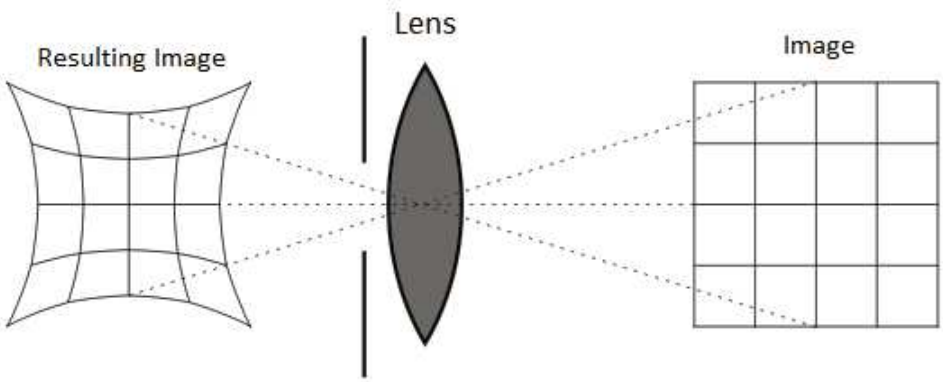


Figure 18 - Pincushion distortion generated when using lenses.

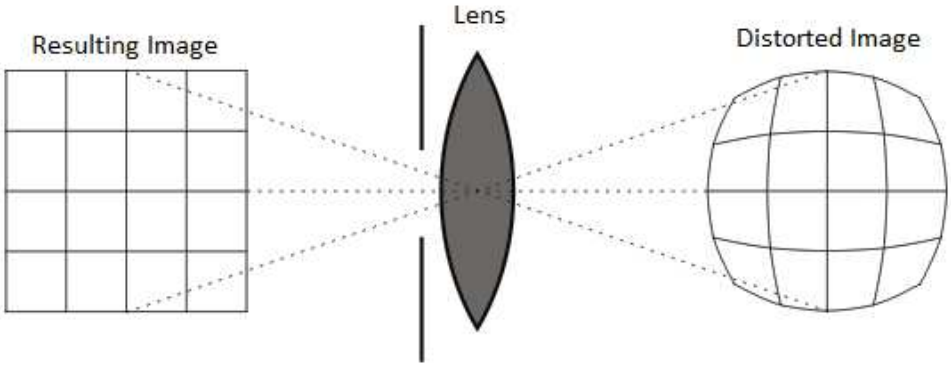


Figure 19 - Resulting image generated compensating the Pincushion distortion using Barrel distortion.

Render Streaming package does not support Virtual reality cameras, and it was necessary to modify the media stream function to accept two cameras, one for each eye. To build a virtual reality camera, we rely on previous versions of Google Cardboard SDK

[3] `https://cutt.ly/ETtCMlB`; Online, accessed in 01/11/2021

[4] `https://developers.google.com/vr/develop/unity/get-started-android`; Online, accessed in 15/12/2021

for Unity. U-DiVE creates two cameras representing the human eyes with a distance of 0.06 in Unity coordinates between them to create a parallax effect. Then, a fragment shader is applied to distort the image and provide a lens correction. We base our fragment shader on the Brown–Conrady model (42) that changes the position of each point by using the formula:

$$X_d = X_u(K_{offset} + K_0 r^0 + K_1 r^1 + ... + K_6 r^6)$$

$$Y_d = Y_u(K_{offset} + K_0 r^0 + K_1 r^1 + ... + K_6 r^6)$$

where (X_d, Y_d) is the distorted image point as projected on image plane using specified lens, (X_u, Y_u) is the undistorted image point as projected by an ideal pin-hole camera, $K_n = n^t h$ is the radial distorted coefficient, r is the radial distance from center defined by the formula:

$$r = \sqrt{(X_u - center_x)^2 + (Y_u - center_y)^2}$$

The K_{offset} coefficient must be set to 1 and the rest of the K coefficients can refer to Google Cardboard SDK parameters. The value of K depends on the effect we want to achieve. The final stereo view for the considered sample scene is presented in Figure 20.

Figure 20 - Stereo image generated through the modified media stream function.

(iii) Device orientation management

For the third step, we focus on the device orientation. The client-side script running on the mobile browser collects the orientation data from the mobile device, in quaternion format, saves it in an array buffer, and sends it to the server. Every time the user moves the head, the values are updated, so no unnecessary data is sent to the server. Inside

Unity, the data is received and applied to the main camera properties. For Unity to accept data from the client-side, a function that assigns the quaternion values received in the Unity control system is added. Thus, coordinates can be used in variables and assigned to the main camera object. The main camera object consists of an empty object and two cameras. If the superior object rotates in any direction the inner cameras representing the eyes follow the movement. When receiving the orientation data, the object coordinates consider that the front of the environment is on the Y-axis. The scene was projected based on this information.

