Foveated Video Streaming for Cloud Gaming

Gazi Karam Illahi

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 22.8.2017

Thesis supervisors:

Prof. Antti Ylä-Jääski

Thesis advisor:

D.Sc. (Tech.) Matti Siekkinen

Aalto University
School of Electrical
Engineering

AALTO UNIVERSITY
SCHOOL OF ELECTRICAL ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Gazi Karam Illahi

Title: Foveated Video Streaming for Cloud Gaming

Date: 22.8.2017 Language: English Number of pages: 8+63

Department of Communications and Networking

Professorship: Data Communication Software

Supervisor: Prof. Antti Ylä-Jääski

Advisor: D.Sc. (Tech.) Matti Siekkinen

Video gaming is generally a computationally intensive application and to provide a pleasant user experience specialized hardware like Graphic Processing Units may be required. Computational resources and power consumption are constraints which limit visually complex gaming on, for example, laptops, tablets and smart phones. Cloud gaming may be a possible approach towards providing a pleasant gaming experience on thin clients which have limited computational and energy resources. In a cloud gaming architecture, the game-play video is rendered and encoded in the cloud and streamed to a client where it is displayed. User inputs are captured at the client and streamed back to the server, where they are relayed to the game. High quality of experience requires the streamed video to be of high visual quality which translates to substantial downstream bandwidth requirements. The visual perception of the human eye is non-uniform, being maximum along the optical axis of the eye and dropping off rapidly away from it. This phenomenon, called foveation, makes the practice of encoding all areas of a video frame with the same resolution wasteful.

In this thesis, foveated video streaming from a cloud gaming server to a cloud gaming client is investigated. A prototype cloud gaming system with foveated video streaming is implemented. The cloud gaming server of the prototype is configured to encode gameplay video in a foveated fashion based on gaze location data provided by the cloud gaming client. The effect of foveated encoding on the output bitrate of the streamed video is investigated. Measurements are performed using games from various genres and with different player points of view to explore changes in video bitrate with different parameters of foveation. Latencies involved in foveated video streaming for cloud gaming, including latency of the eye tracker used in the thesis, are also briefly discussed.

Keywords: Foveated Video Streaming, Cloud Gaming, Real Time Foveated Encoding

Preface

I want to thank Almighty Allah for His benevolence and grace.

I want to thank my instructor Dr. Matti Siekkinen for all the invaluable guidance and encouragement during this thesis, Professor Antti Ylä-Jääski for allowing me to work in his group, and Teemu Kämäräinen for his help in understanding the test bed cloud gaming platform.

I want to pay gratitude to my parents for their kindness, unflailing love and faith in me. Last, but not the least, I want to thank all my friends and family for being there for me.

Otaniemi, 22.8.2017

Gazi Karam Illahi

Contents

Abbreviations

CPU	Central Processing Unit
GPU	Graphics Processing Unit
QoS	Quality of Service
QoE	Quality of Experience
DASH	Dynamic Adaptive Streaming over HTTP
SMPS	Switch Mode Power Supply
HLS	HTTP live streaming
RTSP	Real Time Streaming Protocol
RTP	Real Time Protocol
TCP	Transport Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
AMD	Advanced Micro Devices
SMG	Stream My Game
POV	Point Of View
MOS	Mean Opinion Score
EC2	Elastic Compute Cloud
CDN	Content Delivery Network
GaaS	Gaming as a Service
fps	Frames Per Second
CMOS	Complementary Metal Oxide Semiconductor
CCD	Charge Coupling Device
ITU	International Telecommunication Union Telecommunication Standardization Sector
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
VCEG	Video Coding Experts Group
MPEG	Moving Picture Experts Group
FPS	First Person Shooter
LRS	Little Racers Street
GbE	Gigabit Ethernet
HVS	Human Visual System
POV	Point Of View
AVC	Advanced Video Codig Standard(MPEG-4 AVC)
HEVC	High Efficiency Video Coding
OS	Operating System
CCD	Charge Coupled Device
CMOS	Complementary Metal Oxide Semi-Conductor
JVT	Joint Video Team
RoI	Region of Interest

List of Figures

List of Tables

1 Introduction

More and more rich media and interactive applications are becoming common place. Many of these application require intensive computing power, in particular graphics rendering power. Processing and graphics rendering power is expensive and in many cases it is advantageous to do the processing and rendering remotely. This could be either due to constraints on space available in the device, expense of hardware required, efficient use of resources or a combination of these. In some cases intellectual property, license management and content distribution may be factors that necessitate remote rendering. In any case, cloud based processing and rendering of interactive and rich media applications may reap the same benefits as cloud computing in general. Cloud computing has been studied extensively for quite some time, for example, [6] and [7] discuss the benefits and challenges of cloud computing.

Gaming is a multi-billion dollar business, expected to grow further [8]. Gaming as an industry segment is not only growing vertically but also horizontally. New technologies like virtual and augmented reality are expanding the sensory avenues of immersive gaming while mobile consoles, smart phones and tablets provide new devices for gaming. Graphics intensive games need size-able computing and electrical power to be rendered properly. Games with good graphics require state of the art central processing units (CPU) and graphics processing units (GPU) for good quality of experience (QoE). A GPU is used to carry out the parallel computing needed to render the video output of a game. Further, such intensive computing and graphics rendering operations are power consuming. In mobile devices, both processing resources and power storage come at a premium. Graphics intensive gaming on low end computing devices or mobile devices is a good use case for remote rendering, for example, in a cloud gaming scenario.

In cloud gaming, a server executes the game logic, renders the game-play video, encodes it and then streams it to a thin client. The thin client receives the game video over the Internet, decodes it and plays it on a display. The thin client also captures user control actions, such as key-board presses, and forwards them to the cloud server. The cloud server replays these control inputs to the game logic [4]. Cloud gaming is illustrated in Figure. 1[1, 4]. Games being interactive applications require very low latency and considerable downstream bandwidth for good QoE. The downstream bandwidth is needed for high quality video streaming. The more graphics intensive the game is, the more down-link bandwidth it requires if the game video rendered by the game is to be transmitted faithfully to the client. These constraints are even more pronounced when a wireless telecommunication network is in the path between the server and client.

When streaming any kind of video, provided the source video has high enough quality, the bandwidth available dictates the quality of video streamed. The higher the available bandwidth, the higher the number of bits that can be allocated to each frame of the video and hence the better the quality of stream-able video. Traditional

streaming solutions deal with changes or constraints in the available bandwidth by adapting the quality of the streamed video on the fly. This is typically achieved by temporal adaptation of the streamed video quality. The quality of each frame is dictated by the instantaneous available bandwidth.This is called adaptive bitrate streaming and DASH [9] is widely deployed standard. Typically in such solutions the visual quality is uniform across a frame, that is there is little spatial variation in quality in a frame.

The human eye has a distinctly non-uniform distribution of cone cells which are responsible for vision in bright light. The distribution of cone cells is highest at the back of the eye in a region called the fovea and it drops off quickly with distance from the fovea. This region occupies about 2^{o} of the human visual field [10]. Consequently, the human eye perceives an image with varying visual quality. The region directly in front of the fovea is perceived with the highest acuity and the acuity decreases with the angle of eccentricity from the fovea. This phenomenon is called foveation. Encoding a video with uniform quality across the frame is wasteful considering the visual acuity of the eye.

A video that is encoded such that the region of each frame which will be directly in front of the fovea is encoded with better quality, while other regions are encoded with lesser quality would theoretically be smaller in size and consequently bandwidth efficient in streaming applications. This is called foveated video encoding and has been studied thoroughly [11]. Foveated video encoding, although bandwidth and storage size efficient, is difficult to achieve. The encoder must have information about the region of interest in each frame where the viewers' gaze is likely to point. This can be achieved by analysis of the video before hand, for example, using image processing. However, that may be very difficult for live streaming scenarios where each frame is generated in real time. Another approach of determining regions of interest for each frame is by tracking the gaze of a viewer in real time and providing it to the encoder. This approach, although well known, was not usable outside of research laboratory because of computing and monetary expense of gaze tracking. Recently, with the availability of relatively cheap consumer grade gaze trackers, gaze tracking for real time foveated encoding has become more feasible. This approach is more suited to real time streaming but might not be scalable as a new encoder instance may be needed for each viewer.

1.1 Motivation

This work is motivated by the need to lower bandwidth requirements for high QoE cloud gaming. As mentioned earlier, cloud gaming requires significant downstream bandwidth in order to deliver game play video of sufficient quality to cloud gaming clients. Keeping in view the non uniform acuity of the human visual system, there is scope for reduction of video bitrate without affecting the percieved quality of the video. This provides motivation to explore foveated video streaming in a cloud

gaming context.

1.2 Research Objectives

Cloud gaming is an exciting new paradigm which leverages cloud computing to provide graphics intensive gaming on devices with low compute power. However, for satisfying QoE, low latency and high bandwidth are required. Video rendered by graphics intensive games can be very large in size and streaming it over networks with limited bandwidth may result in poor QoE. This thesis attempts to reduce the bandwidth requirements of high quality video streaming for cloud gaming. Inspired by the non-uniformity of human visual system, the reduction is to be achieved by foveated encoding of game play video, using a gaze tracker at the cloud gaming client to track the user's gaze. The gaze location data is to be sent to the cloud gaming server and which will use it to encode the video with spatially non-uniform quality which corresponds to the acuity of Human Visual System (HVS). Other objectives of the thesis include evaluating the prototype cloud gaming system in terms of video throughput reduction with varying parameters of foveation and a basic feasibility study in terms of latency.

1.3 Research Outcomes

The primary research outcome of this thesis work is a prototype cloud gaming system with foveated video streaming capabalities. The system is based on an open source cloud gaming platform and comprises a modified cloud gaming server of the platform which receives gaze location data from the client and encodes gameplay video with a quality corresponding to the gaze location. The client of the prototype integrates an eye tracker which reports the current gaze location to the server as soon as it is available. The prototype is parametrizable and the area as well as degree of foveation can be controlled.
Other research outcomes of this thesis include measurements of video throughput for different games at various foveation parameters and experiments to characterize latency of the eye tracker used in the prototype.

1.4 Structure

This thesis is further structured into Chapters 2, 3,4, 5,6, 7 and 8. In Chapter 2, the background for this work, i.e cloud gaming, is discussed. In Chapter 3, Video coding and streaming is discussed as video streaming is a primary enabler of cloud gaming. In Chapter 4, Foveated Video Streaming is discussed exploring aspects like physiology of foveation, gaze tracking needed to implement foveated streaming and encoding methodology used.

In Chapter 5 the design and implementation of the foveated video streaming for cloud gaming is described along with the various tools and components used used in the implementation. The tools and components used are introduced, design of the prototype is discussed and design choices are highlighted. In Chapter 6, the prototype is evaluated. The measurement setup is described and the results of measurements are analyzed. Some conclusions are drawn and discussed in Chapter 7 and the thesis is concluded in Chapter 8.

2 Cloud Gaming

2.1 Overview

Video game is a burgeoning industry, with revenues in the year 2016 exceeding 91 billion dollars [12]. The revenue comprises both hardware and game software sales, with software like game executables and add-ons leading the way. The industry is predicted to grow in year 2017 as well [13]. Typically dedicated gamers use high-end hardware or specialized gaming consoles on which the game software is installed. These devices have higher than average computational resources and consume considerable power. A game might be an off-line game or an on-line one, depending on whether or not it needs an Internet connection to be played. In both cases typically the game software is downloaded to the gaming machine and installed thereon. An off-line game can be played without an Internet connection. All of the game logic is executed locally and the game video is also rendered locally. In online multi-player games, the game is installed on the gaming machine but also connects to either peers playing the game or a server. Typically the server, or one of the peers, executes the master version of the game and all other players' actions are forwarded to it, based on which it calculates new game states and instructs the clients or peers to render the same [14]. Rendering, which is computationally quite intensive, is always done locally and thus necessitates use of high end hardware.

Recent trends, however, point to a paradigm shift. More and more gaming is taking place on portable devices like smartphones [12] .For example, the revenue of mobile gaming sector was 41 billion dollars in 2016 out of the 91 billion dollars for the entire gaming industry. Smartphones are quite attractive as gaming devices as they are ubiquitous and make gaming anywhere an achievable vision. Furthermore, smartphones are becoming increasingly powerful in terms of computing power. For example a top of the line Android smartphone in 2017, comprises an Octa-core (2.35GHz Quad + 1.9GHz Quad), 64 bit, 10nm ARM RISC processor [15]. Although these numbers appear competitive with a modern desktop class CPU [16], the performance in terms of raw processing is not comparable [17].Similar is the case with GPUs for desktop and smartphone platforms particularly considering the rendering power needed by modern high end games [18]. From a performance point of view, smartphones suffer from inherent disadvantages. They are constrained by both the available printed circuit board (PCB) real estate as well as the available power budget. Based on a typical battery capacity of 3500 mAh at a voltage of 3.5 V the power budget available to a smartphone processor is of the order a few (about 12) Watts. On the other hand, a standard desktop computer simple switch mode power supply (SMPS), can output anything from 200 to 800 watts. These limitations in smartphone and even mobile computers make remote rendering of graphics intensive applications a feasible and attractive proposition. Modern high-end games can be so compute intensive that they require very-high end hardware components for a good quality of experience. A good gaming computer may cost upwards of 1000 Euros. So remote execution and rendering of games may be a inexpensive and viable, rather

compelling, option for even desktop gaming.

Cloud gaming implements the idea of remote rendering and execution. In cloud gaming, a gamer can play the game on a thin client, such as, a smartphone or a netbook with relatively weak compute power. The execution of the actual game and the rendering of the game play video is done in the cloud by a cloud gaming server. The server encodes this video and streams it to the thin client over the Internet. The client receives this video, decodes it and plays it for the player. The client also captures user inputs and transmits them to the cloud gaming server which replays the inputs to the game as if the user was playing there at the server. With proper availability of network and server resources, the process can be seamless enough that the player does not notice any difference between local gaming and cloud gaming.

Cloud gaming as a commercial segment is in an evolutionary phase. There are many active and under-development cloud services and platforms like GameFly [19], Loud Play [20], Utomik [21], Nvidia Grid [22], Nvidia GeForce Now [23] and PlayStation Now [24]. Sony, which owns PlayStation Now, has also acquired cloud gaming startups like Gaikai [25].

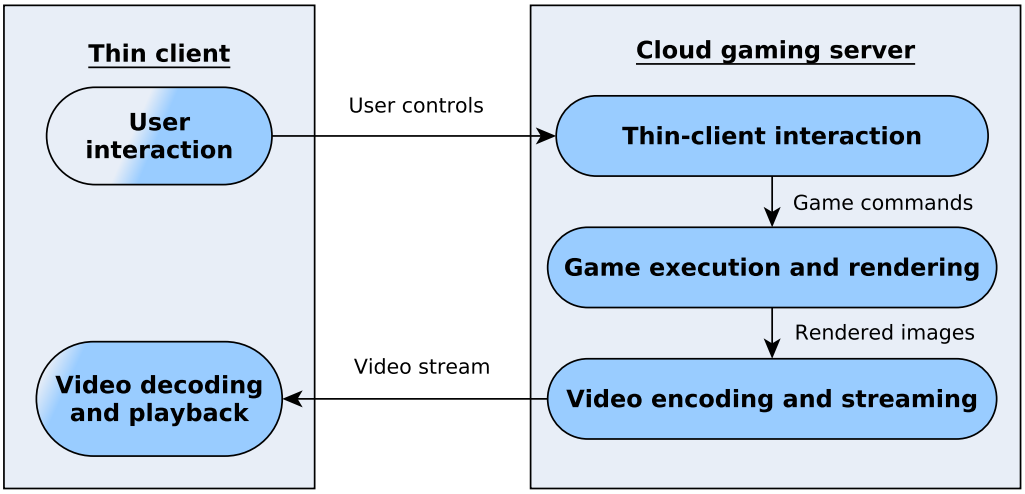


Figure 1: Cloud gaming architecture [1].

2.2 Architecture

Figure 1 illustrates high level cloud gaming architecture. An end user plays a game on a thin client connected to a cloud gaming server over a network, typically over the Internet. In practice a cloud gaming client is installed on a thin client hardware device while the cloud gaming server is run in a virtual instance of an Operating System (OS), suited for the game running in a cloud environment.

- **Cloud Gaming Client:** The client software has at least two components. One for capturing user inputs, such as key presses or mouse clicks and sending them to the cloud gaming server using a suitable protocol. The second component is for receiving game play video (and audio) from the cloud gaming server over the Internet. A streaming client may be used to receive and play the game-play video.

- **Cloud Gaming Server:** The core of a cloud gaming server software consists of three components: one to execute a game, one to receive user inputs from the client, replay them to the game, and one to capture the game-play video generated by the game, encode it and transmit it to the client. The component to receive user inputs receives user input messages from the client, decodes them and replays them to the game logic, mimicking local user action. The component to capture the video may either hook into the GPU operations and get the game play video frames as soon as they are rendered or it may periodically capture video frames from the video buffers of the server. It further encodes the video using an encoder and streams the encoded video using a streaming server. Game-play audio is similarly processed, concurrently with the video.

Figure 1 shows only one client connected to a client. In practice, depending on the nature of the game or the use case, there can be multiple clients connected to the server. Some clients may be regular playing clients of a multi-player game whereas some clients may be viewing clients which view the game play without participating in the game itself.

The architecture Figure 1 is the architecture considered in this thesis. It is game agnostic and works directly with off-the shelf games. Other architectures are also possible. In one alternative architecture rendering load may be shared between the thin client and the server [26]. In another alternative architecture, the rendering may be done at the server, but instead of video, objects are streamed to the client [27]. Both these alternative architectures need modifications to the game engine or ground up design of game engines according to these architectures.

2.3 Benefits and Challenges

Cloud gaming is a natural extension of cloud computing. Cloud computing refers to remote computing on server farms provided as a service. Gaming, as mentioned earlier, is resource intensive and cloud computing provisions resources and ensures their fair usage. Cloud gaming ensures far more intensive usage of hardware resources than when installed on an individual's computer. Cloud gaming offers many advantages for both game developers as well as game players, colloquially called "gamers". Cloud gaming can save the game players from investing in expensive hardware required for high-end gaming. Since hardware development is cyclical, gamers typically invest in new hard ware iteratively with each cycle or every alternate cycle to be able to

enjoy the latest games. This is because game developers develop games keeping in mind the latest hardware to allow their games maximum compute and graphics power. Cloud gaming may ameliorate this need for iterative investment in high end hardware by the end users, instead expediting centralized hardware upgrades by cloud gaming service providers. It can also expedite high-end gaming on portable clients which typically have access to lesser computing resources and smaller power reserves as discussed earlier. For game developers/publishers, games can be valuable pieces of intellectual property and distributing them to end users opens up avenues of piracy, while cloud gaming patches that vulnerability. Further game developers can focus on developing for a single platform and explore new business models [28]. Tradionally, a player has to buy a license to each game they like or would like to try. Cloud gaming may change that by making subscription based services possible. For example, both GeForce Now [23] and PlayStation Now [24], two leading cloud gaming services, have a subscription based business model.

However, for all the advantages of cloud gaming, there remain some key constraints holding back its wide spread adoption, and these are latency and downstream bandwidth requirements. Latency, or more specifically end to end latency, affects QoE profoundly. Higher the latency between user action and resulting game play scene, lower the QoE of game play [29]. In some games even the in-game performance of game player can be affected considerably. The tolerable delay depends on game genre and point of view of the player in conventional online games [30] as shown in Table 1. Theoretically, these latency thresholds should hold for cloud gaming as well.

Game Type	Perspective	Tolerable delay
FPS	First Person	100ms
RPG	Third Person	500ms
RPS	Omni-Present	1000ms

Table 1: Delay Tolerance in traditional games [4].

