UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Application of deep learning upscaling technologies in cloud gaming solutions

Efrén Carles Ramon

Bachelor Thesis
Specialization in technology systems

Director: Alex Pajuelo Gonzalez
1st February, 2022

Resum

Durant tota la seva història, la indústria dels videojocs ha vist com any rere any els requeriments hardware de les entregues més populars del mercat augmentaven amb cada nova sortida. Això, juntament amb l'encariment dels components hardware necessaris per poder executar aquests videojocs causat per la pandèmia i per l'auge del mercat de les criptomonedes ha condicionat al sorgiment d'un nou tipus de servei: el cloud gaming.

Utilitzant tecnología de streaming de video, el cloud gaming és capaç d'oferir als seus usuaris la capacitat de poder jugar a qualsevol de les últimes entregues del mercat des de qualsevol peça de hardware. Aquesta nova aparició, però, no ha ocorregut sense dificultats, dons els alts requeriments pel que fa a latència i ample de banda dificulten a l'usuari a l'hora de tenir una bona experiència amb el servei.

Per tal d'intentar millorar aquesta experiència, en aquest treball s'exploraran les possibles millores que podria aportar l'ús de tècniques de machine learning a l'hora de realitzar un reescalat del tràfic de video per tal d'aconseguir una major resolució final a partir d'una resolució menor de transmissió, reduint l'ample de banda necessari i subseqüentment la latencia del servei.

Resumen

Durante toda su historia, la industria de los videojuegos ha visto cómo año tras año los requerimientos de hardware de las entregas más populares del mercado aumentaban con cada nueva salida. Esto, junto al encarecimiento de los componentes hardware necesarios para poder ejecutar estos videojuegos causado por la pandemia y por el auge del mercado de las criptomonedas ha condicionado a la aparición de un nuevo tipo de servicio: el cloud gaming.

Utilizando tecnología de streaming de vídeo, el cloud gaming es capaz de ofrecer a sus usuarios la capacidad de poder jugar a cualquiera de las últimas entregas del mercado desde cualquier pieza de hardware. Sin embargo, esta nueva aparición no ha ocurrido sin dificultades, porque los altos requerimientos en cuanto a latencia y ancho de banda dificultan al usuario a la hora de tener una buena experiencia con el servicio.

Con el fin de intentar mejorar esta experiencia, en este trabajo se explorarán los posibles beneficios que podría aportar el uso de técnicas de machine learning a la hora de realizar un re-escalado del tráfico de vídeo para conseguir una mayor resolución final a partir de una resolución menor de transmisión, reduciendo el ancho de banda necesario y subsiguientemente la latencia del servicio.

Abstract

Throughout its history, the video game industry has seen the hardware requirements of the most popular titles on the market increase year after year with each new release. This, in conjunction with the rise in the cost of the hardware components needed to run these video games caused by the pandemic and the rise of the cryptocurrency market, has led to the creation of a new type of services: cloud gaming.

Using video streaming technology, cloud gaming is able to offer its users the ability to play any of the latest releases on the market from any piece of hardware. This new appearance, however, has not happened without difficulties, because the high requirements in terms of latency and bandwidth make it difficult for the user to have a good experience with the service.

In order to try to improve this experience, this work will explore the possible improvements that could be made by the use of machine learning techniques when re-scaling video traffic to achieve a higher final resolution from a lower transmission resolution, reducing the required bandwidth and subsequently the latency of the service.

Als meus pares, que han sacrificat el seu temps i la seva salut per a que jo pugui ser on sóc.
En especial a tu, pare. Espero que, siguis on siguis, estigues orgullos de mi.

Contents

List of Figures

List of Tables

1 Context and Scope

1.1 Context

This is a Bachelor's thesis of the Computer Engineering Degree, specializing in technology Systems, taught by the Facultat d'Informàtica de Barcelona of the Universitat Politècnica de Catalunya, done by the student Efrén Carles, directed by Alex Pajuelo.

1.2 Introduction and concept definition.

Gaming is, nowadays, a worldwide phenomenon that started poking it's way at history when in the 1950s and 1960s computer scientists started designing simple games as a way of having fun while testing new technologies [1] . The well known game Pong, a tennis simple simulation with a very primitive AI that challenged the player, was created and developed in 1958 by physicist William Higinbotham [2]. In today's world, those simple simulation games have become multi-million dollar projects developed by teams of thousands of people and the complexity of said games has scaled exponentially.

Not only have we transitioned from a 2D pixel environment to a fully 3D photorealistic one, but the mechanical depth of the industry has also increased. The simple concept of a tennis match with an AI has turned into a plethora of systems and simulations that work together to make sure that the player lives a meaningful and fun experience. Cases like Grand Theft Auto 5, developed and launched by rockstar games in September 17, 2013, where the player has a fully fledged modern city to interact with are possible thanks to a multitude of interconnected systems, such as a traffic simulation, a pedestrian control system, real time weather and day-night cycle, physics simulations, and the list goes on.
Games like that are a testament to the graphical and technical fees the industry is capable of accomplishing. But even though GTA5 is a pretty old game by today's standards, it can be run on almost any machine in today's market. There are plenty of more recently released games that, due to their increasing complexity, can only be executed in very powerful machines. This, combined with the shortage of dedicated graphic devices that has ravaged the gaming market in the past 5 years, has presented us with a problem: nowadays, a lot of people can't play most of the games that are being released.

The price of gaming

As of today's 1st of February, 2022, the price of a computer capable of running all the newly released titles is about 800 to 1100 euros [3]. This machine is not a high-end one, therefore it won't be capable of running all the new games at maximum graphical configurations such as 4K resolutions or ray tracing. This, combined with the fact that most of the newly released titles enter the market with a price of 60 to 70€ makes gaming a very expensive hobby.

Having this in mind, it’s easy to understand why cloud gaming solutions do exist.

Remote game execution

Not being able to run the latest games without upgrading your machine is a reality for a lot of consumers. This, added to the fact that mobile gaming is one of the fastest growing branches of the industry nowadays [4], has conditioned the creation of a different kind of service that offers a solution to those problems.

Much like now obsolete video rental stores, cloud gaming offers a subscription service which allows the client to play a very complete catalog of games without time restrictions, with the added bonus that the game is actually executed in a, more powerful, remote machine. This is a very interesting solution, because with a machine just capable of sending the player’s inputs and receiving a video stream with the game playing is enough to play any game on the market. This method of playing is usually way cheaper than the classical one, not just because you avoid having to buy a PC that will probably cost you several thousand euros, but because the subscription fees usually aren’t more than 30€ a month, offering you not just one game but a full catalog of them [5].

Another added bonus is that the client doesn’t have to download the full game to play, which, considering the fact that the vast majority of games that are released nowadays weigh on average 50 GB or more, it’s not something that can be ignored when considering the advantages of such services.

Old problems of the new solutions

But, even with all of its advantages, cloud gaming is not as popular nowadays as one might think it would be. Some competitors of the market, like Google Stadia [6], can very well reflect the state of the market. In short, it’s not a very popular solution, and the reason for that resides on the very core of the media that it operates on: responsiveness.

Responsiveness is a key, especially in some fast-paced genres, for a good game[7]. There needs to be a correlation between the player’s inputs and the game’s response, and it needs to be almost instantaneous. When the player presses a button, something needs to happen on screen. This is basic and very easy to accomplish when we talk about classic, same-machine, gaming. But when a remote execution is added to the system, things can change.

Latency is a term used to describe the time that is needed to send a packet from a server to a client and receive a response confirming that said packet has arrived without errors.

Latency has been an issue in the World Wide Web since its creation, due to the fact that it's a very decentralized problem. Latency can be added at every step of the two-way communication that occurs on the internet. In the client’s machine, in the client’s network, in

the service provider’s network, in the server’s machine, etc. This is mainly why it was an issue 40 years ago and it’s still a problem nowadays, even though it has improved substantially. But with cloud gaming, things are much more complicated. Having more than 50 milliseconds of latency while playing can totally ruin the player’s experience, rendering the service almost unusable.

It’s true that some other similar services, like real-time video streaming, had faced similar problems and were able to solve them with a plethora of techniques such as buffering. Other services, like websites, use what it’s called a global distribution for static content. A technique that can reduce latency to very low values ranging 10 to 20 milliseconds. But, as exposed previously, cloud gaming is different. The content that needs to be served is completely dynamic. It depends entirely on the actions of the player, therefore it cannot be buffered or statically distributed.

Other techniques native to the gaming media can also be used, like player prediction input. This consists of a prediction model that aims to predict the future player’s input in order to reduce the server’s response time. This, while in theory this could solve the problem of latency, is very computationally expensive and can very often fail, leading to additional latency, instead of reducing it.

1.3 Problem to resolve and potential solution

The power of deep learning

The problem is clear, there’s a latency issue that needs to be resolved in order to improve the usability of the cloud gaming services, but what solution is there to offer? In this thesis I want to test if the use of deep learning technologies can help reduce latency and improve user experience.

But how exactly can deep learning be used to solve this problem? Loss-less resolution upscaling will be the technique this thesis will be based on.

Undoubtedly, the more expensive part of the network transaction between the server and the client is not the player's inputs but the video feed of the gameplay. It's easy to grasp why sending video at 1080p resolution (this is the minimum resolution by today's standards) or more can be very bandwidth expensive. To better identify this, imagine what are the base undivisive packets of the transaction. The video stream is composed of individual frames. These frames can only be displayed one at the time and will only be displayed once the full frame has been received by the client. Having this in mind, we can assume that by reducing the time needed to send and receive a frame we can be able to reduce the reaction time, therefore improving the experience of the user. But this is already possible, if we compress the video stream server-side and then we decompress it in the client, we can greatly reduce the frame size. The problem is that all existing video compression algorithms are not loss-less and have a noticeable impact on video quality. It's true that maybe the most important thing in gaming is responsiveness, but video quality doesn't stand far behind, and there's not much quality we can compromise to achieve the first one.

Here's where deep learning comes into place.

By using a well-trained neural network, we are able to upscale a video feed without the major losses that compressing and decompressing it can add to it. This idea is not new, there are already several existing solutions that use this technique in order to improve the user's experience in classic gaming. NVIDIA's own technology, called DLSS [8], uses a neural network specifically trained to upscale a video feed of a certain game. This is possible thanks to a specific hardware part, included in the 2000 graphics card generation and beyond, called tensor cores.

Tensor cores

Tensor cores are designed with neural network operations in mind, making the process of executing them much more efficient while also relieving pressure on the graphics device [9]. This technique is already being used to upscale the resolution in classic gaming due to the fact that rendering a game at 1080p and then upscaling it to 2k or 4k its way cheaper computationally speaking than rendering directly at 2k or 4k, and the difference in quality between those options is minimall.

Upscaling method

However, in this thesis DLSS or tensor cores won't be used for the following reasons:

1. DLSS is a proprietary software owned by NVIDIA and can only be executed with tensor cores, making it difficult to work on
2. Tensor cores are only included in NVIDIA's 2000's graphics card generations and beyond and we only have available a 1000's generation one, more specifically a GTX 1060 Ti.
3. For the sake of open source code.

As stated in the previous list, I want the code included in this thesis to be open source, therefore, I can only work with already existing code that is also open source. Instead of DLSS, I'll use an open source alternative deep learning technology called FSRCNNX [10]. This technology is open source and can be executed in any graphics device, at the cost of course of being less efficient than the NVIDIA's option.

There are already some projects that use FSRCNNX to upscale games and other video streams, but none are adapted to cloud gaming solutions. The project that I will use as a baseline is an open source tool called Magpie [11], which offers a very complete list of upscaling technologies for various video streaming feeds.

Finally, the end goal of this thesis is to test if, by rendering the game at a lower resolution server side (i.e. 420p or 720p) and then applying upscaling can we improve the end user's experience, making the cloud gaming services more usable.

1.4 General objectives of the project.

In order to successfully perform the tests that we want to perform, we will have to develop a testing platform. However, this is not the objective of this project, the objectives of this project are:

1. To successfully integrate machine learning algorithms into a cloud gaming platform.
2. To test if said algorithms are usable to improve the performance of the service and the user experience.

Then, a set of sub objectives could be:

1. To make the implementation robust.
2. To make the implementation efficient.
3. To make the implementation stable.

1.5 Scope

The end goal of this thesis can be divided into two major goals (that will also be divided in more small, sub-divided tasks).

A very customizable cloud gaming platform will be needed in order to perform the necessary tests to conclude if deep learning can improve the end user's experience. Due to the fact that I don't personally own a cloud gaming platform and that the more commercial solutions that already exist won't offer as much freedom of use as I need in order to perform the necessary tests to extract the conclusion of the thesis, my first goal is to code a small server plus client in C++ that will help me simulate my own cloud gaming platform. This two pieces will work in conjunction in the following way:

1. The server, executed in a remote machine (using windows OS), will capture the screen using various windows API's.
2. After capturing a video stream of the machine's screen, the server will send the uncompressed video to the client.
3. The client, which previously will need to establish a connection with the server, will receive the video feed.
4. Once an entire frame is received, FSRCNNX will be applied to the video feed before displaying it in the client's machine screen.
5. Having this done, I will be able to perform all the tests needed in order to test this thesis idea.

Some may argue that in order to test if this method works i should use a real cloud gaming platform but having in mind that I only have 4 months of time to develop the project I had to take the following thinks in consideration:

1. Complete cloud gaming platforms are big, complicated and expensive, something way out of my reach.
2. Even if I had the means of hosting this kind of platform I would take several months just to set it up
3. Considering that the only aspect of cloud gaming that I want to work on is the video streaming part, it's safe to assume that I don't need a whole platform and that I can work with just a simple client and server that simulates that aspect of the platform.

Once I have the platform and the tests using FSRCNNX have been conducted, my second goal is to train my own neural network to see if I can improve the process of upscaling.

FSRCNNX is a neural network that is trained to upscale any type of game. This, while being handy for a multi-purpose tool, it's probably not as efficient as having a dedicated neural network for each game, so to make the process even more precise, I plan on training a dedicated neural network for the specific game that I choose as a testing game for my thesis.

This second goal is, in my opinion, the one that adds more risk to the thesis. I think this because:

1. I don't have any previous experience training neural networks
2. I don't have a usable dataset, which is strictly necessary in order to train said network
3. The training process is very time consuming.
4. Machines capable of performing this training efficiently are very expensive.

That's why I consider this goal optional and while I will actively try to achieve both goals in the time that I have, I won't consider the thesis failed if I don't achieve it.

1.6 Methodology and version control

For the methodology of work I choose agile as my main way of organizing my work. Agile's methodology's objective is to divide the work into small sub-tasks that can be developed and tested individually. Not only do I think this will help me keep motivated and will make the progress steady but it's also the work methodology that I personally have more experience with.

I will divide the project in different sections that will also be divided into numerous individual tasks. These sections are thematically divided, for example programming is a section on itself while research and technical writing are both part of the same section. This will help me keep the consistency of the project throughout the development process. I'll also top the development of each individual task with a short meeting with the project's supervisor as a way of getting a final review of the work done. This will help me correct any mistakes

that I could've missed during the testing process.

To coordinate all of this I'll use Notion, a very popular solution that is capable of creating workspaces where a team or an individual can coordinate and manage projects. I choose this alternative and not other competitors like jira or trello because for me it's the middle point. On one hand Trello is too simple and I will probably need more features than the ones that it provides. On the other hand, jira is too complicated. I personally think that it is a tool thought to be used by large teams and corporations, and not just individuals.

I'll use github for version control. Even though I'm the only person working on this project I think github is a great tool and I'm very comfortable using it, again having a couple of years of experience with it.

1.7 Stakeholders

Considering the open source nature of this project it's safe to assume that any cloud gaming platform will benefit from this thesis if the conclusions are positive. And even if they aren't, and the use of deep learning doesn't improve the latency and the reaction time, it still can be used just for its efficiency improvements. It has already been proved with NVIDIA's DLSS that rendering a game at 1080 and then upscaling it at greater resolutions is way computationally cheaper than natively rendering that game at greater resolutions. By including this technique in a cloud gaming solution it is possible to make the platform more efficient, therefore making it more economically viable and consequently more cheap to the end user.