4.2 USAGE SCENARIO

The intended use of the U-DiVE framework is the same as that of Google Stadia (43). Google Stadia is the new ambition of Google rooted in the fundamental components of gaming: players, watchers, and developers. The goal of Google is to allow anyone to start playing a game without installation, downloads, and third-party software. The core of Stadia is cloud-computing using the Google Data Center to create a cloud-based gaming experience. This approach means that instead of purchasing a game, the game is executed on a compute node and streamed to the player's device. Our framework is intended to follow Stadia's path, focusing on virtual reality and photorealism without affording expensive hardware. As in Stadia, the data is bidirectional, with inputs flowing from the player and the provider streaming audiovisual data that results from these inputs. Also, as Stadia, U-DiVE is cross-platform, requiring support to browsers only.

Another usage scenario is the Facebook Metaverse , which consists of a virtual world that tries to replicate reality with social interactivity through digital devices. With photorealism, virtual reality and accessibility, the framework meets several requirements that may help Metaverse-like environments possible and immersive with low-cost devices.

[4] `https://about.facebook.com/br/meta/`; Online, accessed in 22/12/2021

5 EVALUATION

This section presents the quantitative analysis of our work. We divide the measurement in two tests: the first test is focused on the rendered frame rate and we use only two scenes, the Cornell box and the living room; The second test we focus on the delay, using all scenes showed in Figure 22 and the grid scene. To extract this analysis we videorecorded the four scenes created and measured latency between movement of the mobile device and reaction response of the scene.

5.1 RECORDING AND DEVICES SETUP

As a low-cost device, we defined, for the client side, that the smartphone must be at least below R$ 1,000 plus the value of cardboard, which can be found for the value of R$ 50. Our setup for U-DiVE evaluation, includes, on the server side, a desktop PC with an Intel i5-9400f processor, a GTX 1660 super video card, 8GB RAM, Gigabit Ethernet. The client side was deployed in a Xiaomi Redmi Note 4 with 4GB RAM, Snapdragon 625 Qualcomm MSM8953 processor, IEEE 802.11 b/g/n/ac WLAN and Google Chrome browser. This device costs around R$ 900 in 2021/2022. The wireless router in the tested was an EchoLife HG8145V5 with frequency range of 5Ghz, mode 802.11a/n/ac, channel width of auto 20/40/80 mhz, route WAN mode, and four ports Gigabit 10/100/1000 Mb/s. To record the scenes, we used an iPhone SE-2 with slow motion video at 240fps.

We were unable to change some of the Unity's algorithms in our work, so measuring latency through exchanging messages from the client and server became an issue. To address this issue we used the approach of recording the movement of the mobile device. As the physical controller and the virtual screen are on the same device, we record the mobile device's movements using a 240 frames per second slow-motion camera. Thus, the measured latency has a precision of 4ms approximately (inter-frame interval).

The experiments were recorded with the mobile device placed on a table for superior stability. The movement of the device with the framework was only in the frontal direction and with similar speeds. Thus, there is no significant variation in pixels between recordings of the same scene.

5.2 SCENES AND VISUAL RESULTS

To perform our experiments, firstly we built a fullscreen grid scene (Figure 21). This scene was created to allow measuring with greater precision when counting the frames of the recordings (Figure 21). Thus, the exact moment when the screen responds to the movement becomes more noticeable. As the grid is just an object it is not influenced by

the colors around it, so the difference between having Ray Tracing and not having it is almost imperceptible to the eyes.

