CODEMILL

Umeå University

and

Codemill

Master Thesis

Multimedia Processing: Real-Time Colour Grading with JIT using the MLT Framework

Pina Kolling

Supervised by

Dr. Cem Okulmus

and

Urban Söderberg

June 4, 2024

Abstract

The topic of this thesis project is multimedia processing, focusing on the user-sided adjustment of RGB values in video streaming using Just-In-Time (JIT) techniques and the Media Lovin' Toolkit (MLT) framework. This is implemented in Codemill's Accurate Player and using Web Real-Time Communication (WebRTC) as a data channel. Colour theory and RGB colour representation are discussed and technical details on the structure and usage of the MLT framework are provided.
The first part of the research question aims to evaluate the feasibility of the real-time colour adjustment. This research question is answered positively by providing an implementation that can address real-world use cases. A comparison of different MLT filters is included, to select the most suitable filter for the RGB adjustment.
The second part of the research question considers the comparison of video colour grading results with MLT filters that were applied on different platforms: The Accurate Player, the command line video editor Melt and the editing software KDEN Live. For this, frames of the different platforms were extracted and subtracted from each to show differences in the colour saturations. The results reveal that the Accurate Player plays back the original video more accurately than the Melt framework. Additionally, the results lead to the assumption that KDEN Live is not using the same Melt filter as the Accurate Player to adjust the RGB values. Those significant differences in the compared frames show the complexity of the topic of colour adjustment and representation.

Keywords:

▶ Multimedia processing ▶ MLT framework ▶ MLT filter ▶ Colour grading ▶ Just-in-time
▶ RGB colour adjustment ▶ Codemill ▶ Accurate Player ▶ Web Real-Time Communication

Contents

1 Introduction

The popularity of streaming is growing, with an increasing number of people using online streaming platforms for their entertainment and approximately 1.8 billion subscriptions to video streaming services [18, 39]. With this increased popularity, the demand for user-driven real-time visual customisation in the web browser for streamed videos has also increased.

1.1 Motivation

The colouring of a video can evoke different emotions and set an overall tone for the perception of the content. For instance, cool colours such as blue can convey cold weather, while warm colours such as yellow can create a feeling of warmth and sunlight. A comparison of the same photo in different tones can be seen in Figure 1. The colouring of a video is a powerful tool that can significantly impact the conveyed mood and atmosphere [21, 40].

Figure 1: The original photo can be seen in the middle, while the picture on the left has more blue tones and the picture on the right has more yellow tones.

The use-cases for on-the-fly adjustments of the video colour arise from different motivations. One of them is that each screen, ranging from computer monitors, TVs, smartphones and tablets to beamers, has unique characteristics that can influence the perceived colour palette. In addition, the light conditions can influence the appearance of the colours. With real-time colour corrections, the user can adjust the colour and cancel out variations that are caused by the device or environment to have a consistent experience [45].

Additionally, users with visual problems or specific colour perception might benefit from real-time colour grading to enhance the visibility and distinction of the on-screen elements. The increased accessibility can improve inclusivity and enable a more diverse range of users to engage with the content [7].

1.2 Research Questions

This thesis revolves around the implementation of video colour grading with the adjustment of the RGB values using Just-In-Time (JIT) techniques and the Media Lovin' Toolkit (MLT) framework. The MLT framework is a multimedia framework tool that can be used for video editing and playback. It is introduced and described in Section 3.1. JIT is used for on-the-fly conversion of video files and optimising them for streaming. The demand for efficient and dynamic video processing tools is increasing and with this, the ability to perform on-the-fly colour correction becomes increasingly valuable. This project lies in the area of multimedia processing, video editing and real-time computing and is done in cooperation with the company Codemill, which is introduced in Section 1.3.

The aim of this thesis is to explore the feasibility of implementing the video colour grading process with JIT, with the focus on real-time colour grading and to implement such a solution, if possible. This solution should contain three sliders in the frontend to adjust the individual RGB values: Red, green and blue. The colour adjustment should then be applied to the video output in real-time, while streaming the video. The following question is relevant:

> Is it possible to obtain colour-graded video results on the fly, meaning in real-time, using JIT and the MLT framework?

If such an implementation is successful, the effectiveness of the grading process will be evaluated. A second research question is therefore:

> Is there variation in the outcomes of filter applications across various applications with a MLT backend and the specified system?

The different applications for the comparison that use the MLT framework as a backend are the command line interface (CLI) video editor Melt and the video editing software KDEN Live.

By answering the research questions above, this project aims to provide insights into the technological and scientific aspects of real-time video processing, exploring the possibilities and potential applications introduced by using JIT and the MLT framework to implement colour grading. The challenges of this thesis project include the understanding of the complex codebase of the Accurate Player and implementing the RGB adjustment in it, comprehending the sparse MLT documentation and finding a valid method to evaluate and compare the results from different platforms.