The latency between user input and visible effect in game play frame at the cloud gaming client—response delay in cloud gaming is due to multiple factors. These are: the latency at the cloud gaming server in processing an input instruction and rendering a corresponding video, the network latency and latency in the client device. These may be respectively called processing delay, network delay and device delay [3]. Device delay occurs in two dimensions, once in taking user input and getting it ready to be transmitted to the server and once in decoding and playing a video frame. These delays may be reduced by using optimized hardware, for example, for encoding at the cloud gaming server and for receiving user input at the client. With development in cloud infrastructure (for example, optimally located edge servers and accessible cloud GPU resources) and advancements in input technologies (USB and touch controllers) sub 100ms latency in cloud gaming may be achievable, even in some wireless network scenarios [1].

In addition to latency, downstream bandwidth requirements for cloud gaming are challenging. High-end and graphics intensive games generate game-play videos which have complex and high graphical detail. Streaming such high quality video requires sizeable bandwidth. Lower available bandwidth can result in frame loss, lower quality per frame and even increase in latency, affecting QoE negatively. Encoding with aggressive compression is a possible solution but higher compression results in lower quality of the decoded video at the client which might be unacceptable for a good QoE. Depending on the game genre and game play pace, the effect of low available bandwidth and the resulting packet loss on QoE may be higher than latency [31]. Current cloud gaming services require high downstream bandwidth for even standard definition video. For example, GeForce Now requires a minimum link capacity of 10 Mbps for the service to even work [32] and PlayStation Now requires a minimum of 5 Mbps to work [33]. Keeping in view the world wide average broadband bandwidth of 7.2 Mbps in 2017 [34], GeForce Now is unplayable for an average global broadband user, while PlayStation Now is somewhat playable, provided the network conditions are constant and no other applications are communicating on the same broad band connection. Considering the above, cloud gaming may become more feasible if downstream bandwidth requirements can be reduced without sacrificing QoE significantly.

2.4 Literature Review

Cloud gaming is an actively researched topic as there are substantial commercialization avenues. The earliest literature in this direction is by Ross [35], in which the idea of remote rendering on powerful cloud resources and use of thin clients for gaming is illustrated. The case for cloud gaming is made on the basis of advancements in GPU technology and a prototype remote rendering technology by AMD and Otoy.

Lee et. al. [29] investigate the cloud-gaming friendliness of various genres of games from a playability perspective, playability being characterized in terms of response latency. Electromyography is used to analyze the players quality of experience. It is verified that all games may not be cloud friendly and cloud friendliness is dictated by a game's real-time strictness. The authors also develop a model to objectively compute the cloud friendliness of a game without doing a QoE user study. This model may be useful to cloud gaming service providers.

Shea et. al. in [4] discuss a generic cloud gaming framework applicable to the cloud gaming service Onlive. The challenges involved, particularly, interaction delay and video quality are enumerated. Interaction delays for Onlive are measured at different network parameters. A comparison is made between local gaming and cloud gaming. The results show that in addition to network latency, the processing time in cloud plays an important part in interaction delay. The processing time in the cloud is needed to execute game logic and to encode the resulting game play video. Further the authors objectively compare video quality of game-play video streamed from cloud servers with locally rendered game play. It is observed that video quality

suffers even with best case scenario of network latency and with increasing latency, the quality deteriorates further.

Claypool et al. in [36] consider the cloud gaming service Onlive from a network turbulence perspective. They observe that Onlive games have a high downlink data rate of 5 Mb/s with packet size around 1000 bytes, while the uplink data rate is much lower at 100 Kb/s with packet sizes around 100 bytes. Further they notice that downlink traffic characteristics do not depend on game genre, but uplink characteristics do. Further they notice downlink traffic of Onlive to be similar to downlink traffic of traditional online games, while the uplink traffic is somewhat similar. Lastly the authors find that the Onlive cloud gaming service adapts downlink data rate to link capacity but not to latency or packet loss and frame rate to link capacity and packet loss, but not latency.

Chen et al. in [37] compare latency performance of Onlive with StreamMyGame (SMG). Onlive, as mentioned earlier was a pioneering cloud gaming service, while SMG provides a server software for the user to install on their own hardware. The work is followed by [38], in which the authors consider Onlive and SMG again and explore which systems provide better Quality of Service (QoS) and which elements constitute a good cloud gaming system by considering network traffic, latency and video quality in more detail. Different games and game genres are considered for measurements.The authors decompose the latency into network delay, processing delay at the game server and play out delay incurred by the video player at the client. It is found that processing delays at Onlive are half of that of SMG. The authors speculate that this is due to extra compute power of Onlive servers and use of hardware encoders by Onlive. The authors also note that Onlive uses differential resource provisioning depending on the game genre. It is also observed that the uplink data rate is game dependent, while downlink data rate is not. Further, the total response delay is found to be dependent on scene complexity and size of the update region in a game scene and processing power of the server and resolution of output video. The authors note the similarity of their results with another study done by Claypool et al. [36] discussed above.

Jarschel et al. in [31] investigate user perceived QoE of cloud gaming. A representative group of casual and leisure gamers are asked to play games of different genres with different points of view (POV) over an emulated cloud gaming platform. The authors consider packet delay and packet loss as relevant parameters in cloud gaming QoE. The test subjects play the games on the emulated cloud gaming platform with different combinations of packet delay and loss introduced in the network path, after which they are asked about their QoE which is expressed as a mean opinion score (MOS). It is observed that with increase in delay, the user QoE decreases, although the decrease depends on the pace of the gameplay. Similar results are observed for packet delay, although for fast paced gameplay, the decrease in QoE with increase in packet loss is somewhat slower and lesser than in medium or slow paced gameplay. It is observed that packet loss affects user perceived QoE substantially more than packet

delay in medium paced games, while delay is more important for fast paced games. For slow paced games, there is no clear tendency as to which is more important: packet loss or delay, apart from content dependency. Further, it is observed that downlink packet loss affect QoE for all games more than uplink packet loss. An important metric observed is that a 120 ms delay may be tolerable.

Choy et al. in [39] study end user latency from a cloud gaming perspective in commercial clouds like Amazon EC2 by sending TCP measurement probes to a sample of users using BitTorrent. It is observed that 70% users within USA suffer a latency of 80ms which the authors deem unacceptable for cloud gaming, considering that latency affects QoE very adversely and that for gaming applications, there will be an inherent processing delay at the server. It is observed that with a considerable number of distributed, rather than centralized, datacenters 90% of the US population may be covered with a latency less than 80ms. The authors propose a hybrid cloud architecture of cloud gaming in which existing cloud data centers are integrated with Content Delivery Network(CDN) edges modified to provide compute resources.

Cai et al. in [26] interpret cloud gaming as gaming as a service (GaaS) and consider different architectures with respect to where the graphics rendering takes place. In remote rendering GaaS, the rendering takes place in the cloud or edge and the gameplay video is streamed to a thin client where the player interacts with the game. This is the cloud gaming architecture generally considered in this thesis. In local rendering GaaS, game logic is executed remotely but gameplay rendering instructions are sent to the client. This architecture is more akin to online gaming. It is observed by the authors that for such an architecture to be truly scalable and platform independent an instruction set has to be developed for remote rendering. In cognitve rendering GaaS, rendering load is distributed between the cloud and the client depending upon various parameters. This architecture is considered by the authors to be most suitable for cloud gaming.

Manzano et al. in [40] analyse traffic characteristics of two cloud gaming platforms active at the time: OnLive and Gaikai. It is observed that OnLive has a higher packet rate than Gaikai, particularly when considering graphics intensive games. The packet size of OnLive traffic is also larger than Gaikai. These observations indicate that OnLive requires higher network performance than Gaikai. They also observe that downstream packet size distributions of both platforms are bimodal with the minor modes being at 250 bytes and 1480 bytes for OnLive and Gaikai respectively. The major modes are found to be at 1400 bytes and 150 bytes respectively. From downstream inter-arrival times, it is observed that in OnLive while 40% packets have an inter-arrival time within micro-seconds, it may reach upto 5 milliseconds (ms). In Gaikai, 20% of the packets have inter-arrival times within microseconds and 50% packets have inter-arrival times around 1ms but it may be as high as 4ms. Upstream traffic from client to server traffic is also analyzed. From the cdf of packet sizes, it is observed that OnLive traffic has modes at 100 bytes, 140 bytes and 240 bytes. Gaiki traffic has modes between 50 bytes to 100 bytes. From inter-arrival times, it is

observed that inter-departure times show a mode at below 1 ms and then a uniform distribution ranging to 12 ms, for Gaikai and between 0ms to 8 ms with a uniform distribution and a mode at 9 ms.

Chuah et al. in [41] discuss cloud gaming from an energy and environmental conservation perspective. It is argued that cloud gaming accrues the green benefits of cloud computing, for example, economy of scale, better provisioning of resources etc. Energy conservation measures possible in the cloud gaming pipeline, like GPUs, encoders, energy optimizing the delivery networks etc. are enumerated. Further some cloud gaming platforms are discussed and a novel cloud gaming architecture is proposed. The novel architecture takes advantage of the GPU capabilities of current mobile devices, sharing rendering load between the cloud and the client.

Semsarzadeh et al. in [42] propose an approach to reduce latency in encoding of game play video in cloud gaming servers. It is proposed to analyze game engines/objects and enable information exchange to the encoder to make motion estimation faster.

Kämäräinen et al. in [1] investigate the possibility of achieving imperceptible latency in mobile cloud gaming. It is observed, by way of thorough measurements, that device based delays play a more important role in total response delay than network delay and placement of cloud gaming servers. The authors conclude that with well provisioned severs and a a short network path, delay below a threshold delay of about 100ms can be achieved even with current (top of the line) smart-phones and predict that device delays will shorten in near future.

Cloud gaming is a field of active research and there are many other publications in the field like: [43] which analyses performance of cloud gaming services from a QoE perspective, [44] which proposes an QoE optimization for cloud gaming using adhoc cloud-lets, [45] which analyses performance of popular thin client platforms in an online gaming context, [46] which discusses the future of cloud gaming and research avenues, [47] which introduces a cognitive platform for mobile cloud gaming which performs flexible resource allocation for improved efficiency of resource usage while delivering on QoS requirements and [28] which is a survey on recent publications in cloud gaming.

3 Video Streaming

3.1 Overview

A stream can be variously defined, but in the context of multimedia communications, it generally refers to a flow of data (e.g., audio and/or video) from a server to one or more clients. Video streaming refers to sending video over a packet network from a server to one or more clients where it is played progressively and concurrently with the data download. It is different from a video file download as at the client device the video is playing as chunks of it are being downloaded. Most of the cloud gaming platforms available today, including the platform used in this thesis, comprise of servers in the cloud which execute game logic, render game play video, encode the game play video and stream it to the cloud gaming client. Video streaming, as such, plays a critical role in cloud gaming.

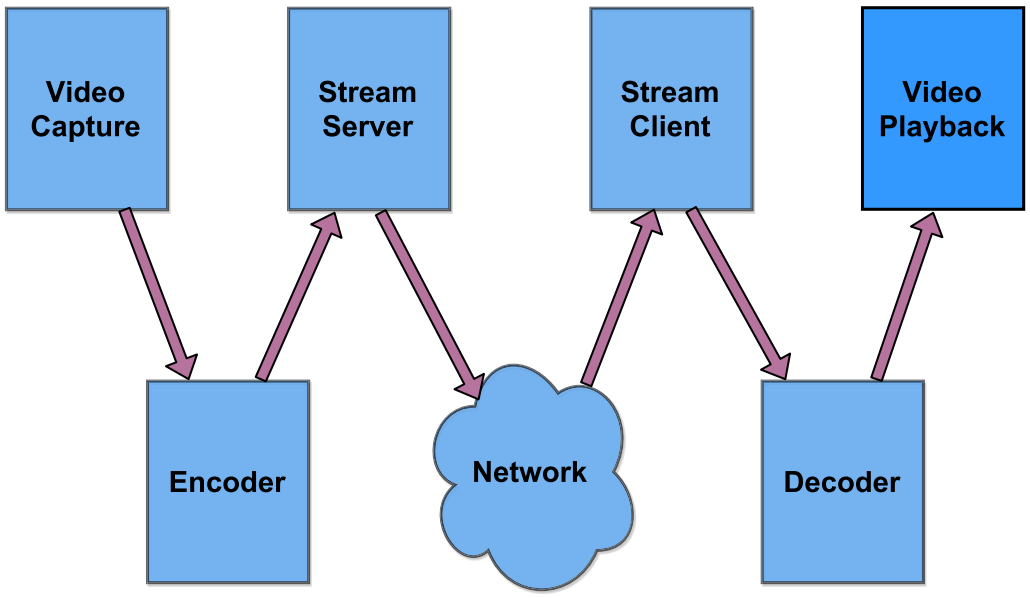


Figure 2: Video Streaming

Raw video, as captured by a capture device, has a large data size. For example, a raw uncompressed video of length one minute at a resolution of 1920x1080 pixels at 30 frames per second (fps) may occupy about 10 Giga-Bytes of storage. Such data sizes are too large for storage or streaming. To make processing video files more feasible, they are encoded. Encoding reduces the file size by representing the video in fewer bits than in its raw size. Encoding achieves this by exploiting spatial and temporal redundancies in the video. Encoding and the resulting compression can be either lossless or lossy. In lossless compression, no information of the video is lost during encoding. In lossy compression some information is lost during encoding. The process and system of streaming is illustrated in Figure 2. The capture device

captures the video and passes it over to the encoder. The encoder compresses the video to reduce its size and adds meta-data like time stamps and decoding parameters to it. The video is then sent to the stream server which breaks down the video into a stream of packets which are sent to the stream client over a packet based network as a stream of packets. The client receives the video stream, arranges the packets in order, and forwards the reconstructed data to the decoder. The decoder attempts to decode the streamed data into raw video frames which are played by the playback device.

The packetization of the video at the stream server and its reconstruction at the stream client is dictated by the streaming protocol stack used. The stream server sends the video over a network to one or more stream clients for playback. The key difference between a simple file transfer and streaming a video is that video is played in real time at the client while it is being sent over by the server. The stream server breaks down the video into packets to be sent over a transport protocol like TCP or UDP. On top of the transport protocol the stream server uses a streaming protocol which enables the streaming client to put back the video together properly. There are various protocols that can be used by the stream server, including but not limited to HTTP Live Streaming (HLS), DASH, RTSP, etc. The protocol dictates the arrangement of video frames into the data packets of the transport layer protocol and adds meta-data, for example the sequence numbers of data packets. These data packets are then sent over a network to the client.

The video is streamed to the client over a packet data network. In practice it is the Internet which comprises disparate networks with disparate physical technologies like Ethernet, wireless, fiber etc. As such the conditions and capacities of the network in between the server and the client are non-deterministic. These conditions dictate the quality of the video streamed as well as the QoE of the user. For good QoE, the encoder, the stream server, or the stream client must take the network conditions into account. The stream client receives the video from the stream server over the network. Typically the client initiates the stream session and asks the server to stream the video. The server transmits the necessary information needed to receive and rearrange the data packets into a decodable video.

The decoder receives the reconstructed video and decodes it. Typically, meta-data needed to decode the video is included at the start of the video. The decoder uses this information to decode the encoded video into a format that can be displayed on a screen. Video playback is done typically on a screen where each pixel is reproduced as it is represented by the data received from the decoder. Screen resolution is the number of independently controllable hardware image elements (screen pixels) comprised in the screen. Another related factor is the pixel density of the screen typically expressed as pixels per inch (ppi). The screen resolution is different from the video resolution which is the number of picture samples (video pixels) in a frame of the video. The resolution of a video and resolution of the screen on which it is displayed and also the pixel density of the screen all play an important role in QoE

of the viewer. A video of high resolution displayed on a screen with low resolution or ppi, a video of low resolution played on a screen with high resolution, or ppi both may result in unsatisfactory QoE.

Video capture is done typically by a video capture device, for example a camera. In scenarios, where the video is computationally rendered, e.g by a game engine, the video capture may be done by hooking into the GPU doing the rendering or by copying the rendering devices' fame buffers periodically. The encoder is, for example, installed on a work station or a cloud server and the stream server is a web server, while the stream client is a web client on a consumer computer, and the decoding and playback happen on the same device. In some cases encoding and decoding may either entirely or partially be done by specialized hardware, however, software implementations of the encoders and decoders are very common. The boundaries between the image capture device, encoder and web server are blurring with services like Facebook Live, Twitch etc coming into vogue. With such services, cloud based servers replay (sometimes also transcode) the content as captured and encoded by a consumer grade device like a smart-phone or a gaming computer. The concepts and stages involved in video streaming, like video capture, video coding and streaming protocols are explained in more details in subsequent subsections.

3.2 Video Capture and Representation

Video capture is essentially capturing a series of images in quick succession which when played back in the same sequence and a similar pace are perceived by the human eye as moving scenes. Video is captured using a camera. The camera shutter speed is set to the desired capture frame rate and the sensor captures images at that rate. The shutter may be a physical shutter or a virtual one, in either case its frequency decides at what rate images are captured by the sensor. The sensor may be a Charge coupled device (CCD) Sensor or a Complementary Metal Oxide (CMOS) Sensor. These sensors comprise an array like arrangement of sensor elements which generate an electric response when light falls on them. This response is representative of the image from which the light is falling on the sensor. In the array of sensor elements, each sensor element captures a part of the image or a "picture element"—pixel in short. Greater the number of these sensor elements, higher the resolution of the captured image and hence the video. These sensor elements are also called pixels (camera pixels) because each sensor element captures a pixel of the image. In color cameras, which is majority of consumer cameras, each sensor corresponding to a pixel may further comprise more than one pixel, each designed to capture a different color. A typical pixel may comprise a sub-pixel for red component of the light and a sub-pixel for blue and one or more sub-pixels to capture the incident green light. The electric response of these small sensors are sampled, quantized and digitized. The quantization and digitization depends on the format in which the raw image is to be converted. The format defines the number of bits used for each pixel and hence the need for quantization and digitization. However, initially the capture

may be made in a format which retains the most information and from which other formats can be obtained by down scaling/sampling. The capture device uses a color space to represent the image captured by the sensor. A color space is a specific organization scheme of colors to allow representation and reproduction of the color gamut perceived by the human eye. Two popular color spaces are the RGB (Red Green Blue) and YCrCb (Luma, Chroma red, Chroma blue). RGB preserves image quality while YCrCb takes less storage space. Color space conversion between the two is easy, however, the reproducibility depends on absoluteness of the base color space (usually RGB). It should be noted that although, color space is different from the format in which a raw video or image is stored, in practice they are somewhat linked. In RGB color space, a color is represented as a combination of levels of red, blue and green in it. While in YCrCb, a color is represented as a combination of levels of brightness (Y), and two chrominance values, Cr and Cb. YCrCb color space is commonly used for digital applications. To represent an image digitally color sub-sampling may be used. Color sub-sampling refers to using fewer samples for color information than luminance information. This is possible because the human eye is more sensitive to brightness than to color. Some commonly used color sub-sampling schemes are given in Table 2. YUV is an older color space used in analog systems. It is close to the YCbCr color space and the terms tend to be used interchangeably, particularly given that many raw formats for image or video use YUVxxx as the file extension, where xxx is the subsampling used.