Finally, I consider myself as a stakeholder too, even if I'm not a main consumer of this type of services, due to the fact that I would love to contribute in a meaningful way to the open source community. I've previously had a couple of projects of the same nature that are available to the public and for me it's a wonderful experience.

2 Project planning

This project lasts approximately 460 hours, distributed between 132 days, lasting from 15/02/2022 to 27/06/2022. The exact date of the project's oral defense has not been decided yet, hence the end date of the project is the earliest date in the range of the turn that i choose to present on. I plan to work 3 to 4 hours every day, but this objective can be disturbed by exams and other student duties.

2.1 Task definition

The tasks that include all the work necessary to finish the project can be divided into 4 different categories: **Programming**, **Research**, **AI training**, **testing** and **project management**. These 5 categories are not in chronological order, meaning that tasks from all

of them can occur simultaneously. This is especially true for project management tasks that can and will occur during all the duration of the project.

The first type of tasks that I will define will be the project management tasks, seeing that they're a crucial first step after I can actually start developing the project. Said tasks will be:

- **Project planning and general direction:** It's important to define what the project is about and what are my objectives. This stage doesn't need to be too specific, but rather define the general direction where I want to move to.

- **Project scope:** This stage is the one where we define the specifics of the project. An achievable goal and a realistic scope are very necessary if I want the project to succeed.

- **Tools and project management software:** Defining the tools based on my needs is a crucial step that will avoid all the errors and trouble that working with unfit tools could cause.

- **Project planning:** Defining the separated tasks and managing them and the time that they require is the basics of project management. This step, even though it's essential, is subject to change in the future.

- **Budgeting and sustainability:** Every project needs to plan for the use of its resources, more specifically its budget, in order to be sustainable.

- **Final project definition:** This step involves defining what would be the desired final state of the project. Considering our goals and the time that we dispose of.

- **Meetings with the tutor:** For this project to succeed, it is crucial that a good communication between the researcher and the tutor is established.

The second type of tasks are the programming tasks, which can easily represent 60 to 70 percent of the total work needed to complete the project, due to the nature of this thesis. The tasks are the following:

- **Screen capture programming:** The first step of our server-client application is to capture the server's screen in order to send the game execution render to the client. This step is not crucial due to the fact that the windows API's that I need to use can be complicated to understand.

- **Network communication between server and client:** This tasc is maybe the easiest of the 3. Socket programming is not complicated and requires almost zero research.
- **AI shader client application:** The second possible obstacle in the programming phase of this project is to be able to apply the AI effect onto the video feed. This can and possibly will be difficult to implement.

Testing is another type of task that is required not chronologically but during all the duration of the project. This includes:

- **Screen capture testing:** This is the most simple of all testing tasks. It only requires to test if the screen can be captured correctly.

- **Network error testing:** Test if the network communication between the client and the server is established correctly and if the traffic is sent without errors.

- **Default AI shader application testing:** This is maybe the most important test of the thesis. The objective of this test is to check if the application of the AI shader can reduce the latency of the overall transaction.

- **Custom AI shader application testing:** Again, this test will only be performed if the time that I dispose of allows it. This test will check if the use of a custom AI improves the performance of the upscaling.

What can be the most complicated type of tasks, hence I've already mentioned that they are optional and part of a second goal, are the AI training tasks. These tasks represent the workload of training an AI for the specific game that I choose to test my thesis on. This can be difficult due to the fact that I've never done something of this nature before:

- **Training data recollection:** In order to train an AI a high volume of data is needed. Considering the fact that this AI is a video-based AI, the data volume is even bigger than other datasets. This data needs to be recollected from somewhere.

- **Training data preprocessing:** Training an AI is not a straightforward process. I can't just put a gigantic dataset of videos throughout the neural network and expect it to work. The data needs to be preprocessed in order to make it suitable for the training process.

- **Training process:** The most time consuming task of all of the AI related tasks. I currently have no idea how much time it will require. It will be done probably in some cloud platform specifically designed for AI training.

A lot of the previously mentioned tasks require knowledge that I don't posses, that's why extensive research will need to be done in order to be able to follow my own planning:

- **Screen capture research:** There are multiple methods to capture the screen in windows, each one with it's pros and cons, which should be analyzed properly to decide what graphic pipeline suits best my needs.

- **Shader application research:** Upscaling the video feed in the client using the AI is not an easy process to implement. Luckily, we have an implementation example in magpie's github repository, but as all external code it needs to be researched and understood in order to use it.

- **AI training research:** As I previously mentioned, it's the first time I work with neural networks and I know very little about their training process. I need to research profusely in order to understand it and to be able to implement it myself with my own dataset.

Once we're finished with all the previous tasks, we will have to document everything. First, we will have to collect all the information obtained in the test part. Afterwards, we can start writing the documentation of the project. Finally, we will have to prepare for the oral defense for the presentation of the project. To do so, we will think about possible questions that may come to mind to the senior FIB TFG tribunal members.

2.2 Resource definition

Human resources

In this project, there are 7 key roles that take part in different tasks in order to assure the success of the project: the **project manager** is responsible for the planning and correct supervision of the project development, the GEP tutor, the project tutor and I will play this role. A **programmer** is also needed alongside a **Tester** in order to correctly develop and test the code of the project, both roles will be played by me. An **AI trainer** will be needed in the case that I reach the second goal of the project, so that an AI can be trained and tested correctly, this role will also be played by me. A **Researcher and technical writer** is also needed, there are a lot of difficult technical subjects that need to be researched so that the programmer and other roles can work correctly. I will be in charge of the research and the technical writer. Finally, the **DevOps** role is responsible for server planning and deployment. This project is inherently related to networking, therefore a DevOps is needed to make sure the servers used to test the project are deployed correctly, I will also play this role.

Hardware resources

During the development of this project I will use two different machines to work, my PC and my laptop. Their respective specifications are:

- Intel core i5-7600K, 16GB RAM, NVIDIA 1060-TI graphics card.(PC)
- Intel core i7-8565U, 16GB RAM, Intel UHD Graphics 620.

Additionally, we also have to take into account all the resources for the connection to the network. Finally, we may also use a USB memory in case it is needed to pass information from one computer to another one.

Software resources

Several software resources will help us throughout the development of the whole project.

To manage the tasks and my time, I will use Notion. To meet with my project tutor I will use Google Meet. To code and controll my code I will use Visual Studio 2019 and GitHub. The Gantt chart will be created using Ganttproject. I will program using C++, with its own compiler and I will use DirectX11 and windows API's. Finally, if I accomplish everything I'll use Google Cloud to host the AI training process.

Material resources

In research projects there is always the need to get knowledge from previous studies and the area in question. In order to obtain this knowledge we will have to read books and more specifically papers.

2.3 Risk assessment and management

The project planning presented in this document is made having in mind a perfect scenario. But that's not realisting, problems and obstacles will appear and I need to be able to handle them accordingly if I don't want this project to fail. In this section I will present possible problematic situations and how I would handle them in case they do appear.

- **Deadline of the project [Risk-high]:** This can be caused due to bad planning and time management. Some tasks can take more than what I initially planned, delaying other tasks that have dependencies on them. As I'm planning this without having started some parts of the project for which I have little experience this risk is very real and possible. This can be solved by re-planning the project and re-arranging the tasks. If for some reason this scenario becomes real, the level of impact would be very dangerous, forcing me to move the presentation date to october.

- **Code errors [Risk-high]:** During any coding project, unexpected bugs and errors can occur and the time needed to fix them is not linear. It's possible that I become stuck while coding any part of the project, which can and will cause delays. This can be fixed by overestimating the time needed to perform these tasks, which I will do. When considering the impact that this would have on the project's development, it depends. Errors while coding are a common occurrence and they usually can be solved within a reasonable timeframe. That's why I believe this scenario to have a low impact on the project.

- **AI training inexperience [Risk-high]:** As I previously mentioned, I have little to no experience in AI training. That's why this represents a very high risk of delay. I tackled this problem by setting this part of the project to be optional. I will only try to do this once I finish the first and essential part of the project. I think that, amongst all the possible error scenarios, this is the one that's more probable to occur. The impact, again, would be quite high, but it would only affect the second part of the project.

- **Unsuccessful research [Risk-medium]:** Not finding a resource, even though nowadays it's quite uncommon due to the big accessibility that the internet provides us, it can still happen. Research is not easy and a scenario where I can't find enough information and resources on a topic is still very possible. To counteract this, I will have at my disposal not only the internet but also the Campus Nord library and many more sources that I can recur in the case that I can't find something online. This scenario's impact would depend on the matter of the unsuccessful research. For example if I can't find any information on a specific tool or resource I can just switch to a different one or change the way I tackle that specific problem. But if instead that the matter affected is one of the core elements of the project, such as the main upscaling algorithm or the used window's API's, then it can be a very blocking and impacting issue.

2.4 Gantt chart

ID	Name	Time (h)	Dependencies	Resources
T1	Project management	80		PC, laptop
T1.1	Project planning and general direction	12		PC, laptop
T1.2	Project scope	8	T1.1	PC, laptop
T1.3	Tools and project manafement software	8	T1.1,T1.5	PC, laptop
T1.4	Project planning	24	T1.1,T1.2	PC, laptop
T1.5	Budgeting and sustainability	8	T1.1	PC, laptop
T1.6	Final project definition	8	T1.1,T1.2,T1.5	PC, laptop
T1.7	Meeting with the tutor	12		PC, laptop, meet
T2	Project programming	114		
T2.1	Screen capture programming	48	T5.1	PC, laptop, VStudio, Github
T2.2	Network communication between server and client	18	T2.1	PC, laptop, VStudio, Github
T2.3	AI shader client application	48	T2.2,T5.3	PC, laptop, VStudio, Github
T3	Testing	52		
T3.1	Screen capture testing	8	T2.2	PC, laptop
T3.2	Network error testing	8	T2.2	PC, laptop
T3.3	Default AI testing	18	T2.3	PC, laptop
T3.4	Custom AI testing	18	T3.3, T4.3	PC, laptop
T4	AI training	80		
T4.1	Training data recollection	8	T5.3	PC, laptop, Youtube
T4.2	Training data preprocessing	36	T4.1	PC, laptop Google cloud
T4.3	Training process	36	T4.2,T5.2	PC, laptop, Google cloud
T5	Research	54		
T5.1	Screen capture research	12		PC, laptop
T5.2	Shader application research	18		PC, laptop
T5.3	AI training research	24		PC, laptop
T6	Project documentation	60		
T7	Oral defense preparation	20		
Total		460		

Table 2.1: Summary of the information of the tasks. [Own creation]

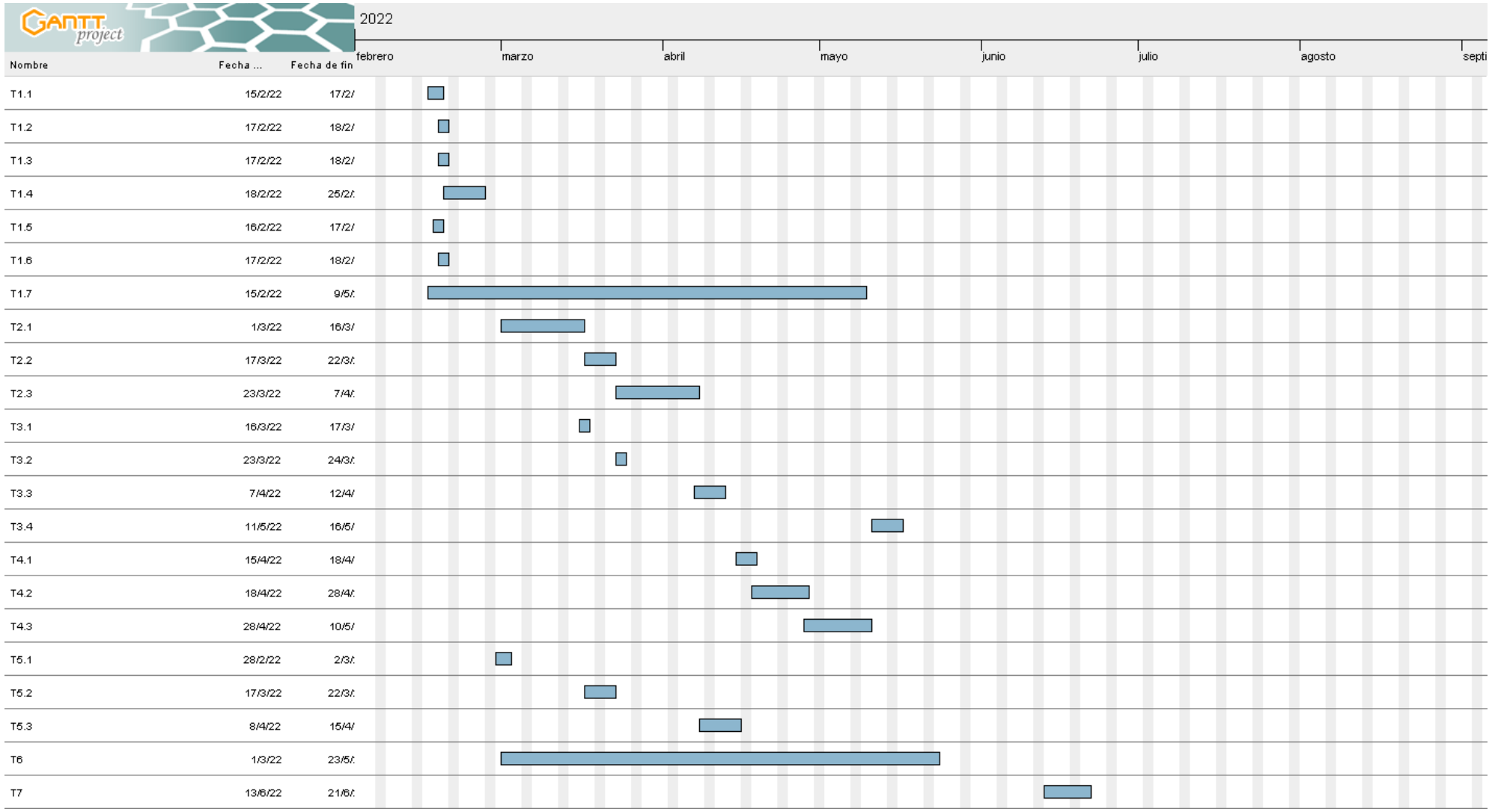

Figure 2.1: Gantt chart of the project. [Own creation]

3 Budget and sustainability

In this part of the project, I will describe what I took for consideration when assessing the budget for this project, including active and passive costs such as equipment and salary but also power consumption and internet access. Finally, the project's sustainability will also be questioned.

3.1 Budget.

Personnel costs per activity

In this section of the project I will compute the human cost of each and every task described in the 2.4 section. My methodology will be to first define what are the key roles of the project and then I will research what is, based on the current market, a normal salary for those roles. As stated before, in this project there are 7 key roles that take part in different tasks in order to assure the success of the project: the **project manager** is responsible for the planning and correct supervision of the project development, the GEP tutor, the project tutor and I will play this role. A **programmer** is also needed alongside a **Tester** in order to correctly develop and test the code of the project, both roles will be played by me. An **AI trainer** will be needed in the case that I reach the second goal of the project, so that an AI can be trained and tested correctly, this role will also be played by me. A **Researcher and technical writer** is also needed, there are a lot of difficult technical subjects that need to be researched so that the programmer and other roles can work correctly. I will be in charge of the research and the technical writer. Finally, the **DevOps** role is responsible for server planning and deployment. This project is inherently related to networking, therefore a DevOps is needed to make sure the servers used to test the project are deployed correctly, I will also play this role.