1.3 Codemill

Codemill was founded in 2008 in Umeå (Sweden) [5, 6]. As of February 2024, they employ over 60 employees [14]. Codemill is an IT-Consulting company that focusses on the distribution of broadcast media. Their Accurate Video Player that is described in Section 5.2, is being used in this thesis and it is a cloud native software that is being used by the world's leading studios, broadcasters and media service providers [10, 13]. The infrastructure of the Accurate Player and its backend components, which are basis of this thesis, are visualised and explained in Chapter 5.

1.4 Related Work

The topic of this thesis project spans across various fields. Due to this, the related work is dispersed among these fields, highlighting the interdisciplinarity of the topic. The areas that are relevant for this project include colour representation, colour grading, multimedia processing, video streaming, the MLT framework, protocol buffers and the comparison of different colour saturations with GIMP.

RGB representation

This thesis revolves around RGB colour adjustment, making the understanding of RGB colour representation an important aspect. Additionally, the results of different methods for the RGB colour adjustment are analysed with the *grain extract* function in GIMP, making it necessary to understand the RGB colour model to understand the functionality and results. One source discussing RGB colour representation and the concepts of additive and subtractive colour models is *Color: An Introduction to Practice and Principles* by Rolf G. Kuehni [31].

GIMP

The book *The Book of GIMP: A Complete Guide to Nearly Everything* by Olivier Lecarme and Karine Delvare describes nearly every function in GNU Image Manipulation Program (GIMP) [32]. This includes the *grain extract* function that is relevant for the MLT filter comparison between different platforms in this thesis project. Furthermore, understanding techniques for evaluating and editing media content is important within the realm of multimedia processing.

Colour grading

To describe the terminology and differences between colour grading and colour correction according to the industry standards, a variety of web pages and articles were consulted. They are used to reflect the usage of the terminology in the field. One web page is titled *Understanding What Color Correction And*

Color Grading Are – And Aren't from Vegas. Vegas (former Sony Vegas) is a video editing program from the company Magix, which is widely known as a prominent company within the video editing field [50]. Other web pages were referenced to confirm the usage of the terms colour grading and colour correction in the industry as described to ensure that the terminology aligns with the commonly accepted standards within the field. Furthermore, the book *The Art and Technique of Digital Color Correction* by Steve Hullfish discusses the topic of colour correction and video post-production in more depth [25].

Effects of different colour

The paper *Color and psychological functioning: a review of theoretical and empirical work* from Andrew J. Elliot discusses theoretical insights and empirical findings on the psychological effect of colours [19]. Furthermore, in *Effects of color on emotions* from Patricia Valdez and Albert Mehrabian, emotional reactions to colours are investigated and attributes are assigned to specific colours that they are associated with [48]. Understanding the effects of different colours is important for the motivation of this thesis and for meeting the users needs and requirements, which is relevant for the implementation in this thesis project.

MLT framework

The MLT framework is a crucial part for this thesis project. The RGB colour adjustment is implemented using MLT as a backend and the results are compared with other applications using a MLT backend. For this, information directly from the MLT website itself was consulted [36]. Especially the list of different MLT filters and plug-ins was considered, to compare the filters and select the most suitable option for the RGB adjustment. Additionally, the code of the MLT framework was examined [37, 38]. The external literature on the MLT framework and its technical aspects is unfortunately limited.

Protocol buffer

The Chapter *Proto Buffer* in the book *Mobile Forensics – The File Format Handbook: Common File Formats and File Systems Used in Mobile Devices* by Chris Currier introduces and discusses the topic of protocol buffers [16]. This is relevant for the data transfer from the frontend to the MLT framework in the implementation for this thesis project.

Multimedia processing

The topic of multimedia processing is widely spread and has many different areas. This resulted in many papers, standards and documentations being used as sources to be combined into the overall context of this thesis project. Individual references have been consulted for various components including WebRTC, JIT and video transcoding, which are relevant for this thesis project.

1.5 Structure

This thesis consists of seven Chapters. The topics and contents of those Chapters are listed in the following. The introduction (Chapter 1) provides an overview of the thesis, including its motivation, research questions, related work and the structure. Chapter 2 contains the theoretical foundation of colour, including the differences between colour correction and colour grading and the effects that distinct colours can have on the viewer. Furthermore, the representation of colours using RGB and the comparison of different colour saturations is described. In Chapter 3 the technical background of the project is explained, including an in depth description of the structure and usage of the MLT framework. Additionally, an overview of video streaming components and additional components such as protocol buffers is given. Chapter 4 outlines the requirements and specifications for the system to be able to execute the code. Furthermore it is described how to execute the code. The design and implementation of the on-the-fly RGB adjustment are described in Chapter 5. The architecture of the system is introduced, describing the frontend and backend of the system: Accurate Player and JIT-WebRTC. Additionally, different MLT filters for RGB adjustment are tested and compared. At the end of the Chapter the implementation of the RGB adjustment is described. Chapter 6 presents the comparison of video colour grading results with MLT filters that were applied on different platforms. The comparison is between the Accurate Player RGB adjustment, that was implemented in this thesis project, the command line video editing tool Melt and the editing software KDEN Live. The conclusion in Chapter 7 summarises the results, contributions and limitations of the thesis project and introduces future work opportunities.