YUV 4:4:4	YUV 4:2:2	YUV 4:2:0
Typically 8 bits per Y, U, V plane No horizontal subsampling No vertical subsampling	4Y samples for every 2U and 2V 2:1 horizontal subsampling No vertical subsampling	4Y samples for every 2U 2:1 horizontal subsampling 2:1 vertical subsampling
24 bits/pixel 8 bits/sample	16 bits/pixel 8 bits/luma sample	12 bits/pixel 8 bits/luma sample

Table 2: Color subsampling schemes with 8 bit color depth [5]

3.3 Video Coding

Raw video, as captured by a camera is quite large in size. Raw video sizes are increasing with the increase in resolution of professional, even consumer grade cameras. To make video sizes manageable for storage and streaming, the videos are encoded before storage or streaming, and decoded before playback. This encoding and decoding process constitutes the area of image and video coding. There are many image and video coding standards and schemes. Some encoding standards just specify the syntax of the data while others also define compression algorithms or targets. Some of the standards are H262, H264 and H265 which have been developed by joint video team (JVT) of ITU-T Video Coding Experts Group (VCEG) and

ISO/IEC JTC1 Moving Picture Experts Group (MPEG). Some other encoding schemes have been developed by private organizations, for example VP9 and VP10 by Google.

A video coding scheme includes an encoder and a decoder. A video encoded by a particular encoder should be decode-able by the corresponding decoder. Encoding is able to compress the file size of a video by exploiting similarity or correlation of samples within an image or within multiple images. Even though the actual algorithms of compression differ between encoding standards, there are some broad common operations involved. At a high level these steps are: analyzing spatial similarities between the samples of an individual frame and temporal similarities between frames of different times for example consecutive frames. These similarities are expressed as differences mathematically and transformed into a more compact form. The transforms are then quantized and digitized. The resolution of quantization typically depends on a parameter called quantization parameter. The digital representation is further reduced in size using entropy coding. The resulting data is then filtered and put in a "container" files. Video decoding typically includes the reverse of these steps to extract raw video from encoded video. These steps are discussed in some detail below, particularly with reference to x264, an open source implementation of the H264 video standard.

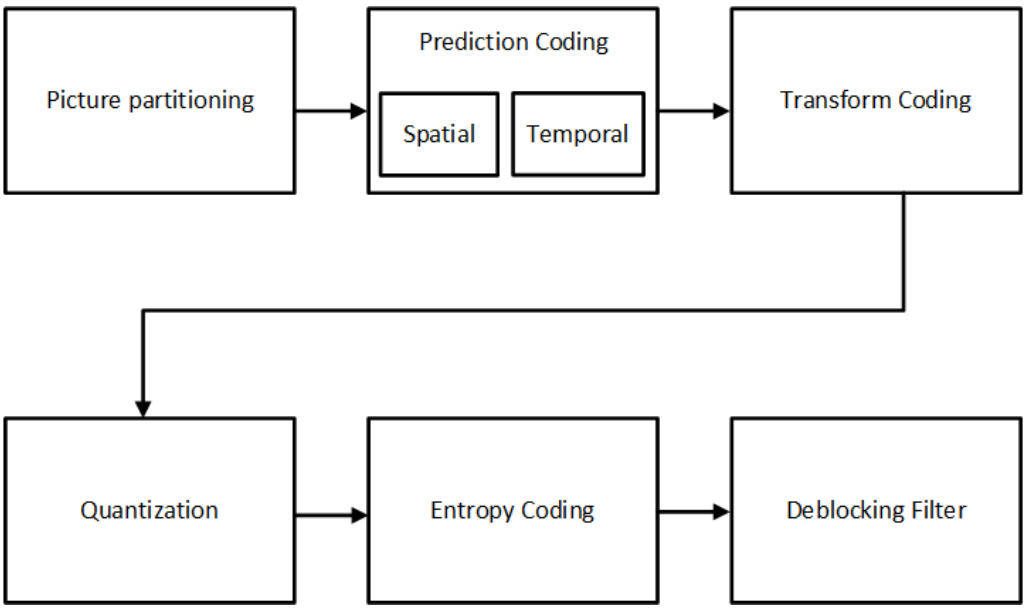


Figure 3: Video Encoding

Figure 3 illustrates the high level steps involved in encoding a video.

1. **Picture Partitioning**: At the start of the encoding process, each picture is divided up into groups of samples(pixels) on which encoding operations take

place. These groups are called blocks and typically a few blocks together are called macroblocks. Macroblocks are the fundamental units of pictures for the purpose of encoding in H264. As an example, a macroblock for H264 may be 16x16 samples of luma and, if 4:2:0 sampling is used, 8x8 samples of chroma. In modern encoders like H264 and H265, the size of the basic coding block may be adjusted. A picture may be partitioned into a few slices each consisting of some macro-blocks. Each slice can be encoded independent of other slices, thus enabling parallel encoding and decoding. In H265, the concept of macroblocks is replaced by Coding Tree Units, which may be up-to 64 x 64 samples in size. A coding tree unit is further divided into coding tree blocks which can be divided into coding units. The coding unit may be further divided into prediction units. This hierarchical partitioning of coding blocks results in higher compression, but the fundamentals of coding mentioned here still apply. H265 also allows partitioning a picture into tiles, which are rectangular groups of macroblocks for even more parallelism.

2. **Prediction Coding:** Prediction coding is at the heart of data compression in video encoding. The core idea is to store/transmit only differential information, rather than all the information representing a picture. In prediction coding a mathematical error is calculated between the original signal and a predicted signal. The error is called the residual. Prediction coding of two types is concurrently used in modern encoders:spatial and temporal. In spatial prediction correlation between samples in the same frame/picture is exploited, for example macroblocks within a frame (or a slice or tile) are represented as a residual between each original macroblock and a corresponding predicted macroblock. The predicted macroblock is one which is already available to the decoder and is usually a previously decoded macroblock. In temporal prediction correlation between samples/macroblocks of different frames is exploited to reduce the amount of data required to represent the current macroblock. For example, in many situations, the difference between the current frame and previous frame may be only that of a few macroblocks changing or some macroblocks "moving" corresponding to the movement of a subject in the video. Instead of re-representing the whole frame only the samples that changed may be encoded, or the macroblocks which have moved may be represented as "motion vectors". Based on the prediction coding used, a frame may be an **I** type frame, a **P** type frame or a **B** type frame. An **I** frame is an intra coded frame. It can be reconstructed only from the data present frame. A **P** frame (predicted frame) is more compressed as it is represented on the basis of previous frames as well. A **B** frame (bi-directional predicted frame) is predicted both from previous frames as well as future frames, as such it is the most compressed.

3. **Transform Coding:** The residuals of original macroblocks and predicted macroblocks are representable as a matrix of coefficients for each frame.This matrix is transformed mathematically to be represented in a more compact form.

Various transforming techniques have been explored including discrete Fourier transform and discrete cosine transforms. Currently x264 based encoders use an arithmetic transform which closely resembles discrete cosine transform.

4. **Quantization:** The transformed matrix is quantized to further represent it in a compact form. This step is where information loss takes place. The granularity of quantization decides how much information is kept and how much is discarded. In x264, a parameter "Quantization Parameter" QP decides the granularity of quantization. QP may be either set by the user or the encoder may set it automatically depending upon other parameters. Higher the QP, more information is lost irreversibly and higher the compression. There is a trade-off between quality and compression efficiency which dictates QP.

5. **Entropy Coding:** Entropy coding further reduces the data size by exploiting the correlation between binary data obtained after quantization. Repeating patterns are represented in terms of symbols. This step is a loss less step of video coding.

6. **Loop Filter:** Since the unit of video coding is a macroblock, adjacent macroblocks may have somewhat different values of luma and chroma and the decoded image may appear "blocky". To prevent this a loop filter is applied at the end to "smoothen" these transitions across macroblock borders, in the process further compressing the video.

Video encoding is a very mature field and as such is not discussed in depth here. For more information the reader is referred to [5] and [48] among many other excellent books. An excellent overview of H264 is presented in [49] and H265 is over-viewed in [50]. Xu et, al survey state of the art video coding approaches in [51].

3.4 Streaming protocols

Video streaming has evolved from simple but sparsely used file download based streaming to adaptive bitrate streaming which constitutes more than 70 percent of the global Internet traffic [52]. The most primitive methods to view video stored at a remote location, typically a web server, were file download methods. In such a system the client, a web browser, requests the media object, downloads it completely and hands it over to a media player. An improvement over this is the progressive download method, wherein the web browser downloads a meta file which it passes to the media player, the media player orchestrates a TCP connection with the web server and starts the download of the media object. As soon as a part of the playable media is downloaded, the media player starts playing it. The download continues concurrently with the media being played. This method has some shortcomings: there is not rate adaption which may cause frequent stalls, control over playback is

limited and the download continues progressively without regard to the fact that the viewer may not consume the complete media object. Current HTTP based streaming technologies are more intelligent and can adapt to network conditions and have richer control gamut for the viewer.

Adaptive bitrate streaming refers to streaming video content while adapting to changing network conditions to provide a smooth viewing experience. In adaptive streaming the quality of video is changed with change in network conditions.Typically the change in video quality is temporal and reactive to network conditions such that the output video bitrate provides the best possible viewing experience at those network conditions. For example a video stream may begin with low quality to provide a fast playback start-up at the client. If the network conditions are right, for example, the link capacity is high and the delay is low, progressive frames may be sent with higher quality. The quality of streamed video may reach a plateau wherein the highest quality video for given network conditions is streamed. The quality may lowered when there are instances of delay and/or congestion. This is temporal adaptive streaming. In spatial adaptive streaming, the quality of certain regions of the frames is reduced or increased as the network conditions change. For example at lower link capacities, pre-determined regions of interest (RoI) may be kept at a higher quality while other regions are streamed at a lower quality to make the best use of available network resources while maintaining an acceptable viewing experience.

3.4.1 HTTP Adaptive Streaming

Adaptive streaming puts constraints on encoding. Typically, for adaptive streaming, the video is encoded as multiple streams, each having a different resolution. Each stream consists of chunks of the video. These chunk boundaries as well as identifiers are synchronized. This allows chunks of one quality to be played followed by chuck of different quality without affecting the linearity of play back—only quality changes. There are several HTTP based propriety adaptive streaming protocols, such as Apple HLS, Adobe HTTP Dynamic streaming and Microsoft Smooth Streaming. These protocols are very similar in that the server stores multiple resolution versions of the same video wherein each version is stored as a series of HTTP compatible chunks(smaller files) which are interchangeable. These protocols are stateless on the server side, but the client maintains the session state. At the start of a streaming session and sometimes also subsequently, the client receives a meta data file which describes the available resolutions (streams) of the video, the arrangement and location of HTTP chunk files for each stream etc. Based on this meta data and the rate adaption method in use, the client decides what resolution chunk to request next from the server. These protocols, although similar are not inter-operable. MPEG-DASH [9] is a standard which attempts to provide an inter-operable standardized streaming mechanism. MPEG-DASH specifies only format of the meta data file called media presentation description, leaving the rate adaptation and codec implementation open to service providers. HTTP based adaptive streaming has the benefit that

existing web infrastructure can be leveraged for video streaming. Also HTTP traffic is generally allowed through firewalls and does not raise any red flags compared to, for example, UDP traffic. The encoding requirements of HTTP based adaptive encoding may be challenging in live streaming scenarios. In live streaming scenarios, this entails re-encoding the video generated by a capture device at multiple resolutions which introduces delay in the whole process. This delay may be acceptable in passive viewing use case scenarios such as watching a sports event or an entertainment event. However, in real time interactive applications, such as cloud gaming, the delay involved in DASH and DASH like standards and protocols is unacceptable and hence these standards can not be used.

3.4.2 Stateful Streaming

For low latency interactive applications, like cloud gaming, session based stateful protocols are more suited. In such protocols, the control and data connections are separated. An example of such a protocol stack, used in the cloud gaming platform this thesis uses, is Real Time Streaming Protocol (RTSP) with Real Time Transport Protocol (RTP) and RTP Control Protocol (RTCP). RTSP sets up the connection and is used for control of the streaming session, such as start, pause, rewind etc. but does not itself perform transport functions [53]. RTP [54] is the transport protocol which carries the audio and video streams. RTCP, used in conjunction with RTP, is used to monitor the stream and its QoS and is also used to synchronize multiple streams if present. Adaptive rate adaptation can be added to RTSP/RTP/RTCP based streaming at the encoder level using for example Scalable Video Coding (SVC) extension of H.264 standard [55]. RTSP/RTP/RTMP can use either TCP or UDP as the underlying transport protocol. When using UDP, there may be issues with traversing firewalls. For low latency applications, RTSP with RTP/RTCP is more desirable as playable video can be transmitted in (play out) lengths smaller than the HTTP file chunks used in HTTP based adaptive bitrate streaming methods.

Another classic stateful streaming protocol is the Real-Time Messaging Protocol (RTMP) [56]. RTMP is a partly proprietary protocol developed by Macromedia now part of Adobe which supports multiple parallel streams of audio, video, data and associated meta-data. It was widely used before HTTP based protocols like HLS took over. RTMP uses a persistent TCP connection to establish a streaming link between a Flash player and a video server. The client and server dynamically decide the size of stream fragments. Stream fragments from different streams, like audio and video, may be interleaved. RTMP has a low packet overhead and multiple versions which extend its functionality. There are versions for security, tunneling over HTTP and for using UDP as the transport protocol.

4 Foveated Streaming

Foveated streaming in this thesis is implemented using gaze location data. Gaze location data is obatined using an eyetracker. In this chapter eye and gaze tracking are discussed from a foveated video coding and streaming perspective, followed by a discussion on foveated video streaming.

Human eye is responsible for the human sense of vision. It is the interface between the the physical world of light and the neurological perception of it by the human brain as images. The eye has five main components which help in imaging: the lens, the cornea, the iris and the retina and the optic nerve. The cornea and lens collect the light reflected from physical objects with the iris controlling the amount of light entering the optical cavity inside the eye. The retina contains photo-receptors which receive this light and generate a corresponding neural response which is sent to the brain for processing via the optic nerve.

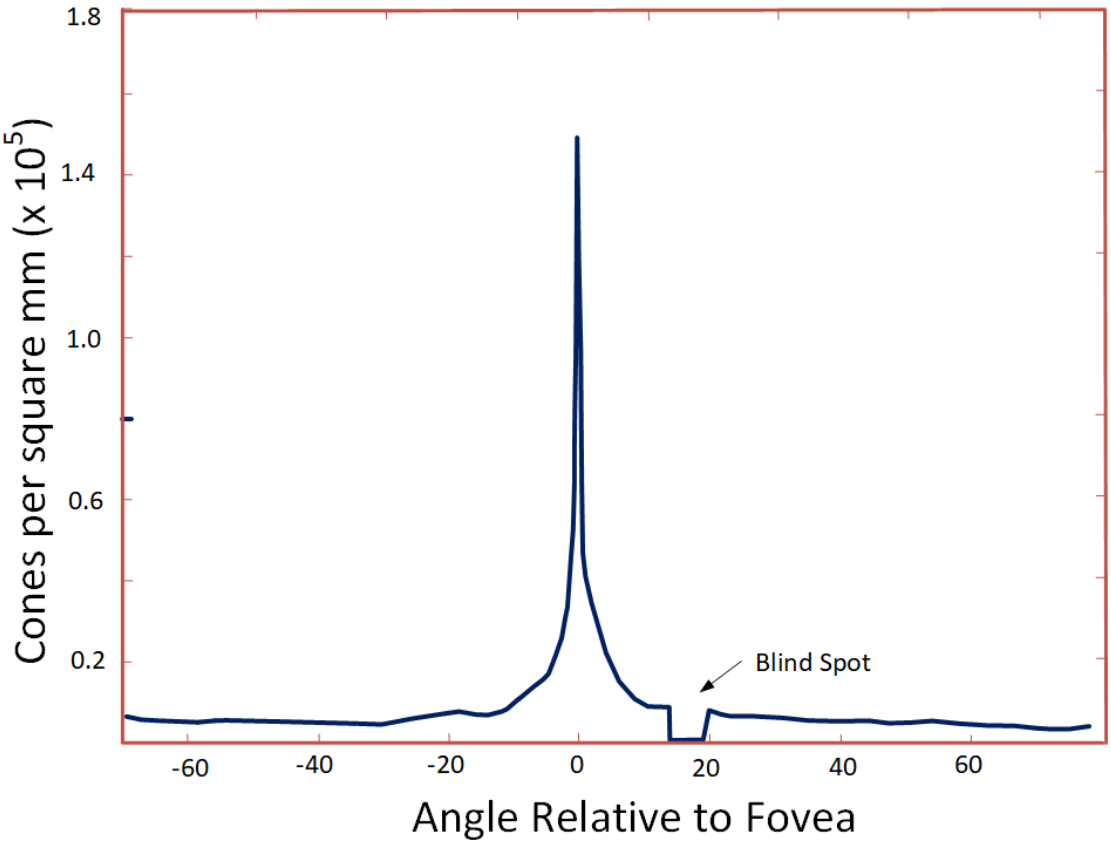


Figure 4: Concentration of cone cells in human eye.

4.1 Human Visual System

Human eye perceives visual scenes with different acuity depending on where it is focused. The eye senses light with the help of the retina which covers the back of the eye. The retina is covered with two types of light sensitive cells, the rods and the cones. The rods are responsive at low light levels while the cones are responsive at higher light levels. Cone cells are also responsible for chromatic vision. The distribution of cones in the retina is not uniform. The center of the retina, which is opposite to the lens has the highest density of cones. This region of the retina is called the fovea centralis or fovea in short. The density of cone cells decreases exponentially with distance from the fovea [57].

4.1.1 Foveation

Figure 4 illustrates the concentration of cone cells in the human eye relative to the angular distance from the fovea centralis. The concentration of cone cells drops exponentially with the angular distance from the fovea, as such human vision exhibits non uniform acuity. The visual acuity corresponds to the concentration of cone cells in the retina. Consequently, parts of a scene directly in front of the fovea, that is at an angular distance of 0^o from it are sampled and therefore perceived with the highest acuity by the human eye. The more the angular distance of a part of a scene from the fovea, the lesser is the visual acuity with which it is perceived. This phenomenon is called foveated vision or foveation. One can observe foveation by focusing their gaze at a word straight ahead in a document covering a substantial portion of their visual field and trying to read from the periphery. As can be noticed, the peripheral region appears blurred and therefore illegible. The foveal region occupies an angular region of 2^o of the human visual field [10]. Thus only a small part of the human visual field is sampled and perceived with high acuity. The area of the region which is perceived with the highest acuity depends upon the viewing distance. At a viewing distance of d, an approximately circular region with diameter of $d = 2 \text{ x } \pi/180$ is perceived with the highest acuity.

4.1.2 Eye Movements

To process a visual scene, our eyes move around a visual field. This is because of foveation [58]. To process an entire visual field, the fovea (more precisely the visual axis) has to be moved around the field, so that salient parts of the scene may be fixated upon by the fovea for sampling. Eye movements are also necessary for cognition and attention, so to perceive new scenes, the the head and eyes move. In addition to using the head to orient the visual axis coarsely, there are four types of eye movements. These are saccades, smooth pursuit movements, vergence movements and vestibulo-ocular movements [59]. Saccades are quick ballistic movements that the eye makes in between fixation periods. Sampling takes place during the fixation periods wherein the eye stays stationary. Saccades occur when the point of visual attention, for example a visual stimulus, changes location in a scene. For example,

while reading, the eye constantly makes saccadic movements to the next word. It should be noted that during fixation periods, the eye is not completely steady and minor eye movements called micro-saccades occur. In smooth pursuit eye movements, the eye smoothly follows a visual stimulus. This occurs when the target of visual attention is moving. Vergence movements occur when the eyes adjust their fovea with targets at different distances from the eyes. Such movements are different in that the eyes may move independently during such movements. Vestibulo-ocular movements serve to stabilize the eyes, more specifically the retina, with respect to the external world, for example by compensating for head movements. This prevents images from sliding on the retina as the head moves. Of these saccadic and smooth pursuit movements are of interest for foveated video streaming.