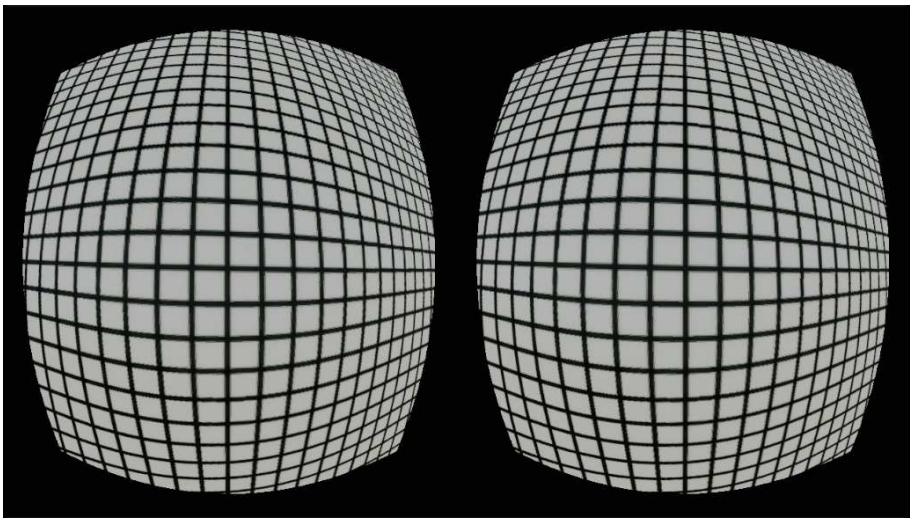


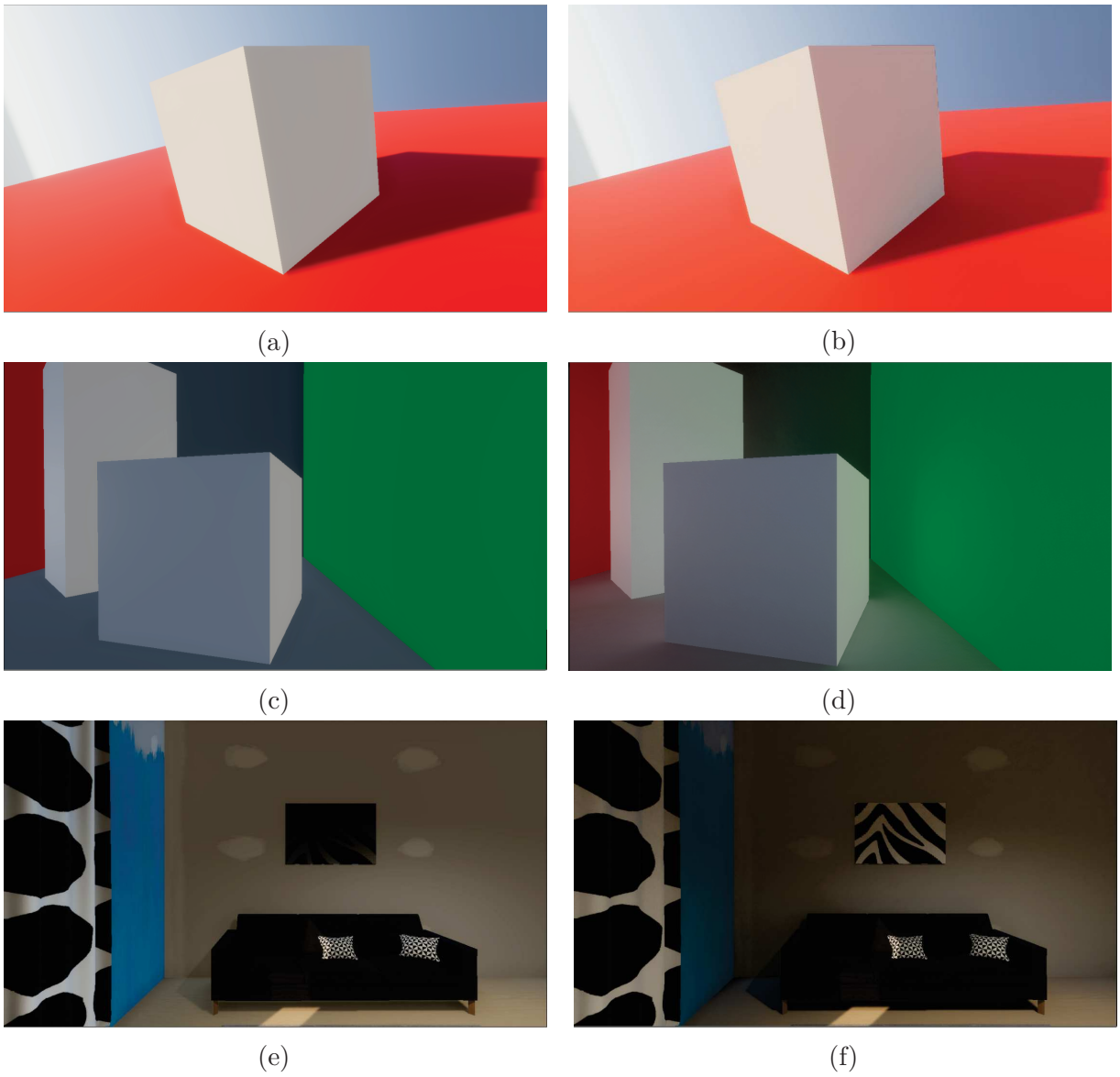
Figure 21 - Grid scene without Ray Tracing.

For visual evaluation we build three main scenes: a scene with a plane and a cube, a simple Cornell box, and a more complex scene that explores more realistic features (Figure 22). We use the same Ray Tracing parameters and virtual reality shaders for both scenes.

As shown in Figure 22a, the first scene has only a white cube centered on a red plane with a directional light. This scene was created to be the simplest and lightest test scene. Using Ray Tracing we can see, in the shaded region of the cube (Figure 22b), a shade of red accentuated due to the light that reflects from the surface of the shaft.

A Cornell box was created as an intermediate scene. This scene will be the middle ground between the simplest and most complex scenes. Our Cornell box consists of a box with the left and right sides colored green and red, respectively. The roof of this box has a light source and two objects on the floor to reflect the wall colors. In Figure 22d) we can see the shades in green in the front cube and red in the back cube when using Ray Tracing.

In the most complex scene (Figures 22e and 22f) we can see a difference in the shading of the sofa and the painting on the wall when using Ray Tracing. A detailed view of this scene can be observed in Figure 23. In Figure 23c, we highlighted hard shadows under the chair and table produced by the lamp. Refractions and shadows are highlighted in Figure 23d. In Figure 23e, we show a mirror reflecting parts of the scene containing reflections, refractions, and shadows. Lastly, we showed Figure 23f, which has hard shadows, soft shadows, and reflections on the glass wall.