Based on what I think are the key roles of the project, now I need to research what are reasonable salaries for each of them. To do this, I will use the online platform known as GlassDoor. GlassDoor is an online tool, used by millions of users to share and report salaries of different technical positions. After some research, I found that those values are:

- Project manager: 51.163€ a year, more specifically 26.64€ an hour.
- Programmer: 43.670€ a year, more specifically 22,74€ an hour.
- Tester: 34.500€ a year, more specifically 17.9€ an hour.
- AI trainer: 35.000€ a year, more specifically 18.23€ an hour.
- Researcher and technical writer: 31.671€ a year or 16.49€ an hour.
- DevOps: 38.697€ a year, or 20,15€ an hour.

*These values were obtained from [12].

Those numbers were acquired from [12] and to find the hourly rates of each job I made the following assumptions: 12 pays, 4 weeks per month, 5 days per week, 8 hours per day.

Now that we have an hourly rate. We can determine the total cost of each task:

ID	Name	Time (h)	Project manager	Programmer	Tester	AI Trainer	Researcher and technical writter	DevOps	Cost (€)
T1	Project management	80	80	12	12	12	12	12	3277.80
T1.1	Project planning and general direction	12	12	0	0	0	0	0	0
T1.2	Project scope	8	8	0	0	0	0	0	0
T1.3	Tools and project manafement software	8	8	0	0	0	0	0	0
T1.4	Project planning	24	24	0	0	0	0	0	0
T1.5	Budgeting and sustainability	8	8	0	0	0	0	0	0
T1.6	Final project definition	8	8	0	0	0	0	0	0
T1.7	Meeting with the tutor	12	12	12	12	12	12	12	12
T2	Project programming	114	0	114	0	0	0	8	2753.56
T2.1	Screen capture programming	48	0	48	0	0	0	0	0
T2.2	Network communication between server and client	18	0	18	0	0	0	8	8
T2.3	AI shader client application	48	0	48	0	0	0	0	0
T3	Testing	52	0	0	52	0	0	0	930.8
T3.1	Screen capture testing	8	0	0	8	0	0	0	0
T3.2	Network error testing	8	0	0	8	0	0	0	0
T3.3	Default AI testing	18	0	0	18	0	0	0	0
T3.4	Custom AI testing	18	0	0	18	0	0	0	0
T4	AI training	80	0	0	0	80	0	0	1458.4
T4.1	Training data recollection	8	0	0	0	8	0	0	0
T4.2	Training data preprocessing	36	0	0	0	36	0	0	0
T4.3	Training process	36	0	0	0	36	0	0	0
T5	Research	54	0	0	0	0	54	0	890.46
T5.1	Screen capture research	12	0	0	0	0	12	0	0
T5.2	Shader application research	18	0	0	0	0	18	0	0
T5.3	AI training research	24	0	0	0	0	24	0	0
T6	Project documentation	60	0	0	0	0	60	0	989.4
T7	Oral defense preparation	20	20	0	0	0	0	0	532.8
Total		460							9942.76

Table 3.1: Total cost of each task compiled with the salary information extracted from [12]. [Own creation]

Generic costs

Amortization

The amortization of the hardware resources will be estimated by considering: The price of the resources when they were purchased, the fact that they most likely have 4 years of useful life of 4 hours daily use. I'll also estimate that 80% of the project will be done using the desktop computer and the other 20% with my laptop. Having this in mind:

Hardware	Price (€)	Time used (h)	Amortization (€)
Desktop computer	1.200€	368	75,61€
Laptop	800€	92	12,60€
Total			88,21

Table 3.2: Total cost of each hardware resource. [Own creation]

Electric

Regarding the electric cost, the actual cost of the kWh is 0.48567 €/kWh [13]. Having this in mind:

Hardware	Power(W)	Time used (h)	Cost(€)

Desktop computer	450W	368h	79,488€
Laptop	120W	92h	5,29€
Total			84,78€

Table 3.3: Total cost of power consumption by the hardware resources. The price for the kw/h used to compute the final numbers was extracted from [13]. [Own creation]

Internet

The internet rate costs 120€* per month. Having in mind that I will work during 4 months and a half, and that I will work approximately 4 hours per day, I can estimate that the internet cost will be 120€ x 4,5 (months) x 0,166 (percentage of day spended working) = 89.64€.

Travel costs

I'll need to travel to Barcelona at least once considering that the oral defense of the project is done presentially and that all the meetings with the project tutor will be done via Google Meet. To travel from my place of living to the FIB faculty and back I need 17€.

Work space

This project will be mainly developed in my house, located in Tortosa (Baix Ebre). The estimated rental cost in my area is approximately 656€ [14] but considering that I live with 3 more people I find it appropriate to divide the rent by 3. Hence the cost of the working space is 218,66€ spanning the 4,5 months of duration of the project this leaves us with a 984€ of total cost for the work space.

Generic cost of the project

To summarize all the generic costs of the project:

Concept	Cost (€)
Amortization	88,21€
Electric cost	84,78€
Internet cost	89,64€
Travel cost	17€
Work space	984€
Total	1263.63€

Table 3.4: Summarization of the generic costs of the project. [Own creation]

Total costs

The total cost expected for the project will be computed using the numbers found in the previous sections. In addition, I will add a 10% margin because it is well known that in almost all IT projects the budget cannot be calculated exactly, and therefore is only an estimation.

Concep	Cost(€)
Personal cost	9942,75€
Generic cost	1263,63€
10% Margin	1120.64€
Total	12327.02€

Table 3.5: Total monetary cost of the project. [Own creation]

3.2 Sustainability

Self-assessment

For me, sustainability only had one meaning: for it to be sustainable, to be environmentally friendly and to not cause harm to the environment, being reusable and recyclable. So in my case, the poll has made me realize that environmental sustainability is only one of the three possible dimensions that sustainability has (environmentally, socially and economically sustainable), moreover, it has made me realize that the IT industry is very closely related to those principles, with its impact on the planet and the human society growing every passing year.

I've also been surprised by the quantity of indicators that exist to evaluate the three dimensions of the problem and the importance of evaluating them in each dimension before starting the project in order to identify the possible problems that we can find and to find a solution to them right from the start.

To sum it up, after doing the poll I was left with the realization that I have pretty much no idea whatsoever about sustainability, so in order to write the following section of the project I'll heavily rely on the information that I've obtained from the previously mentioned poll.

Economic dimension

Regarding PPP: Reflection on the cost you have estimated for the completion of the project

The estimated cost of the project has been calculated in section 3.1. I've taken into consideration Human, hardware, software and material resources. It has also been taken into consideration a 10% margin in order to account for any possible problem or miscalculation. In my opinion, it is a reasonable budget and the project could be carried out in real life by a team of people, although the price of the working space may vary.

Regarding Useful Life: How are currently solved economic issues (costs...) related to the problem that you want to address (state of the art)?

This is a very important part of the project, due to the fact that the current problem (the unpopularity of the services due to their performance problems) are being solved by overprovisioning, consuming more resources. If this project succeeds, it'll help reduce the consumption of energy of the machines used to host the services and the internet traffic produced by them.

How will your solution improve economic issues (costs ...) with respect to other existing solutions?

As stated in the previous section, rendering a game at 1080p to then upscale it with a neural network is way computationally cheaper than rendering it directly at 2-4k. This will allow the cloud gaming services to be hosted in cheaper less powerful machines.

Environmental dimension

Regarding PPP: Have you estimated the environmental impact of the project?

Computers compute, and they do so by using energy. The computational power needed to render a game at 4k is massively bigger than the one needed to render a game at 1080p. By reducing it, we're also reducing the energy consumption of the computer. The traffic between the server and the client is also greatly reduced by lowering the resolution of the video sent to the client.

Regarding PPP: Did you plan to minimize its impact, for example, by reusing resources?

The only possible resource that can be reused is code, which I'm already re-using from various sites and parallel projects. Reducing the time and testing needed to develop the project, consuming less power and human resources.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)?

The problem, viewed from an environmental perspective, is solved by overprovisioning, therefore not taking into consideration the consumption of the services. With this solution, quality of service can be increased while power consumption is reduced.

How will your solution improve the environment with respect to other existing solutions?

As stated before, my solution is more energy efficient than the current solution, therefore it's better if viewed from an environmental view.

Social dimension

Regarding PPP: What do you think you will achieve -in terms of personal growth- from doing this project?

The scale, the complexity and the difficulty of the project is something that I've never seen in any other project that I worked on. I'll need to improve my skill with almost all of the different sets of skills of the roles that I take part in. It's a very new and exciting experience that I'm very pleased to take part in.

Regarding Useful Life: How is currently solved the problem that you want to address (state of the art)?

Currently, the problem is solved by overprovisioning, which is not ideal nor scalable.

How will your solution improve the quality of life (social dimension) with respect other existing solutions?

It will make the cloud gaming services more useful by improving the user experience while also making them more efficient and cheap to host.

Regarding Useful Life: Is there a real need for the project?

I would say there is, considering the low popularity and demand of the cloud gaming services. It will help users by improving the service's quality. I will help middle agents, like internet service providers by reducing the traffic, and it will help cloud gaming service providers, by reducing the power consumption of the servers hosting the service.

4 Creating a testing platform

In order to be able to test the thesis previously presented in this document, we will need to develop a basic platform that mimics a real cloud gaming platform but with the added bonus of being able to add any modifications that might be needed. In this section, I will explain with detail all the parts that will make up the platform. All the described functionalities will be implemented in c++ and the code will be available at this repository: https://github.com/i3q2-sys/DLUpscaling.

4.1 Defining our needs

First, in order to decide what pieces we need to develop we have to analyze what comprises a commercial cloud gaming platform. It's important to define what parts of a commercial solution are needed in our use case and what parts aren't, because for the sake of simplicity and time management we can't afford to code anything but the essential parts. A cloud gaming platform has the following functionalities:

- [Server] Offers a catalog of games to play.
- [Server] Remote execution of a desired game.
- [Server] Video stream capture, encoding and sending of the gameplay
- [Client] Video stream receiving, decoding, and rendering of the gameplay.
- [Client] Player input recording and sending
- [Server] Player input receiving and reproduction.

Observing the previous list, the easiest element to discard is the first one. Our testing platform definitely does not need to offer a catalog of games to play. Being it just a sandbox we can perform all of our tests with just one arbitrary decided game.

Going one by one, we can now analyze for each of the remaining elements of the list if we actually need to implement them in our test platform. The second element (Remote execution of a desired game) is undisputedly needed. How could we perform our tests of efficiency and execution cost reduction if we didn't actually execute a game? Luckily, to implement this functionality we only need to find an existing game seeing that the operating system used to develop this project (Windows) already supports game execution. The next functionality (Video stream capture, encoding and sending of the gameplay) is also needed. In order to perform a bandwidth and latency analysis we need to have a network traffic that mimics the one in a commercial cloud gaming solution. This, consequently, makes the next functionality also needed (Video stream receiving, decoding, and rendering of the gameplay)

for the same reason as the previous element of the list: we need to mimic the network traffic of a real platform. Another added reason is that we cannot perform the FSRCNNX upscaling algorithm without implementing a renderer. Finally, the last two elements of the list could be disputed. Seeing that the player input is also sended through the network it could be argued that, if we want to mimic a real platform's network traffic, we should implement this two functionalities. I, however, choose not to. The improvement in the latency and generally the network performance that we want to test in this project doesn't come from modifying at all the player's input network traffic, therefore in terms of testing results the player's input network traffic is neglectable in our use case. Having this in mind, we can conclude that the last two functionalities are not necessary to implement in our platform.

4.2 General architecture

Now, with a complete list of our needed functionalities, we can proceed to design our platform.

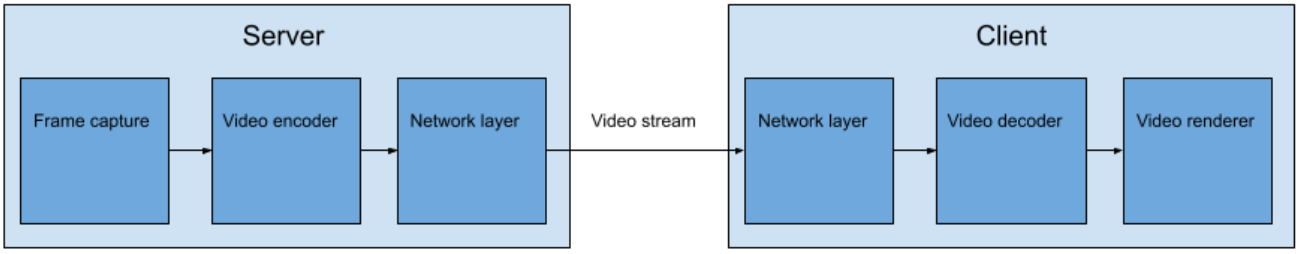

Figure 4.1: General architecture of the platform [Own creation]

The first proposed design follows a client-server architecture. Both one and the other are composed of three internal elements. It's important to note that this diagram (Fig 4.1) is not a class diagram but a functionality diagram, and therefore the final code could include more individual pieces. In order to make this project more understandable to the general public, we will now proceed to explain the functionality of each component without inquiring too much in the code implementation.

4.3 Server side

For the implementation of a working server that's responsible for recording the gameplay, encoding it in a video stream and sending it, we've decided to divide the functionality in three different parts.

Frame capture

The frame capture part of the server is responsible for capturing the screen of the machine that's executing the game. In commercial solutions, this process is made without any middleman: the executed game is not rendered to a screen but to a frame buffer that's encoded directly to a video screen. This, in our case, is not possible due to the fact that to achieve this we should be able to modify the source code of a game or directly implement one ourselves and this could take weeks of extra work. The problem that arises now is that the fact that the game is rendered to the screen and then a frame is captured from the screen could add some extra latency. This is especially dangerous because it could add noise and variance to the test results. Taking this risk into consideration, the screen capturing method selected for this job is the Desktop duplication API[15].

Desktop duplication API

The desktop duplication API provides us with a suiting solution for our problem. The thing that we want to avoid at all costs is added latency to the process of video streaming. Luckily for us, the Desktop Duplication API provides us with an almost zero latency solution. This API based on the Windows API returns the frame that is being displayed on the screen at the moment of calling. Not only that, but this API is also optimized to be called multiple times a second (ideally, our use-case scenario) and only executes when there's been an update in the displayed frame since the last time the API was called. Another added bonus is that due to the fact that it's an operating system call, it can access the frame buffer of the OS without depending on any middle man. The time cost of this access is equivalent to just a memory access, making the process much more efficient and straightforward. Finally, another added bonus is that because it shares its base architecture with DirectX (being both elements of the windows operating system), therefore the output of the Screen Capture calls is compatible with DirectX, also reducing complexity and learning time.

The Frame capture functionality element should be able to capture the screen in two different resolutions: Full hd (1920*1080) and qHD (960*540). This is needed to perform the needed tests that we'll explain in following sections, including the reasoning behind the specific choice of said resolutions.

We know we said we wouldn't be too technical with our explanations, but we find it necessary to include a snippet of the code responsible for the initialization and use of the Desktop Duplication API.