4.2 Gaze Detection and Tracking

Eyes are one of the main sensory input organs in humans. Eyes are also a powerful medium of emotive and cognitive expressions in humans, ranging from expressing needs and desires to cognitive processes. Study of eye and gaze behavior is therefore important from multiple perspectives, including psychological, neurological and opto-neurological etc. Therefore tracking eye and gaze has numerous applications, for example: assitive technologies for the differently abled, e-learning, driver-assistance, security, authentication and for human computer interaction.

Human gaze has been studied and tracked as early as 1908 [60], but reliable non-intrusive and cheap gaze tracking solutions have only recently begun to emerge. Various techniques of gaze detection and eye tracking have been tried over the years which can be broadly divided into two categories: sensor based and computer vision based [61]. Sensor based gaze detection typically involves sensors on the users body (head or eye) to detect changes in electric fields with eye movement. Computer vision based detection is non-intrusive and typically even non-contact. Current gaze tracking hardware, particularly consumer grade hardware, is computer vision based. In computer vision based tracking a real time video of the users' eye or eyes is analyzed using image processing techniques to detect or localize the eye and deduce gaze information.

Figure 5 illustrates a generalized computer vision based gaze tracker. The gaze tracker takes continuous image data (video) and detects the eyes which are then tracked. Simultaneously the head pose is estimated. From the estimated head pose and the tracked eyes, the gaze location/orientation can be calculated. Applications may use the eye and gaze location to provide services depending on the use case.

Computer vision based eye detection may use different models of the eye to detect eyes in an image. The models may be simple elliptical models to complex 3-D models. Further, models may be based on features of the eyes, rather than the shape. One common approach is using pupil detection. Pupil detection is attractive because pupils and iris are typically darker than the surrounding regions. Eye models for

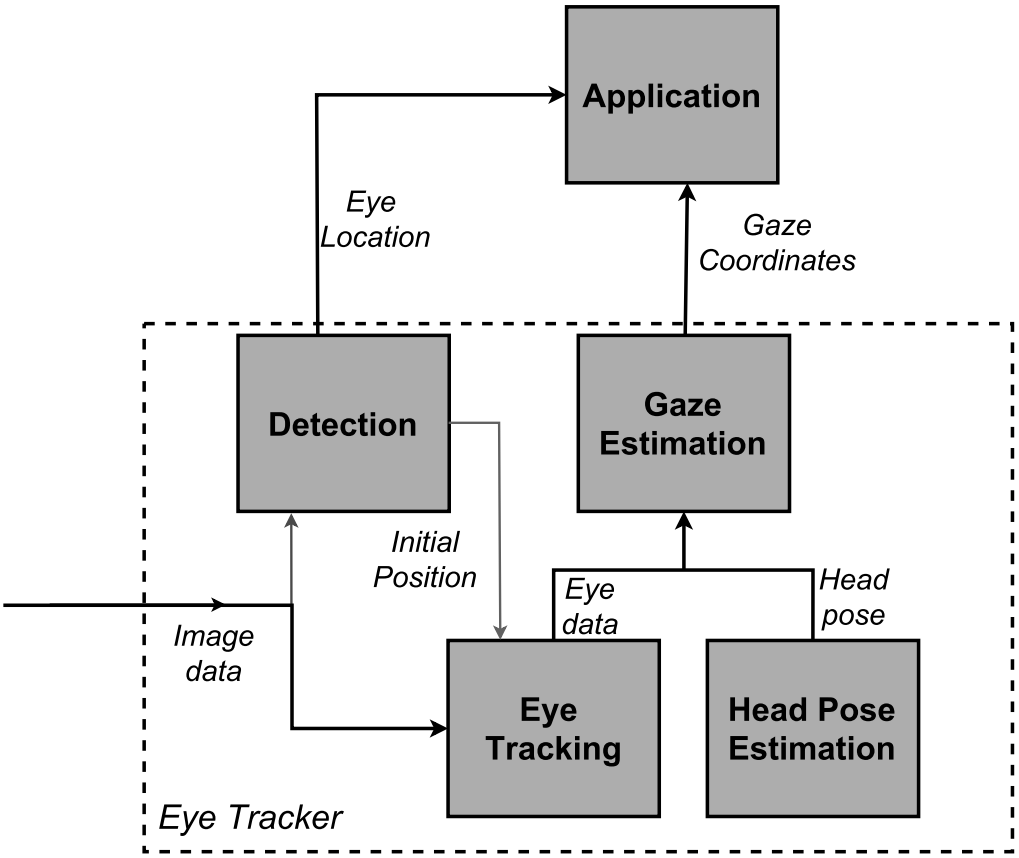


Figure 5: Generalized computer vision based gaze tracker [2]

detection may also be appearance based, wherein appearance is the photometric appearance characterized by color distributions and filter responses. There are hybrid models which combine features of one or more of the previously mentioned models. By far the most common method of eye detection and gaze tracking in modern day is using active IR illumination and pupil detection. [2].

IR illumination based eye detection and gaze tracking works by using one or more IR or near-IR light sources to illuminate the subject, while one or more cameras sensitive to that range record video of the scene. Depending upon the position of the light source(s) with respect to the camera, the pupil either appears darker than its surroundings or brighter than its surroundings, making detection easier. Changes in head pose change the reflection pattern and brightness of the pupil. Often this method of detection may incorporate features of other models and also head pose estimation for more robust eye tracking [61] [2].

Gaze estimation incorporates eye tracking to provide an estimate of where a subject is looking. Gaze refers to the direction in which the eyes are looking. Assuming the subject is looking at a screen, a gaze location on the screen may be estimated. Eye movements can be divided into fixations, saccades and smooth

pursuit. In fixations, the gaze lingers at a small area for a threshold of time typically 80-100 ms [2]. Saccades are sudden jump like motions of the eyes, consequently the gaze, typically between fixations. In smooth pursuit, the eyes and consequently the gaze follows a moving object. A subject may change the direction of their gaze by moving their head as well. Hence, in addition to tracking eyes, head pose estimation is necessary to obtain a sufficiently accurate estimate of the the gaze direction. Head pose information is typically used implicitly by gaze tracking models. State of the art gaze tracking solutions overwhelmingly use active IR illumination to illuminate the eyes of a subject whose gaze is being tracked.

Gaze tracking may be feature based wherein gaze estimate is obtained from features of the eye like contours, corners and the features of eye image captured by the gaze tracking camera. Feature based models may use an eye model or they may be regression based. When an eye model is used, gaze direction is calculated from a geometric model of the eye. The location of gaze is then estimated from the intersection of gaze direction and the object in view. Regression based models map image features to gaze using some parametric or non parametric function.

Gaze tracking may also be appearance based which use image contents, rather than extracted features, to estimate screen co-ordinates of the gaze location. Gaze tracking may also employ natural light illumination rather than IR illumination. Such approaches can be prone to error as conditions of natural illumination change and visible light sources may shine directly into the detection camera.

In IR illumination based eye tracking, the light reflected by cornea, called the corneal reflection, is primarily used to estimate the gaze location with the help of an eye model. This approach, called Pupil Central Corneal Reflection (PCCR), is the most widely used remote and non intrusive method of gaze tracking at present. In PCCR, the eyes of the eye tracking subject are illuminated with IR or near IR light and a bright pupil or a dark pupil is used to detect the eye. Bright pupil and dark pupil refer to the IR illumination setup used. If the IR illuminator is placed close to the image sensor, the pupil image appears bright–hence bright pupil. On the other hand, if the IR illuminator is placed away from the image sensor, the pupil image is dark—hence dark pupil. In addition to extracting the pupil image, the reflection of the IR light from the cornea, called glint, is also extracted. The eye axis, i.e. gaze location, is estimated based on the relative position of the pupil center with respect to the glint. The pupil center and glint provide a pupil-glint vector which with the help of a calibrated gaze mapping function and an eye model gives the fixation point or gaze location. PCCR based methods may use head compensation to take head pose into account as seen in commercially available gaze trackers [62].

All IR illumination based eye tracking solutions need calibration for each subject. During calibration, a subject is prompted to fix their gaze at pre-specified calibration points on a screen while images of the eye are continuously collected. From the images pupil-glint vectors are extracted and together with the known location of

calibration points a pupil-glint vector to gaze location mapping is developed [63]. More information like size and color of subject's pupil, shape of the eye, distance between eyes etc. may also be collected during calibration to improve or customize the eye model used [62].

4.3 Foveated Video Coding

Foveation can be exploited to reduce bitrate of encoded video. If video is encoded such that the quality of the video matches the sampling acuity of the human eye, considerable savings in video bitrate can be made. However, there are considerable challenges. The encoder must know where the location of the viewers' gaze. This is challenging in applications like video broadcasting and streaming where the number of viewers can be large and also the video content may be precoded. If the video content is precoded, it is resource intensive and counter-productive to re-encode it for each viewer. A proposed solution to the issue is region of interest (RoI) encoding wherein the video is pre-analyzed, for example with image processing techniques, to identify regions of interest in each frame where the viewer is likely to focus their gaze. These RoIs in each frame may then be encoded with higher quality. In case of live streaming video, there are two challenges for foveated video encoding: gaze information of the viewer and scalability. In a live stream scenario the number of clients may range from zero to thousands. Gaze information of a viewer is not tracked by state of art viewing clients for example desktop/laptop computers, TVs and mobile devices. This however is a surmountable challenge as commodity eye trackers are becoming cheaper with time due to advancements in chip technology and computer vision techniques. Even commodity cameras, like web-cameras may be used to track gaze location of the viewer [10]. The accuracy of dedicated IR based eye trackers is better as of present. Solutions like Microsoft's Windows Hello [64] [65] which require advanced camera hardware may support eye tracking applications as well. The main challenge in live streaming is again scalability, as each viewers' gaze may be focused differently, needing one encoder instance per viewer. Figure 6 illustrates a screen capture of a game video encoded with quality corresponding to foveation. The user's gaze is fixated around the in-game avatar.

In real time foveated video encoding where the RoI is unknown or un-extractable, real time gaze location is a critical parameter as the region around the gaze location is encoded with high quality/bit rate. Real time gaze location has two aspects, the extraction of gaze location information should be real time as should be the delivery of gaze location data to the encoder. In a cloud gaming context, the delay challenge is to be addressed for game play video delivered to the cloud gaming client and control information delivered to the cloud gaming server as well.

(a) Normal Encoding

(b) Foveated Encoding

Figure 6: Foveated Video Coding. The user gaze is at the overlay bubble.

4.4 Related Work

Foveated video coding has been researched for quite some time. In fact there have been surveys on the proposed techniques as early as 2006 [11] while techniques for foveated video coding have been suggested as early as 1980s and 1990s [66], [67]. Although actively researched upon, there have been few implementations of foveated encoding beyond research labs in part due to lack of commodity eye tracking hardware. There has been a renewed interest in foveated video coding in recent years due to ubiquity of mobile devices , explosive growth of video streaming on them and the bandwidth constraints of telecom networks.

Itti in [68] proposes a neuro-biological model of visual attention for compressed video encoding. In the proposed scheme, the whole video to be encoded is analyzed for "salient pixels". Wang et al. in [69] propose a foveation scalable video coding algorithm which aims to provide the best possible quality at any arbitrary bitrate in terms of foveated visual quality. The same authors had earlier, in [69] proposed an optimized rate control scheme which maximized foveal signal to noise ratio—a metric which measures quality taking foveation into consideration introduced by the same lab in [70].

A foveated video streaming implementation is introduced in [10] by Ryoo et. al. A tile based approach is used wherein each frame is divided into multiple tiles and the tiles are assigned quality on a three tiered system. Each frame is divided into three concentric regions centered around the current gaze location. The innermost region is assigned the highest quality, the middle region is assigned a middle quality level and the outermost region, which is the largest region is assigned the lowest quality. This implementation uses commodity web cameras to detect gaze.

Foveated video streaming for cloud gaming has been investigated earlier as well. Ahmadi et. al in [71] propose video encoding based on a game attention model to reduce video bitrate without significant perceivable quality loss. The game attention model estimates the importance of each macroblock in a video frame which the encoder uses to allocate bits to the macroblock (in other words, control the value of QP for the macroblock). The authors use a Support Vector Machine trained on database of eye tracking data to develop their game attention model. This method, needs eye tracking data for each game before being deployed and can reduce video bitrate by 25%.

Mohammadi et. al in [27] propose an object based coding approach of game play scenes, using the MPEG 4 Part 2 Binary Format for Scenes (BiFS), instead of image based coding. The fact that game engines render scenes object-wise is leveraged. The implementation uses a gaze tracker, as used in this thesis, to gauge which objects are in a players center of attention and the server sends a high quality version of the object. This implementation requires the game engine to be able to output objects to the video encoder (BiFS capable), which would mean either ground up design with this capability in mind or modifications after development. Consequently, this approach may not be feasible for off the shelf games.

5 Design and Implementation

In this thesis a prototype of cloud gaming with foveated video encoding is designed and implemented. An off the shelf gaze tracker is used in the prototype. It is integrated with an open source cloud gaming platform. In addition to control data, the client is configured to forward the gaze data obtained from the gaze tracker to the cloud gaming server. The cloud gaming server is configured to encode game play video in a foveated manner with the foveated area being defined according to the gaze data. The prototype is shown in Figure 7. The prototype consists of a generic laptop with Microsoft Windows running on it as the client and a Linux workstation as the cloud gaming server connected with GbE link. A consumer gaze tracker—Tobii 4C eye tracker is installed on the client machine. GamingAnywhere (GA) is used as the cloud gaming platform. GA is an open source cloud gaming platform which provides client and server software for various operating systems and has support for different encoders and decoders. For the prototype the x264 encoder, which is an open source implementation of the H264 standard, is used. These tools are further described below.

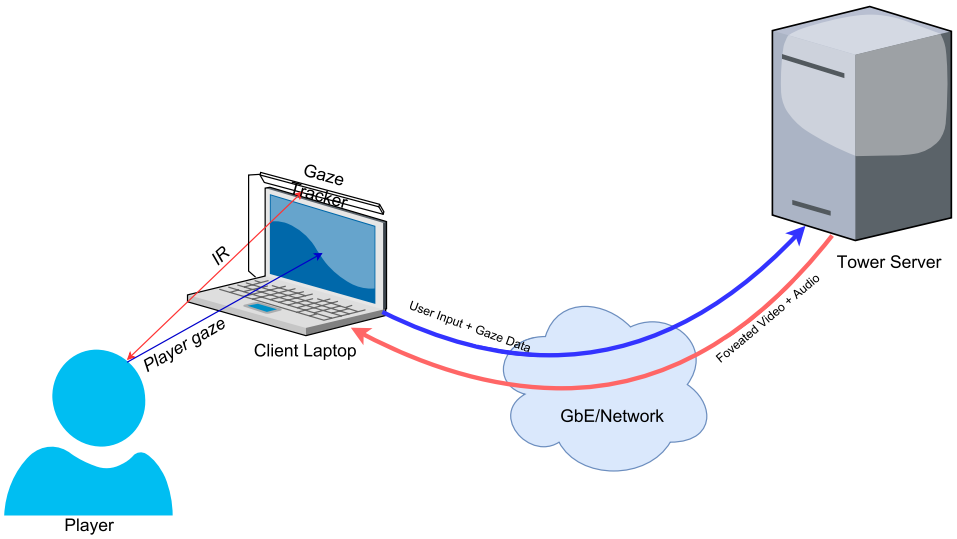


Figure 7: Cloud gaming with foveated video prototype.

5.1 Prototype Components

5.1.1 Gaming Anywhere

GA is an open source cloud gaming system. It is modular, extensible, customizable, multi-platform and portable [3],[72]. The system provides cloud gaming clients for Microsoft Windows, Linux, Mac OS X and Android operating systems. Further it provides cloud gaming servers for Microsoft Windows, Linux and Mac OS X. At the time of its launch GA surpassed commercial cloud gaming service providers operating at the time (Onlive and StreamMyGame) in latency and network load metrics [3]. Another attractive feature is that it can stream/cloudify any off the shelf game without requiring any customizations to the game engine. Figure 8 illustrates the GA architecture at a high level. It shows a GA client connected to a GA server over the Internet.

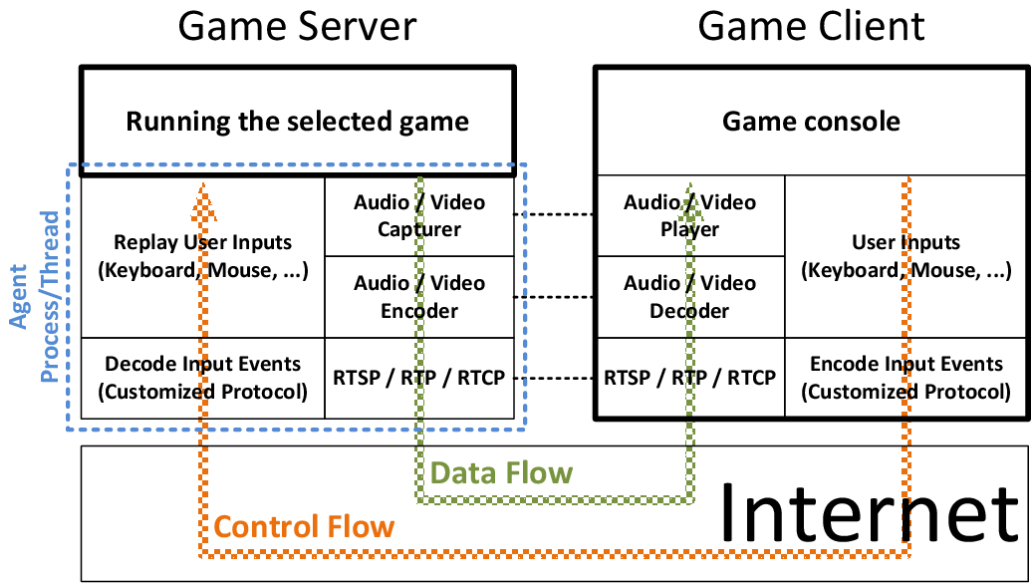


Figure 8: GamingAnywhere architecture [3].

The GA server comprises of modules for capturing audio and video of the gameplay, encoder modules for encoding the audio and video captured and streaming modules to stream the encoded gameplay. Further, there are modules for control information flow parallel and concurrent to the gameplay stream. A module receives and decodes input events as sent by the client and a module replays the input events to the game execution logic. In GA server, gameplay video can be captured in two modes, either event driven or in a periodic fashion. In event driven mode, the capture module hooks into the rendering output of the games, as rendered by the GPU, capturing a gameplay video frame as soon as it is available. In the event driven mode, a user chosen fps may be maintained by using a token bucket rate controller. In periodic mode, the capture module captures a gameplay video periodically by copying video

buffers of the server with a user specified period. The event driven mode is a more attractive design option but depends upon the rendering platform used by the game engine. The encoder module of the GA server is swappable and there is support for many encoders like X264, VP8 and planned support for X265 and VP9. The encoded video is streamed to the client using RTSP and RTP/RTCP. The number of flows depends on the transport protocol used, which again is configurable. If TCP is used, the encoded gampeplay and commands are all sent over RTSP. If UDP is used, the audio and video frames are transported with the RTP over UDP. The gameplay audio capture module captures audio from the gameplay using the ALSA library in Linux and Windows Audio Session API in windows. GA server maintains a shared buffer which the capture module, audio and video encoder modules and the streaming module can access, reducing delay due to memory access operations.

The GA client comprises an RTSP/RTP client-audio/video player module, a user-input capture module and a module which encodes and streams the user input to the server. The multimedia player receives the encoded audio/video gameplay, decodes and plays it. The user input capture module is a key and mouse logger which captures each input event. The captured events, like keystrokes, mouse motion or mouse clicks are encoded in a custom protocol and sent to the server. For a low latency QoE, the the client does not buffer any video frames.