(a) (b)

(c) (d)

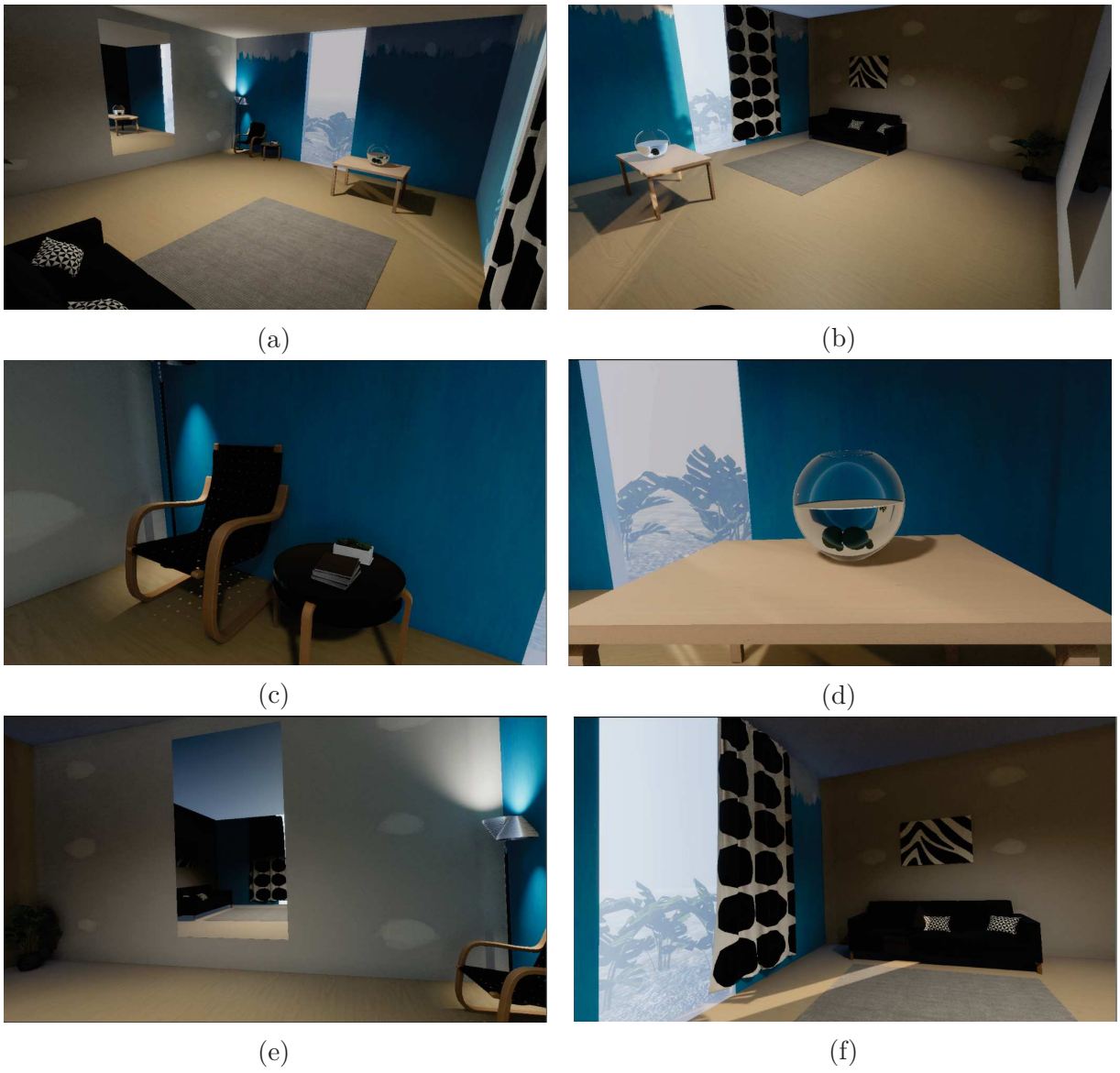
(e) (f)

Figure 22 - Scenes without Ray Tracing on the left and with Ray Tracing on the right.

5.3 TEST RESULTS AND DISCUSSIONS

To perform our first test we use the same Ray Tracing parameters and virtual reality shaders for the Cornell box and the living room scene that explore more realistic features. Our Cornell box consists of a box with the left and right sides colored green and red, respectively (Figure 24). The roof of this box has a light source and two objects on the floor to reflect the wall colors, as shown in Figure 24. With our setup, the Cornell box achieved 125 fps inside Unity and 60 fps in our mobile device.

Despite satisfying the requirements for a virtual reality application, depending on the setup, better performance may be achieved. In the Unity engine, the rendered frame rate depends on the computer hardware and how complex is the scene. From the mobile device perspective, a higher WLAN speed, faster video decoders and faster CPUs may



(a) (b)

(c) (d)

(e) (f)

Figure 23 - Test scene and its details. In 23a and 23b we have the full image in two different points of view, 23c the corner chair with the highlighted shadows, 23d aquarium with refractions and shadows, 23e mirror reflecting parts of the scene containing reflections, refractions, and shadows and 23f the corner sofa which has hard shadows, soft shadows, and reflections on the glass wall.

improve image quality and lower the delay. Different wireless routers and locations may also provide different communication performance.

It is worth noting that 60 fps is the default maximum update rate for the player used in the client-side script on the mobile web browser.

With this scene we reached an average of 39 fps in Unity, fulfilling one of the crucial requirements mentioned by (13). For the mobile device, the stream can reach 60 fps due to the amount of bitrate configured; however, as the scene only reaches 39 fps, the device was capped to this value.

