

CREACIÓ D'UNA APLICACIÓ
DE LIVE VIDEO MIXING AL NÚVOL,
UTILITZANT TECNOLOGIA DE VIDEOJOCOS

laSalle

UNIVERSITAT RAMON LLULL

Treball Final de Grau Universitat la Salle URL Barcelona
Enginyeria Multimèdia - Menció en Videojocs

realitzat per

Àlex Vicente Carpio
Departament d'Enginyeria Multimèdia

Tutor

Gabriel Fernández Ubiergo

Contents

1	Introducció	2
2	Part Teòrica	3
2.1	Explicació del projecte	3
2.1.1	Projecte Actual	3
2.1.2	Requeriments del nou Mixer	4
2.2	Eines Gràfiques	5
2.2.1	Headless Chrome	5
2.2.2	WebGL Headless	8
2.2.3	Unreal Engine 4	8
2.2.4	Unity	10
2.3	Tractament dels fluxes de video	13
2.3.1	Formats Compatibles	13
2.3.2	Protocols de Streaming	15
2.3.3	Llibreries de processament	19
3	Proposició	23
3.1	Comparació: Ventatges i desvantatges de cada eina	23
3.2	Elecció de l'eina i justificació	24
4	Images References	40
5	Annexes	41

1 Introducció

L'objectiu d'aquest projecte és crear una aplicació de live video mixing usant tecnologies emprades en el món dels videojocs. És a dir, una aplicació multimedia amb la que poder gestionar fluxos de video d'entrada i de sortida, convinant-los i gestionant-los d'una manera senzilla, eficaç i ràpida. Tot això creat amb una eina utilitzada als videojocs com és un motor gràfic. En un primer moment pot sorprendre una mica la idea i es pot arribar a pensar que no és viable ja que no està pensada per això. Però anireu veient que tot cobra sentit i els videojocs, o les eines per crear-los, són més potents del que creiem i s'utilitzen a molts llocs que mai hauríem imaginat. Ha estat realitzat a l'empresa Watchity S.L., on he estat treballant com a becari, i a la vegada ho he combinat amb el meu projecte de fi de grau. Ja que es tracta d'una idea molt interessant i innovadora, que em va cridar molt l'atenció desde el primer moment. En aquesta empresa ja tenien una primera aplicació, basada en una eina anomenada Snowmix que tenia les seves limitacions. Aquesta és una de les raons per la qual es van plantejar contractar un becari que refés el sistema desde un altre punt de vista totalment diferent.

En el mercat existeixen algunes altres propostes que fan la competència, però, el que es busca és tenir molta més potència i possibilitats que aquestes eines, i a la vegada ha de ser fàcil d'utilitzar.

Aquest projecte es vol enfocar en un públic professional, i no tant en casos individuals, encara que també seria possible.

Alguns exemples de les empreses/entitats que ja han treballat anteriorment amb Watchity són: *el Correo Gallego, l'Oréal, Generalitat de Catalunya, Honda...*

2 Part Teòrica

2.1 Explicació del projecte

2.1.1 Projecte Actual

Arquitectura Global

El primer que s'ha d'explicar, és la arquitectura del sistema actual de l'empresa Watchity, abans de començar amb el nou. Tenen a disposició dels clients, diferents eines de video, les més importants són:

- **Control Room**, és una eina que serveix per controlar fluxos de vídeo i distribuir-los a diferents xarxes socials. [1]
- **Cut & Share**, serveix per crear petits vídeos fàcilment, a partir d'altres més llargs, i poder-los compartir d'una manera senzilla i còmode. [1]
- Per últim tenim el **Mixer** que és en el que ens centrarem al transcurso d'aquest projecte. Aquest disposa d'un sistema de *Live Video Mixing* que es compona del frontend, i el backend. [3]

En el frontend trobem una aplicació controlada per navegador que té totes les opcions bàsiques per fer una edició en viu dels vídeos. Però, té algunes limitacions, com poden ser: un màxim de 15 capes de vídeos o imatges, no té cap tipus de compatibilitat amb elements 3D, una resolució de 720p al output, no disposa de control per guardar/carregar, limitació de potència per fluxos d'streaming simultanis, poques possibilitats de transicions, no disposa de cap eina per afegir efectes ni de video, ni de so (per exemple, chroma key o reverb, respectivament), impossibilitat per treballar a resolucions de 4K o més.



Figure 1: Frontend Mixer

A partit d'això, es treu un streaming de output en WebRTC RTP, que es pot emetre automàticament a tots els llocs que el client desitgi (Facebook, Youtube, Instagram, etc.)

En quant al backend, s'ocupa de realitzar les tasques demandades desde el frontend explicat anteriorment. A més de controlar les diferents configuracions com poden ser la transcodificació dels vídeos, configuració dels vídeos...

Parts a substituir

Encara que hi han alguns elements que primera vista poden semblar que es poden reutilitzar, no és així, ja que la idea es canviar totalment el sistema desde la base, utilitzant un altre tipus de software totalment diferent, veient que l'actual està molt limitat. Per tant, s'ha decidit no utilitzar res de l'anterior projecte i crear-ho tot desde zero.

2.1.2 Requeriments del nou Mixer

Per crear el nou Mixer s'han definit una sèrie de requisits que ha de tenir, per tal de que compleixi amb les expectatives i sigui útil per dur a terme aquesta “evolució”. És possible que el producte final acabi tenint més funcionalitats de les que s'esmenten a continuació, però sí que reflexa les més bàsiques que ha de tenir.

- Serà necessari que es puguin definir **escenes**, les quals també puguin emmagatzemar les posicions dels elements dins de cada una i totes les seves propietats.
- Poder injectar un **flux d'streaming**
- Poder injectar un video **pre-enregistrat (local)**
- Tot tipus d'elements s'han de poder insertar amb **posicions i paràmetres inicials**
- Injecció d'**imatges** amb transparència, a més de permetre el canal **alpha**
- Possibilitat de tenir fins a **6 fonts** d'streaming **simultàniament**
- Els flux d'entrada poden tenir **diferents resolucions**
- Extreure el render final via streaming (**rtp** o similar)
- Poder definir la resolució i framerate del render final (720p/25fps, 720p/30fps, 1080p/30fps, etc.)
- Poder canviar d'un element a un altre tant per **tall** com per **dissolve**
- **Mute/unmute** de cada font d'entrada
- Control de volum de cada font d'entrada
- Mixing de les fonts no mutejades
- Control de volum de la sortida mesclada
- Mute/unmute de la sortida mesclada
- **Chroma-keying** a les fonts de video (streams o local)
- Possibilitat de treballar fins a **4K**

2.2 Eines Gràfiques

2.2.1 Headless Chrome

Vanilla

La primera opció que es va proposar va ser utilitzar Headless Chrome. A partir de la versió 59 de Chrome (actualment està per la versió 90), és possible executar-lo de manera *headless*, el que significa que es pot utilitzar desde terminal, o més interessant, desde un servidor sense cap tipus de UI. Es va crear pensant en que podria ser molt útil, per fer testos automatitzats de webs o aplicacions online.

Per executar-lo és molt fàcil, només cal afegir uns quants paràmetres d'inici al Chrome. Els paràmetres d'exemple són:

```
chrome \
--headless \
    # Runs Chrome in headless
    mode.
--disable-gpu \
    # Temporarily needed if
    running on Windows.
--remote-debugging-port=9222 \
https://www.chromestatus.com # URL to open. Defaults
    to about:blank.
```

Listing 1: Config Headless Chrome

Encara que està pensat per fer automatitzacions com he comentat abans, es pot aprofitar la potència de Chrome per altres tasques, com poden ser la reproducció de vídeos, crides a APIs, entre altres.

Buscant algun exemple per internet que aprofités Headless Chrome per la reproducció de vídeo, ens vam trobar amb uns quants que havien pensat el mateix.

En un article [4] trobem que van aconseguir fer streaming a Facebook i Youtube Live amb uns resultats bastant satisfactoris. A partir d'una web (video + àudio), ho convertien a un flux de sortida RTMP. Comenta que el més complicat va ser aconseguir que el video i l'àudio estiguessin sincronitzats. Però finalment ho va aconseguir gràcies al controlador d'àudio anomenat PulseAudio [5].

Headless Chrome està pensat per ser utilitzat amb alguna eina d'automatització, i així ho recomanen desde el propi blog oficial de Google Developers [6]. Degut a que no disposem de cap tipus de UI per debugar o interactuar, utilitzarem una eina externa. Al blog de Google ens parlen de Puppeteer, Selenium, WebDriver i ChromeDriver. Totes elles tenen el mateix objectiu, automatitzar tasques al navegador. Podriem dir que la més desenvolupada i consolidada, és Selenium, però Puppeteer també s'ha guanyat molt de renom en els últims temps, per tant, haurem de fer un estudi per veure els pros i contres de cada una i decidir amb

quina d'aquestes ens quedarem.

Selenium

Selenium és una eina gratuïta i open-source [7], que serveix per crear tasques automatitzades en un navegador o aplicació per validar funcionalitats. Es poden utilitzar scripts tant de C#, Python, JavaScript, PHP, Scala... Realment Selenium com a tal és una Suite que es compon d'un grup d'eines; el seu propi IDE, Grid i WebDriver. Nosaltres estem interessats en l'últim, per tant és el que estudiarem.

Selenium WebDriver pot controlar un navegador tal i com ho faria un usuari, ja sigui de manera local o remota en un servidor.

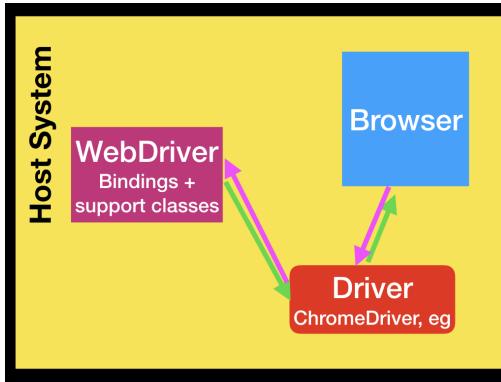


Figure 2: Selenium Driver Graph

Com podem veure a la figura 2, WebDriver es comunica amb el navegador a través d'un driver. Aquest driver és específic de cada navegador. El que ens interessa és el de Chrome, per tant sabem que utilitzarem el ChromeDriver.