GA uses a multi-threading approach to enable low latency cloud gaming. For example, the RTSP server, the audio and capture modules, and user input replay modules each have their own threads. The encoder modules also have separate threads which, depending upon configuration, can spawn more threads. Since the architecture is modular, modules can be extended or swapped. This design modularity is leveraged in implementing the prototype for cloud gaming.

5.1.2 X264 Encoder

The X264 encoder is an open source implementation of the H.264/MPEG-4 AVC standard. It is one of the most feature packed implementations of the popular standard. It provides support for multiple reference frames, CABAC entropy coding, interlaced coding, custom quantization, adaptive mode selection, psycho-visual rate distortion optimizations and arbitrary bitrate distribution etc [73]. X264 is used in many other video encoding softwares and GA utilizes x264 via the ffmpeg library [74]. In addition to a CLI, ffmpeg provides an API with which options can be set for the x264 encoder. Among a size-able number of other customizable options is rate control. In one pass mode, relevant for streaming applications where future frames are unknown, x264 can be configured to maintain: a constant bit rate for encoded video, an average bit rate, a constant quantizer or attempt to maintain a constant visual quality at the lowest possible bit-rate (constant rate factor) [75]. Among these constant bit rate is suitable for real time applications as a user can specify a constant bit rate depending upon the available bandwidth. However, from a QoE perspective

constant quality as achieved by the constant rate factor (CRF) mode is better suited.

Further, x264 supports adaptive quantization. The default adaptive quantization mode in x264 is variance based adaptive quantization which allocates more bits to macroblocks with less detail (flatter areas), that is the quantization step size is decreased. However it also enables an application to manipulate the quantization parameter (QP), i.e. the quantization step size, on a frame to frame basis as well as macro-block to macroblock basis. In practice this is implemented by allowing an application to add an offset to the QP calculated by x264 algorithms [76]. This functionality is leveraged to implement real time foveated video encoding in this thesis. The QP offsets are calculated for each macroblock based on the current gaze location.

5.1.3 Tobii 4C Gaze Tracker

A Tobii 4C gaze tracker from Tobii Eye Tracking [77] is used in the prototype. The Tobii 4C eye tracker is a consumer grade eye tracking device directed towards gaming and interactive applications. It has an on board eye tracking ASIC, near IR (850nm) illuminators and sensors. The Tobii 4C gaze tracker claims to use an enhanced version of PCCR gaze tracking technique with proprietary algorithms in a multi-illuminator multi-sensor setup. The eye tracking computation is done in the on board ASIC chip. It has also head tracking support, but the head tracking computations are done in the host computer. During callibration, the gaze tracker analyses the user's eyes to choose the best method, between bright pupil and dark pupil, for eye tracking of that particular user. The gaze tracker is capable of providing individual eye, gaze, gaze fixation and head location data as a data stream. The eye tracker comes with an SDK which allows extraction of the data. The SDK allows the developer to set filters on the data stream from the eye tracker. In the prototype, the "light" filter is used. The light filtering smoothens out sudden eye movements, such as micro-saccadic movements and noise based on current and past data points as well as velocity of eye movements [78].

5.2 Prototype

The prototype as mentioned earlier comprises of a Windows client on which the gaze tracker and a GA client is installed and the server comprises of a Linux workstation on which a GA server module is installed. Both the GA client and GA server are modified to enable foveation. Figure 9 illustrates the various modules and data flows between the server and the client and the modules therein.

5.2.1 Server and Client

The client sets up a connection over RTSP with the server, which responds with game play audio-video stream, the encoding parameters being dictated by the configuration in the server. Before receiving a connection request from a client, the server keeps the encoders and user input relay modules on standby. GA also supports view only mode

where the user input relay modules are not activated for a client. On the receipt of a client request, the server activates the the encoder modules and if the client is an active player, also the user input receive and replay modules. The client receives the audio video stream, decodes it and plays it on a display. Further, it captures user input, encodes it and sends it to the server. In the prototype, simultaneous with the client sending a request to the server, a gaze tracker module is also started. The gaze tracker module extracts lightly filtered gaze location data from the Tobii 4C gaze tracker. The gaze location data, obtained in the form of co-ordinates (in pixels) on the client display, is encoded into a light weight data format and transmitted to the server. The design decision to use a separate data stream and a light weight data format is made to ensure that gaze location is available at the server with the lowest possible delay. The data format comprises of just three fields to avoid processing and network overhead, the x and y coordinates of the gaze and a relative timestamp. The gaze location is sent to the server as soon as available from the gaze tracker over a TCP or UDP connection. Although UDP can be faster, TCP is used during evaluation experiments to circumvent firewalls. The GA server has modules to work on the video pipeline and transmit the encoded audio video stream and modules to handle user interactions. In the prototype the video pipeline is modified to receive gaze data and to encode foveated video. In the module feeding frames to the encoder for encoding, a new component is added to receive the gaze location data and to decode it in an independent thread. The gaze data is used to calculate the quantization offsets for each frame

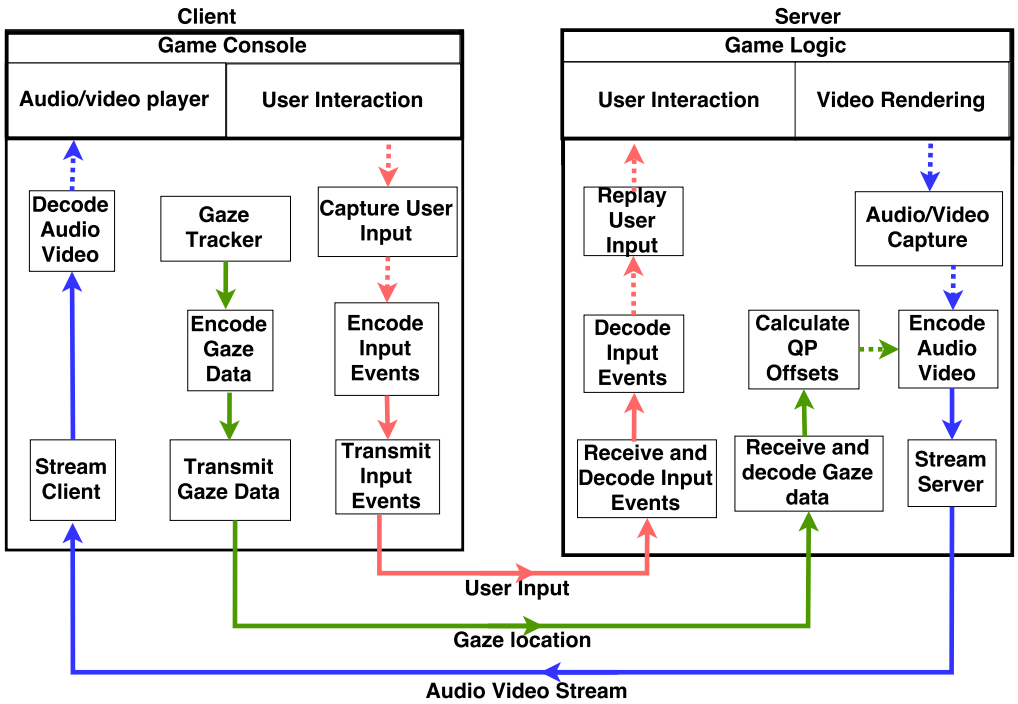


Figure 9: Architecture of the prototype

5.2.2 Quantization Offset Calculation

As discussed in Chapter 3, macroblocks are typically the smallest image units for video encoding. The x264 encoder allows an application to alter quantization parameter at the granularity level of a 16 x 16 pixel macroblock. An application can do this by specifying a quantization offset (QO) for each macroblock of an image. The x264 encoder then adds this offset to the QP it has calculated for each macroblock based on its own algorithms. The offset calculating module in the prototype checks the dimension of each incoming frame in pixels and calculates the dimensions in terms of macroblocks. Further, it takes the latest available gaze location in pixels and translates it to a location in terms of macro-blocks. Then it calculates an array of quantization offsets according to Equation 1.

$$QO(i,j) = QO_{max}\left(1 - \exp(\frac{(i-x)^2 + (j-y)^2}{2(W)^2})\right) \tag{1}$$

Parameter	Description
i, j	Indicies of a macroblock
$QO(i,j)$	Quantization offset for the (i,j)th macroblock
$Q0_{max}$	Maximum possible value of quantization offset
x, y	Indicies of the macroblock where gaze is located
W	parameter to control the spread/width of the high quality region. At a distance of W from the gaze location, the value of QO is about 40 % of QO_{max}

Table 3: Table of parameter of Equation 1

In Equation 1, QO_{max} is the maximum offset which may be added to a QP for a macroblock, i and j are indices of the matrix of macroblocks comprising the frame, x and y are the indices of the macroblock where the center of the gaze is, and W controls the width of the foveal region (as dictated by width of the 2d Gaussian curve of Equation 1). Since W corresponds to the standard deviation of a Gaussian curve, it is the radius of a circular region around the gaze location within which the QO values are less than about 40% of QO_{max} [79]. In this thesis, foveal region, as controlled by W, refers to the region which is encoded with high quality. The parameters of the equation are summarized in Table 3.

The human visual system (HVS) has a non uniform acuity, as discussed in Chapter 4, which corresponds to the density of photreceptors in the eye. The acuity of HVS is illustrated in Figure 10. As can be seen, the region of highest acuity occupies a very narrow angular region, around 2^o and drops exponentially with the angular eccentricity from the fovea.

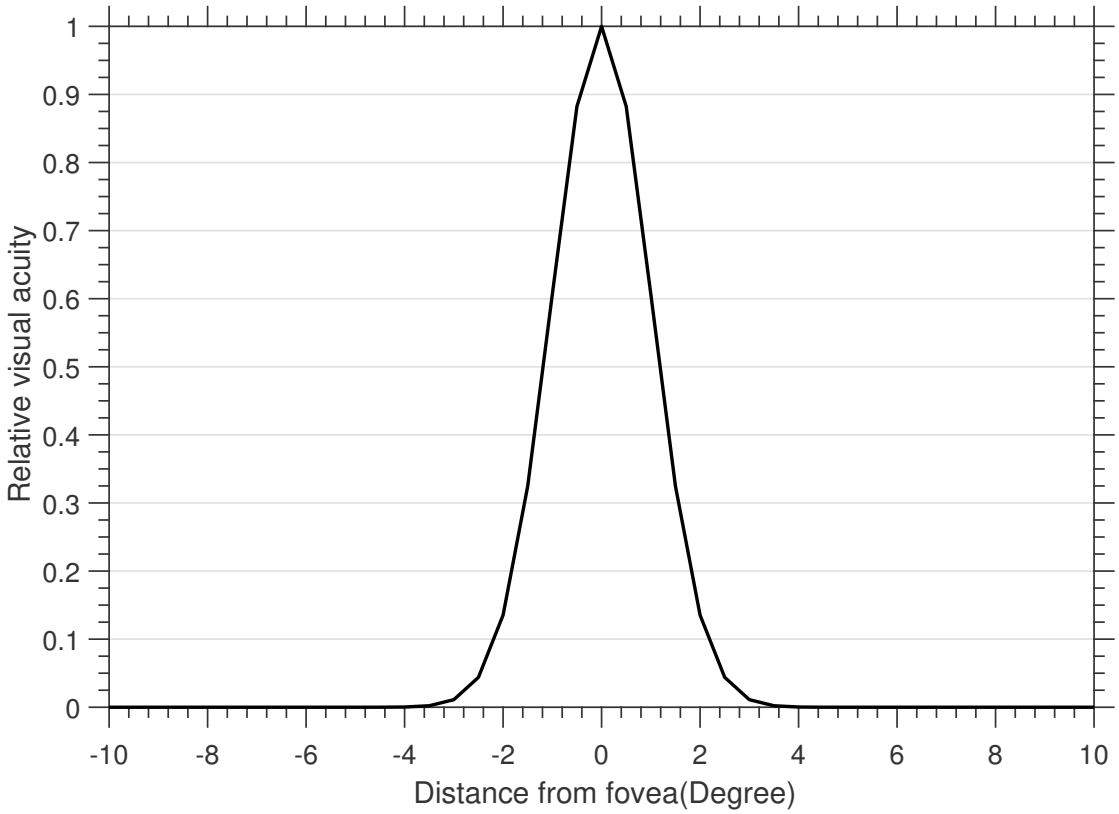


Figure 10: Visual acuity of the human eye

Equation 1 calculates QOs in according to a two dimensional Gaussian curve whose "width" can be changed. The Gaussian curve is chosen because the drop of the Gaussian curve is not as steep as the approximately quadratic nature of the HVS acuity. It is assumed that this spread and comparatively gradual drop somewhat compensates for any minor inaccuracies in the gaze location data.

The normalized QOs generated by Equation 1 for various values of W expressed in terms of frame width FW are illustrated in Figure 11. Figure 12 illustrates the offsets calculated by Equation 1 for a 1280 by 1280 pixel (80 by 80 macroblocks, each macroblock of 16px x 16px) image as a heat-map, with the gaze location at the center of the image(at the macroblock 40,40), QO_{max} set as 10 and W set as $FW/4$ (320pixels, 20 macro blocks). It is clear that the quantization offset for a macro-block increases as with its distance from the gaze location. The macroblocks at and around the gaze location have 0 or low quantization (lesser than 1) offsets. At even a very generous value of W, it can be seen that a majority of macroblocks have a quantization offset of more than 5, substantially reducing the bitrate needed to encode the frame. At an average viewing distance of 50 cms, the area subtended by the foveal region is nearly 2 cms in diameter which on a 34.5 cm screen of 1366 px resolution— a typical laptop configuration, corresponds to about 80 pixels or 5 macroblocks of 16 x 16 px.

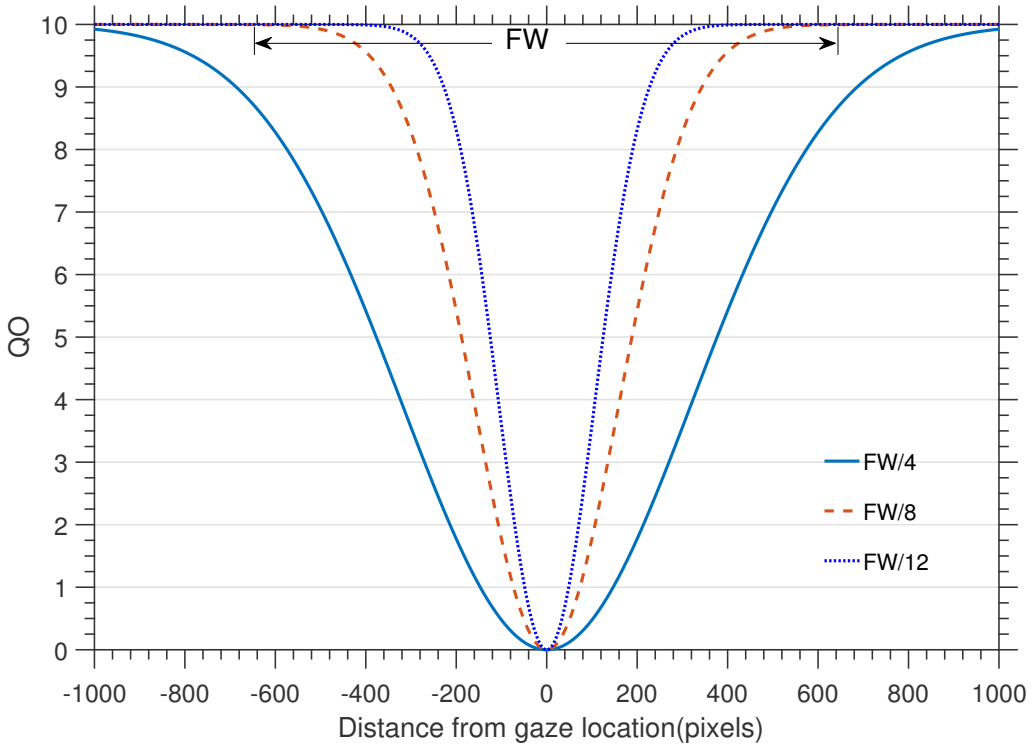


Figure 11: Foveation and QO calculation. FW is the width of the output frame in pixels.

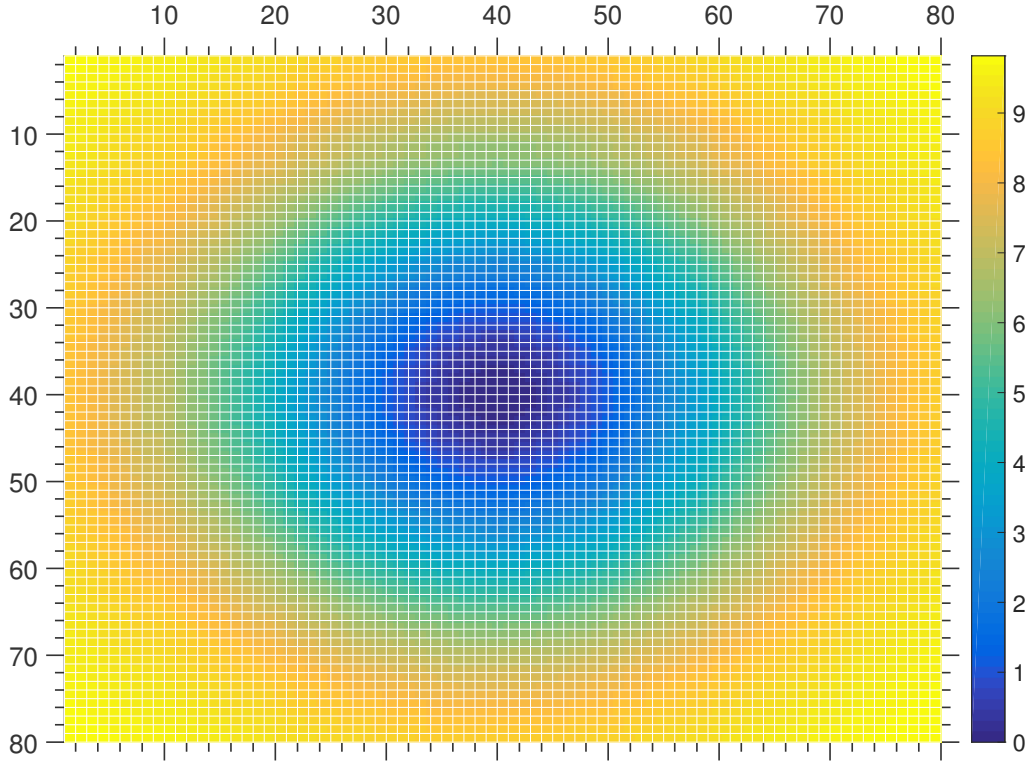


Figure 12: Heatmap of QOs for a 1280x1280px image, the gaze location being at the center and QO_{max}=10,W= Frame Width/4

Ideally, the width of the foveal region should correspond exactly to the region of an image perceived with the highest visual acuity by the human eye. However, there are many factors that affect the size of the region perceived with the highest acuity by the human eye, for example visual stimuli and their motion. Further, the gaze tracker may not be accurate enough or there may be latency in the calculation and reporting of gaze location. Another aspect that affects the decision of size of the foveal region is how fast a players gaze moves in a game. These aspects are explored in the experiments reported in Section 6.

6 Evaluation

Here the effect of foveated encoding on game play video bitrate is investigated. Video games of different genres present different POVs for the player. POV of a game affect the graphical details of the gameplay video generated and it is expected that foveated video encoding will have different characteristics and constraints for each genre and also the throughput will be reduced to different degrees once these constraints are taken into account. Three video games, AssaultCube, Trine2 and Little Racers Street (LRS) are considered in the bandwidth measurements, representing three different genres of video games. AssaultCube is an open source first person shooter game (FPS) similar to a very popular FPS game Counter Strike. Trine2 is a side-scrolling action platform and puzzle game with fairly complex graphics. LRS is a birds eye view racing game where the scene and players' gaze location changes frequently. Gaze data of these and an additional game Formula Fusion is also studied later to analyze latency feasibility of foveated video streaming for cloud gaming. Formula fusion is futuristic racing game with a traditional behind the car POV for the driver.