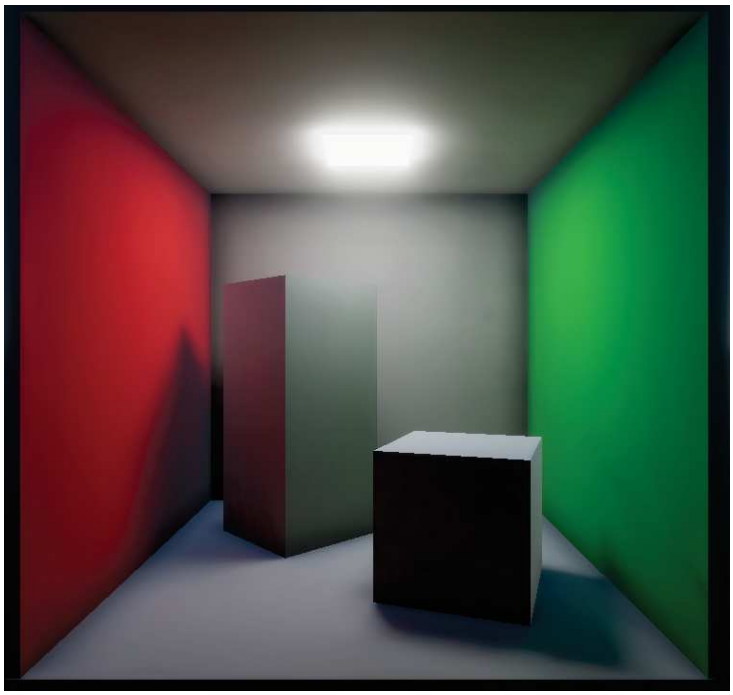


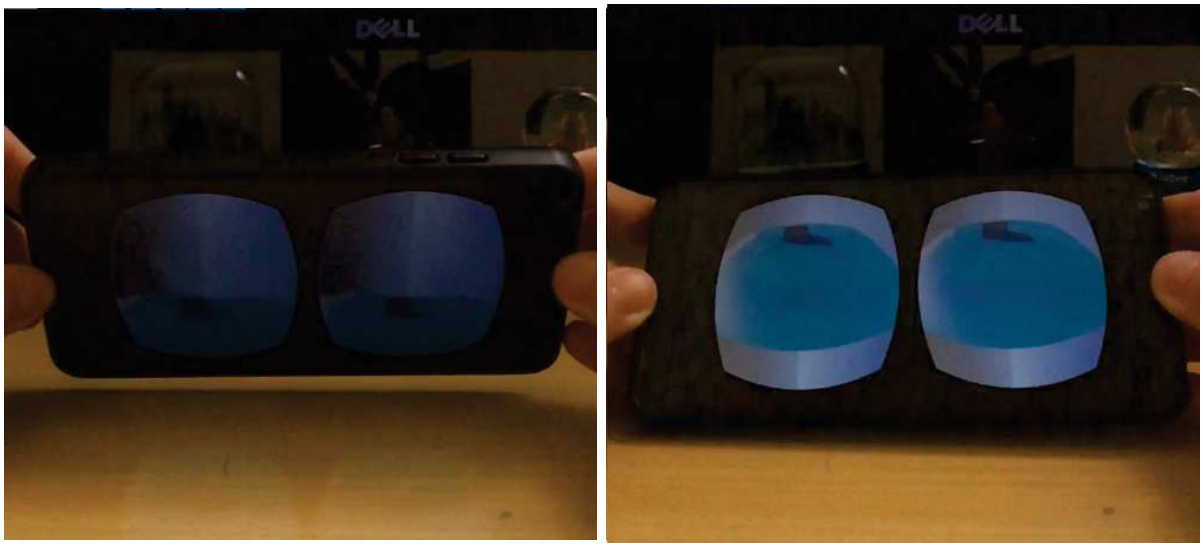
Figure 24 - Cornell box.

Figure 25 - Image including all reflection, refraction and shadow elements discussed.

In the tests, both scenes were executed as expected. User orientation was performed as if the scene was executing locally and a slight latency was noticed, but nothing hindered the overall experience.

As the network may suffer some kind of instability, for the latency test, five experiments were videorecorded for each scene, and we computed the average between them. Each experiment has the same movement direction and speed of the device so that

the results are similar between the recorded experiments. Figure 26 shows two captured frames from a recorded experiment, illustrating the device movement. After the experiment is recorded, an analysis is made from the moment the mobile device is moved until the scene's visual response expected for that movement. The number of frames between these moments are counted.



(a) (b)

Figure 26 - Mobile device being recorded with repeated and constant movements.

To achieve latency measurement, we need to take the number of frames from the device's response and multiply by 4.1, a value obtained by dividing a second, in millisecond, by 240 frames. For each video, we perform this process, then we removed the highest and lowest values and averaged the remaining values.

Table 2 shows the results of the recordings and their average latency. The same scenes and positions were recorded with and without Ray Tracing. We can observe that, in each recording, a different scene obtained greater latency. At first it was the cube scene with Ray Tracing, in the second it was the complex scene with Ray Tracing, in the third and fourth it was the complex scene but without Ray Tracing and in the fifth it was the grid scene with Ray Tracing. This shows that the latency is not exclusively dependent on the server hardware, but in fact on the whole Round Trip Time (RTT). RTT is the time it takes the orientation leaves the browser and arrives in Unity to generate a new frame and the frame to leave Unity and arrive in the browser back again. RTT can be influenced by the distance a signal has to travel, how quickly a request is received by a server and routed back to a user, the number of network hops, traffic levels and, of course, the server response time.