Els navegadors que suporta són: Chrome, Chromium, Firefox, Internet Explorer, Opera i Safari. A la seva documentació [8] no comenten en cap lloc compatibilitat amb Headless Chrome, però degut a que està basat en el propi navegador, i no fa cap canvi brusc en el seu funcionament, podem assegurar que també serà compatible. Ho podem corroborar amb un article on ho experimenta [9], en el que expliquen que només hem d'indicar el driver corresponent a Chrome, i escriure a les opcions, el flag «`--headless`».

Puppeteer

Per altre banda tenim un altre eina bastant més nova, anomenada Puppeteer [10]. Les funcionalitats principals venen a ser molt semblants a Selenium, controlar totes aquelles coses que podries fer manualment al navegador, fer-ho de

manera automàtica. A més de funcionalitats molt útils com fer captures de pantalla i crear PDFs.

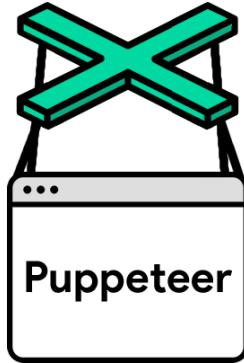


Figure 3: Puppeteer Logo

És open source i està sota la llicència d'Apache 2.0 [11] pel que es pot utilitzar i comercialitzar de manera totalment lliure tot aquell contingut que tingui les seves eines.

És molt fàcil d'instalar, ja que només cal instalar un paquet npm i per utilitzar-lo es fa tot amb javascript mitjançant les comandes que disposen a la seva documentació. A més de tot això, la major ventatja que té és que està pensat per ser utilitzat amb Headless Chrome, dóna moltes facilitats per realitzar totes les tasques sense estar veient cap tipus de UI, i a la vegada poder debuggar de manera còmode i senzilla.

Un petit codi d'exemple que ens mostren a la seva documentació per comprovar com de fàcil és utilitzar-lo. El que fem és afegir puppeteer al nostre script de js amb require, iniciem el navegador en mode headless, obrim el link <<https://example.com>>, fem una captura i tanquem el navegador.

```
const puppeteer = require('puppeteer');

(async () => {
  const browser = await puppeteer.launch();
  const page = await browser.newPage();
  await page.goto('https://example.com');
  await page.screenshot({ path: 'example.png' });

  await browser.close();
})();
```

Listing 2: Puppeteer Get Started Example

2.2.2 WebGL Headless

WebGL és una API escrita en Javascript utilitzada per la renderització de gràfics en 3D al navegador. Està basat en la Open Graphics Library (OpenGL), pel que només necessitarem que sigui compatible amb aquest [12], sense cap altre dependència que podria dificultar la compatibilitat. Hem de tenir en compte que encara que el navegador sigui compatible, per tota la renderització 3D, per molt mínima que sigui, necessitarem una targeta gràfica, Això pot ser un problema ja que es vol executar tot en un model cloud, al donar estabilitat i professionalitat. La majoria de gent que utilitza servidors té unes idees bastant diferents al que es vol fer aquí, i potser necessiten potència de CPU o molta RAM, però no GPU, pel que potser és complicat trobar una màquina remota amb la potència suficient que es necessita.

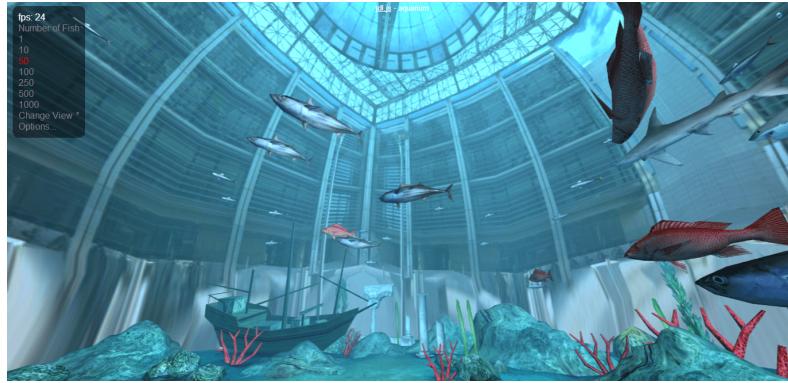


Figure 4: Exemple WebGL

Per una banda aquesta solució és molt atractiva ja que es faria tot el sistema desde 0 sense utilitzar cap tipus de dependència, però a la vegada també depèn de les polítiques o limitacions de potència del navegador. Mai serà capaç d'utilitzar el 100% de la màquina i no és comparable amb una aplicació standalone.

2.2.3 Unreal Engine 4

Un dels motors gràfics més utilitzats dels últims temps és Unreal Engine. Està creat per Epic Games, programat en C++ i amb una potència sorprenent que fa que sigui el més utilitzats en projectes fotorrealistes, o inclús, en projectes d'arquitectura. Sobretot a partir de 2019, on van juntar-se amb l'empresa Quixel, dedicada entre d'altres coses, a crear «megascans». És una tecnologia que diposa textures amb grans resolucions i qualitats sorprenents, però a la vegada molt eficients pel desenvolupament de videojocs. Així aconseguint un renderitzat molt realista a temps real, i destacant entre els altres motors competidors.



Figure 5: Captura Quixel Demo Megascan a UE4 (Youtube)

Durant el transcurs del 2021 es llençarà la versió d'Unreal Engine 5, on han demostrat que poden inclús superar-se i crear uns entorns sorprenents juntant la tecnologia que ja havien estat desenvolupant de Quixel, amb una nova anomenada Lumen, que controlarà totes les llums i farà ús del Ray-Tracing a temps real, també dit RTX a les targetes gràfiques de Nvidia. Amb totes aquestes novetats i evolucions tecnològiques que han anat fent en els últims anys, és lòtic que s'hagin guanyat entre la comunitat el nom a millor motor gràfic en quant a qualitat visual [13]. Però com bé sabem, això no és l'únic que importa.

L'Unreal disposa d'un sistema de programació anomenat Blueprints, és una metodologia de programació totalment visual i basada en nodes. Bàsicament són petites capces que internament tenen trossos de codi en C++, els quals juntem entrades i sortides.

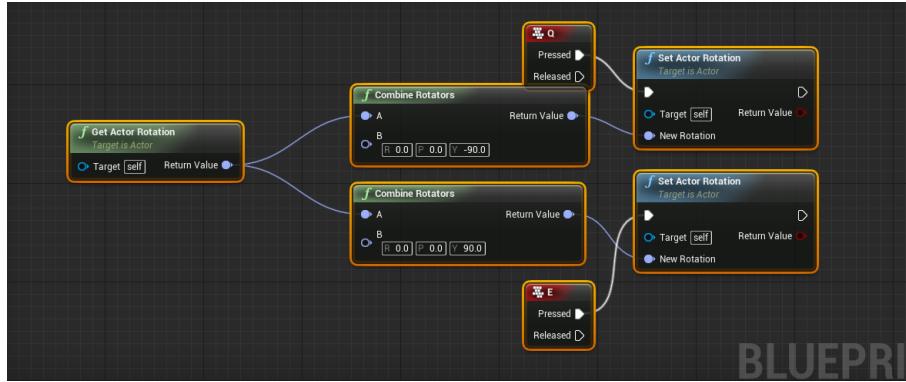


Figure 6: Rotació Programada en Blueprint UE4

Això pot ser molt còmode i és un punt a favor per la gent que comença a desenvolupar videojocs, si estàs acostumat a utilitzar-ho o no es tenen els

coneixements necessaris de programació i vols fer scripts amb una complexitat baixa-mitjana. Però pel contrari si es volen fer scripts més complexes pot ser que ens trobem amb certs inconvenients i poden quedar scripts amb milers de nodes i que sigui molt difícil d'interpretar o debugar.

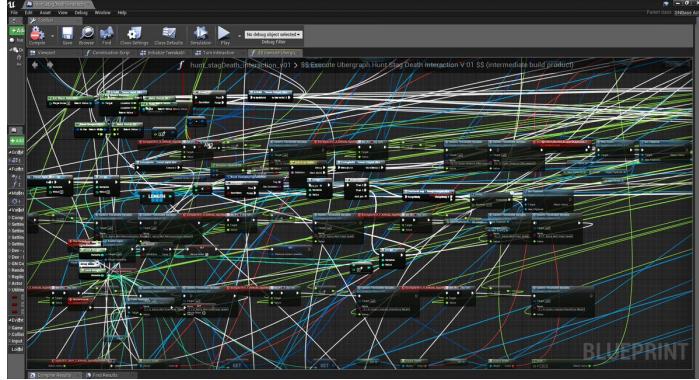


Figure 7: Exemple Blueprint Caòtic UE4

Altres inconvenients poden ser que les idees que es tenen surtin fora de la norma, que els nodes estiguin massa limitats o que simplement no t'agradi, per això també et donen la opció a crear un script buit i fer-ho tot amb C++. I llavors veiem que la documentació que es pot trobar és molt inferior a la dels blueprints, i possiblement sigui més complicat i més llarg de fer que els seus competidors, com poden ser Unity o Godot.

2.2.4 Unity

El motor de videojocs més famós i més utilitzat desde els seus orígens és sense dubte Unity [14]. Va ser creat al 2005 a Copenhaguen per 3 amics, David Helgason, Joachim Ante i Nicholas Francis.