Initialization:

```cpp
ScreenCapture::ScreenCapture(VideoEncoder* p)
{
    hr = E_FAIL;
    //Capturing Device creation
    for (UINT DriverTypeIndex = 0; DriverTypeIndex < gNumDriverTypes; ++DriverTypeIndex)
    {
        hr = D3D11CreateDevice(
            nullptr,
            gDriverTypes[DriverTypeIndex],
            nullptr,
            0,
            gFeatureLevels,
            gNumFeatureLevels,
            D3D11_SDK_VERSION,
            &lDevice,
            &lFeatureLevel,
            &lImmediateContext);

        if (SUCCEEDED(hr))
        {
            // Device creation success, no need to loop anymore
            break;
        }
        lDevice.Release();
        lImmediateContext.Release();
    }
```

```cpp
    // get device
    CComPtrCustom<IDXGIDevice> lDxgiDevice;
    hr = lDevice->QueryInterface(IID_PPV_ARGS(&lDxgiDevice));
    if (FAILED(hr)) std::cout << "Failed QueryInterface" << std::endl;

    //get adapter
    CComPtrCustom<IDXGIAdapter> lDxgiAdapter;
    hr = lDxgiDevice->GetParent(
        __uuidof(IDXGIAdapter),
        reinterpret_cast<void**>(&lDxgiAdapter));
    if (FAILED(hr)) std::cout << "Failed GetParent" << std::endl;
    lDxgiDevice.Release();

    //Get output
    UINT Output = 0;
    CComPtrCustom<IDXGIOutput> lDxgiOutput;
    hr = lDxgiAdapter->EnumOutputs(Output,&lDxgiOutput);
    if (FAILED(hr)) std::cout << "Failed getting output" << std::endl;
    lDxgiAdapter->Release();
    lDxgiAdapter = nullptr;

    //QI for output
    CComPtrCustom<IDXGIOutput1> lDxgiOutput1;
    hr = lDxgiOutput->QueryInterface(IID_PPV_ARGS(&lDxgiOutput1));
    if (FAILED(hr)) std::cout << "Failed QI" << std::endl;
    lDxgiOutput.Release();

    //create desktop duplication
    hr = lDxgiOutput1->DuplicateOutput(lDevice,&lDeskDupl);
    if (FAILED(hr)) std::cout << "Failed creation of desktop duplication" << std::endl;
    lDxgiOutput1.Release();
```

```
    //Create 2d texture
    lDeskDupl->GetDesc(&lOutputDuplDesc);
    D3D11_TEXTURE2D_DESC desc;
    desc.Width = lOutputDuplDesc.ModeDesc.Width; //this can be substituted with the game's resolution
    desc.Height = lOutputDuplDesc.ModeDesc.Height; //same as above
    desc.Format = lOutputDuplDesc.ModeDesc.Format;
    desc.ArraySize = 1;
    desc.BindFlags = D3D11_BIND_FLAG::D3D11_BIND_RENDER_TARGET;
    desc.MiscFlags = D3D11_RESOURCE_MISC_GDI_COMPATIBLE;
    desc.SampleDesc.Count = 1;
    desc.SampleDesc.Quality = 0;
    desc.MipLevels = 1;
    desc.CPUAccessFlags = D3D10_CPU_ACCESS_READ;
    desc.Usage = D3D11_USAGE_DEFAULT;
    hr = lDevice->CreateTexture2D(&desc, NULL, &lDestImage);

    this->pVe = p;

}
```

Figure 4.2: Initialization of the Desktop Duplication API. (Own elaboration)

Frame capture:

```
bool ScreenCapture::CaptureScreen()
{
    HRESULT hr;
    hr = E_FAIL;

    CComPtrCustom<IDXGIResource> lDesktopResource;
    DXGI_OUTDUPL_FRAME_INFO lFrameInfo;
    hr = lDeskDupl->AcquireNextFrame(
        250,
        &lFrameInfo,
        &lDesktopResource);

    if (FAILED(hr))
    {
        return false;
    }

    if (lFrameInfo.LastPresentTime.HighPart == 0)
    {
        hr = lDeskDupl->ReleaseFrame();
        return false;
    }
```

```
    // QI for ID3D11Texture2D
    hr = lDesktopResource->QueryInterface(__uuidof(ID3D11Texture2D), reinterpret_cast<void**>(&lAcquiredDesktopImage));
    //lDesktopResource->Release();
    lDesktopResource = nullptr;
    if (FAILED(hr))
    {
        hr = lDeskDupl->ReleaseFrame();
        return false;
    }
    //copy aquired texture to a CPU accessable 2dtexture
    lImmediateContext->CopyResource(lDestImage, lAcquiredDesktopImage);
    hr = lDeskDupl->ReleaseFrame();
    return true;
}
```

Figure 4.3: Fragment of code responsible for the capture of a single frame. (Own elaboration)

Video encoder

While the Screen Capture module is in charge of capturing each frame of the video streaming we cannot just send them raw through the network and the reason why it's pretty simple. Let's suppose that we capture the screen in a 1080*1920 resolution at 30 frames per second (which again, is still not up to the industry standard, it should be 60 frames per second). We can compute the bandwidth needed to send all of that information through the network with the following equations:

$$bandwidth = frame\,rate * frame\,size$$
$$frame\,size = resolution * pixel\,size$$

Having in mind the fact that each pixel is represented with 4 bytes (Red, green, blue and transparency) and that our resolution is 1920*1080 pixels, we can calculate the size of each frame to be 82944000 bytes or, to work with smaller numbers 7.91 megabytes. If we have a frame rate of 30 fps that means that we are sending 30 frames each second, which is equivalent to 237.30 megabytes per second of bandwidth used to send the video streaming. This is a very unacceptable number. To put it in perspective, the mean bandwidth in the European Union as of June 22, 2021 is 103.3 megabits per second [16], which equates to 12,91 megabytes per second. This makes the platform unusable for the vast majority of the population. Luckily, this problem was faced and solved by video streaming companies a long time ago.

Video encoding is the process of converting RAW video files into a more suitable format that is not saved as individual whole frames but instead is stored as a fluid video. This process can be used to compress already existing video files to a smaller file size but it can also be used to transform individual images into a video stream. It's important to note that while we want to reduce the data size to be able to send it through the internet we absolutely don't want to sacrifice too much image quality. As stated in previous sections, image quality is one of the most important factors to consider when a cloud gaming platform wants to offer a good service, therefore, if we reduced the quality of image when encoding the video stream just for the sake of network bandwidth reduction we wouldn't be getting realistic network traffic metrics.

Video encoding is a very complicated matter, with lots of parameters, codecs (which we'll explain later) and algorithms. Implementing our own encoder would be a titanic task, so the best alternative would be to use an existing library to perform that operation. Based on its popularity, its availability in our coding language and its quality and efficiency, ffmpeg will be our chosen video encoding library.

FFMPEG

FFMPEG[17] is a leading multimedia framework capable of performing a vast array of different operations to video and audio streams. The project was started in the early 2000 by

Fabrice Bellard [18] and was led by Michael Niedermayer from 2004 to 2015. Said framework exists in the form of an open source tool written in C language, this makes using ffmpeg in our project a bit complicated. One way to include it would be to call console commands from our code to summon the tool and then, when it's finished, collect the result. This however is very inefficient. To call a command line tool from an already executing piece of code it's necessary to summon the operating system, which performs a context swap before executing the desired command line tool. If we need to summon ffmpeg once every frame (approximately 30 to 60 times a second) this way of using ffmpeg is not viable.

Luckily for us, there's another way. We can embed the code of the tool in our own code as statically linked libraries. This means that we only need the headers and the compiled binary to be able to summon ffmpeg. This way of using ffmpeg is highly efficient and much more elegant than just calling it from the command line, but it generates two problems that previously didn't exist

- FFMPEG's code is written in bare C and our project is written in C++.
- In order to properly use FFMPEG's code, we need to understand its architecture in a quite detailed way.

The first issue is not that complicated to fix. C++, being a language that derives directly from C, has a compatibility option that allows us to use C code without changing our architecture that much. We only have to include FFMPEG's code under the correct "extern" directive.

The second problem requires us to investigate FFMPEG's architecture in order to understand it and to use it.

FFMPEG has 4 major elements that we need to use in order to properly encode and decode a video stream: **AVPackets**, **AVFrames**, **Encoders**/**Decoders** and **codecs**. To provide a better understanding of the encoding/decoding process to the reader we'll proceed to explain each of those 4 elements.

First, the most basic one, the codec. A codec is nothing more than an implementation of a compression algorithm. It's the piece of software that is in charge of executing the compression algorithm and it has to be selected before the encoding process starts. There are a vast number of codecs, each one with its own pros and cons and with its use cases. The only two defining factors that we need to consider in order to select what codec we want to use is compression rate and data loss. Compression is usually a lossy process, which means that some information will inevitably be lost in the process of compression and decompression. This is especially true in video compression, seeing that the volume of data that we need to compress and the compression rate that we want to achieve are not possible without some data loss. In our testing project we'll be using the H.264 codec [19]. The h.264 is, undisputedly, the most widely used video codec in the world. Both because it's very efficient and because the quality of the video stream produced by it is superior to most of all its competitors by a big margin. One thing important to note is that this codec is not free, it's been licensed by a group of enterprises called MPEGLA, which contains all of the companies that claim to have some impact on the technological development of the tool. In

our case we can use the codec without paying any fee due to the fact that this is an open source project from which we don't plan to earn any monetary revenue.

The next piece to be explained will be the AVFrame. The AVFrame, in conjunction with the AVPacket, are both input and output in the encoding and decoding process of a video stream. In our case, the server will capture the screen (for more information, see section 4.3 - Desktop duplication) and it will extract the raw data from it. This data will be properly formatted and it will be inserted into an AVFrame, which is a data structure that only contains raw frame data and some other frame parameters. This AVFrame will be inputted into the video compressor, which on itself will output an AVPacket. This AVPacket is the core and basic unit that composes our video stream, and it also is the main information that we need to send through the network. Before that there's still one crucial step left. Seeing that an AVPacket is a data structure that cannot be sent through the network as it is, we have to serialize all of its attributes into a byte array. This process is going to be reversed in the client, so the order of the serialized attributes will be important later.

Finally, the encoder/decoder are both sides of the same coin. They're the objects responsible for applying the codec algorithm to the inputted AVFrame to create an AVPacket but also to reverse the codec in an AVPacket to output an AVFrame. Both the encoder and the decoder have to work in conjunction and synchronization, with the same configuration parameters, if we want to get acceptable results. In most real cases, this synchronization of parameters happens before any video packet is sent. In our case, seeing that we don't need to change said parameters dynamically, we can just hard-code them.
Once an AVFrame is encoded into an AVPacket, and the later one has been serialized, the resulting byte array is sent to the network layer.

Now, the code responsible for all the aforementioned functionality will be presented. Starting with the initialization of the FFMPEG encoder

```
VideoEncoder::VideoEncoder(int width, int height, NetworkLayer* nl)
{
    pNl = nl;
    pC = avcodec_find_encoder(AV_CODEC_ID_H264);
    if (!pC) {
        std::cout << "No codec found";
        exit(1);
    }

    pCC = avcodec_alloc_context3(pC);
    if (!pCC) {
        std::cout << "Could not allocate video codec context\n";
        exit(1);
    }

    pPck = av_packet_alloc();
    if (!pPck) {
        std::cout << "Alloc packet fail" << std::endl;
        exit(1);
    }

    if(FRAME_X == 960)pCC->bit_rate = 3000000;
    else pCC->bit_rate = 8000000;
    pCC->width = width;
    pCC->height = height;
    pCC->time_base = AVRational{ 1, FRAMERATE };
    pCC->framerate = AVRational{ FRAMERATE, 1 };
    pCC->gop_size = 60;
    pCC->max_b_frames = 3;
    pCC->pix_fmt = AV_PIX_FMT_YUV420P;
```

```
    if (pC->id == AV_CODEC_ID_H264) av_opt_set(pCC->priv_data, "preset", "ultrafast", 0);
    //av_opt_set(pCC->priv_data, "tune", "film", 0);
    av_opt_set(pCC->priv_data, "profile", "extended", 0);
    av_opt_set(pCC->priv_data, "crf", "18", 0);
    pts = 0;
    ctx = sws_getContext(FRAME_X, FRAME_Y, AV_PIX_FMT_BGRA, FRAME_X, FRAME_Y, AV_PIX_FMT_YUV420P, 0, 0, 0, 0);
}
```

Figure 4.4: Fragment of code responsible for the initialization of the video encoder. (Own elaboration)

And the encoding of a frame:

```cpp
void VideoEncoder::EncodeAndPush(const unsigned char* frame)
{
    //Frame construction
    int ret;
    AVFrame* F = av_frame_alloc();
    if (!frame) {
        std::cout << "Could not allocate video frame\n";
        exit(1);
    }
    F->format = pCC->pix_fmt;
    F->width = pCC->width;
    F->height = pCC->height;
    ret = av_frame_get_buffer(F, 0);
    if (ret < 0) {
        std::cout << "Could not allocate the video frame data\n";
        exit(1);
    }
    ret = av_frame_make_writable(F);
    if (ret < 0) {
        std::cout << "make frame writable fails" << std::endl;
        exit(1);
    }
```

```
uint8_t* inData[1] = { (uint8_t*)frame };
int stride[1] = { 4*FRAME_X };
sws_scale(ctx, inData, stride,0,FRAME_Y,F->data, F->linesize);
F->pts = pts;
pts++;
//frame encoding
avcodec_open2(pCC, pC, NULL);
if (ret < 0) {
    std::cout << "Could not open video codec \n";
    exit(1);
}
ret = avcodec_send_frame(pCC, F);
//*pPck = encode2(pCC, F);

if (ret < 0) {
    std::cout << "Error sending a frame for encoding\n";
    exit(1);
}
while (ret >= 0) {
    ret = avcodec_receive_packet(pCC, pPck);
    if (ret == AVERROR(EAGAIN) || ret == AVERROR_EOF)
        break;
    else if (ret < 0) {
        std::cout << "Error during encoding\n";
        exit(1);
    };
    break;
}
```

```
    //pts
    BYTE pts_b[sizeof(pPck->pts)];
    std::memcpy((void*)pts_b, &(pPck->pts), sizeof(pPck->pts));
    //dts
    BYTE dts_b[sizeof(pPck->dts)];
    std::memcpy((void*)dts_b, &(pPck->dts), sizeof(pPck->dts));
    //data
    BYTE* data_b = (BYTE*)malloc(pPck->size);
    std::memcpy(data_b, pPck->data, pPck->size);
    //data size
    BYTE size_b[sizeof(pPck->size)];
    std::memcpy(size_b, &pPck->size,sizeof(pPck->size));
    //streamindex
    BYTE streamindex_b[sizeof(pPck->stream_index)];
    std::memcpy(streamindex_b, &pPck->stream_index, sizeof(pPck->stream_index));
    //flags
    BYTE flags_b[sizeof(pPck->flags)];
    std::memcpy(flags_b, &pPck->flags, sizeof(pPck->flags));

    BYTE side_data_elems_b[sizeof(pPck->side_data_elems)];
    std::memcpy(side_data_elems_b, &pPck->side_data_elems, sizeof(pPck->side_data_elems));
    int total_size = sizeof(pts_b) + sizeof(dts_b) + pPck->size + sizeof(size_b) +
        sizeof(streamindex_b) + sizeof(flags_b) + sizeof(side_data_elems_b);
    BYTE* Packet_b = (BYTE*)malloc(total_size);
    memcpy(Packet_b, pts_b, sizeof(pts_b));
    memcpy(Packet_b + 4, dts_b, sizeof(dts_b));
    memcpy(Packet_b + 8, size_b, sizeof(size_b));
    memcpy(Packet_b + 12, streamindex_b, sizeof(streamindex_b));
    memcpy(Packet_b + 16, flags_b, sizeof(flags_b));
    memcpy(Packet_b + 20, side_data_elems_b, sizeof(pts_b));
    memcpy(Packet_b + 24, data_b, pPck->size);
    std::string s = std::to_string(pPck->pts);
    pNl->AddFrame((const char*)Packet_b, total_size);
    free((void*)data_b);
    av_frame_free(&F);
}
```

Figure 4.5: Fragment of the code responsible for the encoding of a single frame into the video stream. (Own elaboration)

Network layer

The network layer may be the simplest of all the three pieces that compose the server. Its only job is to constantly send the serialized AVPackets generated by the encoder, however this process is not straightforward. Seeing that the size of the AVPackets is not constant (some packets are key frames and others are interpolation frames, for more information consult [19]) we can't just start sending the stream and expect the client to be able to distinguish one packet from another. To fix this, we'll divide the sending process into two sections. First, we'll send the AVPacket size in bytes and then we'll send the AVPacket. This allows the client to first read an integer containing the size of the AVPacket and then with that information read the correct amount of bytes from the socket.

4.4 Client side

The client side of this project is, in my opinion, the most important and complex of the both. It's the part that applies the upsampling, which is the core premise of this bachelor's thesis. Before applying the upscaling algorithm in the rendering stage, we need to perform several other operations, starting with the receiving of the video streaming.