If we look at the average of each scene, we can observe that there is no significant changes between the scene with active and non-active Ray Tracing, Figure 27. With the measured latency of all recordings, we can see that the difference between the lowest

Tabela 2 – Latency comparison table.

VR Scene	Time(ms)					
	1	**2**	**3**	**4**	**5**	**Mean**
Cube	184,5	209,1	184,5	176,3	168,1	181,77
Cube with RT	225,5	180,4	159,9	172,2	180,4	177,67
Grid	151,7	155,8	172,2	184,5	176,3	168,10
Grid with RT	155,8	151,7	180,4	188,6	188,6	174,93
Cornell Box	168,1	188,6	176,3	143,5	159,9	168,10
Cornel Box with RT	172,2	176,3	164	147,6	168,1	168,10
Complex Scene	176,3	184,5	213,2	209,1	180,4	191,33
Complex Scene with RT	184,5	221,4	205	172,2	184,5	191,33

latency, 147.6, and the highest latency, 225.5, is 77.9. This shows that latency can vary widely between different sessions. We can also conclude that the observed mean latency is around 177 (ms), considering all scenes and sessions.

During recording and testing we observed a problem with pixel loss in the frame. We notice this problem only in the Grid scene. This loss occurs when the lines that create the Grid move through the screen and the pixel needs to go from the lightest (white) to the darkest (black).

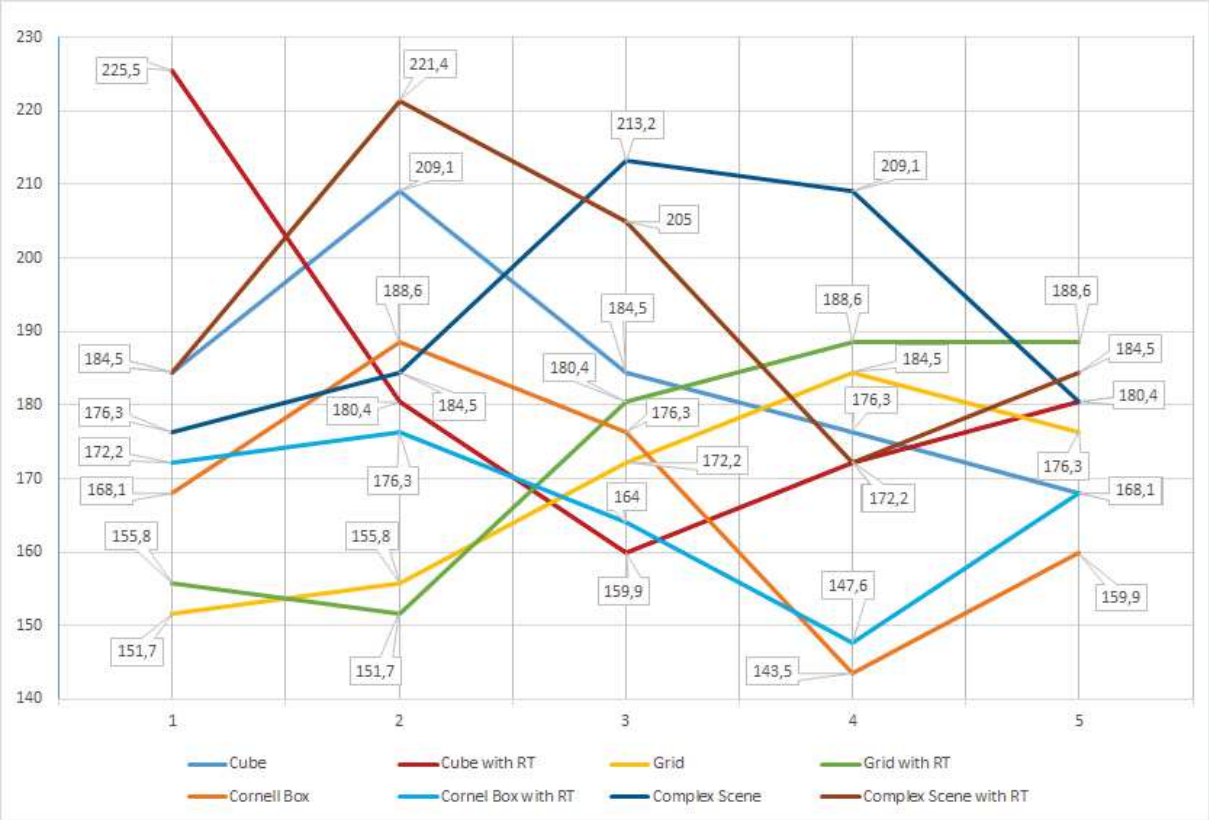


Figure 27 - Line Chart of Table 2. Each labeled column corresponds to a test of each scene and each row represents the value obtained in these tests.