2 Theoretical Foundation of Colour

In this Chapter, the terms colour correction and colour grading are introduced and the effects of different colours in media and their association are explained. Additionally, the RGB colour model is presented and a method for the comparison of different colour saturations using GIMP is shown.

2.1 Colour Correction and Colour Grading

Colour correction and colour grading are both processes used in film making, photography and digital art to adjust and enhance the colour of images or videos. While those two processes are related, they serve different purposes and are typically performed at different stages of production. Colour correction aims to correct technical imperfections and achieve a natural-looking image, while colour grading focuses on enhancing the visual aesthetics and storytelling aspects [35, 50].

Colour correction is the process of adjusting the overall colour balance, contrast and exposure of an image or video to achieve a more accurate representation of the scene as it was captured by the camera. It involves correcting potential technical issues such as white balance, exposure inconsistencies and colour inaccuracies caused by the camera or lighting conditions. This is typically done in the early stages of post-production to ensure that the footage looks natural and consistent across different shots and scenes [25, 35, 50].

Colour grading, on the other hand, is the creative process of altering the colour and mood of an image or video to evoke a specific aesthetic or emotional response. The effects of different colours are described in Section 2.2. Colour grading involves making stylistic choices to enhance the visual storytelling or the overall cinematic look or perception of the footage. It can involve manipulating individual colours, adjusting contrast, saturation and various other image manipulation techniques. Colour grading is often performed after colour correction and is considered a more artistic and subjective process. It allows film makers and photographers to establish a unique visual style and enhance the narrative impact of their work [35, 50].

In summary, colour correction and colour grading serve different purposes and are performed at different stages of post-production. Colour correction ensures technical accuracy and consistency, while colour grading adds artistic components and enhances the visual storytelling and perception of the footage. Regarding the definition of these two processes, the RGB adjustment in this thesis project is classified as colour grading [25, 35, 50].

2.2 Effects of Different Colours

Different colours can convey a different understanding of a scene to the viewer or even influence the viewers emotions. Different colours have distinct psychological associations, which influence how viewers perceive and interpret the content [19, 48].

In different visual media, including photos and videos, the strategic use of colour can effectively communicate themes, evoke emotions and guide viewer interpretation [19, 35, 48, 50].

In the following, different colours with their psychological associations are listed and example pictures are shown.

Red

Figure 2: Picture of a Amanita muscaria with alarming red colour.

The colour red is often associated with intense emotions such as passion, excitement and danger. Viewing it in clothing on self or others has even been shown to increase the perception of aggressiveness and dominance [34]. When wearing the colour in sport competitions and events, red has been shown to enhance performance and the perceived performance [19, 24].
In visual media, the use of red can evoke feelings of urgency, power and love. For example a video with flashes of red light may create a feeling of tension or alarm while a female wearing red can increase the perceived attraction [19, 20].
Beyond its symbolism in human emotions and cultural contexts, red also serves as a prominent warning colour in nature. One example of this is the toxic fly agaric mushroom (Amanita muscaria), which can be seen in Figure 2. The instinctual reaction to red is used in various contexts, including traffic signals and emergency alarms, where the colour serves as a clear and universally understood signal [19].

Blue

The colour blue is often linked to feelings of calmness and stability. In videos and pictures, the presence of blue tones can create a sense of relaxation. For instance, the photo in Figure 3 of a lake scene in Sweden with shades of blue in the water and the sky can evoke a peaceful mood.
Additionally, blue stores and logos have been shown to increase the perception of expected quality and trustworthiness [48, 52].
But this is not the only association with the colour blue. While it might evoke feelings of calmness or trustworthiness in certain contexts, it can also represent sadness or coldness depending on the situation [48].

Figure 3: Picture of a lake with mainly blue tones.

Green

Figure 4: Picture of a forest with mainly green tones.

Green is associated with nature, growth and harmony. In visual media, the use of green can symbolise renewal, freshness and balance. Green is a colour that often appears in nature, because it is the result of chlorophyll, a substance in plants that helps them to turn sunlight into energy [29, 48].
For example, the picture in Figure 4 of a forest in Sweden can evoke feelings of vitality and may convey a sense of natural beauty and harmony, because the plants in this picture show a lot of green tones.
Additionally, green is often used in environmental campaigns and eco-friendly branding to signify sustainability and a connection to the earth [3].

Yellow

Yellow is often associated with happiness, optimism and energy. When used in videos or pictures, yellow can evoke feelings of warmth, cheerfulness and positivity. For instance, a photograph capturing a bright yellow sunrise as seen in Figure 5 can convey a sense of warmth, hope and optimism.

Figure 5: Picture of a sunrise with mainly yellow tones.

In media editing, the colour yellow can be increased to convey the impression of sunshine or a sunny day to the viewer.

Moreover, yellow is used in the realm of marketing and advertising. Brands often apply the colour yellow to evoke positive associations with their products, because of its cheerful and optimistic connotations. It also plays a role in interior design: A room painted in soft yellow tones can convey feelings of warmth, making it an ideal choice for homes [48].