És el motor que suporta més plataformes actualment [15], com Windows, Mac OS, Linux, iOS, Android, WebGL, Playstation, Xbox, Nintendo Switch, Stadia i moltes altres més. Això és un punt molt a favor, ja que necessitem desenvolupar l'aplicació per un servidor Linux, i Unity és el motor més professional dels que donen suport. Altres motors com Unreal Engine, Game Maker o Cry Engine no accepten el sistema operatiu Linux com a opció de compilació, o funcionen amb bastants problemes.

La programació dels scripts es fa amb llenguatge C# el qual forma part de la plataforma .NET propietaria de Microsoft. És un llenguatge derivat de C/C++ i és dels més utilitzats juntament amb Java. Si ens fixem en el rànking de plataforma TIOBE [16], que és una web que s'encarrega d'ordenar els llenguatges de programació de més utilitzat a menys, podem veure que en el top 5, tenim els

3 llenguatges de C, els quals tenen una base molt semblant. Després tindriem Java i Python, tots dos són llenguatges d'alt nivell molt senzills i que faciliten molt el flux de programació.

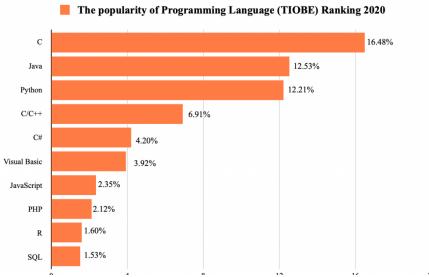


Figure 8: Llenguatges de programació més utilitzats al 2020

A més de que Unity utilitzi C# que ja és un llenguatge d'alt nivell de per si, s'utilitza la API MonoBehaviour [17], que estalvia molta feina i permet fer tasques que a priori són complicades o llargues, en pocs minuts i en poques línies.

MonoBehaviour és la classe base d'on parteixen tots els scripts de Unity. Només cal indicar que el nou script s'estén de la API.

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class NewBehaviourScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