6 CONCLUSIONS AND FUTURE WORK

This work aimed to create a framework, called U-DiVE, that allows the use of photorealistic virtual reality scenarios with low-cost mobile devices. This framework was built over the Unity engine, which has all the support to create a photorealistic virtual reality environment, a rendering pipeline with native photorealism algorithms, and a package that allows for performing interactive streaming of the application. The streaming package is based on WebRTC, allowing the application to perform real-time interactive streaming.

Besides Unity having all the support needed to build the framework, some parts of Unity cannot be modified and therefore it was not possible to build a ping test to identify latency and delay within the software. To work around this problem, we used the approach of recording the movement of the mobile device. As the physical controller and the virtual screen are on the same device, we recorded the mobile device's movements using a 240 frames per second slow-motion camera.

As no framework for comparison was found with the same features, various scenes to evaluate performance were created. The first scene built is a plane with a cube. This scene is considered the simplest scene and serves as a basis for comparing the other scenes. The second scene contains a vertical plane with a grid, and it was created to be more accurate when recording scenes and tests. The third scene is a replica of the Cornell box. We created this scene to have a comparison with a scene widely used in literature. Lastly, a complex scene with reflective, refractive effects and shadows.

We verified that the frame rate on all scenes was above the minimum required for a virtual reality system, and the latency of the stream on the mobile device maintained high quality. The performance in the complex scene was below 60 fps, so the stream is limited to the number of frames generated on the server. We verified that the delay of the scenes with and without Ray Tracing has a small difference, showing that even a complex scene does not interfere in the transfer of data between client and server. Although Ray Tracing is a heavy algorithm, the evaluation of the data showed that using the algorithm has no relevant impact. Due to the sudden change of color in the grid, from white to black, some flaws in the images received by the client could be noticed.

In terms of performance, the proposed framework meets the minimum requirements for a virtual reality system and uses photorealistic techniques in the scenes created. We can conclude that the results achieved with our synthetic scenes show that, for a low-cost device, a complex and realistic environment can be executed as expected, depending only on the server hardware and the WLAN connection.

In future works, we intend to change the distortion shaders to a vertex displacement based solution that eliminates the need to render a intermediate texture. In this approach,

the distortion is in the geometry itself using a custom vertex shader, and no shader pass is needed saving a step of copying the rendering into a texture. We plan to do a qualitative test with multiple users to evaluate user experience. These tests were scheduled but unfortunately they were unable to be executed due to the COVID-19 pandemic. Another future work is to make the U-DiVE work as an edge technology, which can accompany the user, through computing capabilities of radio base stations in mobile networks for example (44). We can create techniques to predict the user's head movement, so we can generate the next frame before receiving the actual orientation. Creating an environment with multiple cameras and avatars for users to have a social interaction as well is the goal of the metaverse.

REFERÊNCIAS

1 Soojeong Yoo and Judy Kay. Vrun: running-in-place virtual reality exergame. In *Proceedings of the 28th Australian Conference on Computer-Human Interaction*, pages 562–566, 2016.

2 Won-Jong Lee, Seok Joong Hwang, Youngsam Shin, Jeong-Joon Yoo, and Soojung Ryu. Fast stereoscopic rendering on mobile ray tracing gpu for virtual reality applications. In *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pages 355–357. IEEE, 2017.

3 Won-Jong Lee, Youngsam Shin, Jaedon Lee, Shihwa Lee, Soojung Ryu, and Jeongwook Kim. Real-time ray tracing on future mobile computing platform. In *SIGGRAPH Asia 2013 Symposium on Mobile Graphics and Interactive Applications*, pages 1–5, 2013.

4 Arthur Pedro Godoy and Cesar AC Teixeira. An architecture to promote the use of mobile devices on interactions with media synthesized remotely. In *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*, pages 143–150, 2014.

5 Mohammed S Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. Toward low-latency and ultra-reliable virtual reality. *IEEE Network*, 32(2):78–84, 2018.

6 Hamed Ahmadi, Omar Eltobgy, and Mohamed Hefeeda. Adaptive multicast streaming of virtual reality content to mobile users. In *Proceedings of the on Thematic Workshops of ACM Multimedia 2017*, pages 170–178, 2017.

7 Seyedmohammad Salehi, Abdullah Alnajim, Xiaoqing Zhu, Malcolm Smith, Chien-Chung Shen, and Leonard Cimini. Traffic characteristics of virtual reality over edge-enabled wi-fi networks. *arXiv preprint arXiv:2011.09035*, 2020.

8 Martin J Prins, Simon NB Gunkel, Hans M Stokking, and Omar A Niamut. Togethervr: A framework for photorealistic shared media experiences in 360-degree vr. *SMPTE Motion Imaging Journal*, 127(7):39–44, 2018.

9 Kai Rohmer, Wolfgang Büschel, Raimund Dachselt, and Thorsten Grosch. Interactive near-field illumination for photorealistic augmented reality on mobile devices. In *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 29–38. IEEE, 2014.

10 Simon NB Gunkel, Rick Hindriks, Karim M El Assal, Hans M Stokking, Sylvie Dijkstra-Soudarissanane, Frank ter Haar, and Omar Niamut. Vrcomm: an end-to-end web system for real-time photorealistic social vr communication. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 65–79, 2021.

11 Michael Csongei, Liem Hoang, Christian Sandor, and Yong Beom Lee. *Global illumination for Augmented Reality on mobile phones*. IEEE, 2014.