Black and white

Figure 6: Picture of skyscrapers in Dubai in greyscales.

Black and white or greyscales are often used to evoke a sense of timelessness or simplicity. In visual media, the absence of colour can draw attention to shapes, textures and contrasts. For example, a black and white photograph of a city skyline can emphasise the architectural details and create a dramatic atmosphere or, depending on the context, it may evoke a sense of nostalgia. In Figure 6, houses in Dubai can be seen. The greyscale filter of this photo enhances the focus of the viewer on the architecture and the structures of the buildings. Furthermore, black, white and greyscale are often used to convey elegance and simplicity [48].

2.3 RGB Colour Representation

RGB colour representation is a model that attempts to mathematically represent colours as perceived by human vision. RGB stands for Red, Green and Blue. Each of those colour channels (red, green and blue) has a value ranging from 0 to 255. When all three colours are at their maximum value with `(255,255,255)`, the outcome is white. On the contrary, when all three colour channels are at their minimum value with `(0,0,0)`, the result is black.

The RGB colour model combines three colours (red, green and blue) with different values to create the broad spectrum of colours. In Standard Dynamic Range (SDR), each of the red, green, and blue values uses 8 bits. This results in 16 777 216 possible colours [31]. There are commercial video formats that also support High Dynamic Range (HDR) colour spaces, which can use 10 or even 12 bits per colour channel [17].

As an example, the colour red can be created with mixing full red (255 in the red channel), no green (0 in the green channel) and no blue (0 in the blue channel). This leads to a pure red as `(255,0,0)`. This red can be adjusted by varying the values in the other colour channels. By decreasing the red value, tones such as a dark maroon with `(130,0,0)` can be achieved and with a higher red and green value a vibrant crimson with `(200,50,10)` can be achieved. On the other hand, by increasing blue while maintaining a high red channel, tones including scarlet `(255,40,0)` or coral `(255,120,80)` can be created. In addition to this, introducing equal amounts of green and blue alongside the high red value can transition the colour into a softer tone, such as rose `(255,150,150)`. The described colours can be seen in Figure 7.

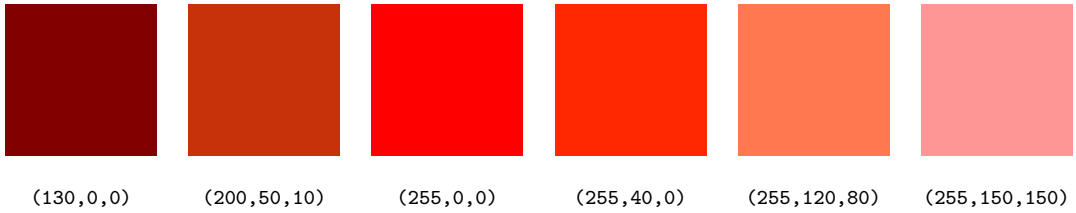


Figure 7: Different shades of red represented by different RGB values.

Each pixel on a standard LCD display consists of three sub-pixels that each displays one of the colours red, green or blue. By adjusting the intensity of each sub-pixel, a wide range of colours can be achieved. The RGB value of a pixel indicates the amount of each colour. This means if for example the red value is set to 0, the red sub-pixel is turned off. When the red value is set to 255, the red sub-pixel is turned fully on [8, 31].

The RGB colour model is an additive colour model, which is the opposite to subtractive colour models that apply to paints and other substances where the colour reflects certain components of the light. In the subtractive model, a paint filters out all colours but its own and two blended colours filter out all colours but the common colour between them. In the additive model, the colour has a brightness that is the sum of the brightness of the two mixed colours. Because of this, mixed colours appear brighter than individual colours [31].

A colour diagram that shows the additive model of RGB can be seen in Figure 8 and the subtractive model can be seen in Figure 9.

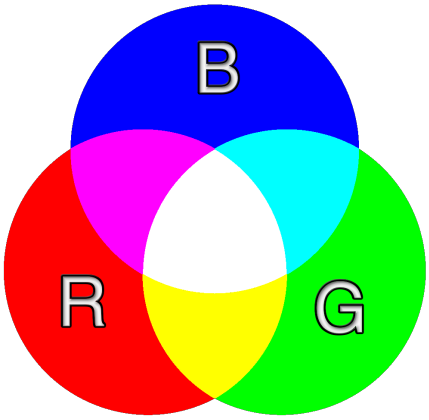


Figure 8: Three coloured circles in red (R), green (G) and blue (B) showing the function of the additive colour model of RGB.