6.1 Experimental Setup

The client and server are setup as illustrated in Figure 7. They are connected over the campus GbE network with negligible latency to remove any network related bias in the results. Network latencies in cloud games and their effects have been well studied as discussed in Section 2.4 and Section 6.3. Further, the client and server are configured to use TCP for audio/video data, control data and gaze data.

6.1.1 Encoding parameters

The x264 encoder is set with the following parameters based, in part, on the recommendations in [3]:

```
    --profile main --preset ultrafast --tune zerolatency --crf 28
--ref 1 --me_method dia --me_range 16 --keyint 48 --intra-refresh
--threads 4
```

The parameters are explained below:

- `profile` indicates the profile of the H264 standard to be used for encoding. It is set to `main` as recommended by [3].`main` profile is suited for standard definition video.

- `preset` indicates the the preset speed of encoding, which is set to `ultrafast` keeping in view the real-time nature of the application, although slower encoding may result in better compression

- `tune` is an optional parameter which specifies the quality of input video. It is set to `zerolatency` to correspond to cloud gaming.

- `crf` indicates the constant rate factor to be used. `crf` is a measure of "constant quality" which the encoder aims to achieve. It can be set to values between 0-51 for 8 bit color depth, 0 being lossless encoding and 51 being the worst. When `crf` based rate control is used, QPs for each macroblock are adaptively chosen frame by frame. Higher the `crf`, higher the quantization step size, higher the quantization loss introduced and hence lower the bitrate. The `crf` is set to 28 in the measurements as 28 provides reasonable quality of experience in real time streaming scenarios. Note that on top of the QPs calculated by the `crf` algorithms, quantization offsets are added for foveated encoding in the protoype.

- `ref` indicates the size of Decoded Picture Buffer. It indicates the number of decoded pictures that a predicted frame can reference. The decoder has to store as many decoded pictures as indicated in `ref`. It is set as 1 to keeping in view that higher values typically mean more time needed to encode and decode, although there might be gains with respect to compression.

- `me_method` indicates the method of motion estimation to be used. The value set in the experiments `dia` refers to diamond motion search which, being the simplest motion estimation method is the fastest. Again, possible compression gains are sacrificed for low latency.

- `me_range` indicates the range of motion estimation in terms of pixels. It is set to 16 which is the maximum possible value when `me_method` is set to `dia`.

- `keyint` indicates the frequency of picture refresh. It can either be by inserting I-frames or by periodic intra-refresh. It is set at 48 so that the picture refreshes approximately every second when the streaming frame rate is 40-45fps.

- `--intra-refresh` enables the use of periodic intra refresh, instead of key frame or I- frame insertion. Periodic intra refresh inserts a column of intra coded macroblocks in each encoded picture. This ensures robustness when dealing with for example frame loss as I frame as a single point of failure in a Group of Pictures is eliminated . Also a more uniform frame size is achieved with intra-refresh.

- `threads` specifies the number of threads to be used during encoding. It is set to 4 to improve speed of encoding.

6.1.2 Traffic Capture

The data flow between the cloud gaming server and the cloud gaming client of the prototype of Figure 7 is captured using tcpdump [80]. Tcpdump is started on the

server with a filter to capture traffic only coming from and going to the client. Both upstream and downstream traffic between the server and the client are captured, but the upstream data stream contains only the control (user control) data and gaze data. Thus reduction, if any, in throughput are a direct result of foveated encoding. The throughput per second is extracted from the capture files using Wireshark [81]. For each game multiple iterations of the experiment are conducted over the same period of time, varying parameters QO_{max}, and W. W is varied in terms of the output frame width FW. For each iteration of the experiment for a game, the gameplay is kept as identical as possible both in terms of actions of the player's actions and length.

6.1.3 Game Specific Experimental Procedure

The games and the corresponding game specific measurement setups are described below.

AssaultCube:

AssaultCube is a FPS game. FPS games are very popular and other game genres, like action adventure, horror games etc. tend to use the same point of view. In AssaultCube, the player holds and controls a weapon. There is a cross-hair projected for the weapon indicating where the weapon targets, if used. The player can move the character and the weapon as well. Movement of the player is executed as steps, jumps, crawling and climbing, while movement of the weapon is executed either as arm movement or rotation of feet. The scene is displayed the same way it would appear if the player was part of the scene. In other words, the view displayed is eponymous to the genre.

For the experiment, the prototype cloud gaming server is started in event driven mode and the gameplay resolution is set at 1366x768 to be able to play it at full resolution on the prototype cloud gaming client ($FW = 1366 pixels$). To keep the gameplay consistent in each measurement, an AssaultCube tutorial in which the player actions are predefined is played for a preset length of time, instead of an active player game. The player, however tries to focus on the action in the game in each measurement.

Trine2:

Trine2 is a "platformer" game with a side scrolling gameplay. The player looks at their in-game avatar which it controls, traversing platforms and solving puzzles. The game play is two dimensional, in that the action moves sideways or up and not down not into the depth of the scene. Trine2 is a representative of modern platformer games in graphics complexity.

For the experiment, the prototype cloud gaming server is started in periodic mode

and the gameplay resolution is set at 1366x768 to be able to play it at full resolution on the prototype cloud gaming client ($FW = 1366 pixels$). To keep the gameplay consistent in each iteration of the experiment, the same level of the game is played with the player making an effort to replicate the actions of the in game avatar.

LRS:

LRS is a racing game with a with a birds eye POV. The player looks at a car which the control from the above, racing other cars. The underlying map of the track moves predictively only to accommodate the cars, because the map is typically bigger than the screen. The birds eye point of view is common in other games, like strategy games.

For the experiment, the prototype cloud gaming server is started in periodic mode and the gameplay resolution is set at 1366x768px to be able to play it at full resolution on the prototype cloud gaming client ($FW = 1366 pixels$). To keep the gameplay consistent in each iteration of the experiment, the same race of the game is played on the same race track with the player making an effort to replicate the actions of the steering the car.

6.2 Throughput Reduction

Figure 13 shows box plots of bandwidth measurements for the game AssaultCube. There is a dramatic reduction in video bit rate when the parameter W is set to 1/8th of the screen size and QO_{max} is set to 5. With QO_{max} at 10, the video bitrate reduces even further. The reduction in bitrate is not as significant when QO_{max} is set lower. When QO_{max} is set constant at 10, and W is varied, the drop in video bitrate is less steep with smaller W, but significant when compared to no foveation. With W decreasing below $FW/8$, the gains in bit rate reduction are smaller. This may be due to the fact that with FW becoming smaller, relatively few macroblocks have high quality. However, it should be noted that the bitrate reduction depends on the size of the output frame.

Figure 14 shows box plot for throughput of Trine2. Similar pattern of video bitrate reduction with varying QO_{max} and W. Trine2 being more graphics intensive, the highest bitrate without foveated encoding is higher than the highest bitrate without foveated encoding in AssaultCube. However, when foveated encoding is used, the highest bitrate is also reduced.

Figure 15 shows box plot for throughput of LRS. The median throughput without using foveated encoding is the highest in LRS in spite of the fact that graphical complexity of an average scene in LRS may not be as high as Trine2.This can be explained by the fact that even though the scene is not complex, it changes omni-directionally (in four directions on a 2-d plane) at a high frequency with almost every control input of the user, courtesy of the birds eye view. It can be also noticed that

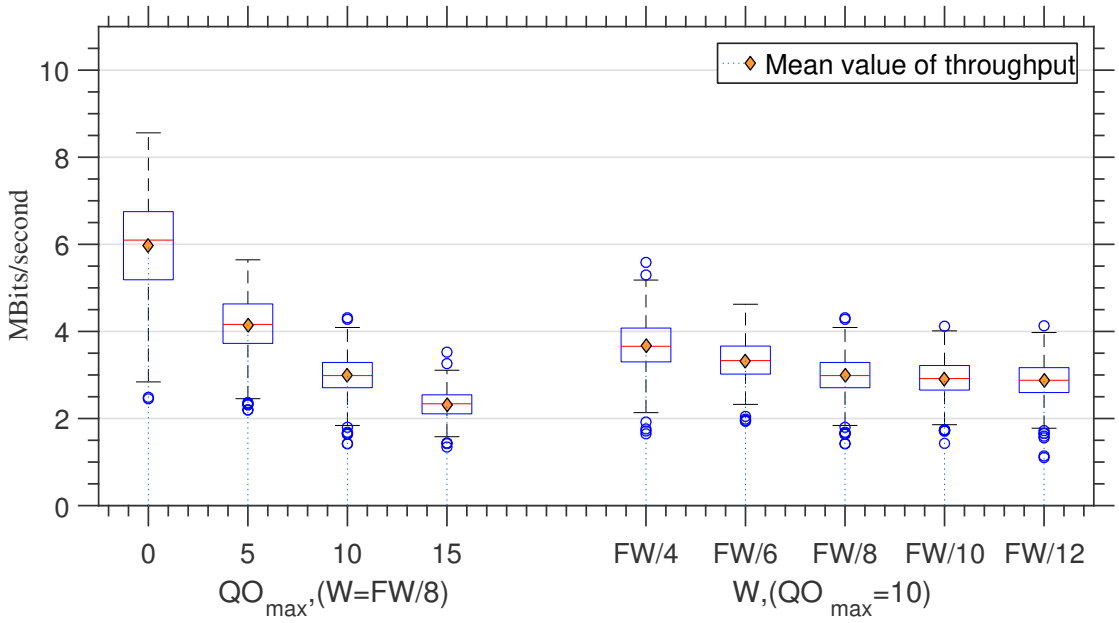


Figure 13: Throughput at different values of QO_{max} and W for AssaultCube

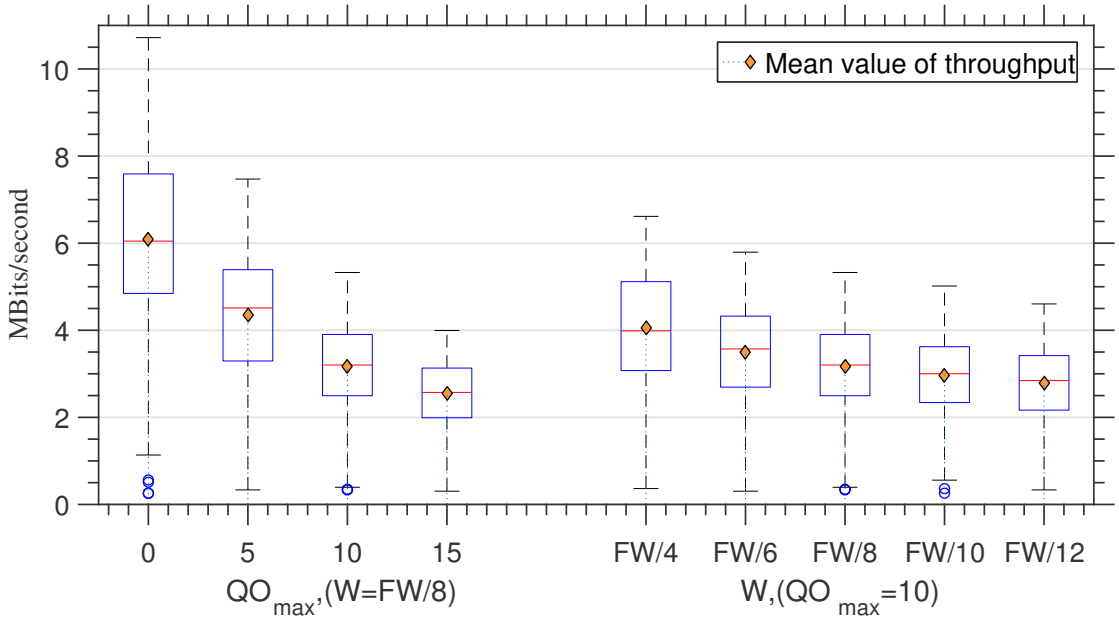


Figure 14: Throughput at different values of QO_{max} and W for Trine2

there a quite a few outliers below the minimum throughput. These occur when the a control input results in change of the position of the car in the game, but not a displacement of the map.

In all the three measurements illustrated in Figures 13,14, 15, it is noticeable that

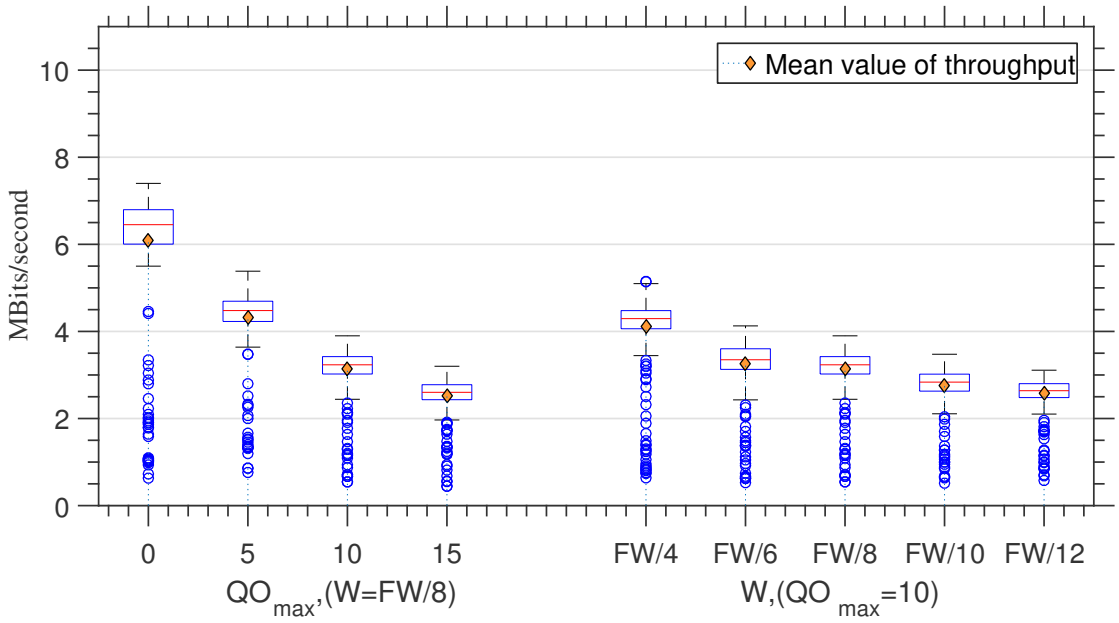


Figure 15: Throughput at different values of QO_{max} and W for LRS

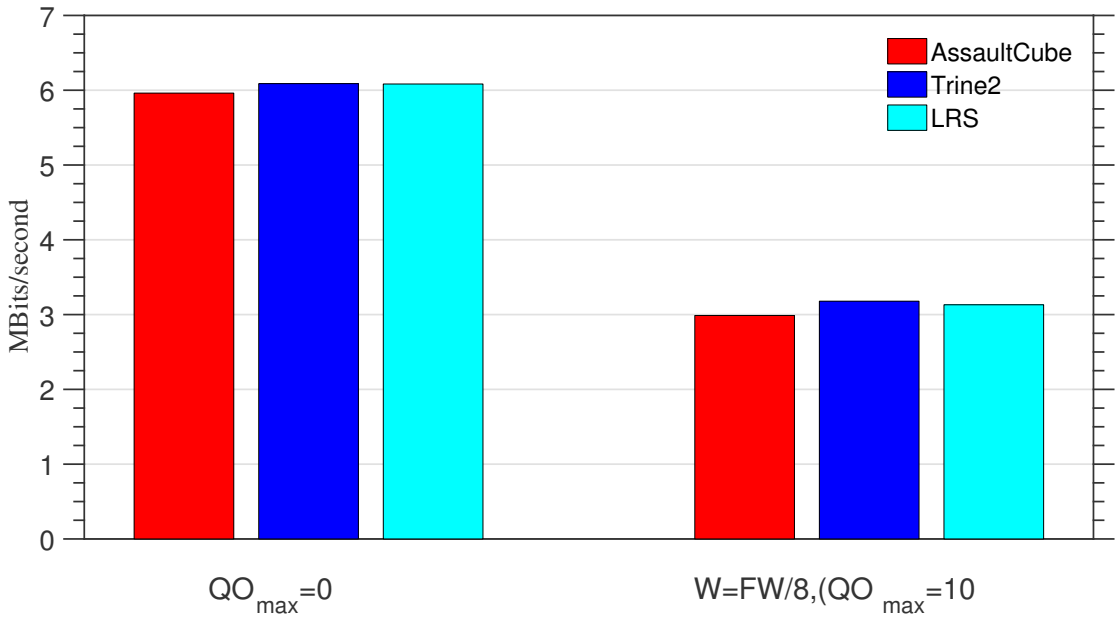


Figure 16: Average throughput without foveated encoding and with foveated encoding at $QO_{max} = 10$ and $W = FW/8$ for AssaultCube, Trine2 and LRS

QO_{max} affects bit rate more than W. The reduction in video bitrate is lower as W is reduced beyond $FW/10$ the reduction in video bitrate becomes flatter. This is due to the fact that at the current FW ($1366 pixels$), the number of macroblocks kept at high resolution is very small beyond $FW/8$. At higher screen sizes this will change.

It can also be noticed, that for the three games considered here, the average video bitrate without foveated encoding and with foveated encoding with identical parameters is in the same ball park range. This is further illustrated in Figure 16 where average throughput of the three games without foveated encoding and with foveated encoding at $QO_{max} = 10$ and $W = FW/8$. With these parameters, it is also noticeable that video bit rate is reduced by up-to 50%.

The region of an image perceived with highest acuity by the HVS depends on viewing distance as discussed in Chapter 4. At a viewing distance of 50cm, considering a 2^o angle of foveation, the region of highest perceived visual acuity is a region with diameter $d = 2 \times \pi/180 \times 50cm$ which comes out as 1.745 cms $\approx$ 2cm. At $W = FW/8$, for a screen width of 34.5 cm, (equal to frame width in full screen mode), the W value is 4.3 cms. This means a circular region of diameter 4.3 cm around the gaze location is encoded with $QO \leq 40\%$ of QO_{max}. Figure 17 shows screen captures of the game Trine2 as seen on the cloud gaming client with no foveated encoding, foveated encoding at QO_{max}=10,W=$FW/8$ and foveated encoding at QO_{max}=30,W=$FW/8$. The players' gaze is indicated by the overlay bubble in each screen grab. From visual inspection, it is hard to distinguish the quality of picture within the overlay bubbles in the three cases. Farther from the overlay bubble, some loss of quality in Figure 17 b compared to Figure 17 a can be observed. Comparing Figure 17 c with Figure 17 a, significant loss of quality occurs away from the overlay bubble. However, it should be noted that these screen captures have been taken when the in game avatar is actively moving, to accentuate the loss of quality further from the gaze location in foveated encoding. When the in game avatar is stationary, the video frames from the first two cases illustrated here look identical to the naked eye. Further, it is imperative to note that user experience and QoE are highly subjective, and this parameterization i.e. QO_{max}=10 and W=$FW/8$ may not be universal.

6.3 Gaze and Latency

Latency is one of the main challenges of cloud gaming. Game QoE is highly dependent on responsiveness of the game play, i.e. the delay between a player input and on screen action. In a desktop gaming setup, the only factors affecting the responsiveness of the game are latency between the user input device to game engine and the latency of game engine itself. However, in cloud gaming additional compute and transport steps are introduced which exacerbate the latency. Latency in gaming is being studied actively and there have been many publications on the subject, some of which are briefly discussed in Section 2.4.