}

```

Listing 3: Unity Code Basic Template

En les últimes versions de Unity, han apostat molt per competir directament amb Unreal Engine, en un dels punts que tenien més diferenciats, l'atractiu visual hiperrealista.

Quan Unity va començar, com que era relativament fàcil d'utilitzar i gratis, van aparèixer una quantitat immensa de jocs de mòbil creats amb aquest motor, pel que es va guanyar una fama errònea, de que aquest motor només servia per jocs petits, amb uns gràfics tirant a simples i sense molts efectes.

Per tant, al 2019 van llençar amb la seva última actualització, una nova funció anomenada HDRP (High Definition Render Pipeline) [18], la qual millorava tot el sistema del render del motor, aconseguint uns resultats que van sorprendre a tota la comunitat, i que no tenien res a envejar amb Unreal Engine.



Figure 9: Captura Book of the Dead Unity Demo HDRP (Youtube)

A més, podem trobar que el videojoc més realista que s'ha creat mai en quant al gènere de FPS (First Person Shooter), van decidir crear-lo amb el motor Unity, enllot d'Unreal, i van aconseguir uns resultats molt bons, fins al punt que està considerat el joc amb millors gràfics, creat amb aquest motor. Es tracta del videojoc *Escape from Tarkov*.



Figure 10: Captura Escape from Tarkov Pre-Alpha

2.3 Tractament dels fluxes de vídeo

2.3.1 Formats Compatibles

L'aplicació haurà de rebre tot tipus de vídeos, de tots els formats possibles, entenent formats el conjunt de còdecs i contenidors. Primer s'ha de fer un anàlisi dels formats que existeixen i com de populars són.

Còdecs

Hi ha molts tipus de còdecs, però és cert que alguns s'utilitzen molt poc, o ja s'han quedat antiquats.

- H.261: Aquest còdec va ser el primer que es va tornar popular, degut a que era el que donava millors resultats en el seu moment. Però evidentment, ha passat molt temps i la tecnologia ha evolucionat suficient com per deixar aquest còdec antiquat.
- H.263: Va ser el primer que còdec realment eficient, però a la vegada demanava molts recursos de l'ordinador per codificar un vídeo, i a la vegada, per reproduir-lo necessitarem una potència relativament alta.
- H.264: És el més popular a l'actualitat de manera professional, també conegut com MPEG-4 AVC, compta amb una compressió molt alta amb una pèrdua de qualitat imperceptible. A més de ser molt polivalent al acceptar molts tipus de bitrate d'entrada.
- H.265: També anomenat HEVC, és el successor del H.264. A vegades pot donar uns resultats pitjors al seu predecessor, però sobretot pels vídeos en 4K obtenim uns molts millors resultats ja que comprimeix molt més obtenint uns resultats de qualitat molt semblants. En els vídeos 4K es pot obtenir fins a un 64% de reducció de bitrate en comparació amb H.264.

- **MPEG-1:** És un códec bastant antic que ja no s'utilitza, però va ser molt utilitzat per discs de dades CD-ROM i inclou compressió tant de video com d'àudio.
- **MPEG-2:** Dona bons resultats amb el DVD, però requereix pagar llicència per poder-lo utilitzar.
- **MPEG-4:** S'utilitza molt per contingut de vídeos a la xarxa, ja que té una gran compressió de vídeo i àudio, però la qualitat pot baixar al utilitzar baixos nivells de bitrate.
- **X264:** Molt similar al H.264 amb la gran diferència de ser open-source, el que fa que sigui el més utilitzat de manera gratuita, amb els millors resultats.
- **DivX:** Comunament utilitzat per comprimir pel·lícules DVD i famós per impulsar la pirateria al voltant de l'any 2000 al permetre comprimir fins a 7 pel·lícules en un sol disc DVD [19].

Contenidors

- **AVI:** Cada cop és menys utilitzat, però a l'anterioritat va ser molt popular per la gran compatibilitat de la que disposava en diferents sistemes operatius.
- **MOV:** A diferència de l'AVI, aquest códec creat per Apple, era molt poc o gens compatible amb altres sistemes operatius, pel que només s'utilitzava amb els seus dispositius.
- **MPG:** Són un conjunt de formats, i alguns d'ells han arribat a ser els més famosos, sobretot el MPG-4, també anomenat MP4. Que té molta qualitat, comparable al MOV, i actualment és pràcticament impossible trobar algun ordinador que no sigui capaç de reproduir-lo.
- **WMV:** Windows Media Video és un format propietari de Windows, i necessitarem reproduir-lo amb el Windows Media Player per aconseguir els millors resultats d'aquest.
- **MKV:** Acrònim de Matroska, és un format molt utilitzat per emmagatzemar més d'un arxiu en un sol. Per exemple podem trobar un arxiu MKV que contingui més d'una pista d'àudio (per diferents idiomes), alguns arxius de subtítols o inclús diferents arxius de video en un sol.
- **FLV:** El format de video de Flash Player, que és molt compatible amb tots els ordinadors, però cada cop s'està quedant més antiquat.

Els reproductors més famosos, com són VLC, MPV, o MPlayer, accepten tots els tipus de códecs i contenidors, pel que serà necessari i pràcticament requisit que l'aplicació també els accepti tots per a que no es quedí enrera. Ja sigui

creant un reproductor propi, o un basat en llibreries de tercers.

Imatges

En quant a les imatges, també trobem molts formats diferents i s'haurà de tenir en compte per decidir si es podrà donar suport a tots ells, o pel contrari s'haurà de prescindir d'alguns. Com a mínim hauria d'acceptar els més famosos, que són:

- BMP: Creat per l'empresa Microsoft, és l'estàndard del sistema operatiu Windows.
- JPEG: Un dels formats més utilitzat degut a la seva gran compressió, mantenint una qualitat molt acceptable.
- GIF: El GIF és un format diferent a la resta, ja que consta d'un conjunt d'imatges que es reproduïxen de manera continua, formant un petit video. Això si, la seva qualitat queda molt limitada.
- PNG: Disposa d'una gran compressió mantenint la qualitat, com podria ser la del JPEG, però, amb la gran ventatja de que disposem de l'atribut alpha, el que permet tenir imatges amb transparències.

Àudio

Els vídeos tindràn el seu propi àudio, i encara que no està pensat com a requisit que es pugui reproduir àudio de manera independent, com podria ser una música de fons o una veu en off. Els formats que s'haurien de suportar per poder tenir un mínim de compatibilitat són:

- WAV: Desenvolupat per Microsoft i un dels més utilitzats, és compatible amb pràcticament tots els còdecs. Accepten arxius d'un tamany gran fins a 4GB, i poden estar molt comprimits o poc.
- MPEG: Un dels formats més famosos per ser un estàndard en àudio i sobretot, en la música, podem trobar per exemple els còdecs MP2 (més utilitzat per aplicacions de broadcast) o el MP3.

2.3.2 Protocols de Streaming

En quant a la sortida del vídeo/àudio resultant, s'haurà de decidir quin protocol s'utilitza, ja que es voldrà una sortida en streaming. El vídeo en streaming, o retransmissió en directe, és la emisió d'un flux de dades a temps real, o amb el menor retard possible, a un destí. Per exemple, quan mirem la televisió, tenint en compte que és per internet i no per antena, estem rebent un flux de dades que arriba desde uns servidors centrals de la cadena en particular. Es van començar a fer experiments de televisió fa molts anys, però la majoria van ser un fracàs degut al gran ample de banda que demanaven els vídeos. Per tant el que va sorgir primer van ser els serveis de VOD (Video on Demand). No s'ha de

confondre el VOD amb el Streaming ja que són dos conceptes totalment diferents i que no utilitzen les mateixes tecnologies. Un exemple de servei VOD que va triunfar és la plataforma Youtube. Aquesta tecnologia va anar avançant fins que van millorar les connexions a internet i això va provocar que per fi el Streaming fos possible. L'objectiu del directe, és tenir un retard mínim, amb una qualitat bona. Actualment el mínim de qualitat per una plataforma professional és la resolució 1920x1080. La resolució 4K dona molt bona qualitat, però requereix una quantitat molt alta d'ample de banda i actualment grans plataformes de streaming com Youtube o Twitch no ho permeten com a opció. Encara i això serà un repte a tenir en compte i seria un gran punt a favor respecte a la competència.

Els protocols de streaming són els encarregats de posar les «normes» en com es transmetrà el flux de dades, a més de contemplar els errors i minimitzar-los en tot lo possible.

Un protocol no té res a veure amb un códec o un format, sinó que és complementari a aquests i només s'ocupa de la transmissió.

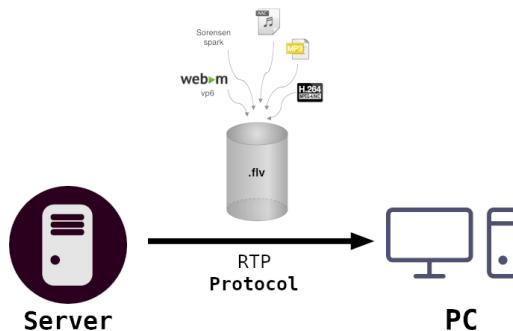


Figure 11: Diagrama Protocol-Format-Códec

Tots els protocols hauràn d'utilitzar un dels dos mètodes de transmissió que existeixen, que són TCP (Transmission Control Protocol) o bé, UDP (User Datagram Protocol). La diferència més important i notòria entre aquests, és que el TCP, requereix d'una resposta del client, forçant a que tots els paquets arribin, el que fa una connexió més segura, però més lenta. Quan TCP no aconsegueix enviar un paquet correctament, repeteix el procés fins a aconseguir-ho. Per altre banda, UDP ignora això, simplement envia els paquets, i el receptor ja s'ocuparà de rebre-ho correctament. Això provoca que a vegades es pugui percebre fragments en el video i pèrdua de qualitat, però és molt més ràpid i té menys retard.

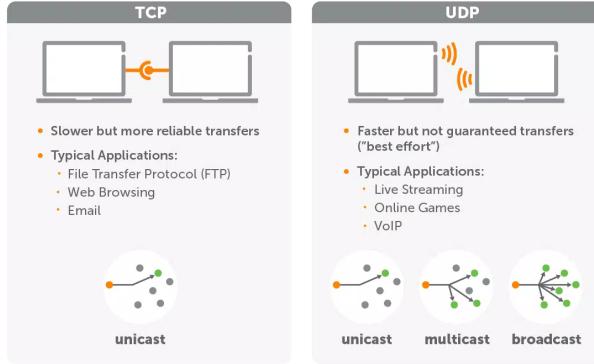


Figure 12: Comparació TCP-UDP

Tornant als protocols de live streaming, hi ha molts per escollir, pel que es comentaran els més importants.

- RTMP (Real Time Messaging Protocol): Protocol TCP. És propietat d'Adobe, pel que era molt utilitzat a les webs que disposaven d'Adobe Flash Player. No obstant, s'ha quedat bastant obsolet i presenta problemes de compatibilitat amb còdecs moderns i la seguretat és baixa. Per altre banda, els resultats que dona són de bona qualitat i té molt suport amb les diferents plataformes i software. Els codecs més recomanats per aquest protocol són H.264 pel vídeo, i AAC per l'àudio.
- WebRTC: Protocol TCP i UDP. Utilitzat per la gran majoria de desenvolupadors web, degut a la seva gran compatibilitat amb tots els navegadors moderns: Chrome, Firefox, Opera, Safari, Edge... És de gran qualitat i soporta els còdecs VP8 i VP9, clarament superiors al H.264 en quant a streaming de vídeo. Per l'àudio, tenim que WebRTC suporta el còdec Opus, gairebé el més utilitzat per vídeos en directe. A més, amb la ventatja de ser open source, el projecte està en una constant evolució, i pròximament suportarà el còdec H.265, i més important encara el AV1, que promet una gran millora en tots els aspectes [21]. Té el millor resultat de latència, podent arribar a valors menors a 1 segon.
- FTL (Faster Than Light): Protocol UDP. Creat per l'empresa Microsoft, dedicat a la plataforma ja extinta Mixer [22]. Una proposta que no va sortir del tot bé, i que pretenia competir directament amb Twitch [23]. Aquest protocol té com a objectiu ser el més ràpid, com indica el seu nom, per poder comunicar-te amb els visualitzadors del directe d'una manera pràcticament instantània. Tenint en compte que Twitch utilitza RTMP, i ho converteix a HLS [24] (molt més lent que la proposta de Mixer), no era una mala idea. Encara i això, Twitch ja era massa gran i estàndard com per poder competir.

- SRT (Secure Reliable Transport): Protocol UDP. També es un protocol open-source, al igual que WebRTC, i és considerat com a l'evolució de RTMP. És de molta qualitat i bastant més estable als anteriors. Els objectius d'aquest projecte són crear un protocol que transmeti vídeos sense soroll al senyal (jitter) i que eviti la pèrdua de paquets.

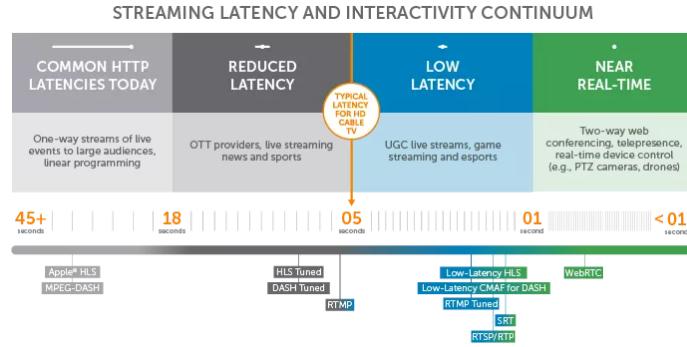


Figure 13: Gràfic Latències Protocols de Streaming

Resumint, no hi ha protocols més bons o més dolents, depenen el projecte que es vulgui fer, serà més adequat un o altre. Pel projecte que s'ha proposat en aquest treball, es necessitarà sobretot un protocol amb poc retard.

	RTMP	WebRTC	FTL	SRT
Pros	Estabilitat, gran compatibilitat i qualitat	Open-source, còdecs actuals i molt baixa latència	Qualitat adaptativa i molt ràpid	Molta qualitat, estabilitat i compatibilitat de codecs
Contres	Molta latència, més insegur que altres opcions, còdecs antics	Constantment en desenvolupament i pot ser inestable	Qualitat inferior a la resta i poca compatibilitat	Està en desenvolupament, no hi ha gaire documentació
Video Còdecs	H.264	VP8, VP9, H.264 (H.265 i AV1 futur proper)	H.264	-
Àudio Còdecs	AAC	Opus	Opus	-
Latència	3-30 segons	1 segon o menys (menor a WebRTC)	1 segon o menys	1 segon o menys

Table 1: Taula resum dels protocols esmentats

2.3.3 Llibreries de processament

La gran majoria d'aplicacions que utilitzem a diari que disposen de productors multimedia, o qualsevol tipus de processament de video/àudio, utilitzen llibreries externes per fer aquest procés més senzill. Hi ha varietat d'opcions de llibreries de processament de contingut multimèdia. Totes elles tenen un desenvolupament complex, és per això que hi ha poca rivalitat, i totes disposen d'una API per utilitzar-les. Les més utilitzades i conegudes, són dues que es comentaran a continuació:

FFMPEG

És la llibreria de processament de video/àudio més gran que existeix, i és utilitzada per la gran majoria de desenvolupadors.



Figure 14: Logo FFmpeg

Concretament, el que defineix FFmpeg, és un conjunt de software lliure (framework), que tal i com diu a la seva documentació [25], les funcionalitats que permet són:

- Decodificar: és l'acció de convertir un flux de dades en un contingut multimedya visible i amb sentit per a un receptor.
- Codificar: és l'acció contraria a decodificar, convertir un contingut multimedya a un flux de dades, ja sigui per una transmissió més còmode, comprimir, o el que es desitji.
- Transcodificar: significa convertir un contingut multimèdia d'un códec a un altre. Per exemple, ens podem trobar amb un contingut en H.265, que el volem reproduir en un ordinador relativament antic, que no accepta aquest códec, pel que haurem de transcodificar-lo a un altre que l'ordinador si que «entengui» com podria ser el H.264.
- Multiplexar: si tenim video i àudio per separat i ho volem juntar, el que haurem de fer es un mux, és a dir, juntar dos o més fluxes d'entrada, en un de sortida.
- Demultiplexar: és el contrari de multiplexar, tenint un sol flux d'entrada, extreiem dos o més.
- Stream: si enlloc de voler generar un fitxer de sortida, el que volem és directament crear un streaming, utilitzant el protocol més apropiat, ho podem fer tot desde ffmpeg i sense utilitzar cap eina externa.
- Filtres: per si totes les funcionalitats anteriors fóssin poques, també permeten aplicar filtres als àudios o als vídeos, el que ho fa molt còmode per poder automatitzar tasques desde terminal, sense haver d'utilitzar editors de vídeo o àudio. En quant a filtres d'àudio disposem de 105 efectes, entre ells els més senzills com el volum, compressors... I en quant al vídeo, disposem de 264 efectes, com per exemple, fer un crop, escalar, rotar... [26]
- Reproduir: FFmpeg també inclou una eina anomenada FFplay, que permet reproduir tot tipus d'arxius sense problemes. Normalment s'utilitza com a eina de test.

Es va desenvolupar principalment per GNU/Linux, però està disponible per tot tipus de plataformes com Windows i MacOS. S'utilitza a través d'un terminal, però també es pot integrar en tot tipus d'aplicacions per utilitzar-lo internament, com fa Audacity, Youtube, Chrome, OBS Studio, i molts altres grans projectes.