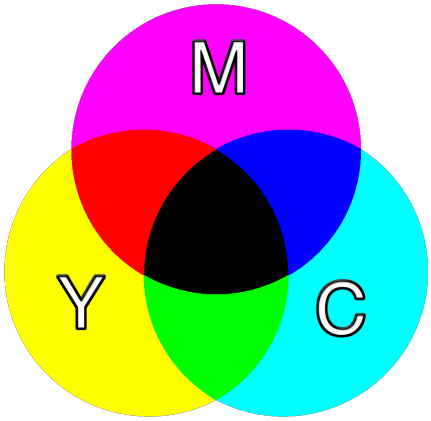


Figure 9: Three coloured circles in magenta (M), yellow (Y) and cyan (C) showing the function of the subtractive colour model.

2.4 Comparison of Different Colour Saturations with GIMP

In this Section, a method for analysing the differences between two images is explained, where the colours of the images are subtracted from each other by using the image manipulation program GIMP. This method in GIMP is called *grain extract* and it subtracts the colour values of corresponding pixels from two images to show any differences. It is a visual representation of the colour subtraction process, where subtracting identical pixels from each other results in a grey image, indicating no differences [32].

When subtracting one image from another, the RGB pixel values from one image are subtracted from the corresponding pixel values in the other image and an offset is added. The result reflects the relative differences in the colour values between the two subtracted images. For example with the subtraction of two red toned images,

areas where the subtracted image has a less intense red tone will result into red pixels after the subtraction and analogically, areas where the subtracted image has a more intense red tone will result into green or blue pixels in the result of the subtraction. The result of the subtracted pixels depends on the difference between the two colour values.

Two examples of the colour subtraction with different red values can be seen in Figure 10. In the Figure, the red colours `RGB(202, 95, 95)` and `RGB(255,0,0)` are subtracted from each other with the *grain extract* function. This results in a blue colour (`RGB(75, 223, 223)`) and a different red tone (`RGB(181, 33, 33)`).

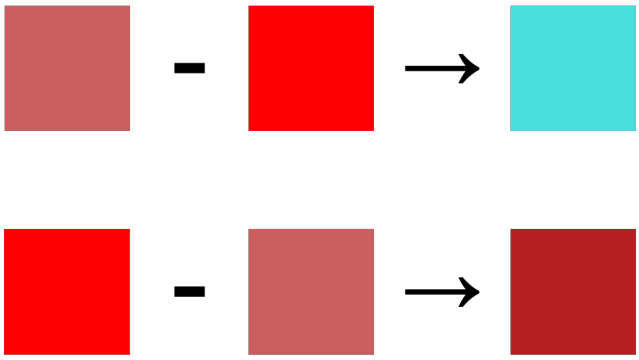


Figure 10: Subtraction of two different red values from each other with the GIMP function *grain extract* to visualise the differences.

The calculation to retrieve the new RGB values is defined and explained in the following. Let g be the *grain extract* function in GIMP that adds 128 to two subtracted RGB values. The input values as well as the result are ranging from 0 and 255. The function is defined as follows:

$$g : [0, 255] \times [0, 255] \to [0, 255]$$
$$g(x, y) = (x - y + 128) \bmod 255$$

Let $\texttt{RGB}_1$ and $\texttt{RGB}_2$ be the RGB values of a pixel in the same position in two different images that are being subtracted.

$$\texttt{RGB}_1 = (R_1, G_1, B_1) \qquad \texttt{RGB}_2 = (R_2, G_2, B_2)$$

The function g can be applied to the RGB value pairs to retrieve the new values of the colours after the subtraction.

$$\begin{aligned}
\texttt{RGB}_1 - \texttt{RGB}_2 &\Rightarrow (g(R_1, R_2), g(G_1, G_2), g(B_1, B_2)) \\
&\Leftrightarrow (((R_1 - R_2 + 128) \bmod 255), \\
&\quad ((G_1 - G_2 + 128) \bmod 255), \\
&\quad ((B_1 - B_2 + 128) \bmod 255))
\end{aligned}$$

For the examples that are shown in Figure 10, the calculations are presented in the following, with the RGB values $\texttt{RGB}_1 = (202, 95, 95)$ and $\texttt{RGB}_2 = (255, 0, 0)$.

$$\begin{aligned}
\texttt{RGB}_1 - \texttt{RGB}_2 &\Rightarrow (g(202, 255), g(95, 0), g(95, 0)) \\
&\Leftrightarrow (((202 - 255 + 128) \bmod 255), \\
&\quad ((95 - 0 + 128) \bmod 255), \\
&\quad ((95 - 0 + 128) \bmod 255)) \\
&\Leftrightarrow (75, 223, 223)
\end{aligned}$$

$$\begin{aligned}
\texttt{RGB}_2 - \texttt{RGB}_1 &\Rightarrow (g(255, 202), g(0, 95), g(0, 95)) \\
&\Leftrightarrow (((255 - 202 + 128) \bmod 255), \\
&\quad ((0 - 95 + 128) \bmod 255), \\
&\quad ((0 - 95 + 128) \bmod 255)) \\
&\Leftrightarrow (181, 33, 33)
\end{aligned}$$

The results are the light blue colour `RGB(75, 223, 223)` and the red tone `RGB(181, 33, 33)`, which can be seen in Figure 10.