Network layer

The network layer component of the client is responsible of receiving the raw video stream. As stated in the previous section, due to the fact that the packet size of the video stream is not constant, we can't just read the video stream straightforwardly. First, we need to read the size of the incoming packet and after that we can read the packet.

One thing worth noting is that the only thing that the network layer does is read and store the raw byte array. This byte array needs to be deserialized, but it would be incorrect to do it in the network layer for two reasons:

- It would require the network layer to know about ffmpeg and its elements.
- It would break the software functionality division that we've been trying to maintain throughout the development of this project.

Because of this, the job of the network layer ends here, when this part of the project pushes the received packet as a byte array to the video decoder.

Video decoder

The video decoder is responsible for de-serializing the raw byte array into an AVPacket to then decode it into an AVFrame. This resulting frame will contain the raw pixel data of a frame of the video stream, which will be passed to the renderer.

FFMPEG

We won't explain again the structure and functionality of ffmpeg seeing that we've already explained it in section 4.3 - FFMPEG. The only thing worth mentioning regarding ffmpeg is that, as stated previously, encoders and decoders are both sides of the same coin. The process of initializing them is almost identical in both cases and the only differences are the input-output types (contrary to the encoder, the decoder inputs an AVPacket and outputs an AVFrame).

Once the frame is decoded, we have to pass it to the Renderer component somehow. Due to some reasons that we'll explain in the following sections, we can't just overwrite whatever data structure that holds the previous frame with the new one. We also can't perform any operation directly on said data structure because that would require the video decoder to know about DirectX. To solve this problem, we've implemented a small

interface in the renderer to allow the video decoder to communicate with it. This interface accepts as an input raw pixel data in the form of a byte array.

Renderer

The renderer component might be the most important and complex piece of not only the client but all the project. It's responsible for rendering each received frame to the screen after applying the up sampling algorithm. This process is very complicated and requires the creation of a large number of DirectX's resources.

DirectX

DirectX is a collection of API's designed to handle tasks related to multimedia purposes. It's usually used in video game use cases, acting as the low-level communications level between a game engine and the underlying OS. This interface is what allows us to render things to the screen. It's a very powerful and flexible tool that allows us to do practically anything. But one question that could be asked is: If we only need to render a 2D frame into the screen, couldn't we just use any other simpler tool? The key to answering this question lies in the upscaling step.

As we'll explain in the following sections, the FSRCNNX effect will be applied to each individual frame as a shader, more specifically an HLSL shader. This fact directly disqualifies every other option that doesn't allow us to execute an HLSL shader during the rendering process. OpenGL, another very big and powerful graphics API is also disqualified by this, seeing that it only accepts GLSL shaders. This last statement is not entirely a problem, seeing that HLSL and GLSL have almost the same capabilities, so we could translate one shader from one shading language to another. This process, however, would not only increase the required time to develop the testing platform, but it would also make a lot of pre-existing code unusable (See section 4.4 - Effect compilation for more information).

But what does said shader do? It's most certainly not a black box, and we can and will explain exactly what operations does it perform to an image in order to achieve a greater resolution.

Image upscaling

Traditional image resizing is a well known technique widely used across the world. It's been done using digital FIR filters such as box, triangle, Mitchell Netravali and Lanczos for nearest neighbor, bilinear interpolation, bicubic interpolation and windowed sinc scaling. Said filters are very good at interpolating existing information in order to fill missing information, but they cannot recreate it. When resizing an image using those techniques it loses a lot of high-frequency information. This loss of information is caused by the multiple mean operations performed during the process. For example a bilinear filter interpolates two existing values and creates a straight line between them in order to create new values and therefore upscale an image.

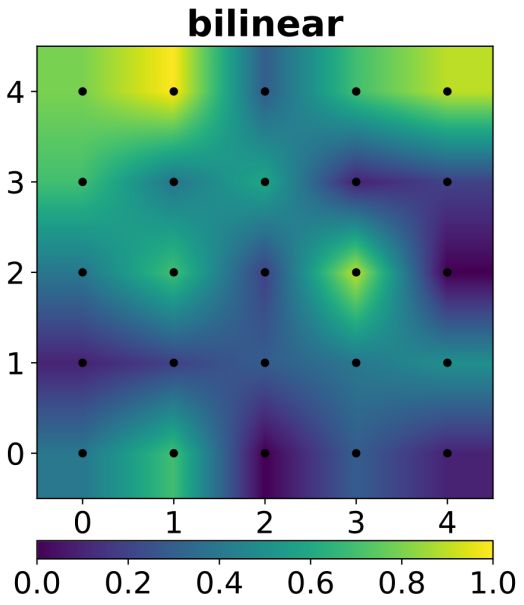

Figure 4.6: Results of a bilinear interpolation filter. (Source: https://en.wikipedia.org/wiki/Bicubic_interpolation)

.

A better result can be achieved by taking more than two values into account while doing the interpolation and creating a polynomial curve into the existing discrete points. This technique is what is called bicubic interpolation.

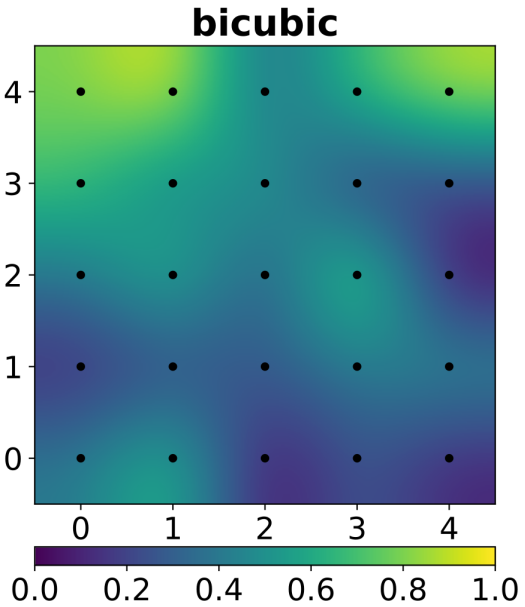

Figure 4.7 Results of a bicubic interpolation filter. Source: https://en.wikipedia.org/wiki/Bicubic_interpolation

There are other existing FIR filters that can be used. Don P.Mitchell and Arun N. Netravali authored a paper, Reconstruction filters in computer graphics [22] in which they proposed the Mitchell Netravali filter which achieves better results than a simple bilinear or bicubic interpolation. However, said results are still not acceptable. If we downsize a popular picture of the Swedish model Lena Forsén to half its size and then we upscale it using the

Mitchell Netravali filter, we can see that the resulting image is considerably blurrier than the original one.

Figure 4.8 Results of a Mitchell Netravali filter application Source: https://artoriuz.github.io/blog/super_resolution.html

The difference between them is, as explained previously, mostly contained in the high frequencies or edges that get lost when downscaling an image and cannot be recuperated. The upscaling process is creating new pixels but not new information, said existing information is being “diluted” in the new pixels. This loss of information is one of the key aspects that the machine learning upscaling techniques focus on improving from the classic counterparts.

The human eye has a lower acuity for differences in color than in luminance. This fact can be useful to determine which image data is more important and therefore which data should be treated by the upscaling algorithm. Chroma subsampling is a well known technique used in a vast number of video codecs (H.264 is one of them, for more information read section 4.4 FFMPEG) that consist of having fewer samples for chromatic information and more samples of luminance, usually in a 4:2 ratio. The YCbCr family of color spaces already divides pixel image data into one component of luminance and two components of chrominance, which makes chromatic subsampling easier with said colospace.
Having all previously mentioned in mind, we can define two statements about the machine learning algorithm that we’ll use:

- It will operate in YCbCr color space.
- It will only upscale luminance

The last statement is important if we want to reduce the computational demand of the execution of the algorithm.

FSRCNN

The FSRCNN algorithm is based on convolutional neural networks[23]. This is present on its very own name, which stands for Fast Super-Resolution Convolutional Neural Network. Convolutional neural networks are a class of artificial neural networks (ANN) that specialize in the analysis and operation of imagery. The FSRCNN algorithm was created as an attempt to make its predecessor, the SRCNN algorithm, faster in execution and training.

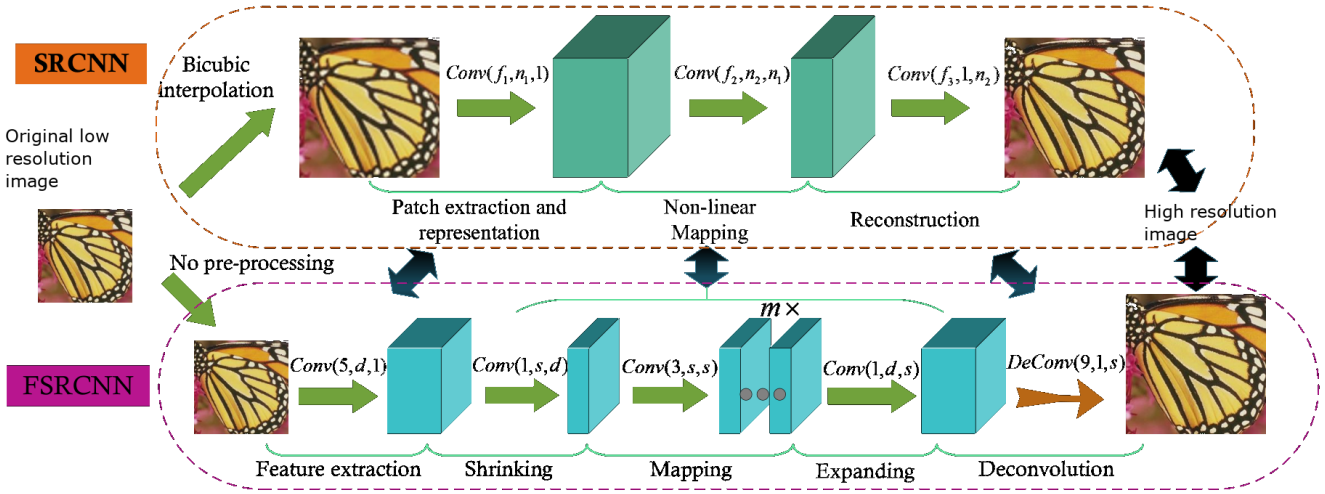

Figure 4.9 Differences between SRCNN and FSRCNN. Source: http://mmlab.ie.cuhk.edu.hk/projects/FSRCNN.html

The main difference between these algorithms is that FSRCNN receives the original image as input, instead of a processed version of the original with a bicubic interpolation applied. This, combined with the fact that the inner layer neural networks are smaller and shallower (the number of neurons is not reduced, there are more neural networks but they're smaller) makes the execution of the FSRCNN algorithm much more efficient in terms of computational power. When it comes to output fidelity, FSRCNN achieves a comparable or better image quality than its predecessor.

Original image

Upscaled with SRCNN to

Figure 4.10: Comparison between two upscaled images, one using FSRCNN and the other using SRCNN with the original image. Source: https://lankning.github.io/Super-Resolution/FSRCNN.html

As it can be seen in the previous images, the upscaled image is not 100% equal to the original image but it's noticeably better than the one upscaled with SRCNN.

And as stated previously, the resulting image is not perfect. This imperfection comes from two main sources: The limitations of the algorithm but also the encoding/decoding process, which in itself also reduces the quantity of information available to perform the upscaling.

Those conditions will need to be taken into account when assessing the quality of the resulting images to determine if the application of machine learning to upscale the video stream of a cloud gaming platform is really beneficial without sacrificing too much quality of service.

Initialization

Before we can start rendering all the frames that we receive from the decoder, DirectX and the FSRCNN effect require us to perform some extra steps, mainly initializing all the required data structures.

The **Window** is the main and lowest-level data structure of the client program. It's created by summoning a windows OS API and it contains and defines the boundaries in which we can render images. It also defines the maximum definition and number of pixels that we can render. To perform our tests and having in mind that the screen that we have

has a maximum resolution of 1920*1080, we defined the window to also have a 1920*1080 size.

The app window is initialized by the following code:

```
App::App(HINSTANCE hInstance, int width, int height)
{
    //Start of the window cration and registration
    HRESULT hr;
    pL = new Logger();
    this->hInstance = hInstance;
    WndWidth = width;
    WndHeight = height;
    wc = { 0 };
    const LPCTSTR pClassName = "hw3dbutts";
    wc.cbSize = sizeof(wc);
    wc.style = CS_OWNDC;
    wc.lpfnWndProc = HandleMsgSetup;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = nullptr;
    wc.hCursor = nullptr;
    wc.hbrBackground = nullptr;
    wc.lpszMenuName = nullptr;
    wc.lpszClassName = pClassName;
    wc.hIconSm = nullptr;
    RegisterClassEx(&wc);

    LPCTSTR a = "HWRLD";


    RECT wr;
    wr.left = 100;
    wr.right = width + wr.left;
    wr.top = 100;
    wr.bottom = height + wr.top;
    AdjustWindowRect(&wr, WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU, FALSE);
```

```
    hWnd = CreateWindow(pClassName, a, WS_CAPTION | WS_MINIMIZEBOX | WS_SYSMENU, CW_USEDEFAULT, CW_USEDEFAULT,
        wr.right - wr.left, wr.bottom - wr.top, nullptr, nullptr, hInstance, this);
    ShowWindow(hWnd, SW_SHOW);
    //end of the window creation and registration
    //graphics layer creation
    pRL = std::make_unique<RenderLayer>(hWnd);
    pRL->InitializePrimitiveGeometry();
    //network layer creation, we'll comment this for now.


    //creation of video decoder
    pD = new Decoder(FRAME_X, FRAME_Y, pRL.get());

    //creation and connection of network layer
    pNL = new NetworkLayerClient(pD, pL);
    pNL->Connect();

}
```

Figure 4.11: Fragment of the code responsible for the initialization of the window of the app. (Own elaboration)

The **DirectX Device** is the first native DirectX data structure that we need to initialize. It references the GPU of the system and it's used to create and initialize other necessary data structures. It also describes the capabilities and compatibility of the system's device, which helps us decide what features we can implement. The main use that we'll give it in our project is the creation of DirectX 2D textures, which can hold pixel information of a texture that needs to be rendered.

The **DirectX Context** is the main DirectX structure. It's used to perform most of the graphical operations that we need to perform during the initialization and rendering process. It's the object that helps us pass data to the CPU. It helps us set the corresponding shader. It can also be used to perform the draw operation, to present the back buffer, to set the viewport, etc. It's created from he DirectX device (the real name of the object is not DirectX Context but DirectX Device Context) and it only has to be initialized once, in the start of the program, before we render anything.

The code responsible for the initialization of the DirectX device and context is the following:

```
DXGI_SWAP_CHAIN_DESC sd = {};
sd.BufferDesc.Width = 0;
sd.BufferDesc.Height = 0;
sd.BufferDesc.Format = DXGI_FORMAT_B8G8R8A8_UNORM; //this may change based on the texture format
sd.BufferDesc.RefreshRate.Numerator = 0;
sd.BufferDesc.RefreshRate.Denominator = 0;
sd.BufferDesc.Scaling = DXGI_MODE_SCALING_UNSPECIFIED;
sd.BufferDesc.ScanlineOrdering = DXGI_MODE_SCANLINE_ORDER_UNSPECIFIED;
sd.SampleDesc.Count = 1;
sd.SampleDesc.Quality = 0;
sd.BufferUsage = DXGI_USAGE_RENDER_TARGET_OUTPUT;
sd.BufferCount = 1;
sd.OutputWindow = hWnd;
sd.Windowed = TRUE;
sd.SwapEffect = DXGI_SWAP_EFFECT_DISCARD;
sd.Flags = 0;

D3D11CreateDeviceAndSwapChain
(
    nullptr,
    D3D_DRIVER_TYPE_HARDWARE,
    nullptr,
    0,
    nullptr,
    0,
    D3D11_SDK_VERSION,
    &sd,
    &pSwap,
    &pDevice,
    nullptr,
    &pContext
);
```

Figure 4.12: Fragment of the code responsible for the initialization of the DirectX device and context. (Own elaboration)

The **Effect Drawer** is a custom object, which means that we implemented it. It's an object that is responsible for the application of the FSRCNN shader. But applying a shader in DirectX is pretty easy, it can be done in 10 lines of code. So why do we need a custom data structure to handle that? The answer is that the FSRCNN is not a common shader. The FSRCNN, as shown in the previous section (section 4.4 - FSRCNN) is an algorithm that performs multiple operations in the whole frame. Those operations require the input to be the output from the previous operation (except the first operation, which requires the original image as input). This requires each operation to be performed entirely before we can execute the next operation. To do this with a simple shader is impossible. A simple shader performs the same operation to the whole frame until it ends. So to be able to execute the FSRCNN algorithm we would require to have multiple shaders, each one with different operations, executed sequentially to the frame. This is both an inefficient and inelegant solution, however it's not the only thing that we can do. There are a kind of shaders, called multipass shaders, that can solve our problem much more efficiently but that require a more complicated setup.