```
ffmpeg -i input -map 0 -c:v libx264 -crf 18 -c:a copy  
output.mkv
```

Listing 4: Exemple Transcodificació Arxiu H.265 a H.264

La instalació d'aquesta eina és molt senzilla, sobretot si s'utilitza Linux, o concretament Ubuntu, es pot aconseguir introduint un parell de línies al terminal.

```
sudo apt-get update  
sudo apt-get install ffmpeg
```

Listing 5: Instalació FFmpeg Ubuntu

També és molt utilitzat per la compressió de vídeos de manera automàtica, per exemple, si pujem un vídeo a «Twitter», no es pujarà el vídeo original, ja que el pes de l'arxiu seria excessiu per emmagatzemar-ho al seus servidors, sinó que es fa una copia reduïda del vídeo, i aquest és el que es guarda.

LibVLC

Per altre banda trobem un altre gran framework, l'únic capaç de competir amb FFmpeg, i és LibVLC. Està desenvolupat per la organització VideoLAN, on tenen com a objectiu, desde la seva fundació, desenvolupar aplicacions open-source gratuïtes. Van començar el projecte a una universitat de França l'any 1996, i va començar a ser open-source a partir de l'any 2001, però, a l'any 2009 van decidir seguir amb el projecte de manera independent i trencar tots els vincles amb l'escola [27]. Ara mateix tenen desenvolupadors per 40 països diferents, i animen a la comunitat a contribuir i recolzar el projecte ja sigui programant, traduint, donant components i material, o amb alguna aportació econòmica [28].

El major projecte creat per aquesta organització és el mundialment conegut VLC, un reproductor multimèdia basat en les seves pròpies llibreries LibVLC.



Figure 15: Captura de la primera versió de VLC

LibVLC no està tant pensat per utilitzar-lo desde terminal com podria ser FFmpeg, i encara que també es pot integrar a qualsevol projecte, és més complicat de instal·lar i utilitzar [29].

```

#include <stdio.h>
#include <stdlib.h>
#include <vlc/vlc.h>

int main(int argc, char* argv[])
{
    libvlc_instance_t * inst;
    libvlc_media_player_t *mp;
    libvlc_media_t *m;

    /* Load the VLC engine */
    inst = libvlc_new (0, NULL);

    /* Create a new item */
    m = libvlc_media_new_location (inst, "http://
        mycool.movie.com/test.mov");
    //m = libvlc_media_new_path (inst, "/path/to/test.
        mov");

    /* Create a media player playing environment */
    mp = libvlc_media_player_new_from_media (m);

    /* No need to keep the media now */
    libvlc_media_release (m);

#endif

```

```

/* This is a non working code that show how to
   hooks into a window,
* if we have a window around */
libvlc_media_player_set_xwindow (mp, xid);
/* or on windows */
libvlc_media_player_set_hwnd (mp, hwnd);
/* or on mac os */
libvlc_media_player_set_nsobject (mp, view);
#endif

/* play the media_player */
libvlc_media_player_play (mp);

sleep (10); /* Let it play a bit */

/* Stop playing */
libvlc_media_player_stop (mp);

/* Free the media_player */
libvlc_media_player_release (mp);

libvlc_release (inst);

return 0;
}

```

Listing 6: Codi base d'exemple de libVLC

3 Proposició

3.1 Comparació: Ventatges i desventatges de cada eina

Està clar que cada eina té les seves ventatges i desventatges, tampoc hi ha una que sigui millor que l'altre, sinó que depèn del projecte que es vulgui fer les solucions seran molt diferents. A continuació s'analitzaran els pros i contres de cada eina, desde un punt de vista subjectiu pel projecte que es vol fer:

	Chrome Headless	Unreal Engine 4	Unity
Pros	Simplicitat, alta compatibilitat, no requereix utilitzar plugins	És professional, millor qualitat d'iluminació	Molta documentació, molts plugins open-source, optimitzat per escenes senzilles, alta compatibilitat
Contres	Pot quedar limitat en quant a potència; efectes de video	Relativament poca documentació, no tants plugins i comunitat com la competència, poca compatibilitat amb Linux	Menys potent que Unreal, eines externes no professionals creades per la comunitat

Table 2: Comparativa subjectiva de les possibles eines

3.2 Elecció de l'eina i justificació

Per poder decidir amb criteri i assegurar-se de que es prèn la decisió correcte sobre quina és l'eina que s'ha de utilitzar, es farà una prova amb cada una de les eines esmentades (Chrome Headless, Unreal Engine 4 i Unity).

Chrome Headless

És molt útil per tasques automatitzades, no es necessita cap tipus de software extra, com a molt es poden necessitar llibreries de nodejs pels scripts de javascripts, però en tot cas, quedaria tot empaquetat en un sol projecte. Al utilitzar el navegador de Chrome, això ens dona moltes ventatges en quant a simplicitat de les tasques, però també ens limita la potència, compatibilitat de còdecs i formats, pel que mai serà exactament igual que si executéssim les mateixes tasques de manera nativa al ordinador.

Si busquem projectes creats amb aquesta eina que s'assemblin a l'objectiu que volem aconseguir, trobem una aplicació creada per «Sebastian Pereyro», de la web Empirical [30]. Headless Chrome està disponible a partir de la versió 59, actualment van per la 91, pel que no seria cap problema utilitzar aquesta eina. Es poden carregar pàgines web i executar tests o tasques, a més de generar PDFs i fer captures de pantalla repetidament. A més comenta que es pot compartir de manera directa el contingut «Screencast content», això si, fent múltiples captures de pantalla. L'experiment que es va proposar va ser de capturar àudio i video de manera automàtica, tant d'una web pròpia com d'una web externa i

transmetre el contingut a Facebook, Youtube, Twitch...

El que utilitza com a eines, són Node.js, el navegador Headless Chrome en qüestió, Pulseaudio que és el motor de video que utilitzarà per la captura, i FFmpeg que farà la transmisió final del flux de Headless Chrome cap a les destinacions pertinents. Tot això ho vol fer utilitzant un servidor que s'ocupa de totes les tasques. Tenint una aplicació client, en aquest cas farà ús de Postman [31], es comunica amb el servidor central que tindrà tota la programació, agafa el contingut multimèdia d'un altre web, i l'envia als servidors multimedia.

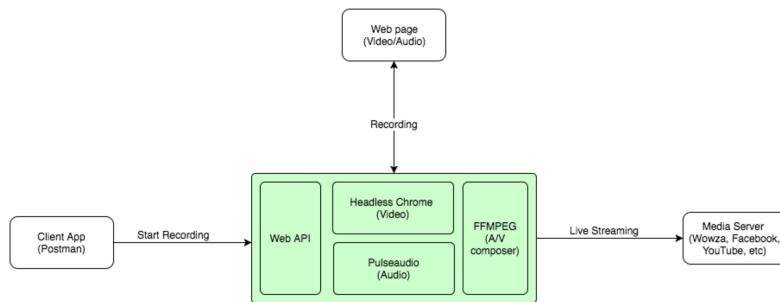


Figure 16: Gràfic Experiment Headless Chrome

```

// Init remote interface to chrome port
function initRemoteInterface(chrome){
    const port = chrome.port;
    logger.log("Initialize Remote Interface on port: "
        + port);
    return ChromeRemoteInterface({port: port});
}

//Load Page
async function loadPage(url){
    await Page.navigate({url: url});
}

//Start screencast
function startCapturingFrames(){
    logger.log("Starting capturing screen frames... ");
    return Page.startScreencast({
        format: "jpeg",
        quality: 100
    })
}

```

Listing 7: Experiment Headless Chrome Empirical

En el codi que ens proporciona 7, podem comprovar com fa la captura de la pantalla. Es tracta de una captura total de la finestra del navegador. Per a fer-ho, utilitza una funció integrada al Chrome, anomenada startScreencast. Pels paràmetres de configuració que necessita i veient la documentació de Chrome DevTools [32], podem deduir que es tracta un petit script que envia la comanda de fer captura de pantalla, en el format que desitjem (en el cas d'exemple és en jpeg), i junta aquests frames per aconseguir tenir un video. En quant a l'àudio, ens comenta que encara que era un dels seus reptes, no està suportat per Chrome, pel que no es podrà capturar de manera nativa. La opció més vàlida serà capturar-lo per separat, i més tard juntar-ho, el que farà que hi hagi problemes de sincronització d'àudio/video, un problema molt comú en aquest tipus de projectes. La qualitat dels vídeos tampoc podrà ser molt alta, ja que headless chrome no suporta acceleració per hardware amb GPU, ho veiem a l'exemple 1 amb el flag –disable-gpu extret directament de la documentació de Google [6]. Un altre cosa que comenta és que el framerate serà variable, nosaltres podem enviar 30 comandes per segon de que faci la captura de pantalla, però és pràcticament impossible que vagi totalment sincronitzat i faci exactament les 30 captures cada segon, un dels problemes que comporta això, és que s'anirà desincronitzant poc a poc, cada cop més, l'àudio.

Després de fer totes les captures, o mentres s'estan fent, es pot utilitzar FFmpeg per juntar les imatges en una seqüència de video, juntar-li l'àudio, i transmetre-ho a on es desitji, ja sigui en directe o a un fitxer local.

En aquest projecte, no s'ha utilitzat cap eina d'automatització de navegador, sinó que s'ha fet tot amb Headless Chrome Vanilla. Pel que es farà una prova utilitzant totes les eines del projecte, i a més, amb Puppeteer per poder-ho automatitzar tot.

Per la primera prova, es programarà tot en HTML i Javascript. Per una banda tindrem el index.html, un arxiu molt senzillet que contindrà un o dos vídeos locals que es reproduiràn automàticament a dins un canvas, al obrir l'arxiu.

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="index.css">
  </head>
  <body>

    <div id="element-to-record">
      <video autoplay muted controls class="myvideo" id="myvideo">
        <source src="/localVideos/
```

```

        NightcrawlerTrailer.mp4" type="video/mp4">
    </video>
    <video autoplay muted controls class="myvideo2" id="myvideo2">
        <source src="/localVideos/
            StalkerTrailer.mp4" type="video/mp4">
    </video>
</div>

<canvas id="background-canvas" style="position
    :absolute; top: -9999999px; left
    : -999999999px;"></canvas>

</body>
</html>
```

Listing 8: Test Chrome Headless | index.html

L'aspecte de la web és correcte, tenim un video, concretament, el tràiler de la pel·lícula *Nightcrawler* [33] ocupant tota la pantalla en el fons, i a la cantonada superior esquerra, ocupant un quart de la pantalla tenim el tràiler de la pel·lícula *Stalker* [34].