3 Technical Background

In this Chapter, the structure and usage of the MLT framework is explained and a brief overview of video streaming components and additional components is given. Furthermore, protocol buffers are introduced and explained with examples.

3.1 MLT

The MLT framework is a multimedia framework that can be used for video editing and playback. It is written in C and provides a set of tools, libraries and services for handling multimedia content, including video and audio. Melt is the name of the command line tool, which exposes various functionality of the framework and was developed for testing the MLT framework [36].

A variety of multimedia tasks can be performed with MLT, including video editing, rendering, the application of filters on audio or video files and converting data between different formats. The areas of applications include film production, television broadcasting and digital content creation. Additionally, the MLT framework supports a wide range of media formats and codecs.

Structure of the MLT framework

In the following, the relevant components of the MLT framework's structure will be explained. This is based on the documentation as well as the code of the framework [36, 38]. *Producer*, *Consumer* and *MLT frame objects* are fundamental components in the MLT framework.

A MLT frame object is a data structure that is used for representing multimedia content. It is a container that holds multimedia data. Each MLT frame object contains a single uncompressed frame image and its associated audio samples. These frame objects can be processed individually, which allows the user to manipulate and transform multimedia content efficiently. MLT frame objects can store the multimedia data in various formats and they support metadata annotations, which allows to attach additional information or properties to the multimedia content. This includes filters for manipulating the content, which are discussed in Section 5.4.

A producer generates or provides data – it produces the MLT frame objects. It reads data from files or other data sources. The producer serves as the starting point of the data processing pipeline. It retrieves the raw data and passes it on to other components, for example the consumer. This can be seen in Figure 11.

A consumer requests MLT frame objects from the producer. It is responsible for consuming or processing the data that was produced by the producer or other components that are between the producer and consumer. The consumer represents the endpoint of the data processing pipeline and produces the final output or results.

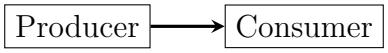


Figure 11: Producer-Consumer Relationship in the MLT framework.

Applying filters in the MLT framework is a technique for manipulating multimedia content, including videos. MLT filters can perform a wide range of visual and audio operations, including colour correction, blurring, sharpening, cropping and resizing. A list of the filters can be found on the MLT website and example applications of a wide selection of visual filters can be seen in Appendix C [37].

To apply a filter in Melt, a filter instance has to be created. The filter instance has to contain the name of the filter and, if applicable, the filter parameters with corresponding values. Filters are processing units that manipulate the multimedia data as it flows through the data processing pipeline. They are placed between the producer and consumer, allowing them to modify the data before it reaches the consumer. A MLT data processing pipeline containing a filter is shown in Figure 12.

Figure 12: Producer-Filter-Consumer Relationship in the MLT framework.

Usage of the MLT framework

In this Section, the usage of the Melt framework is explained with code examples. The function of each code segment is explained. This is based on the implemented code for this project, the code of the MLT framework and the MLT documentation [36, 38].

The code structure described below refers to the creation of the necessary MLT components to apply a filter.

Initialise the factory

This command initialises the MLT factory, to set up the environment for creating and managing multimedia objects.

```
mlt_factory_init(const char * directory);
```

The function takes a directory path as an optional parameter. In this directory, custom modules can be stored, which can then override the default settings.

Create a profile

In this line, a multimedia profile is created to define the video properties.

```
mlt_profile profile = mlt_service_profile(mlt_service self);
```

The function `mlt_service_profile()` retrieves the profile that is associated with a given service. It takes a service as a parameter and returns its profile.

Create a producer

In this command, a producer object is created. The producer object generates or provides the data.

```
mlt_producer producer = mlt_factory_producer(mlt_profile profile,
                                             const char* service,
                                             const void* resource);
```

The function `mlt_factory_producer()` creates a producer based on the profile, service and resources, which are the parameters of this function.

Create properties

This line retrieves the properties associated with a specified producer parameter within a property object.

```
mlt_properties properties = MLT_PRODUCER_PROPERTIES(producer);
```

The macro `MLT_PRODUCER_PROPERTIES` internally accesses the properties of the producer.

Create a consumer

This line creates a consumer object that processes the data.

```
mlt_consumer consumer = mlt_factory_consumer(mlt_profile profile,
                                             const char* service,
                                             const void* input);
```

The function `mlt_factory_consumer()` creates a consumer object based on the provided parameter profile, service and optional input. If the service object is not specified, it defaults to `MLT_CONSUMER`.

Create a filter

In this command a filter is created. The name of the filter is given as an argument in form of a string.

```
mlt_filter filter = mlt_factory_filter(mlt_profile profile,
```

```
                                                const char* service,
                                                const void* input);
```

The `mlt_factory_filter()` function retrieves a filter based on the parameters. Those are the profile, the name of the filter as service and an optional argument of the filter.

Create and set filter properties

This code creates a property object and sets the filter properties.

```
mlt_properties filter_properties = mlt_filter_properties(filter);
```

The function `mlt_filter_properties()` retrieves the properties of a filter. The filter is given as a parameter.

```
mlt_properties_set(mlt_properties self,
                        const char* name,
                        const char* value);
```

The function `mlt_properties_set()` assigns a new value to a property, in this case to a filter. The parameters are the filter properties that were retrieved in the previous step, and the filter parameters and associated values as strings.