6.3.1 Gaming and Gaze Location

To investigate gaze movement characteristics and their effect on foveated encoding, gaze of a player is tracked while playing different games. Each gaming session lasts about 15 minutes and the games played are of different genres. The games for which gaze is tracked are AssacultCube, Trine2, LRS (bandwidth measurements are also

(a) QO_{max}=0 (normal encoding)

(b) QO_{max}=10,W=$FW/8$

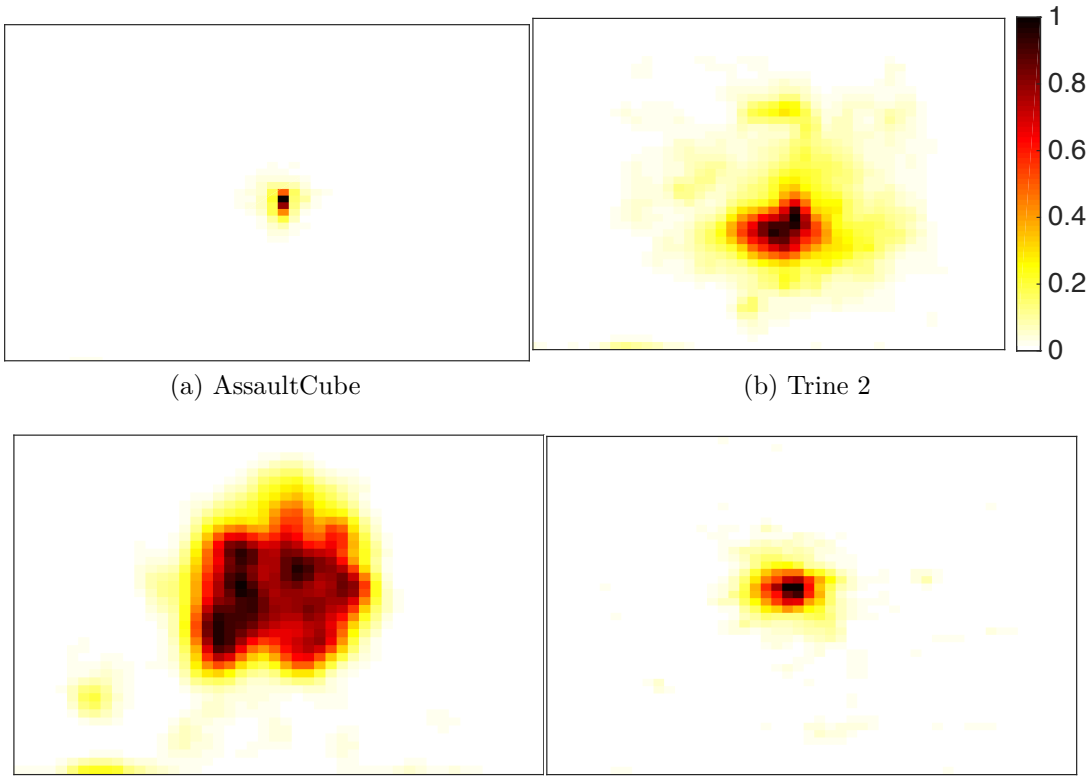
(c) QO_{max}=30,W=$FW/8$

Figure 17: Screen Captures of Trine2 without and with offsets. The overlay bubble indicates the gaze location.

done for these three games) and Formula Fusion. Formula Fusion is a futuristic racing game, but the POV is that of a conventional racing game i.e. behind the vehicle. Figure 18 illustrates the heatmaps of gaze locations for each game. The heatmaps are computed by bivariate Gaussian kernel estimation. It can be observed that for all games, the gaze is mostly targeted towards the center of the screen. Glances to the periphery are so small in number, that if they show up in the heat map, the color intensity/heat is low. This is to be expected as typically, the center of the action in a game is in and around the center of the screen. The heatmap of AssaultCube is clustered around the centre of the screen with very little spread. This is because AssaultCube is an FPS and typically the player is looking at, and hence the gaze location is fixed at, the crosshairs of their weapon. The updates to game video and state happen such that the crosshairs are always at the center of the screen. In Trine2, the gaze locations, although clustered around the center, have a broader spread. This is due to the exploratory nature of the game. The player may focus their gaze at the in-game avatar most of the time, but there are enough exploratory glances to objects around the center and even at the bottom left of the screen to register on the heatmap. In LRS, the focal area for the player is larger, even though centered. This is due to the birds eye view nature of the game. In LRS the player controlled vehicle as well as the map of the game move (laterally) with user input, resulting in a broader cluster of gaze locations. In Formula Fusion, the gaze locations show a tight grouping around the center as well. This is again due to somewhat first person view of the game. The game keeps the track and the player controlled car in the center of the screen. It is evident from the gaze heat maps that the game characteristics will dictate the quality and seamlessness of foveated video for cloud gaming. To illustrate how long the player gaze stays stationary, gaze moments are calculated from the gaze data. Gaze moments are defined as periods of time during which the gaze stays within a circular region of a particular size (D). CDFs of gaze moments with $D = FW/8$ and $D = FW/4$ are shown in Figure 19. It can be seen that the vast majority of gaze moments —upto 90%,even when the size of localization is small ($FW/8$), last more than 100ms. Gaze moments are longer in Assault Cube and Formula Fusion, expectedly given the gaze heatmaps in Figure 18.

Rate of change of gaze is also calculated by taking the distance between subsequent gaze locations, as reported by the eye tracker, and dividing it by the time difference between the two samples. The CDFs of rate of change of gaze are shown in Figure 20. Again it can be seen that the majority of changes in gaze location are slow,more than 90% of gaze changes translate to 100px per 100ms. Some trans-screen glances, where the player gaze moves across the screen, do cross 1Kpx per second that is about 100px in 100ms. These are probably the most challenging instances for foveated encoding. Further it can be observed that the rate of change of gaze is higher in LRS than in other games.

Latencies of 100ms to 300ms are considered viable depending upon the genre of the game [4]. The Tobii eye tracker samples eye locations at 90Hz and outputs gaze location coordinates at nearly the same frequency, that is every 11.11 ms. Although



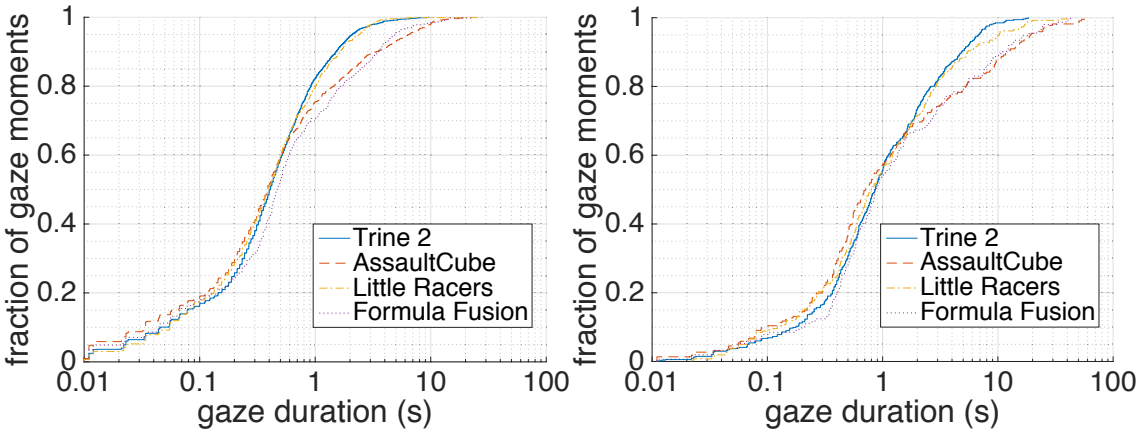
(a) AssaultCube (b) Trine 2

(c) Little Racers STREET (d) Formula Fusion

Figure 18: Gaze tracking heatmaps from 15 minute gameplay sessions. The color scale is normalized.

this delay occurs parallel to other device related delays, even adding this delay to the achievable delays investigated, in for example, [1], a delay of 110ms is achievable. This would be the delay between eye movement and resulting change in video at the client, assuming the delay in Tobii tracker corresponds to its sampling rate and is 11.11ms. Looking at Figure 20, vast majority of gaze changes take more than 110ms which means the gaze location is still at the point around which the encoded video quality is highest.

The challenge of latency can also be addressed in a game specific manner. Based on the player gaze characteristics of a game the parameter W can be set. For example, if in a game the gaze rate of change is high, for example more than 200px per 100ms, the W can be set so that a region of 200px around the gaze location is encoded with high quality. For the games considered here, played on the prototype of Figure 7, with majority of gaze changes at a rate below 100ms/100 pxs, $W = FW/8$ translates approximately to a region of diameter 170 pix with quality offset of 0 to 4, which can be considered high quality. So majority of gaze changes still occur within a region which is encoded with high quality.



(a) $W = FW/8$ (b) $W = FW/4$

Figure 19: CDF plots of the duration of gaze moments with different sizes of foveated regions.

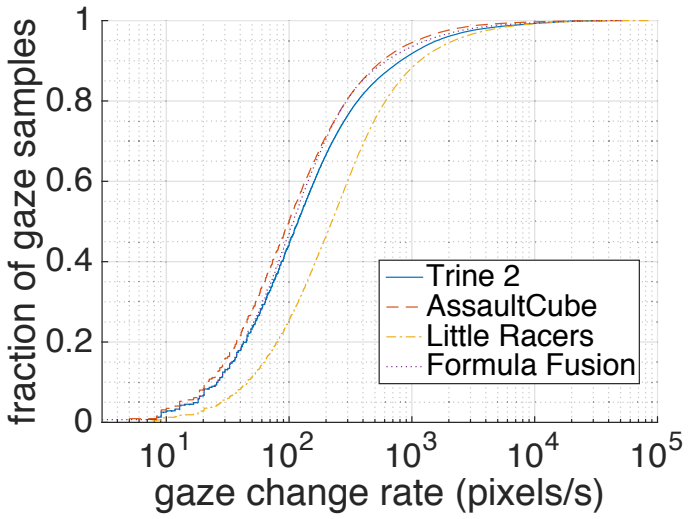


Figure 20: CDF of rate of gaze shifting.

6.3.2 Gaze Tracker Latency

To calculate the end to end latency between gaze location change and play out of accordingly foveated video, the latency of the gaze tracker is needed. The latency of the gaze tracker is the time between the moment the gaze tracker receives an image of the eyes and the moment the gaze tracker reports the co-ordinates of user's gaze corresponding to that image at its API. To measure the gaze tracker latency this thesis conducts two experiments. In the first experiment the gaze tracker is installed as recommended by the vendor on a laptop and it is configured to track a user's gaze by following a set of calibration instructions displayed by the gaze tracker software. For the experement, the user is instructed to fixate on a visual stimulus as soon as

it appears on the screen of the laptop. Each round of the experiment consisted of generating 5 visual stimuli at known locations. As a stimulus is generated, its location is stored with a system time-stamp. Concurrently gaze is recorded by the gaze tracker and the gaze data points available at the gaze tracker API with corresponding system timestamps are stored.It should be noted that the Tobii EyeTracker 4C provides a timestamp with each reported gaze location, however, this timestamp is relative to an arbitrary point in time and hence does not carry any latency information. The latency is measured as the difference between the time stamp of a visual stimulus and the time stamp of the first gaze location reported by the gaze tracker within 100 px of the visual stimulus. From multiple rounds of measurements, latencies of 260-400ms are measured. Subtracting the average saccade latency of 200ms [82] from this latency, we get latencies of 60-200ms for the gaze tracker. However, this experiment has mulitple limitations from a statistical, physiological and cognitive point of view. First, human saccade latency is highly variable and depends upon the type, nature and number of stimuli presented [83]. Second, there is delay in detecting a stimulus and firing neurons to cause muscles to move the eye in that direction. Third, the test subject may be subconciously biased and may move their eyes too fast or too slow.

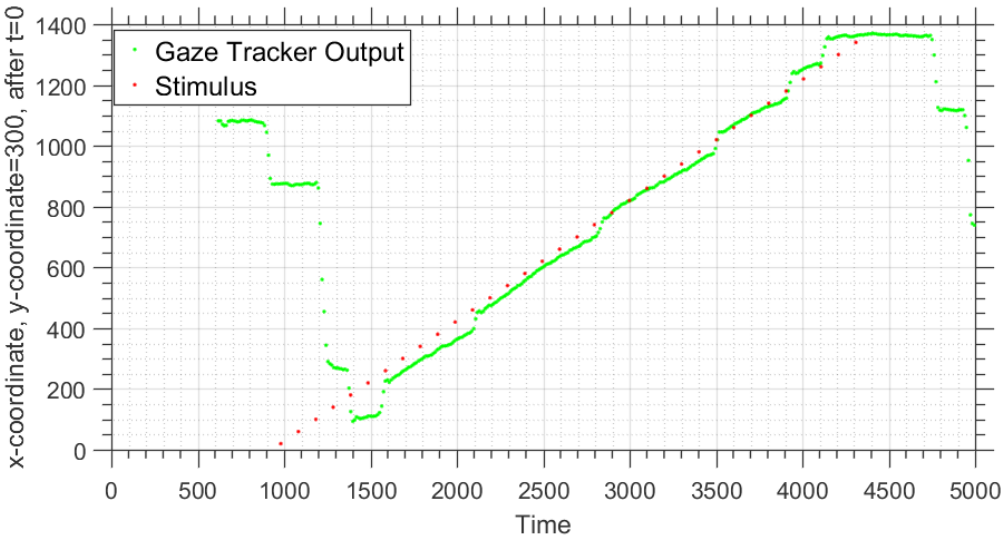


Figure 21: Gaze tracker latency. The y-coordinates of the stimuli and hence the gaze locations reported by the gaze tracker are constant(within a range).

To overcome some of these challenges, particularly the variability of saccade latency, a second experiment is designed. In the second experiment, an attempt is made to use smooth pursuit movements of the human eye to measure the latency of the gaze tracker. In smooth pursuit eye movement, as the name suggests, the eye smoothly follows a visual stimulus. The experiment comprises of generating a growing horizontal line on the screen of a laptop on which the gaze tracker is appropriately

configured (calibrated and positioned as per vendor instructions). The test subject is asked to follow the tip of the line. The stimulus line grows at about 10cm/sec on a 34.5 cm screen which corresponds to an angular speed of about $10.4^o/sec$ which should be a low enough speed to ensure smooth pursuit [84]. The line is grown by adding a new stimulus at the edge of the previous stimulus. Each stimulus location, along with its system time-stamp is logged as is the gaze location from the gaze tracker and its corresponding system time stamp. The difference between the time-stamp of a gaze location and the corresponding stimulus at the same location, should in theory provide a measure of latency of the gaze tracker. Figure 21 illustrates one such measurement. It can be noticed that after the initial delay of finding the target, the latency is highly variable. It ranges from 0 ms to 200ms. There are occasions where the gaze reaches a point even before the stimulus is shown. This may be due to the eye overshooting or other oculo-muscular movements, investigating which are out of the scope of this thesis. Further, this may be due to the test subject not following instructions or getting over-zealous. In short, there are variables and phenomenon involved which are not in the scope of this work.

From Figure 21, a conservative conclusion can be made that gaze tracker latency is highly variable, difficult to condense into a single representative number and will depend upon multiple factors, such as the velocity of the eye. Design and implementation of experiments to meaure latencies of eye trackers, which involve multiple physiological, neurological, cognitive and psychological variables is challenging. Methodologies for robust and accurate measurements of gaze tracker latency are a field of their own. Researchers have suggested such methodologies as well as conducted some measurements. Gibaldi et. al. in [85] conduct a thorough study of Tobii EyeX gaze tracker which is a predecessor of the Tobii Eye Tracker 4C used in this thesis. The methodology used in [85] for measuring gaze tracker latency is based on end to end latency measurements of a gaze contingent display suggested by Suanders et. al in [86]. For the Tobii EyeX gaze tracker, the latency measure in [85] is less than 50ms. As Tobii Eye Tracker 4C is supposed to be an improvement over the Tobii EyeX gaze tracker [87], this latency figure is encouraging. Indeed, Tobii Eye Tracker 4C has a sampling frequency of 90Hz compared to Tobii EyeX gaze tracker's 60Hz. Assuming Tobii Eye Tracker 4C has a sub 50ms latency and this latency occurs in parallel to other device latencies, a sub 110ms-150ms e2e latency is still achievable, which is optimisitic considering the discussion in Section 6.3 and measurements in [1]. However, this optimism must be taken with caution as a reliable and accurate latency figure for Tobii Eye Tracker 4C is not available.

7 Discussion

Cloud gaming provides a green and cost effective solution for video gaming. It also makes possible the paradigm of gaming anywhere, which entails playability of high graphics games across platforms and across hardware. This is prima- facie an enticing proposition. However latency, device and network, and high bandwidth network requirements are challenges which need to be addressed. There have been various studies on latency and low latency is becoming more and more achievable with improvements in device technology and with strategic placement of servers [1]. This thesis investigates a possible solution to the bandwidth challenge without affecting the quality of experience by leveraging the phenomenon of foveated vision. From Section 5.2 and Chapter 6 it can be concluded that cloud gaming with foveated encoding is achievable and accrues sizeable reduction in video bit rate. Depending on parametrization, the gameplay video bitrate can be reduced by more than 50%. In the prototype, a 150$ gaze tracker is used together with an average net-book for playing games, the total cost being still much lower than a "gaming laptop". It is conceivable that the price of consumer gaze trackers will drop further with advancements in gaze tracking and commoditazion of gaze trackers. Latency is also affected by the reduced bitrate and network delays might be shortened by this approach.

7.1 Future Work

Quality of service and foveated encoding can be improved if game specific RoIs are incorporated. For example many games have 'meters' or status bars or small navigational maps whose position is typically the same in all the frames. These widgets are generally small in size relative to the frame size and can therefore be encoded at high quality, without the use of gaze tracking and without affecting the bitrate too adversely. In such an encoding scheme, occasional glances towards these regions can be left out of the gaze tracker based foveated encoding, improving overall QoE. For better latency performance of foveated encoding, a combination of game attention/location patterns and real time gaze tracking can be used to predict gaze movements and ameliorate delay in gaze tracking and gaze tracking reporting. For some games, for example FPS games, the gaze location is stationary for most of the time. In such games, a gaze tracking may not be needed at all.

New encoders, like x265—an open source implementation of the HEVC standard, support macroblock level tuning of quantization as well and thus can be used in cloud gaming for even more bitrate savings as HEVC provides more compression compared to AVC. Such implementations, with two pronged bitrate reduction, will be particularly useful where network conditions are poor with only sub-broadband bandwidth available, enabling "pro" gaming experiences on low end or mobile computers with poor connectivity, for example in aspirational developing economies. HEVC also supports tiling of the video, which can also be leveraged to implement foveated encoding in a cloud gaming application.

Another important implementation avenue is using hardware already present in consumer devices for eye tracking instead of a dedicated eye tracker. Advancements in computer vision have made web camera based eye tracking possible even extending it to tablets and smart-phones [88], [89], [90], [91]. Since integrated front facing cameras are ubiquitous in laptops, tablets, and smart-phones, web-cam based eye tracking can enable foveated video and cloud gaming with foveated video on millions of devices. Further, modern devices are starting to incorporate infrared cameras typically for authentication applications, can also be used to track gaze, thus eliminating the need for separate eye trackers.