12 Siddhant Prakash, Alireza Bahremand, Linda D Nguyen, and Robert LiKamWa. Gleam: An illumination estimation framework for real-time photorealistic augmented reality on mobile devices. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 142–154, 2019.

13 Frederick P Brooks. What's real about virtual reality? *IEEE Computer graphics and applications*, 19(6):16–27, 1999.

14 Bill Fleming. *3D photorealism toolkit / Bill Fleming.* John Wiley, New York, 1998.

15 Ze-Nian Li, Mark S Drew, and Jiangchuan Liu. *Fundamentals of multimedia*. Springer, 2004.

16 Kadi Bouatouch and Christian Bouville. *Photorealism in computer graphics*. Springer Science & Business Media, 2013.

17 Google cardboard. `https://arvr.google.com/cardboard/`. Accessed: 2021-07-30.

18 Nikiforos M Papachristos, Ioannis Vrellis, and Tassos A Mikropoulos. A comparison between oculus rift and a low-cost smartphone vr headset: immersive user experience and learning. In *2017 IEEE 17th International Conference on Advanced Learning Technologies (ICALT)*, pages 477–481. IEEE, 2017.

19 Jian He, Mubashir Adnan Qureshi, Lili Qiu, Jin Li, Feng Li, and Lei Han. Rubiks: Practical 360-degree streaming for smartphones. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 482–494, 2018.

20 Marco C Jacobs, Mark A Livingston, and Andrei State. Managing latency in complex augmented reality systems. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 49–ff, 1997.

21 Tarek El-Ganainy and Mohamed Hefeeda. Streaming virtual reality content. *arXiv preprint arXiv:1612.08350*, 2016.

22 Gi Sook Jung and Soon Ki Jung. A streaming engine for pc-based 3d network games onto heterogeneous mobile platforms. In *International Conference on Technologies for E-Learning and Digital Entertainment*, pages 797–800. Springer, 2006.

23 Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.

24 Georg Rainer Hofmann. Who invented ray tracing? *The Visual Computer*, 6(3):120–124, 1990.

25 Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45, 1968.

26 Jon Peddie. *Ray Tracing: A tool for all*, volume 5. Springer, 2019.

27 Turner Whitted. *An Improved Illumination Model for Shaded Display*, page 119–125. Association for Computing Machinery, New York, NY, USA, 1998.

28 Edward R Freniere and John Tourtellott. Brief history of generalized ray tracing. In *Lens Design, Illumination, and Optomechanical Modeling*, volume 3130, pages 170–178. International Society for Optics and Photonics, 1997.

29 James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

30 Henning Schulzrinne, Stephen Casner, Ron Frederick, Van Jacobson, et al. Rtp: A transport protocol for real-time applications, 1996.

31 Roger Pantos and William May. Http live streaming, 2017.

32 ISO/IEC 23009-1. Information technology - dynamic adaptive streaming over http (dash) - part 1: Media presentation description and segment formats, 2014.

33 Adam Bergkvist, Daniel C Burnett, Cullen Jennings, Anant Narayanan, and Bernard Aboba. Webrtc 1.0: Real-time communication between browsers. *Working draft, W3C*, 91, 2012.

34 Salvatore Loreto and Simon Pietro Romano. Real-time communications in the web: Issues, achievements, and ongoing standardization efforts. *IEEE Internet Computing*, 16(5):68–73, 2012.

35 Mark R Mine. Characterization of end-to-end delays in head-mounted display systems. *The University of North Carolina at Chapel Hill, TR93-001*, 1993.

36 Anthony Steed. A simple method for estimating the latency of interactive, real-time graphics simulations. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, pages 123–129, 2008.

37 Jiandong Liang, Chris Shaw, and Mark Green. On temporal-spatial realism in the virtual reality environment. In *Proceedings of the 4th annual ACM symposium on User interface software and technology*, pages 19–25, 1991.

38 Dorian Miller and Gary Bishop. Latency meter: a device for easily monitoring ve delay. In *Proceedings of SPIE*, volume 4660, 2002.

39 Sebastian Friston and Anthony Steed. Measuring latency in virtual environments. *IEEE transactions on visualization and computer graphics*, 20(4):616–625, 2014.

40 Ding He, Fuhu Liu, Dave Pape, Greg Dawe, and Dan Sandin. Video-based measurement of system latency. In *International Immersive Projection Technology Workshop*, volume 111. Citeseer, 2000.

41 Christoph Anthes, Rubén Jesús García-Hernández, Markus Wiedemann, and Dieter Kranzlmüller. State of the art of virtual reality technology. In *2016 IEEE Aerospace Conference*, pages 1–19. IEEE, 2016.

42 Duane C Brown. Decentering distortion of lenses. *Photogrammetric Engineering and Remote Sensing*, 1966.

43 Adam C Desveaux and Valerie S Courtemanche. The effects of latency in commercial cloud video gaming services. *Worcester Polytechnic Institute*, 2020.

44 Alexandre Martins Gama de Deus, Eduardo Pagani Julio, and Marcelo Ferreira Moreno. Join-me: Uma arquitetura para integração entre operadores de redes móveis e provedores de serviços. In *Anais do XI Workshop de Pesquisa Experimental da Internet do Futuro*, pages 26–31. SBC, 2020.