Connect elements

The filter is connected to the producer and the consumer to the filter, if a filter is applied.

```
mlt_filter_connect(mlt_filter self, mlt_service producer, int index);
```

The function `mlt_filter_connect` takes the filter and the producer as parameters and connects them.

```
mlt_consumer_connect(mlt_consumer self, mlt_service producer);
```

The function `mlt_filter_connect` takes the consumer and the filter as parameters and connects them.

If no filter is applied, the producer is being connected to the consumer directly.

```
mlt_consumer_connect(mlt_consumer self, mlt_service producer);
```

The function `mlt_filter_connect` takes the consumer and the producer as parameters and connects them.

Start consumer

This line starts the consumer.

```
mlt_consumer_start(mlt_consumer self);
```

The function `mlt_consumer_start()` initiates the consumer in the MLT framework. It takes the consumer object as a parameter.

Close the components

The following functions are responsible for closing the associated component within the MLT framework. These functions mark the end of a MLT process by closing and cleaning up resources.

```
mlt_consumer_close(mlt_consumer self);
mlt_producer_close(mlt_producer self);
mlt_profile_close(mlt_profile profile);
mlt_factory_close();
```

3.2 Overview of Video Streaming Components

In this Section, an overview of video streaming components that are relevant for this thesis project is given.

Transcoding is the conversion of one digital data format into another. One form of video transcoding is the compression of videos with the codec h.264: The video compression codec h.264 can be used to compress and decompress video streams in the context of real-time communication [2, 51]. File formats such as Audio Video Interleave (AVI) often serve as input files for video transcoding processes. AVI can contain audio and video information to allow synchronised playback of audio and video components. It is a container format, which means that it can contain multiple streams of audio and video data, along with other multimedia data such as subtitles. AVI files can undergo transcoding to optimise their compression, compatibility or quality, with h.264 being a commonly used codec for such purposes [33, 51].

WebRTC, is a free, open-source project for real-time communication between web browsers and other applications that was developed by Google in 2011. It enables direct peer-to-peer (P2P) communication without the need for intermediary servers. WebRTC supports various video codecs, including h.264, which is used for video conferencing, streaming and other real-time applications. This means that the video streams exchanged between the browser and the backend are encoded and decoded using h.264 [47, 51].

Representational State Transfer (REST) is an architecture for network applications. A REST Application Programming Interface (API) exposes a set of endpoints, which are Uniform Resource Locators (URLs) that serve as addresses for communication

between different software systems over the internet using hypertext transfer protocol (HTTP) methods [4, 26, 43].

JIT or Just-In-Time in the context of video streaming refers to a dynamic software solution employed for real-time video transformation. It is used for on-the-fly conversion of video files, optimising them for streaming. The process includes the transcoding of a video file, potentially one with non-web-friendly formats, into a more suitable format for the playback in web browsers. JIT operates by dynamically preprocessing video content as it is requested by users. This is in contrast to the common technique of pre-converting video files into formats that are suitable for streaming or browsers in advance. This approach allows for greater flexibility and efficiency, as it minimises the need for storage space and allows real-time adjustment such as the adjustment of the RGB colours that is implemented and described in this thesis project. Additionally, JIT allows to adjust video resolution, bitrate and other parameters in real-time to adapt the results to the user's bandwidth [44].

3.3 Overview of Additional Components

This Section introduces additional components that are relevant for this thesis project.

Docker is a platform that provides an isolated environment where applications can run. It enables the usage of these isolated environments that are called containers. Those Docker containers can reduce the problems that arise from environmental discrepancies. Docker enables testing in local environments while enhancing the reliability and comparability [27].

Node.js is a JavaScript (JS) runtime environment that enables the execution of JavaScript code outside of a web browser. It enables developers to use JavaScript both on the client and server sides of web applications. Node.js is a Free and Open-Source Software (FOSS) and the community contributes to its evolution. This community-driven development model has led to a big selection of modules and libraries [9, 11, 22]. To manage project dependencies and packages for Node.js, package managers such as Yarn or Node Package Manager (npm) can be used. These package managers allow developers to install, update and manage libraries and tools used in projects by providing a centralised repository of libraries and tools [9, 28].

Web services are software systems for communication between different applications over a network. They allow data exchange and function execution across platforms and programming languages. Java-based web services are commonly used in modern

software development, due to the robustness and scalability of the Java platform for building distributed systems over the web. Web services enable machine-to-machine (M2M) interaction – the communication and data exchange between devices or systems [43].

KDEN Live is a video editing software that offers a wide range of features such as cutting, trimming and arranging video clips and adding effects, transitions and titles to videos. It also supports various file formats. KDEN Live is freely available, open-source and uses the MLT framework as its backend [30].