Essentially, a multipass shader is a shader that has different functions. Each function corresponds to a different execution to the shader, and it operates to the whole frame. We can use this to perform the different operations required by the algorithm with a single-file

contained shader. The process of compilation and initialization of the shader is quite complicated, therefore, we'll explain it in more detail in the next section.

Effect compilation

The effect compilation is performed using three different data structures: The effect compiler, the effect drawer, the effect and the pass structure. We'll proceed to explain them starting with the one in the lowest level of functionality.

The **Pass** data structure is the smallest of the four. It contains the necessary attributes to perform only one of the several operations of the FSRCNN algorithm. Said attributes are mainly a virtual shader (a shader that's compiled from the file that contains all the shaders), and the list of inputs and outputs that need to be bound and accessed by that shader. Each pass is compiled from the same file, which needs to be parsed by the effect compiler.

The **Effect** data structure is the one that holds all the passes. The renderer object has no need to know the pass data structure. It would be tedious and complicated to call the execution of each pass from such a high level structure. To solve this, the Effect object was created to contain all the necessary pass objects, amongst other attributes needed to execute the whole effect.

The **Effect Drawer** is just an interface that helps the renderer call the execution of an effect. It contains all the functionality of the effect and its only purpose is to simplify the interactions with said piece of code.

The **Effect Compiler** is perhaps the most complicated piece of code of the project. Its function is to, from a single file, compile multiple shaders needed for the execution of the algorithm. Not only that but it also has to initialize the different resources (mainly textures) needed by each pass. Finally, it has to initialize the effect and fill it with all the resources generated. To perform this compilation we used a couple of external libraries, mainly the muparser [24] library, which is a parsing library that helps us interpret certain arbitrary notations in the shader file. These notations contain information about the necessary attributes, resources and inputs/outputs of each pass.

It's important to note that the architecture and part of the code of these 4 components is an adaptation from another existing project. This project has been mentioned several times in this document. Magpie [11] is an existing upscaling tool that upscales the screen execution of a desired game. This upscaling process is also performed using the same 4 components that we mentioned earlier. To ignore the help that the Magpie project has indirectly provided us would be not only disingenuous but it would also interfere with the morals and objectives of the open source community.

Input overwriting

To feed the decoded frames into the rendering process we can't just overwrite the data structure that contains the currently rendered frame. This operation is prohibited for several reasons, the first being that the decoder doesn't know if the current frame is still being rendered. If it is, by overwriting it we would interfere with the rendering process, therefore corrupting the render and possibly causing a crash.

The other reason is that the data structure (which is an ID3D11Texture2D, more information about that texture in [25]) isn't accessible by the CPU. This happens because DirectX optimizes the use of its resources to achieve better performance. If we want to render a texture, it has to be accessible by the GPU. But if we want to write to a texture it has to be accessible by the CPU. By rendering a texture that is accessible by both the CPU and the GPU we would be accessing a different memory region (shared memory, much slower than GPU memory) than by rendering a only-GPU texture. To solve this problem we created a staging resource; a texture that is accessible by both the GPU and the CPU but that it's never rendered. The decoded frame is copied into that texture by the CPU and then, when the render finishes rendering the previous frame, it copies the contents of the staging texture to the main rendering texture (this operation is performed by the GPU).

With this round abound, we can achieve both a better consistency with our renderer-decoder communication and a better performance in the rendering process.

4.5 General workflow of the functionality modules

We've given a lot of information about the individual components of our architecture, explaining their interaction with each other, but we believe that, for the sake of clarity, a general explanation of the workflow of the platform will help the reader better understand the whole process, from the screen capturing to the final rendering.

First, the screen of the server is captured while playing the desired testing game. The generated frame is then encoded into individual packets which are serialized into byte arrays, which are sent by the network layer to the client. The client's network layer reads the data stream and sends it to the decoder for the deserialization and decoding of the packets. The generated frame is fed to the renderer, which upscales it using the FSRCNN algorithm by performing multiple draws using different shaders. Finally, the upscaled frame is then rendered to the screen. This process is executed in a loop several times a second to display a video stream.

5 Performance evaluation

Now that we successfully created a solid and flexible testing platform, it's time to start performing the corresponding tests. Before that, and to be able to interpret the results correctly, we have to describe the hardware environment in which we're performing said tests.

5.1 Hardware environment description

As stated in one of the GEP sections, we'll perform the tests using two different machines. One will execute the server and one the client. For the server execution we'll use a laptop, an ASUS laptop to be precise. This laptop has an Intel Core i7-1165G7 @ 2.8GHz [27] and an NVIDIA GTX 1060 mobile as a graphics card. The machine also has 16 GB of DDR4 ram. Its screen of 13'' has a resolution of 1920*1080 pixels, ideal for our testing. The disk space/type is won't be relevant for the nature of the tests.

The client will be executed on a desktop computer. This machine has an Intel core i5-7600k @ 3.8 Ghz[28]. For the GPU it has an NVIDIA GeForce GTX 1060 with 3GB of graphics memory [29]. It has 16 GB of DDR4 RAM. A screen with a 60Hz refresh rate, 24'' and 1920*1080 pixels of resolution. Again, the disk space/type won't be relevant for the nature of the tests.

When talking about the network aspect of the hardware, we'll perform the test using two Tp-link network cards that both can get up to 300 Mbps of bandwidth [30]. These cards were selected because they supply us with more than enough bandwidth and because they're popular and cheap.

5.2 Environment configuration

To perform the tests, we'll run two different configurations.

Native configuration

First, we'll run the selected game in the server at native FullHD (1920*1080) and we'll stream a minute of gameplay to the client. The client, while receiving the video stream, will render on the screen the video **without** upscaling it. While doing so, the client will collect certain metrics about the video stream and the rendering process (metrics that we'll discuss in the following section) and will log them in a log file for future inspection.

Upscaled configuration

The second run will execute the game in the server at HD (960*540), capture the screen and send the video stream to the client. The client will receive the video stream, decode it, and render each frame while **performing** the upscaling. While doing so, the client will collect the necessary data and metrics about the video stream and the rendering, and log them in a file for later inspection and comparison with the metrics of the previous run.

All the metrics that we'll get will be the mean of one minute of video streaming. The tests will be performed 5 times to avoid outliers and instability in the results.

FFMPEG configuration

In a video stream, the active configuration of the video encoder used during the process of encoding can change the stream's attributes drastically. The image quality, the bit rate, the frame rate… Almost every aspect of the video stream is entirely dependent on the configurable parameters of the encoder. If we want to perform an accurate analysis of the possible benefits of using FSRCNN on our platform it's critically important to guarantee that said parameters are configured according to our needs. Ffmpeg, in conjunction with the h.264 codec has a vast array of parameters that we can configure in order to achieve the best video quality.

The first, and maybe the easiest, parameter of the list is the frame rate. It's very important that, if we want to accurately compare the bit rate of both video streams, they're encoded using the same frame rate. To have better stability in our measurements we decided to set an arbitrary bandwidth of 30 frames per second on both configurations.

Secondly, we have to configure the video resolution of both streams. For the native configuration, seeing that we capture the whole screen and send it without altering the frame, we have to configure a 1920*1080 resolution, also known as FullHD. For the Upscaled configuration, we have to capture only ¼ of the screen due to the fact that the FSRCNN algorithm performs its upscaling in a factor of two. ¼ of the original FullHD resolution can be achieved by halving both the X and Y axis of the screen, ending with a 960*540 resolution, also known as HD.

The next parameter, the GOP size, requires a bit of explanation about how the video encoding process is generally performed [38] .

The necessary bandwidth to send a raw FullHD 30 fps video, considering that each pixel is represented using 4 bytes, is 248,83 MBps. However, this bandwidth is calculated assuming that each frame is sent in whole. In a traditional video, each frame is typically very similar to those that precede and follow it. Maybe the mouth of a character moves, or something in the background changes its position, but the rest of the frame stays the same. Video encoders take advantage of this by only sending some frames in full (they're called keyframes) and some frames only containing the information that changed from the previous frame (they're called p-frames).

So, technically, we could first thing first when the string starts, send a key frame and then keep sending just the difference between the original frame and the current one. This, even though it sound good on paper, it's not advisable for several reasons, the main ones being:

1. The visualization of a video stream is not sequential. The user may want to reproduce a part of the video that has already been sent.
2. Delivery media is imperfect. Packets get lost, bits flip, noise gets introduced.
3. The theory behind this key frame difference division breaks when a change in scenery happens, and most of the pixels of the screen change.

That's why we can't just send one key frame, instead, we send one periodically. This is what the GOP size means, GOP standing for Group Of Pictures. GOP size is the distance in frames between two key frames.

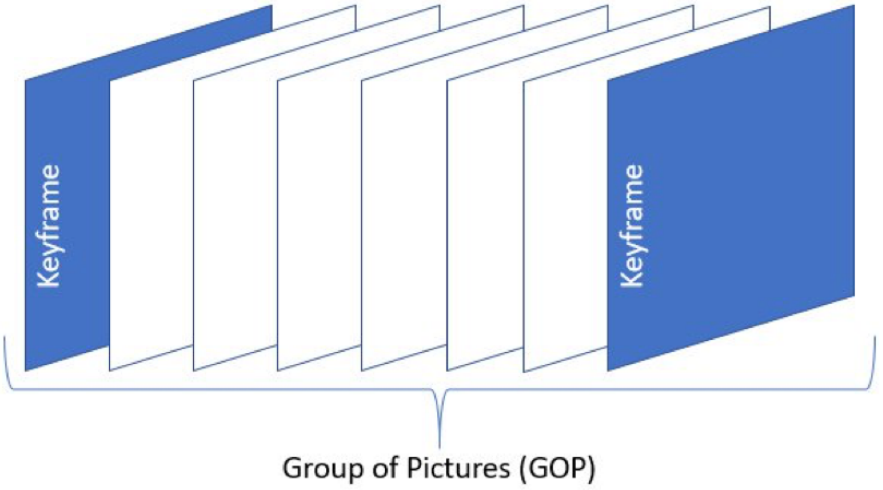

Fig 5.1: Visual representation of a GOP. (Source: [38])

Usually, real-world applications use 0.5-2 seconds of GOP size, which in a 30 fps video is 15-90 frames. In our case, we've decided to use 2 seconds, or 60 frames, as our GOP size, this number being based on values used in real use-cases.

The next parameter is also related to the previous topic: max_b_frames. We know about key frames and p-frames, but what are b-frames? B-frames, in contrast to p-frames, contain references to not only a previous key frame but also a future one. This is possible because frames are not sent sequentially, instead key frames are sent first. This way, the decoder can operate with b-frames without lacking information. These are combined with p-frames and key frames to perform a very space-efficient compression. In ffmpeg, max_b_frames describes the maximum number of b-frames allowed in between non-b-frames. We've decided to set that attribute to 2 based on the fact that it's one of the most common values used in real-world applications.

For the pixel format, we've decided to use YUV 4:2:0. As we explained in section 4.4 - FSRCNN, the human eye is more sensitive to luminance information than to chrominance information. The h.264 takes advantage of that by performing a more lossy compression to the chrominance planes and a more loss-less one to the chrominance, because the spectator won't be able to see said loss. The YUV pixel format already contains the 3 necessary Y Cb and Cr planes required by the h.264 codec and that's why we decided to use it.

All the previously explained parameters were required by ffmpeg to work. From now on, the explained parameters will be related only to the h.264 codec.
The first and most important parameter to configure on the h.264 codec is the CRF attribute, also known as Constant Rate Factor. CRF is the default quality and rate control setting for the h.264 codec. It's increasing and exponential, meaning that the value 0 offers the worst compression (loss-less) but the best image quality and 51 offers the best compression in exchange for the worst image quality. We decided to go for a CRF of 18. The default value, 23, introduced too many image imperfections, so we decreased the CRF until we found a good balance between data compression and image quality.

Finally, the last configured parameter is the profile [40]. The H.264 standard defines a number of profiles. Each profile uses a subset of the coding tools defined by the H.264 standard. Said tools are algorithms or processes used for video coding and decoding. There are 7 [40] different profiles with different capabilities, the baseline profile being the most basic one. For the sake of simplicity, we're not going to explain what all of the profiles do. Instead we'll focus on the extended profile, the one that we used.

The extended profile is a superset of the baseline profile. The extended profile extends the baseline profile with several error resilience techniques. It uses B slices and supports interlaced video coding. This profile is targeted at streaming video. It's characterized by higher compression but also higher complexity. Unlike the other profiles, this one supports special slices designed for streaming: the SI and SP slices. These allow the server to switch between different bit-rate streams when needed. All of this features made us decide to use this profile, seeing that the next on in the complexity scale, the high profile, sacrificed too much image quality in exchange for more compression.

5.3 Metrics to look for

Deciding what metrics we need to evaluate the possible change in performance is key to have a good resulting analysis of the tests. The original objective of this project was to see if the machine learning upscaling technologies were able to improve the recurring problem of high latencies in cloud gaming solutions. This, however, is not the only possible improvement that we can get from using said techniques, and it's important to take them into account.

For the first and most important measurement, the latency, we'll need to collect data about the bandwidth differences between the HD and FullHD video streams. It might be a bit counterintuitive to talk about measuring the bandwidth when we want to improve the latency, but in reality it makes sense. Imagine we have to send 300Mb of data from point A to point B. And point B has a network card that can only send 150Mbps. The time that B will need to send said data will be 2 seconds + the added latency of the general network. Applying machine learning upscaling we can reduce the size of the data, and therefore reduce the 2 second part of the total time that we'll need to send 300Mbs of data and, therefore, reduce latency.

Another conclusion that we can extract from the previous statement is that while real use cases happen in the general open network (the cloud gaming server is located outside the local area network of the client) we can perform our tests having both the client and the server in a local area network. This is possible because, as previously stated, the improvement in the latency comes directly from the reduction of the needed bandwidth. While working in a local network, we can eliminate external sources of latency coming from the outside network present in a general use case so we can better isolate the improvement caused by the upscaling. This will help us get cleaner and more reliable and stable results of the network metrics.

Another aspect about the possible improvements that machine learning upscaling can bring us is performance improvement. As mentioned in a previous section, rendering a video game is very resource consuming. However, by rendering the game in a lower resolution we can reduce the resources used by the rendering process. This improvement comes from the reduction of the number of pixels that the GPU needs to calculate. Less resolution equals less pixels, which in itself equals to less number of graphical calculations. To measure the difference in computational power consumption between the two executions we're required to use an external tool, seeing that it's impossible for us to access GPU metrics from our code regarding other processes. To perform this analysis, we've decided to use MSI afterburner [31]. MSI afterburner is one of the best GPU tuning utilities on the market. It's easy to use, fast, free and allows us to access the game performance data that we require to do a good analysis of the performance improvement.

Finally, one aspect that we can't ignore is the image quality. This metric, however, is entirely subjective to the observer. We've seen in previous sections that the quality between native FullHD and upscaled FullHD (from HD) has noticeable differences, however, it's the job of the observer to determine if said differences in quality are game breaking or if they are ignorable. Even though it is a subjective conclusion, we'll try to perform a good analysis whatsoever.