Figure 17: Resultat index.html

Un dels primers problemes que han sorgit, ha sigut amb la reproducció dels vídeos, degut a que un navegador no està pensat exclusivament per aquestes funcions, les seves polítiques d'ús poden canviar dràsticament d'un dia per l'altre i destrossar un projecte d'aquest estil. Tant és així que a l'any 2017, van intro-

duir una nova política a la versió 71, en la que no estava permès auto reproduir els vídeos automàticament al obrir una web, sinó que el usuari havia de clicar manualment al vídeo. Això si, permetia reproduir-los si estaven sense so. Una política probablement introduïda per evitar elús de publicitat molesta, però que hauria canviat per complet aquest projecte i hauria fet que deixés de funcionar. Per tant, les opcions que es tenen, és reproduir els vídeos sense àudio (inviable ja que l'àudio és un dels requisits), o utilitzar una eina externa com Puppeteer per fer «trampes» i clicar allà on volguem fent creure al navegador que ho està fent un usuari qualsevol.

```

const puppeteer = require('puppeteer');
const PuppeteerVideoRecorder = require('puppeteer-
    video-recorder');
var path = '/Output/';

init_puppeteer();

var global_browser;
const videosPath = "/Output/";

async function init_puppeteer() {

    // Execute Puppeteer
    global_browser = await puppeteer.launch({headless:
        true, executablePath: '/usr/bin/google-chrome',
        ignoreDefaultArgs: ['--mute-audio']});

    check_login()
};

async function check_login()
{
    try {
        const page = await global_browser.newPage();
        const recorder = new PuppeteerVideoRecorder();
        await recorder.init(page, videosPath);
        await page.setViewport({width: 1920, height:
            1080});

        await page.goto('http://localhost:3000/', {
            timeout: 60000})
            .catch(function (error) {
                throw new Error('TimeoutBrows');
            });
    };
}

```

```

//Cliquem el boto que executra el script per
//desmutejar els videos
await page.click('#unmute');

//Executem el script per a fer captures
await page.click('#startStreaming');

}

catch (e) {
    console.log(' LOGIN ERROR
    -----');
    console.log(e);
}

function delay(time) {
    return new Promise(function(resolve) {
        setTimeout(resolve, time)
    });
}
}

```

Listing 9: Test Reproducció Videos | videoPlayback.js

Amb aquest codi ja tindrem uns scripts que guardaran 30 captures per segon a una carpeta local. A la vegada, podem executar una comanda FFmpeg, per convertir aquestes captures en una seqüència de video, i guardar-ho a un arxiu local. Això ho podrem aconseguir amb aquesta sèrie de comandes a la terminal.

```

//Executem el script de node videoPlayback.js
node videoPlayback.js

//Iniciem el ffmpeg per llegir totes les captures i
//convertir-les a un streaming rtmp
cd /Output/images/
ffmpeg -r 24 -pattern_type glob -re -i '*.jpg' -f flv
rtmp://example.com

//FFplay Stream per reproduir-lo
ffplay rtmp://example.com

```

Listing 10: Comandes FFmpeg

Pel FFmpeg s'utilitzen els flags:

- -r 24: indica el número de frames per segon que tindrà el video resultant.
- -pattern_type glob [36]: indiquem el patró del nom dels arxius per una

lectura correcte, en aquest cas ens ajudavem d'un txt que contenia tots els noms dels arxius, generat automàticament pel javascript.

- -re: indica que el processament s'ha de fer amb una velocitat de 1x, per evitar que processi les imatges més ràpid de 24 FPS i es vegi més ràpid del que hauria.

Aquesta solució no ha donat uns resultats gaire atractius, sobretot pel fet d'estar capturant la pantalla amb screenshots, i sense poder transmetre el àudio. Pel que s'ha provat un altre alternativa, basada en un altre projecte creat per l'usuari de Github FBSamples [37].

En aquesta altre proposta, trobem algunes variants interessants, el més diferent és que utilitza un servidor de websockets com a intermediari.

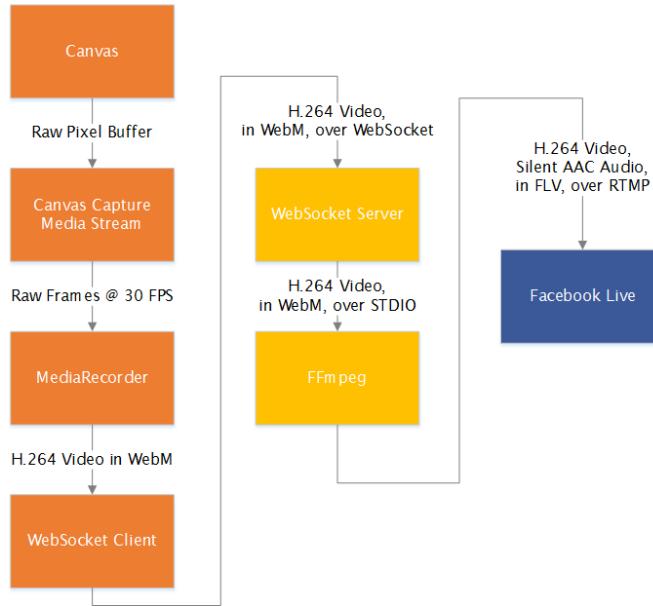


Figure 18: Arquitectura Projecte Canvas Streaming

Això sí, per capturar el canvas, utilitza MediaRecorder, que és el mateix plugin que s'utilitzava en la primera prova. Fent algunes modificacions al projecte de github de prova obtenim resultats amb el html creat anteriorment.

```

// Creacio del servidor websockets
//.....
//Al iniciar servidor
wss.on('connection', (ws, req) => {

```

```

// Ensure that the URL starts with '/rtmp/' , and
// extract the target RTMP URL.
let match;
match = "rtmp://example.com";

const rtmpUrl = "rtmp://example.com";
console.log('Target RTMP URL:', "rtmp://example.
com");

// Launch FFmpeg to handle all appropriate
// transcoding, muxing, and RTMP
const ffmpeg = child_process.spawn('ffmpeg', [
  // Facebook requires an audio track, so we
  // create a silent one here.
  // Remove this line, as well as '-shortest', if
  // you send audio from the browser.
  // '-f', 'lavfi', '-i', 'anullsrc',
  // FFmpeg will read input video from STDIN
  '-r', '30',
  '-i', '-',
  // Because we're using a generated audio source
  // which never ends,
  // specify that we'll stop at end of other input
  // . Remove this line if you
  // send audio from the browser.
  // '-shortest',
  '-f', 'pulse',
  '-i', 'alsa_output.output_name.analog-stereo.
monitor',
  '-async', '1',
  '-ac', '2',
  // If we're encoding H.264 in-browser, we can
  // set the video codec to 'copy'
  // so that we don't waste any CPU and quality
  // with unnecessary transcoding.
  // If the browser doesn't support H.264, set the
  // video codec to 'libx264'
  // or similar to transcode it to H.264 here on

```

```

        the server.
        '-vcodec', 'copy',

        // AAC audio is required for Facebook Live. No
        // browser currently supports
        // encoding AAC, so we must transcode the audio
        // to AAC here on the server.
        '-acodec', 'aac',

        // FLV is the container format used in
        // conjunction with RTMP
        '-f', 'flv',

        // The output RTMP URL.
        // For debugging, you could set this to a
        // filename like 'test.flv', and play
        // the resulting file with VLC.
        rtmpUrl
    ]);

    //FFmpeg error and close functions
    //.....
};

}

```

Listing 11: Servidor Websockets | server.js

Les ventatges amb aquest canvi és que aconseguim molta més qualitat de imatge, i es captura automàticament el so. Per altre banda, es segueix utilitzant Pulseaudio de manera autònoma, pel que seguirà havent un retard variant de video/àudio.

Les conclusions que es treuen d'aquest experiment és que aquesta eina és interessant per fer algun petit projecte o que no requereixi de so, però no acaba d'encaixar en els requisits que es proposaven.

Unreal Engine 4

Per la prova amb Unreal Engine, s'ha utilitzat una tecnologia molt utilitzada de manera professional a la televisió, i que recentment ha estat integrada a Unreal com a plugin extern. Es tracta del software NDI (Network Device Interface) [38]. Creat per la empresa NewTek i amb una llicència privativa, permet una gran qualitat d'imatge, amb poc retard i una estructura molt estable. Dóna un grau important de professionalitat i compta amb un suport per part de NewTek i un fòrum de comunitat. Les desventatges són, que al ser una empresa que recolza el software privatiu i tancat, no té cap mena de suport per Linux, només per Windows i MacOS. Sumant això a que la compatibilitat d'Unreal Engine tampoc és extraordinària, és bastant inabastable aconseguir bons resultats amb

Unreal en una màquina Linux. Per tant, les proves que es faràn seran desde una màquina Windows per comprovar la potència d'aquesta tecnologia.

Per començar, crearem un streaming de NDI desde VLC, per poder comprovar que funciona correctament tant entrada com sortida de fluxos. Un cop installem les Tools de NDI, el plugin de VLC per poder treballar amb aquest software, s'instalarà automàticament. Només cal entrar a les opcions de VLC i marcar com a motor de sortida «NewTek NDI Video Output».