Careful parametrization of Equation 1 can improve QoE. Parameters like QO and W should be chosen according to the game to be played. In evaluation experiments Chapter 6, W is chosen in terms of output frame/screen width to implement a degree of scalablity with screen size. Since screen size usually dictates viewing distance, the actual size of foveal region is automatically changed with viewing distance. A more active approach where the viewing distance is actively measured, for example, at the start of the gaming session, the client reports the viewing distance. The Tobii gaze tracker used in the prototype in Section 5.2 is capable of detecting the viewing distance as well.This can be leveraged to initially or continually re-parameterize Equation 1 at the cloud gaming server. There are many improvements that can be incorporated in the server and client of the prototype to improve latency and QoE performance. On the client side, gaze location data may be sent in a non-continuous fashion, forwarding gaze location only when gaze location changes by a predefined threshold, which may be defined for example in relation to W. On the server side, gaze location data may be further filtered such that gaze location changes within a certain threshold distance or time are discarded. This can improve speed of encoding. Also the algorithm can be modified such that sudden gaze changes across longer distances do not affect QoE by, for example, triggering non-foveated video streaming till the gaze settles.

Future work should include studying foveated video streaming for cloud gaming from a QoE perspective. This would include user studies for a variety of game genres. Further the nature of gamers should be taken into account, the QoE can vary from an enthusiast gamer to a casual gamer. Although gaze tracking and latency is briefly considered here and the results look optimistic, this is an aspect that needs to be investigated in deeper detail. Another promising direction as mentioned earlier, is leveraging components like web cameras and IR cameras already present in state of the art devices for gaze tracking.

7.2 Limitations

This work does not investigate QoE of the prototype, which is necessary to validate foveated video streaming for cloud gaming. Foveated video streaming for cloud gaming is desirable only if QoE is at most slightly lower than QoE with un-foveated video streaming. As such, carefully designed user studies with replicable conditions and

standardized testing methodologies are imperative to validate the foveation encoding technique used, the end to end latency of foveation as well as the parametrization chosen.

Another limitation of the work is it uses AVC coding standard. The newer and more efficient HEVC standard should be more efficient in terms of bandwidth requirement reduction. The most feasible option would be integrating x265, an open source implementation of HEVC, with GamingAnywhere. There is some work being done by the developers of GA in that direction. x265 API is similar to x264 API, so integrating the solution presented in this thesis should not be complicated.

The analysis of latency in Section 6.3 is not very rigorous and may suffer from inaccuracies in that the latency of Tobii Eye Tracker 4C is assumed to be same as its sampling frequency. A more rigorous approach would consider the latency of the eye tracker and even physiological latencies. Another significant limitation of this work that the latency of the eye tracker is not (accurately) measured. Tobii Eye Tracker 4C SDK is quite opaque about how the gaze location is calculated and what light filtering–used in the prototype, entails. Further, the experiments designed for measuring the latency of the Eye Tracker have too many unknowns and involve neurological delays, such as saccade delays and cognitive delay between registering a visual stimulus and actual eye movements, which makes the measurements inaccurate. However, latency measurements of a previous generation Tobii eye tracker (EyeX) conducted in [85] are encouraging.

8 Conclusion

In this thesis a prototype for foveated video streaming for cloud gaming is developed and evaluated. The background of and motivation for cloud gaming is introduced. The need for video throuhput reduction is established and foveated video streaming is introduced as a possible solution. The non uniform acuity of the HVS is discussed with respect to the phenomenon of foveation. Then gaze tracking and its need in foveated video encoding and streaming is introduced. Other elements of a cloud gaming system like video streaming, encoding etc are also discussed.

The elements of the prototype are introduced and discussed. The protype consisting of a gaze tracker and modified GA client and server software which also includes an x264 encoder installed respectively on a Windows laptop and a Linux tower server is discussed. The client software is modified to include a TCP client which sends gaze location data to the server as soon as possible. In the server an additional TCP server is introduced which recieves the gaze data from the gaze tracker. The gaze data is fed into an module which calculates quantization offsets for each macroblock of the currently available game play frame. These offsets are applied to the QP parameters which the X264 computes for each macroblock. This gameplay video is then streamed to the client.

Gaze location is briefly studied for some games to establish latency feasiblity of foveated video streaming for cloud gaming. The analysis shows that gaze location tends to be within small regions for most of the time and that rate of change of gaze is below 100px or 100 ms for vast majority of gaze locations which is encouraging from a latency point of view. Considering achievable latency in cloud gaming, as explored in other works, it is concluded that cloud gaming with foveated streaming is feasible.

The effect of foveation on video bitrate and network throughput is studied by capturing the traffic between the protoype client and the prototype server while playing a few test games with and without foveated encoding enabled. Foveated encoding is implemented at different parameters with each game. The results show video bandwidth reduction by more than 50% depending on the parametrization used. Some optimal parameters are suggested for the test games which show promise in terms of QoE as observed from visual inspection of the gameplay video at the prototype client.

References

[1] T. Kämäräinen, M. Siekkinen, A. Ylä-Jääski, W. Zhang, and P. Hui, "A measurement study on achieving imperceptible latency in mobile cloud gaming," in *To Appear in the Proceedings of the ACM Multimedia Systems Conference*, ser. MMSys '17. New York, NY, USA: ACM, 2017. [Online]. Available: https://users.aalto.fi/~siekkine/pub/kamarainen17mmsys.pdf

[2] D. W. Hansen and Q. Ji, "In the eye of the beholder: A survey of models for eyes and gaze," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 478–500, March 2010.

[3] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "Gaminganywhere: An open cloud gaming system," in *Proceedings of the 4th ACM Multimedia Systems Conference*, ser. MMSys '13. New York, NY, USA: ACM, 2013, pp. 36–47. [Online]. Available: http://doi.acm.org/10.1145/2483977.2483981

[4] R. Shea, J. Liu, E. C.-H. Ngai, and Y. Cui, "Cloud gaming: architecture and performance," *IEEE Network*, vol. 27, no. 4, pp. 16–21, 2013.

[5] B. Bing, *Next-generation video coding and streaming*. Wiley, 2015.

[6] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010. [Online]. Available: http://dx.doi.org/10.1007/s13174-010-0007-6

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[8] D. Takahashi, "Pwc: Game industry to grow nearly 5Jun 2016. [Online]. Available: https://venturebeat.com/2016/06/08/the-u-s-and-global-game-industries-will-grow-a-healthy-amount-by-2020-pwc-forecasts/

[9] I. 23009, "Dynamic adaptive streaming over HTTP (DASH)," 2012.

[10] J. Ryoo, K. Yun, D. Samaras, S. R. Das, and G. Zelinsky, "Design and evaluation of a foveated video streaming service for commodity client devices," in *Proceedings of the 7th International Conference on Multimedia Systems*, ser. MMSys '16. New York, NY, USA: ACM, 2016, pp. 6:1–6:11.

[11] Z. Wang and A. C. Bovik, "Foveated image and video coding," *Digital Video, Image Quality and Perceptual Coding*, pp. 431–457, 2006.

[12] "Games data and market research, market brief - year in review 2016." [Online]. Available: https://www.superdataresearch.com/market-data/market-brief-year-in-review/

[13] "Pwc predicts moderate growth for u.s. video games, fast growth for vr and esports," Jun 2017. [Online]. Available: https://venturebeat.com/2017/06/06/pwc-predicts-moderate-growth-for-u-s-video-games-but-fast-growth-for-vr-and-esports/

[14] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," 2001. [Online]. Available: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization#Basic_Architecture_of_a_Client_.2F_Server_Game

[15] "Specifications | samsung galaxy s8 and s8," 2017. [Online]. Available: http://www.samsung.com/global/galaxy/galaxy-s8/specs/

[16] "Intel® core™ i7-7700t processor," 2017. [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/core/i7-processors/i7-7700t.html

[17] E. Blem, J. Menon, and K. Sankaralingam, "Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2013, pp. 1–12.

[18] S. Wang and S. Dey, "Adaptive mobile cloud computing to enable rich mobile multimedia applications," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 870–883, June 2013.

[19] "Gamefly," 2017. [Online]. Available: https://www.gamefly.com/#!/streaming

[20] "Loud play." [Online]. Available: http://www.loud-play.com/

[21] "Utomik." [Online]. Available: https://www.utomik.com/

[22] "Nvidea grid." [Online]. Available: http://www.nvidia.com/object/cloud-gaming.html

[23] "Geforce now." [Online]. Available: https://www.nvidia.com/en-us/shield/games/#geforcenow/

[24] "Playstation now – ps now subscription for ps3 games." [Online]. Available: https://www.playstation.com/en-us/explore/playstationnow/

[25] "Gaikai.com :: History." [Online]. Available: https://www.gaikai.com/#!/history

[26] W. Cai, M. Chen, and V. C. M. Leung, "Toward gaming as a service," *IEEE Internet Computing*, vol. 18, no. 3, pp. 12–18, May 2014.

[27] I. S. Mohammadi, M. R. Hashemi, and M. Ghanbari, "An object-based framework for cloud gaming using player's visual attention," in *2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, June 2015, pp. 1–6.

[28] W. Cai, R. Shea, C. Y. Huang, K. T. Chen, J. Liu, V. C. M. Leung, and C. H. Hsu, "A survey on cloud gaming: Future of computer games," *IEEE Access*, vol. 4, pp. 7605–7620, 2016.

[29] Y.-T. Lee, K. T. Chen, H.-I. Su, and C. L. Lei, "Are all games equally cloud-gaming-friendly? an electromyographic approach," in *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, Nov 2012, pp. 1–6.

[30] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, vol. 49, no. 11, pp. 40–45, Nov. 2006. [Online]. Available: http://doi.acm.org/10.1145/1167838.1167860

[31] M. Jarschel, D. Schlosser, S. Scheuring, and T. Hoßfeld, "An evaluation of qoe in cloud gaming based on subjective tests," in *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, June 2011, pp. 330–335.

[32] Geforce now system requirements. [Online]. Available: http://shield.nvidia.com/support/geforce-now/system-requirements

[33] Playstation now on pc. [Online]. Available: https://www.playstation.com/en-gb/get-help/help-library/services/playstation-now/playstation-now-on-pc/

[34] "State of the internet report | akamai," 2017. [Online]. Available: https://www.akamai.com/us/en/about/our-thinking/state-of-the-internet-report/

[35] P. E. Ross, "Cloud computing's killer app: Gaming," *IEEE Spectrum*, vol. 46, no. 3, pp. 14–14, March 2009.

[36] M. Claypool, D. Finkel, A. Grant, and M. Solano, "Thin to win? network performance analysis of the onlive thin client game system," in *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, Nov 2012, pp. 1–6.

[37] K.-T. Chen, Y.-C. Chang, P.-H. Tseng, C.-Y. Huang, and C.-L. Lei, "Measuring the latency of cloud gaming systems," in *Proceedings of the 19th ACM International Conference on Multimedia*, ser. MM '11. New York, NY, USA: ACM, 2011, pp. 1269–1272. [Online]. Available: http://doi.acm.org/10.1145/2072298.2071991

[38] K. T. Chen, Y. C. Chang, H. J. Hsu, D. Y. Chen, C. Y. Huang, and C. H. Hsu, "On the quality of service of cloud gaming systems," *IEEE Transactions on Multimedia*, vol. 16, no. 2, pp. 480–495, Feb 2014.

[39] S. Choy, B. Wong, G. Simon, and C. Rosenberg, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, Nov 2012, pp. 1–6.

[40] M. Manzano, J. A. Hernández, M. Urueña, and E. Calle, "An empirical study of cloud gaming," in *2012 11th Annual Workshop on Network and Systems Support for Games (NetGames)*, Nov 2012, pp. 1–2.

[41] S. P. Chuah, C. Yuen, and N. M. Cheung, "Cloud gaming: a green solution to massive multiplayer online games," *IEEE Wireless Communications*, vol. 21, no. 4, pp. 78–87, August 2014.

[42] M. Semsarzadeh, M. Hemmati, A. Javadtalab, A. Yassine, and S. Shirmohammadi, "A video encoding speed-up architecture for cloud gaming," in *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, July 2014, pp. 1–6.

[43] Z. Y. Wen and H. F. Hsiao, "Qoe-driven performance analysis of cloud gaming services," in *2014 IEEE 16th International Workshop on Multimedia Signal Processing (MMSP)*, Sept 2014, pp. 1–6.

[44] W. Cai, Z. Hong, X. Wang, H. C. B. Chan, and V. C. M. Leung, "Quality-of-experience optimization for a cloud gaming system with ad hoc cloudlet assistance," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 2092–2104, Dec 2015.

[45] Y. C. Chang, P. H. Tseng, K. T. Chen, and C. L. Lei, "Understanding the performance of thin-client gaming," in *2011 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, May 2011, pp. 1–6.

[46] K. T. Chen, C. Y. Huang, and C. H. Hsu, "Cloud gaming onward: research opportunities and outlook," in *2014 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, July 2014, pp. 1–4.

[47] W. Cai, C. Zhou, V. C. M. Leung, and M. Chen, "A cognitive platform for mobile cloud gaming," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, vol. 1, Dec 2013, pp. 72–79.

[48] M. Wien, *High efficiency video coding: coding tools and specification.* Springer, 2015.

[49] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, July 2003.

[50] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec 2012.

[51] M. Xu, Y. Liang, and Z. Wang, "State-of-the-art video coding approaches: A survey," in *2015 IEEE 14th International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC)*, July 2015, pp. 284–290.

[52] (2016) Cisco visual networking index: Forecast and methodology, 2016–2021 - cisco. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html#_Toc484531491

[53] H. Schulzrinne, "Real time streaming protocol (rtsp)," 1998.

[54] V. Jacobson, R. Frederick, S. Casner, and H. Schulzrinne, "Rtp: A transport protocol for real-time applications," 2003.

[55] S. Wenger, Y.-K. Wang, T. Schierl, and A. Eleftheriadis, "Rtp payload format for scalable video coding," Tech. Rep., 2011.

[56] H. Parmar and M. Thornburgh, "Adobe's real time messaging protocol," *Copyright Adobe Systems Incorporated*, pp. 1–52, 2012.

[57] B. Wandell, *Foundations of vision*. Sinauer Associates, 1995.

[58] K. Rayner and M. Castelhano, "Eye movements," *Scholarpedia*, vol. 2, no. 10, p. 3649, 2007, revision #126973.

[59] D. Purves, G. J. Augustine, D. Fitzpatrick, L. C. Katz, A.-S. LaMantia, J. O. McNamara, and S. M. Williams, "Neuroscience. sunderland," *MA: Sinauer Associates*, 2001.

[60] E. B. Huey, *The psychology and pedagogy of reading*. The Macmillan Company, 1908.

[61] A. Al-Rahayfeh and M. Faezipour, "Eye tracking and head movement detection: A state-of-art survey," *IEEE Journal of Translational Engineering in Health and Medicine*, vol. 1, pp. 2 100 212–2 100 212, 2013.

[62] "An introduction to eye tracking and tobii eye trackers," 2010. [Online]. Available: http://www.acuity-ets.com/downloads/Tobii%20Eye%20Tracking%20Introduction%20Whitepaper.pdf

[63] Z. Zhu and Q. Ji, "Eye gaze tracking under natural head movements," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, June 2005, pp. 918–923 vol. 1.

[64] "Windows hello | windows 10." [Online]. Available: https://www.microsoft.com/en-us/windows/windows-hello

[65] J. Baxter, "Windows hello face authentication." [Online]. Available: https://docs.microsoft.com/en-us/windows-hardware/design/device-experiences/windows-hello-face-authentication

[66] E. L. Schwartz, "Computational anatomy and functional architecture of striate cortex: A spatial mapping approach to perceptual coding," *Vision Research*, vol. 20, no. 8, pp. 645 – 669, 1980. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0042698980900905

[67] C. F. R. Weiman, "Video compression via log polar mapping," pp. 266–277, 1990. [Online]. Available: http://dx.doi.org/10.1117/12.21244

[68] L. Itti, "Automatic foveation for video compression using a neurobiological model of visual attention," *IEEE Transactions on Image Processing*, vol. 13, no. 10, pp. 1304–1318, Oct 2004.

[69] Z. Wang, L. Lu, and A. C. Bovik, "Foveation scalable video coding with automatic fixation selection," *IEEE Transactions on Image Processing*, vol. 12, no. 2, pp. 243–254, Feb 2003.

[70] S. Lee, M. S. Pattichis, and A. C. Bovik, "Foveated video quality assessment," *IEEE Transactions on Multimedia*, vol. 4, no. 1, pp. 129–132, Mar 2002.

[71] H. Ahmadi, S. Zad Tootaghaj, M. R. Hashemi, and S. Shirmohammadi, "A game attention model for efficient bit rate allocation in cloud gaming," *Multimedia Syst.*, vol. 20, no. 5, pp. 485–501, Oct. 2014.

[72] Gaminganywhere - an open source cloud gaming system. [Online]. Available: http://gaminganywhere.org/

[73] x264. [Online]. Available: http://www.videolan.org/developers/x264.html

[74] Ffmpeg. [Online]. Available: http://ffmpeg.org/

[75] L. Merritt and R. Vanam, "x264: A high performance h. 264/avc encoder," *[online] http://neuron2.net/library/avc/overview_x264_v8_5.pdf*, 2006.

[76] git.videolan.org git - x264.git/blob - x264.h. [Online]. Available: http://git.videolan.org/?p=x264.git;a=blob;f=x264.h

[77] Tobii eye tracking. [Online]. Available: https://www.tobii.com/

[78] Tobii, "Developer's Guide tobii EyeX SDK for C/C++," 2015. [Online]. Available: http://developer-files.tobii.com/wp-content/uploads/2016/03/Developers-Guide-C-Cpp.pdf

[79] W. R. Leo, "The gaussian or normal distribution." [Online]. Available: https://ned.ipac.caltech.edu/level5/Leo/Stats2_3.html

[80] tcpdump. [Online]. Available: http://www.tcpdump.org/

[81] "Wireshark." [Online]. Available: https://www.wireshark.org/

[82] R. Carpenter, "Movements of the eyes 2nd edition (london: Pion)," 1988.

[83] J. Findlay and R. Walker, "Human saccadic eye movements," *Scholarpedia*, vol. 7, no. 7, p. 5095, 2012, revision #122018.

[84] C. H. Meyer, A. G. Lasker, and D. A. Robinson, "The upper limit of human smooth pursuit velocity," *Vision research*, vol. 25, no. 4, pp. 561–563, 1985.

[85] A. Gibaldi, M. Vanegas, P. J. Bex, and G. Maiello, "Evaluation of the tobii eyex eye tracking controller and matlab toolkit for research," *Behavior Research Methods*, vol. 49, no. 3, pp. 923–946, Jun 2017. [Online]. Available: https://doi.org/10.3758/s13428-016-0762-9

[86] D. R. Saunders and R. L. Woods, "Direct measurement of the system latency of gaze-contingent displays," *Behavior Research Methods*, vol. 46, no. 2, pp. 439–447, Jun 2014. [Online]. Available: https://doi.org/10.3758/s13428-013-0375-5

[87] "What's the difference between tobii eye tracker 4c and tobii eyex?" [Online]. Available: https://help.tobii.com/hc/en-us/articles/212814329-What-s-the-difference-between-Tobii-Eye-Tracker-4C-and-Tobii-EyeX-

[88] E. Wood and A. Bulling, "Eyetab: Model-based gaze estimation on unmodified tablet computers," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '14. New York, NY, USA: ACM, 2014, pp. 207–210. [Online]. Available: http://doi.acm.org/10.1145/2578153.2578185

[89] "Gazepointer." [Online]. Available: https://sourceforge.net/projects/gazepointer/

[90] "openeyes - open source software." [Online]. Available: http://thirtysixthspan.com/openEyes/software.html

[91] E. Miluzzo, T. Wang, and A. T. Campbell, "Eyephone: Activating mobile phones with your eyes," in *Proceedings of the Second ACM SIGCOMM Workshop on Networking, Systems, and Applications on Mobile Handhelds*, ser. MobiHeld '10. New York, NY, USA: ACM, 2010, pp. 15–20. [Online]. Available: http://doi.acm.org/10.1145/1851322.1851328