5.4 Additions to the architecture

When designing the architecture for the testing platform we omitted the logging part for two reasons: To make the first design simpler and because a log layer can be added after the main project development is completed without much difficulty.

We'll try to keep the changes simple, seeing that a big addition at this stage of the project can be very dangerous for the stability of the software. The new architecture will look like this:

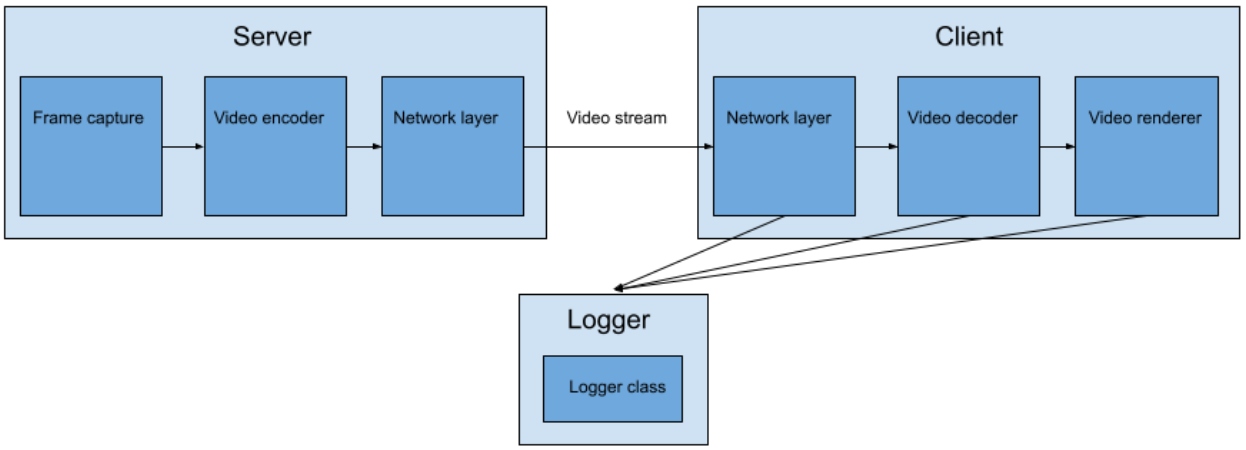

Figure 5.2: General architecture of the platform(modified) [Own creation]

The logger will only be accessed by the server, seeing that for the bandwidth we only need one of the two pieces of the project to have measurements, and for the fps we can only measure them in the client.

The logger class.

The logger class is the only code addition that we need to add to the project. It contains just the necessary attributes to calculate the metrics. A very basic file writer to store them in a file for the later inspection, and a simple API to allow the other three components to communicate and log information to it.

Logger API

The logger API only has two functions: SubmitBandwidth and LogFrame.

SubmitBandwidth is the function responsible for logging the bandwidth of the application. Every time the NetworkLayer receives a packet, it calls this function passing as a parameter the size of the received packet. Once every second, the Logger stores all the bandwidth received in a vector of bandwidths and resets the bandwidth counter.

Something similar happens with the LogFrame function. Every time the renderer finishes rendering a frame, it calls the function to indicate to the logger that one frame has been rendered. The logger, once every second, stores the number of frames logged into a frame rate vector and resets the frame counter. Finally, once a minute, the logger is called from the main App loop to calculate the needed metrics. This essentially computes the mean of all the values of both the bandwidth vector and the fps vector and writes it in a file called log.txt. The numbers stored in that file contain the mean frame rate and bandwidth of one minute of execution.

5.5 Game selection for the testing

The selection of a proper game to perform the tests on is actually not an easy one. We need to select a game that allows us to change the rendering resolution at our will in order to perform the tests correctly. We also need to be able to execute it in our server machine and, seeing that it's not a high-end machine, this factor can potentially eliminate a lot of possible candidates. Another positive factor could be the price of the game, so selecting a free to play game would be preferable over a paid game.

After considering all the requirements, 4 candidates were selected amongst a large pool of possible games. Said candidates were DoomEternal [32], Terraria [33], Minecraft[34] and CS:GO Global offensive [35]. Next, we'll evaluate the pros and cons of each game to help us select the best candidate for the testing.

Doom Eternal

Pros:

- Game relatively new.
- High-end graphics.
- Good performance in low-end hardware.

Cons:

- Unable to be rendered at desired resolution without modifying the game's code.
- High price of acquisition (60€).

Terraria

Pros:

- Extremely low hardware requirements.
- High flexibility in terms of resolution.
- Cheap price of acquisition (5€).

Cons:

- Very low-end graphics.
- Was released some years ago.

Minecraft

Pros:

- Very popular game.
- Relatively cheap price.
- High flexibility of resolution.

Cons:

- Low-end graphics.
- Bad hardware performance.

CS:GO Global offensive

Pros:

- Free to play.
- Can be rendered at the desired resolution.
- Very good hardware resolution.

Cons:

- Outdated graphics.

After some deliberation and consideration of the pros and cons of all the previously mentioned candidates, we've decided that the game that we'll use to perform the tests will be **GS:GO Global offensive**. Even though it's not a game with high-end graphics it vastly surpasses the graphical aspect of both Terraria and Minecraft. Graphically, it's way behind Doom Eternal, but it still delivers a good enough image quality.

One of the deciding factors was that we can execute the game at 960*540 natively without requiring any modification of the source code of the game. It's also free in the Steam [36] marketplace which is also a positive factor to consider.

Even though our first thought were that Doom Eternal would be the selected game, the fact that we cannot render said game at 960*540 natively directly disqualified it from being selected.

In the case of Minecraft and Terraria, both are very flexible in terms of rendering resolution but the fact that the graphical esthetic of both games heavily leans towards pixelated graphics could possibly make the evaluation of the quality of the resulting image too difficult.

5.6 Performing the tests

As stated previously, to perform the test we'll play and stream one minute of gameplay several times for each of the two configurations. Those play sessions will be performed playing the same part of the game in both configurations for a better consistency. This process will be performed 5 times for each configuration and the results will be combined in a mean for each metric.

5.7 Results

After performing the desired test we can present the obtained results:

5.7.1 Bitrate

	Esec 1	Exec 2	Exec 3	Exec 4	Exec 5	Mean
Upscaled 1080	329.908	339.595	345.589	335.295	319.296	333.936,6
Native 1080	887.401	804.457	935.568	844.058	865.853	864.467.4

Table 5.1: Table with the adjusted results of the bitrate (in bytes per second) to a standard frame rate of 30 frames per second. It also shows the mean of the results of the 5 executions (Own elaboration).

It's worth noting that the differences in the total bandwidth of each execution of the same configuration come from the difference in the gameplay, seeing that each one of the 5 executions played a different part of the game. On the contrary, both executions of the same index of execution played the same part of the game in both configurations.

5.7.2 Game performance

GPU load during the performance of the upscaled 1080 test.

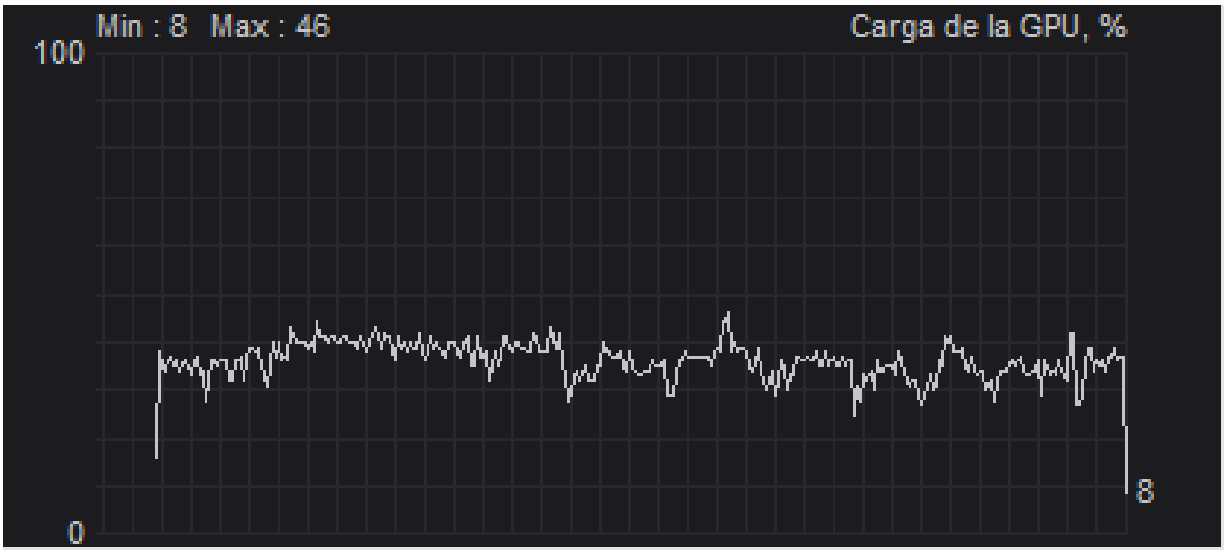

Figure 5.3: GPU load during the execution of the testing game at 960*540 pixels of resolution (Own elaboration).

In the figure 5.3, previously presented, we can observe the GPU load percentage during the execution of the testing game with the HD configuration. It's worth noting that during the whole execution the load is pretty stable, and it's always under 50%.

GPU load during the performance of the native 1080 test.

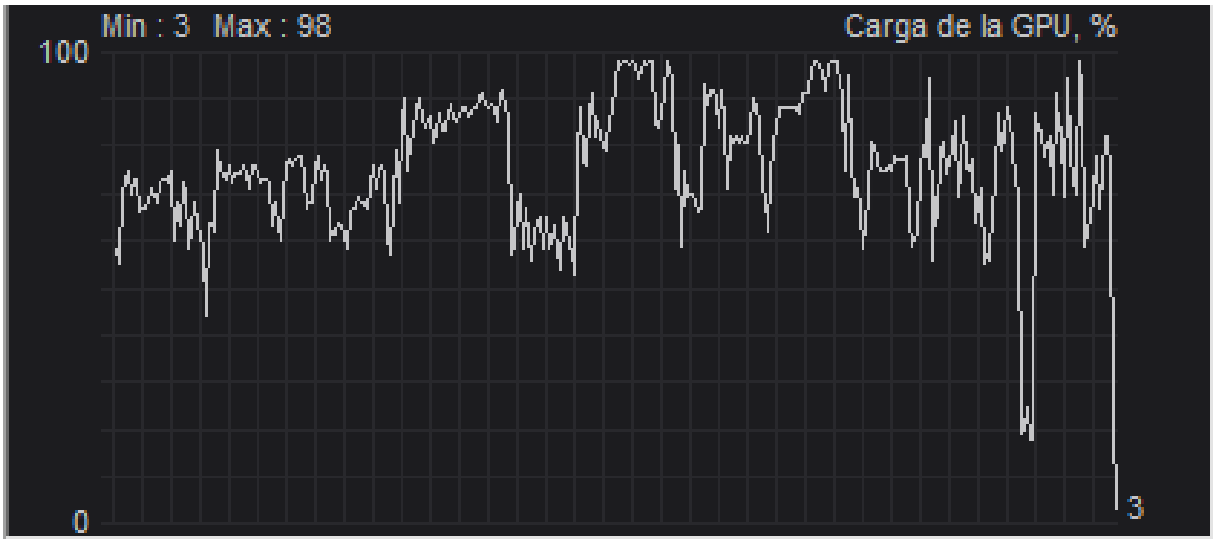

Figure 5.4: GPU load during the execution of the testing game at 1920*1080 pixels of resolution (Own elaboration).

In the figure 5.4, previously presented, we can observe the GPU load percentage during the execution of the testing game with the FullHD configuration. This execution in comparison with the previous one is much more unstable and inefficient, reaching peaks of 98% GPU load.

It has to be noted that both tests were performed by playing the same section of the testing game, so we can really see independently of the game requirements for a specific scene if there's a noticeable difference between both configurations.

5.7.3 Image quality

Original frame, not streamed:

Figure 5.5: Screenshot of the testing game in the server machine being executed at 1920*1080 pixels of resolution (Own elaboration).

In the figure 5.5 we can see the original rendered frame of the testing game in the client, before being streamed.

Native 1080 frame, streamed:

Figure 5.6: Screenshot of the game being rendered at the client machine while being streamed at 1920*1080, without upscaling (Own elaboration).

In the figure 5.6 we can observe a frame of the FullHD configuration, being rendered after being streamed to the client.

Upscaled 1080 frame, streamed:

Figure 5.7: Screenshot of the game being rendered at the client machine while being streamed at 960*540, upscaled at 1920*1080 (Own elaboration).

In the figure 5.7 we can see a frame of the HD configuration, being rendered before being streamed to the client and upscaled.

6 Result interpretation

Now that we have all the results from the performed tests we can start interpreting them.

Bitrate interpretation

The difference between the bit rates of both video streams is very noticeable. We speculated from the start of the project that we would see something similar to the results we got but we couldn't really predict exactly what would be the improvement. By comparing both average values we can approximate the difference to be an improvement of 61,37%. Or in other words, in the upscaled configuration the bit rate is reduced by 61.37% from the native configuration. If we calculate the difference in frame size between both streams, we see that the frame of the upscaled configuration is ¼ the size of the frame of the native configuration. This could make us think that the difference between one bit rate and the other should be 75% instead of the result that we got. This, however, is not true due to the fact that bit rate doesn’t have a linear correlation with frame size, thanks to the H.264 codec. This massive decrease of the necessary bit rate to transmit the video stream could potentially reduce by the same factor the possible latency of the operation (seeing that bit rate and latency have a linear correlation).

GPU load interpretation

Another speculated difference between both configurations was the server’s GPU load related to the execution of the game. By reducing the resolution at which the game is executed, we’re reducing the number of pixels our GPU needs to calculate to render a full frame of the game. In the figures 5.3 and 5.4 we can observe a noticeable difference between the GPU loan in both executions. In the execution of the game at FullHD we have peaks of 95-97% of GPU load. Even though the graphics show a very irregular chain of measurements, we can see that in most cases the GPU load rounds in the 70-90%. Said irregularities in the different measurements come from the fact that different sections of the testing game have different requirements in terms of GPU power. This, however, is important to have in mind while comparing both executions, because if one execution was made on sections of the game that are more demanding than the other, the results won’t be valid. We had that in mind while performing the tests, so both the HD and the FullHD executions were measured in the same game area and under the same conditions.

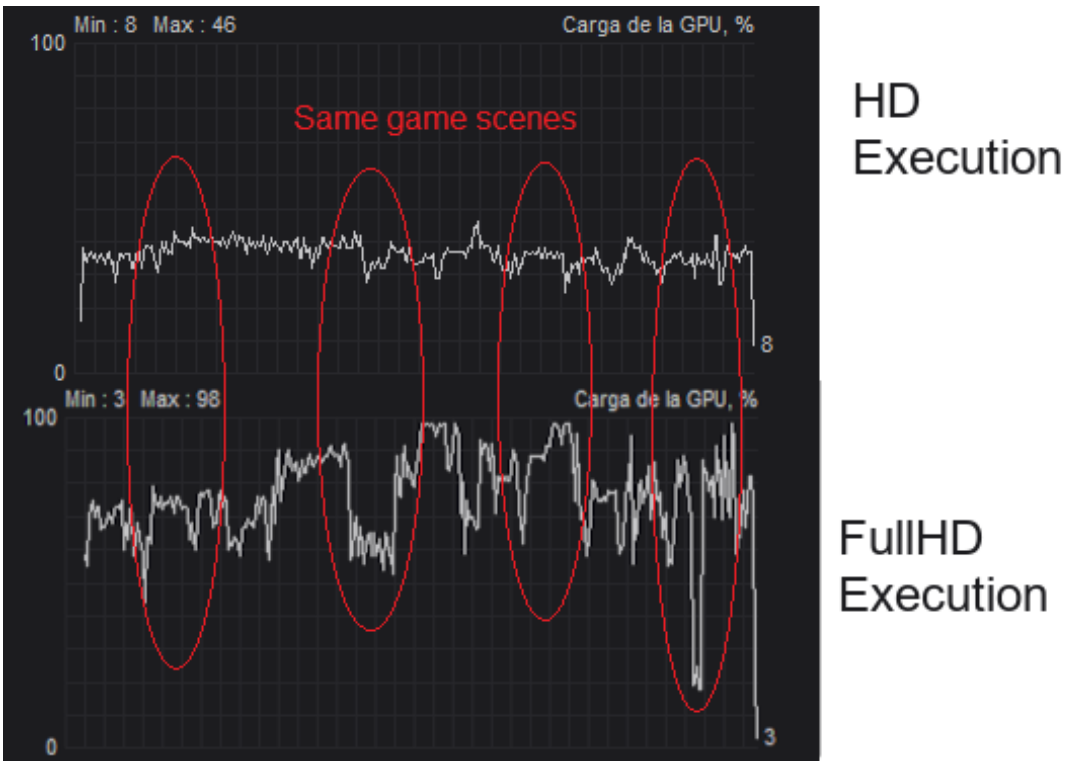

Figure 6.1: Parallelism in scenes between executions. (Own elaboration)

In the figure 6.1 we can see the parallelisms between both executions.