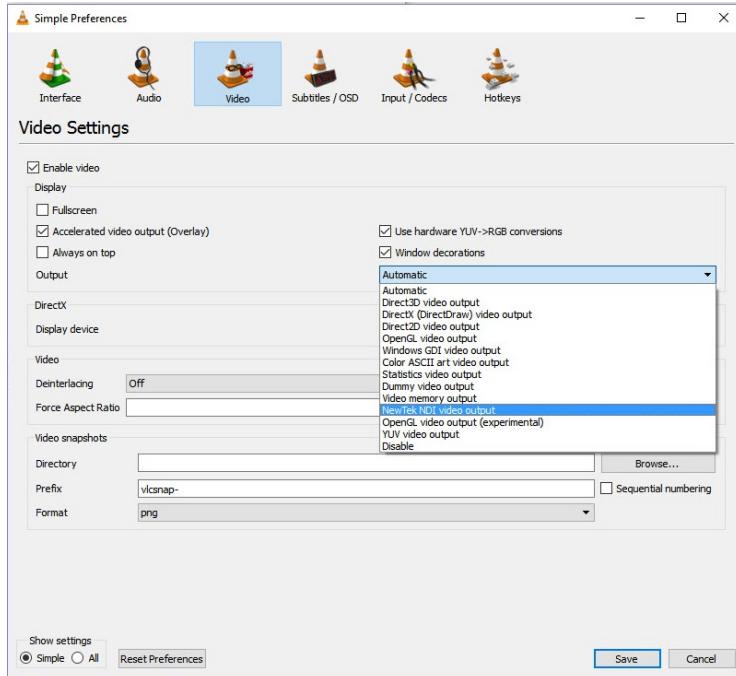


Figure 19: Configuració Sortida NDI a VLC

Encara que per aquesta prova s'hagi fet la sortida amb VLC, es pot fer de moltes altres maneres, com pot ser FFmpeg o una eina propia de NDI dedicada a això.

Un cop ja tenim un flux de video creat, l'haurem de rebre a l'Unreal, per fer-ho, hem d'utilitzar el plugin corresponent, i seguir els passos de la seva documentació [39].



Figure 20: Node NDI Receiver

Ara que ja està tot programat, ja comencem a veure alguns resultats d'aquest software. Per aprofitar l'Unreal i provar altres efectes també, s'ha escollit un video amb un chroma per comprovar com es comportava devant un streaming calculant-lo a temps real.

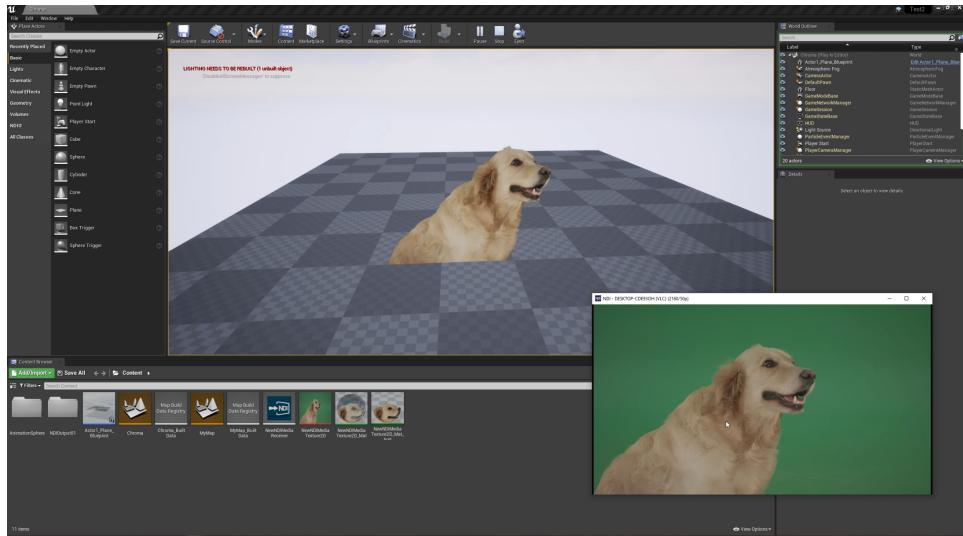


Figure 21: Resultat NDI Input

Els resultats d'aquesta prova han sigut molt bons, la latència del NDI era molt baixa, a més d'una qualitat sorprendent. Ha donat alguns problemes d'estabilitat en algun moment, però s'ha de tenir en compte que el plugin d'Unreal encara és molt nou i és normal que doni algun problema que altre, encara i això, són resultats prou estables i té potencial per poder ser utilitzat com a eina de televisió professional.

Unity

Per la prova del Unity, s'ha creat un projecte que contigui diversos elements. Un vídeo de fons, un element 3D amb un vídeo com a texture, un logo 3D i un PNG amb transparència. Per a la reproducció de vídeos s'ha escollit el plugin UMP (Universal Media Player) [40], considerat el reproductor més complet de tota l'Asset Store (plataforma de descàrrega de plugins).



Figure 22: Funcionalitats UMP

A més, és l'únic compatible amb Linux, el sistema operatiu més utilitzat en sistemes al núvol [41]. Pel UMP, haurem de posar un element que controla la reproductor, i definir quines són les malles que han de renderitzar el video. Per aquesta prova es té un controlador que reproduirà el video del fons, i un altre que reproduirà el video de la càpsula, l'altre element 3D. A més, tindrem una animació pel logo 3D que anirà girant, i un altre per la capsula que s'anirà movent per la pantalla.

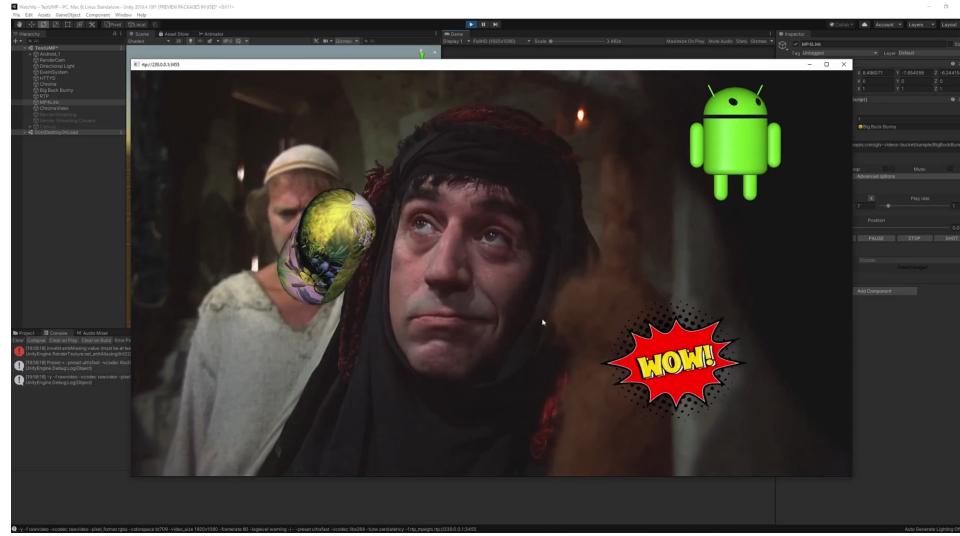


Figure 23: Prova Unity

Els resultats amb Unity han sigut molt satisfactoris, no ha donat cap problema i la reproducció ha sigut la més fluida. El plugin UMP suporta pràcticament tot tipus de formats de vídeos ja que utilitza les llibreries de libVLC. El codi és mitjanament obert pel que es podrà editar fàcilment. La imatge en PNG té la transparència perfectament aplicada, pel que veiem que el Unity suporta els valors d'alpha d'aquest.

Ara que ja s'han fet proves suficients amb totes les eines, es pot decidir sense cap tipus de dubte, que la millor opció per aquest projecte és Unity.

References

- [1] Watchity. (2019b, September 13). ¿Qué es el Control Room (Live Distribution)? Watchity - Help Center. <https://help.watchity.com/hc/es/articles/360017584293--Qu%C3%A9-es-el-Control-Room-Live-Distribution->
- [2] Watchity. (2020, July 8). Share cuts on Social Networks. Watchity - Help Center. <https://help.watchity.com/hc/en-us/articles/360023619094-Share-cuts-on-Social-Networks>
- [3] Watchity. (2019, September 9). ¿Qué es el Mixer? Watchity - Help Center. <https://help.watchity.com/hc/es/articles/360018866273--Qu%C3%A9-es-el-Mixer->
- [4] Pereyro, S. (2018, February 09). Live streaming with headless chrome - Empirical. Empirical. <https://blog.goempirical.com/how-to-use-headless-chrome-to-screencast-audio-and-video-to-an-rtmp-endpoint-216ccfdde4db>
- [5] FreeDesktop. (2021, January 16). PulseAudio. <https://www.freedesktop.org/wiki/Software/PulseAudio/>
- [6] Google. (2021, February 25). Getting Started with Headless Chrome | Web [.]. Google Developers. <https://developers.google.com/web/updates/2017/04/headless-chrome>
- [7] SeleniumHQ. (2018, December 19). SeleniumHQ Browser Automation. Selenium. <https://www.selenium.dev/>
- [8] SeleniumHQ. (2021a, July 7). Driver requirements :: Documentation for Selenium. Documentació Selenium. https://www.selenium.dev/documentation/en/webdriver/driver_requirements/
- [9] Smirnov, A. (2020, November 18). How to run a headless Chrome browser in Selenium WebDriver. Medium. <https://itnext.io/how-to-run-a-headless-chrome-browser-in-selenium-webdriver-c5521bc12bf0>
- [10] Google. (2021a, February 11). Puppeteer | Tools for Web Developers [.]. Google Developers. <https://developers.google.com/web/tools/puppeteer>
- [11] Apache. (2004, January 1). Apache License, Version 2.0. Apache License. <https://www.apache.org/licenses/LICENSE-2.0>
- [12] Mozilla. (2021, May 27). WebGL: 2D and 3D graphics for the web - Web APIs | MDN. WebGL Support by Browser. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API#browser_compatibility

- [13] Dealessandri, M. (2020, May 14). What is the best game engine: is Unreal Engine right for you? GamesIndustry.Biz. <https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unreal-engine-4-the-right-game-engine-for-you>
- [14] Eric Peckham, E. P. (2019, October 17). TechCrunch is now a part of Verizon Media. Extra Crunch. <https://techcrunch.com/2019/10/17/how-unity-built-the-worlds-most-popular-game-engine/>
- [15] Technologies, U. (2021, January 1). Multiplatform. Unity. <https://unity.com/features/multiplatform>
- [16] TIOBE. (2021, July 1). index | TIOBE - The Software Quality Company. <https://www.tiobe.com/tiobe-index/>
- [17] Technologies, U. (2021b, January 1). Unity - Scripting API: MonoBehaviour. Unity. <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>
- [18] Lagarde, S. (2020, February 24). HDRP: Out of Preview in 2019.3. Unity Blog. <https://blog.unity.com/technology/hdrp-out-of-preview-in-2019-3>
- [19] Texto: Luz Fernández. (2002, December 28). La copia de películas se hace fácil en Internet. Cinco Días. https://cincodias.elpais.com/cincodias/2002/12/28/tecnologia/1041322677_850215.html
- [20] Bychok, A. (2021, May 21). Streaming Protocol Comparison: RTMP, WebRTC, FTL, SRT – Restream Blog. Ultimate Live Streaming Hub – Restream Blog. <https://restream.io/blog/streaming-protocols/>
- [21] Fernández, Y. (2019, November 7). Qué es el códec AV1 y cuáles son sus ventajas. Xataka. <https://www.xataka.com/basics/que-codigo-av1-cuales-sus-ventajas>
- [22] Hao, D. T. G. (2019, August 7). Mixer's Faster Than Light streaming protocol explained. Dot Esports. <https://dotesports.com/streaming/news/mixer-faster-than-light-streaming-protocol-explained>
- [23] Rus, C. (2020, June 23). Microsoft cierra Mixer, su alternativa a Twitch en la que tenía en exclusiva a superestrellas de eSports como . . . Xataka. <https://www.xataka.com/videojuegos/microsoft-cierra-mixer-su-alternativa-a-twitch-que-tenia-exclusiva-a-superestrellas-esports-como-ninja>
- [24] Twitch. (2015, December 18). Twitch Blog | Twitch Engineering: An Introduction and Overview. Twitch Blog. <https://blog.twitch.tv/en/2015/12/18/twitch-engineering-an-introduction-and-overview-a23917b71a25/>

- [25] FFmpeg. (2021, January 1). About FFmpeg. <https://ffmpeg.org/about.html>
- [26] FFmpeg. (2021b, January 1). FFmpeg Filters Documentation. <https://ffmpeg.org/ffmpeg-filters.html>
- [27] VideoLAN. (2021, January 1). Free Software and Open Source video streaming solution for every OS! - VideoLAN. <https://www.videolan.org/videolan/>
- [28] VideoLAN. (2021a, January 1). Contribute to the project - VideoLAN. <https://www.videolan.org/contribute.html>
- [29] VideoLAN. (2019, March 4). libVLC Tutorial - VideoLAN Wiki. WikiVLC. https://wiki.videolan.org/LibVLC_Tutorial/
- [30] Pereyro, S. (2017, October 31). Screencast with Headless Chrome. Google Docs. https://docs.google.com/presentation/d/1b-HvxKmeFYE3qoNDJDHfbgVwEbi0QgmTvpH8w4CK5Tw/edit#slide=id.gc6f90357f_0_0
- [31] Postman. (2021, January 1). Postman | The Collaboration Platform for API Development. <https://www.postman.com/>
- [32] Google. (2021a, January 1). Chrome DevTools Protocol. Page Domain. <https://chromedevtools.github.io/devtools-protocol/tot/Page/#method-startScreencast>
- [33] Filmaffinity. (2014, January 1). Nightcrawler (2014). <https://www.filmaffinity.com/es/film779937.html>
- [34] Filmaffinity. (1979, January 1). Stalker (1979). <https://www.filmaffinity.com/es/film534365.html>
- [35] Google. (2017, September 13). Autoplay policy in Chrome. Chrome Developers. <https://developer.chrome.com/blog/autoplay/>
- [36] FFmpeg. (2021c, January 1). FFmpeg Formats Documentation. <https://ffmpeg.org/ffmpeg-formats.html#image2-1>
- [37] FBSSamples. (2020, May 8). GitHub - fbsamples/Canvas-Streaming-Example: This project contains example code showing how to go live on Facebook using a element as a source. GitHub. <https://github.com/fbsamples/Canvas-Streaming-Example>
- [38] NewTek. (2021, January 1). About NDI - Network Device Interface. <https://www.ndi.tv/about-ndi/>
- [39] NewTek. (2020, November). NDI Unreal Documentation. NDI Unreal Documentation. http://514f211588de67e4fdcf-437b8dd50f60b69cf0974b538e50585b.r63.cf1.rackcdn.com/Utilities/SDK/NDI_SDK_Unreal_Engine/4.22.0/Documentation/NDI-Unreal-Documentation.pdf

- [40] Unity Direction Kit. (2016, June 4). UMP (Win, Mac, Linux, WebGL) | Video. Unity Asset Store. <https://assetstore.unity.com/packages/tools/video/ump-win-mac-linux-webgl-49625>
- [41] Verma, A. (2015, August 30). Access denied | fossbytes.com used Cloudflare to restrict access. Fossbytes. <https://fossbytes.com/ubuntu-linux-is-the-most-popular-operating-system-in-cloud/>

4 Images References

- 1 Watchity. (2019, September 9). Frontend Mixer [Graph]. Frontend Mixer. https://help.watchity.com/hc/article_attachments/360027399494/Tipo_Production.png
- 2 SeleniumHQ. (2021, July 7). Selenium Driver Graph [Graph]. Selenium Driver Graph. https://www.selenium.dev/documentation/images/basic_comms.png?width=400px
- 3 pupeteer. (2018, January 12). Puppeteer Logo [Graph]. Logo. <https://user-images.githubusercontent.com/10379601/29446482-04f7036a-841f-11e7-9872-91d1fc2ea683.png>
- 4 WebGL Example. (2017, March 17). [Illustration]. Aquarium. <https://1stwebdesigner.com/wp-content/uploads/2014/03/aquarium.png>
- 5 Quixel. (2019, March 20). Quixel Megascan [Render]. UE4 Demo. <https://www.youtube.com/watch?v=9fC20NWhx4s>
- 6 Spiris, S. (2014, December 25). Blueprint [Graph]. UE4 Simple Blueprint. <https://answers.unrealengine.com/storage/temp/24917-rotate90degrees.png>
- 7 Neginfinity, N. (2013, January 27). Chaos Blueprint [Graph]. UE4 Blueprint Example of Bad Node Programming. <https://forum.unity.com/proxy.php?image=https%3A%2F%2Fpbs.twimg.com>
- 8 Daria, D. (2021, July 12). Ranking Languages 2020 [Graph]. Darly Solutions. <https://darly.solutions/the-most-popular-programming-languages-in-2021/>
- 9 Unity. (2018, January 16). Book of the Dead - Unity Interactive Demo - Teaser. YouTube. https://www.youtube.com/watch?v=DDsRfbfnC_A
- 10 BATTLESTATE GAMES LIMITED. (2015, December 22). Pantalla de Pre-Alfa 19. www.escapefromtarkov.com. <https://www.escapefromtarkov.com/media/id/29?lang=es>
- 11 Aakanksha, A. (2016, May 12). Diagrama Protocol-Format-Códec [Graph]. StreamShark. <https://streamshark.io/blog/understanding-codecs-and-formats/>
- 12 Ruether, T. (2021, March 16). TCP vs UDP [Graph]. Wowza. https://www.wowza.com/wp-content/uploads/Graphic-UDP-Vs-TCP-Diagram_1150x685.webp
- 13 Ruether, T. (2021a, March 16). Graph Latency Streaming Protocols [Graph]. Wowza. <https://www.wowza.com/wp-content/uploads/latency-continuum-2021-with-protocols-700x300-1.webp>
- 14 FFmpeg. (2013, April 10). FFmpeg Logo [Logo]. Wikipedia Media. https://upload.wikimedia.org/wikipedia/commons/thumb/5/5f/FFmpeg_Logo_new.svg/1280px-FFmpeg_Logo_new.svg.png
- 15 Brinkmann, M. (2009, July 7). VLC 1.0 [Photograph]. Ghacks.Net. <https://www.ghacks.net/2009/07/07/vlc-media-player-1-0-released/>
- 16 Pereyro, S. (2017, October 31). Screencast with Headless Chrome. Google Docs. <https://docs.google.com/presentation/d/1b-HvxKmeFYE3qoNDJDHfbgVwEbi0QgmTvpH8w4CK5Tw/>

- 17 Vicente A. (2021, March 9) Test Headless Chrome Index. Own elaboration.
- 19 NewTek. (2016, August 31). NDI VLC Output Config [Config]. NewTek. <https://www.newtek.com/blog/tips/vlc-media-player-and-newtek-ndi-vlc-plugin/>
- 20 NewTek. (2020b, November 20). NDI Receiver Unreal [Graph]. NewTek. <https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRMEVIL0BGn1XzKBxR2VKpZowQagdDYYENP>
- 21 Vicente A. (2021, June 1) Test NDI Receiver. Own elaboration.
- 22 Unity Direction Kit. (2016a, June 4). UMP Functionalities [Graph]. Asset Store. <https://assetstorev1-prd-cdn.unity3d.com/key-image/9f96a930-72d3-4972-a650-f4ea4f6e326b.webp>
- 23 Vicente A. (2021, January 14) Test Unity UMP. Own elaboration.

5 Annexes