For the HD execution, we can see in the figure 5.4 that it's not only much more stable but it's also far less demanding in terms of GPU power. This execution stabilizes at a range of 30% to 45% of GPU load, which when compared to the 70% to 90% values of the other execution shows us an improvement of more or less 50% in the efficiency of the game execution.

Image quality interpretation

As stated before, this may be the section of the analysis that's more subjective to the observer than the others. Even so, we'll try to perform an analysis as objective as possible.
At first glance, the image quality of the upscaled configuration looks pretty promising. The general gameplay elements are recognizable and there's not much blur present in the scene. What is noticeable is a swift change in the color palette of the frame. The colors look darker and less saturated, which is logical considering the fact that the h.264 encoder sub samples the chrominance components of the frame.

The most obvious defects are notorious dents that appear in some straight diagonal lines. Some wall edges, signs and objects present a hard denting making them appear a bit to sharp but still recognizable.

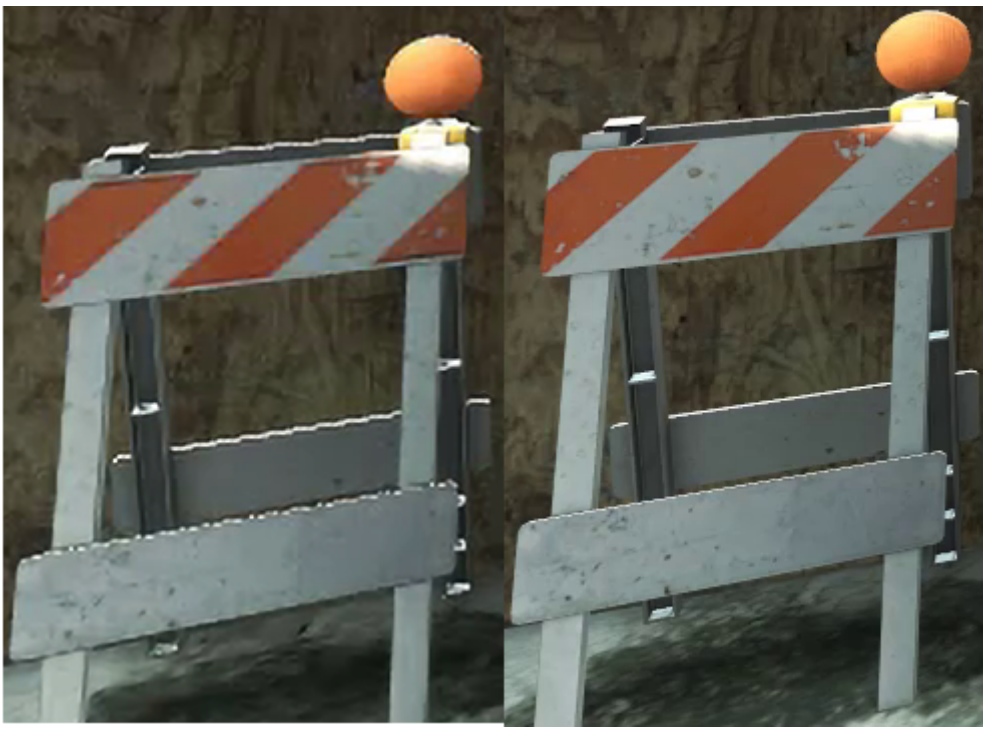

Figure 6.2: Comparison between both configurations of the same fence. Upscaled (left) and native (right).(Own elaboration)

The worst defect that appears in the upscaled image is the loss of small details, more concretely letters and drawings. This has to be taken into account because it could affect the user's experience negatively. For example in this case the mini map, present in the upper-left part of the screen, is heavily distorted and dented. This, interfering with a core element of the gameplay, cannot be ignored. Some games won't have this problem but games like CS:GO that depend on a precise UI will. This problem can be seen in the following figure.

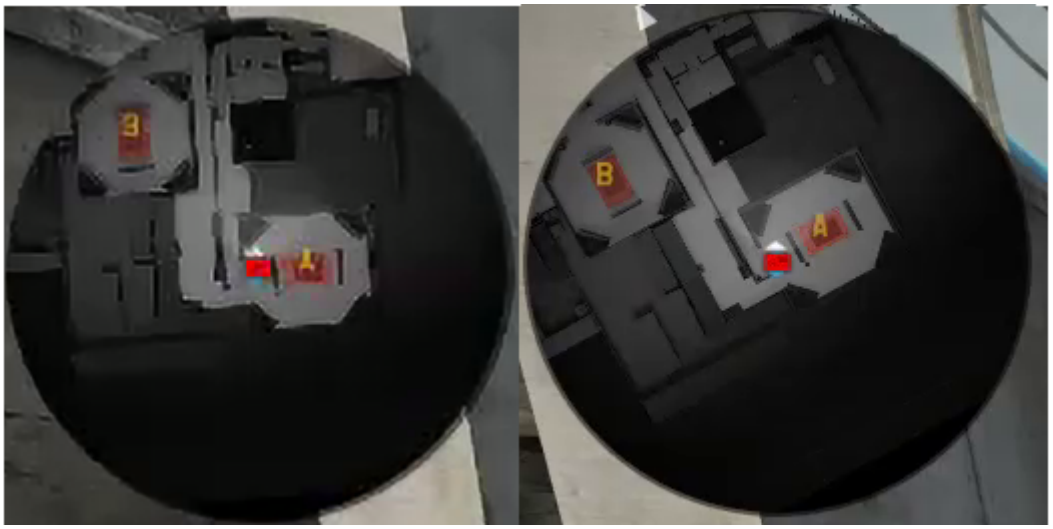

Figure 6.3: Comparison between both configurations of the minimap. Upscaled (left) and native (right).(Own elaboration)

Finally, there are quite a few interferences that appear all around the screen, reducing the overall image quality. It would be concerning if not for the fact that they also appear in the native FullHD version of the image. These imperfections come not from the upscaling process but from the streaming process. As stated previously, the video streaming process implements a lossy compression that introduces image imperfections. In our testing platform, these imperfections could be more noticeable than in other commercial platforms due to the fact that our platform was coded in 3 months and that we have considerably less knowledge about video streaming than other professional cloud gaming solutions. It's also worth noting that these imperfections appear only when there's a lot of change in the screen.

Figure 6.4: Comparison between both configurations of the compression induced imperfections. Upscaled (left) and native (right).(Own elaboration)

Overall, the image quality of the upscaled configuration is quite high, considering that it was generated with 75% less information than the native FullHD one. There are some imperfections and some elements of the frame are distorted. This, depending on the game that's being played, could heavily affect the gameplay experience, so it has to be taken into account when deciding if using machine learning upscaling techniques is worth it.

7 Conclusion

To conclude this bachelor's thesis, yes, the use of a machine learning algorithm upscaling algorithm could potentially improve the performance of a cloud gaming service, notoriously reducing its latency. It is also true that this technique could potentially make the hosting of said services much more computationally efficient and subsequently cheaper, however, to ignore the noticeable reduction in image quality would be reckless.

The reality is that we're reducing in 75% the quantity of information that is sent to the client and it's impossible to achieve the same results as what we could achieve with 100% of it. And even though the resulting image quality is pretty high, again, considering the fact that it's been generated with 75% less information, the differences between native FullHD and upscaled FullHD are noticeable. Certain elements of the gameplay could be distorted to the point of being unrecognizable, for example small letters and details. To apply this technique, the cloud gaming service providers should be cautious. Always making sure that the user experience is not damaged whatsoever. One of the best approaches we think would work best would be to offer this as an option, rather than making it the default standard of the industry. In the end, each user is different and each game has different requirements.
Some games tolerate high latencies better than others. Some games tolerate worse graphical details than others.

User availability is, in our opinion, what should always be the priority in those kinds of services and, by offering the option of using said techniques, a platform can become available to a user base that previously couldn't, for any reason, access said services.

Bibliography

[1] [Online; Accessed Feb 20, 2022] (url:)
https://en.wikipedia.org/wiki/Early_history_of_video_games

[2] [Online; Accessed Feb 20, 2022] (url:)
https://en.wikipedia.org/wiki/William_Higinbotham

[3] Francis Locknear “Average cost of a gaming PC (2022)” (A:) thecostguys.com
[Online; Accessed Feb 20, 2022] (url:)
https://thecostguys.com/gaming/average-cost-of-a-gaming-pc#:~:text=A%20typical%20gaming%20PC%20will,pay%20as%20much%20as%20%242%2C000.

[4] Davis Silver. “Mobile gaming trends 2022” (A:) is.com Jan 25, 2022
[Online; Accessed Feb 21, 2022] (url:)
https://www.is.com/community/blog/mobile-gaming-trends/

[5] Tamara Dutina. “Expert opinion on cloud gaming Industry: challenges and solutions” (A:) superadmins.com jan 27, 2021 [Online; Accessed Feb 20, 2022] (url:)
https://superadmins.com/expert-opinion-cloud-gaming-industry-challenges-solutions/

[6] Karl Bode. “The failure of Google stadia”
(A:) techdirt.com feb 18, 2022 [Online; Accessed 21 Feb 21, 2022] (url:)
https://www.techdirt.com/2022/02/08/google-stadias-failure-is-almost-complete/

[7] “6 reasons why cloud gaming can fail”
(A:) purecloudgaming.com Oct 1, 2020 [Online; Accessed Feb 21, 2022] (url:)
https://purecloudgaming.com/6-reasons-why-cloud-gaming-can-fail/

[8] Scharon Harding. “What is DLSS? NVIDIA’s AI-powered graphics technology” (A:) tomshardware.com June 23, 2021 [Online; Accessed Feb 21m 2022] (url:)
https://www.tomshardware.com/reference/what-is-nvidia-dlss

[9] Ángel Aller. “¿Qué son los tensor cores de una GPU NVIDIA? Descúbrelo” (A:) profesionalreviews.com Dec 19, 2020 [Online; Accessed Feb 22, 2022] (url:)
https://www.profesionalreview.com/2020/12/19/tensor-cores/

[10] Sik-Ho, Tsang “Review: FSRCNN (Super Resolution)” (A:) towardsdatascience.com Oct 27, 2018 [Online; Accessed Feb 21, 2022] (url:)
https://towardsdatascience.com/review-fsrcnn-super-resolution-80ca2ee14da4

[11] Binlue. [Online; Accessed Jan 20, 2022] (url:)
https://github.com/Blinue/Magpie/releases

[12] [Online; Accessed March 8, 2022] (url:)
https://www.glassdoor.es/member/home/index.htm

[13] [Online; Accessed March 8, 2022] (url:)
https://tarifaluzhora.es/info/precio-kwh

[14] [Online; Accessed March 8, 2022] (url:)
https://www.enalquiler.com/precios/precio-alquiler-vivienda-tortosa_31-45-49291-0.html

[15] [Online; Accessed March 12. 2022](url:)
https://docs.microsoft.com/en-us/windows/win32/direct3ddxgi/desktop-dup-api

[16]Federico Caruso, Ornaldo Gjergji ”Europe’s internet speeds are faster than ever, but not for everyone” (A:) balcanicaucaso.org june 22, 2021 [Online; Accessed june 09, 2022] (url:)
https://www.balcanicaucaso.org/eng/Areas/Europe/Europe-s-internet-speeds-are-faster-than-ever-but-not-for-everyone-212940#:~:text=Internet%20performance%20in%20Europe%20has,103.3%20Mbps%20in%20June%202021.

[17][Online; Accessed June 12. 2022](url:)
https://en.wikipedia.org/wiki/FFmpeg

[18] [Online; Accessed June 12. 2022](url:)
https://en.wikipedia.org/wiki/Fabrice_Bellard

[19] Jan Ozer “What is H.264” (A:)www.streamingmedia.com [Online; Accessed june 19, 2022] (url:)
https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=74735#:~:text=The%20H.&text=264%20Spec-,H.,Video%20Coding%2C%20or%20AVC).

[20]joão Vitor Rafael Chrisóstomo “Machine learning super-resolution” [Online; Accessed june 20, 2022](url:)
https://artoriuz.github.io/blog/super_resolution.html

[22]Don P.Mitchell “Reconstruction filter in computer graphics” Aug. 1998
(A:https://dl.acm.org/)
[Online; accessed june 18, 2022] (url:)
https://dl.acm.org/doi/10.1145/378456.378514

[23][Online; Accessed June 12, 2022](url:)
https://en.wikipedia.org/wiki/Convolutional_neural_network

[24][Online; Accessed May 20, 2022](url:)
https://beltoforion.de/en/muparser/

[25][Online; Accessed April 15, 2022](url:)
https://docs.microsoft.com/en-us/windows/win32/api/d3d11/nn-d3d11-id3d11texture2d

[27][Online; Accessed june 12, 2022](url:)
https://ark.intel.com/content/www/us/en/ark/products/208921/intel-core-i71165g7-processor-12m-cache-up-to-4-70-ghz-with-ipu.html

[28][Online; Accessed june 12, 2022](url:)
https://www.intel.com/content/www/us/en/products/sku/97144/intel-core-i57600k-processor-6m-cache-up-to-4-20-ghz/specifications.html

[29][Online; Accessed june 12, 2022](url:)
https://www.nvidia.com/es-la/geforce/products/10series/geforce-gtx-1060/

[30][Online; Accessed june 12, 2022](url:)
https://www.tp-link.com/es/home-networking/adapter/tl-wn821n/

[31][Online; Accessed june 15, 2022](url:)
https://www.msi.com/Landing/afterburner/graphics-cards

[32][Online; Accessed june 15, 2022](url:)
https://bethesda.net/es/game/doom

[33][Online; Accessed june 15 2022](url:)
https://terraria.org/

[34][Online; Accessed june 15, 2022](url:)
https://www.minecraft.net/es-es

[35][Online; Accessed june 15, 2022](url:)
https://blog.counter-strike.net/

[36][Online; Accessed june 15, 2022](url:)
https://store.steampowered.com/

[37][Online; Accessed june 16, 2022](url:)
https://support.google.com/youtube/answer/1722171?hl=en#zippy=%2Ccontainer-mp%2Caudio-codec-aac-lc%2Cvideo-codec-h%2Cframe-rate%2Cbitrate

[38] Brian Samis “Back to basics: GOP explained” 28 May 2020
(A:) aws.amazon.com
[Online; Accessed June 16, 2022] (url:)
https://aws.amazon.com/es/blogs/media/part-1-back-to-basics-gops-explained/#:~:text=Simply%20put%2C%20a%20GOP%20is,30%20frames%2C%20or%201%20second.

[39][Online; Accessed June 16, 2022] (url:)
https://trac.ffmpeg.org/wiki/Encode/H.264

[40][Online; Accessed June 16, 2022] (url:)

https://www.vocal.com/video/profiles-and-levels-in-h-264-avc/