3.4 Protocol Buffer

Protocol buffers are a binary data format developed by Google to compress data for sending it efficiently over a network. The structure of this format is defined in a `.proto` file. The `.proto` files are used to generate code for reading and writing to the protocol buffer [16].

In this project, protocol buffers version 2 (proto2) were used. In the first line of the `.proto` file, the syntax is defined with the proto version that is being used. As an example, the syntax for proto2 is shown below.

```
syntax = "proto2";
```

Messages and Fields

Data structures are defined using message declarations. Messages can contain fields for the data, each with a unique identifier and a specified data type. The data types for proto2 are listed in Table 1 [23].

In the following, an example message is shown and described. The example message represents a person's information, including their name, age and email address. This is defined as follows:

```
message Person {
  required string name = 1;
  required int32 age = 2;
  optional string email = 3;
}
```

Person is the label of the message and name, age and email are the fields of the message with the data types `string` and `int32`. The keywords `required` and `optional` define whether the field is mandatory or optional. The numbers 1, 2 and 3 are the fields' unique identifiers.

Data Type	Description
double	Double-precision floating-point number
float	Single-precision floating-point number
int32	Signed 32-bit integer
int64	Signed 64-bit integer
uint32	Unsigned 32-bit integer
uint64	Unsigned 64-bit integer
sint32	Signed 32-bit integer (suitable for negative numbers)
sint64	Signed 64-bit integer (suitable for negative numbers)
fixed32	Unsigned 32-bit integer (four bytes)
fixed64	Unsigned 64-bit integer (eight bytes)
sfixed32	Signed 32-bit integer (four bytes)
sfixed64	Signed 64-bit integer (eight bytes)
bool	Boolean value (true or false)
string	Variable-length text string
bytes	Variable-length sequence of bytes

Table 1: Data types for fields in protocol buffers (proto2) [23].

Enums

Proto2 supports the definition of enumeration types as enum. Enums allow for the declaration of a set of named constants, each associated with a unique integer value. This can be used for representing a fixed set of options or states within a message definition [16, 23].

In the following, an example enum is shown and described. It represents the gender of a person and is defined as follows:

```
enum Gender {
  UNKNOWN = 0;
  MALE = 1;
  FEMALE = 2;
  DIVERSE = 3;
}
```

Gender is the name of the enum and `UNKNOWN`, `MALE`, `FEMALE` and `DIVERSE` are the enum constants with the associated integer values 0, 1, 2 and 3. It allows to represent the gender options unknown, male, female or diverse, providing a way to represent a fixed set of options or states.

4 System Requirements and Specifications

In this Chapter, the technical requirements for the system are described. These requirements provide the necessary standards to ensure the system's functionality. Furthermore, the execution of the code is described.

4.1 Technical Requirements

Regarding the hardware requirements, the system should be capable of running Node.js, which is necessary for executing the frontend development tool Yarn. Additionally, the system should be able to execute the Python backend and MLT framework within a Docker container. While implementing, it was observed that the system should have a minimum of 16 Gigabyte (GB) of Random Access Memory (RAM) to ensure smooth execution of the development environment and Docker containers.

The software requirements include information about the operating system, applications and frameworks. While testing, the code encountered difficulties running on Windows. The code should be operational on Windows after adjusting the configuration. For compatibility reasons and the ease of use via the command-line interface, the code for this thesis project was executed on a Linux system (Ubuntu 22.04). Regarding the documentation in the README files, the Mac operating system should be able to run the code after making adjustments, but this was not tested in the scope of this thesis [12].

In addition to this, the system relies on several software components. Docker is needed to use Docker containers for encapsulating the system's backend. This allows consistent and efficient results across various devices. Node.js serves as the runtime environment for executing JavaScript code for web applications and Yarn is the package manager for Node.js applications that is being used in this project. Additionally, Git is essential for version control. While it is not strictly required for the system's execution, Git is used to pull Git repositories containing the code and necessary dependencies and frameworks, such as the MLT framework [46]. An installation of Melt is not needed for the execution or development of this system, but can be useful for testing or exploring the functionalities of the MLT framework. Furthermore, the system adheres to standard security practices, including using the secure communication protocol HTTP.

4.2 Execution of the Code

In this Section, the execution of the code is described and information regarding the test video file are given.

The codebase for this project consists of two directories: `Accurate-player-3-core` and `jit-webrtc`. The system's architecture is described in Section 5. Additionally, the READMEs of the two main components of the system can be seen in Appendix A. The files for the frontend code are in a directory that is called `accurate-player-3-core`. For the execution of the frontend, it is necessary to navigate to the directory `demo` within the `packages` folder, followed by executing the `yarn start` command with the backend parameter in the command line interface (CLI):

```
~/accurate-player-3-core/packages/demo$ JIT_BACKEND=http://localhost:8080 yarn
   start
```

For updating dependencies during development, the following command should be executed and running in addition:

```
~/accurate-player-3-core/packages/jit$ yarn start
```

To run the backend, the folder `jit-webrtc` needs to be accessed in the CLI to start the script `main.sh`, which starts the backend in a docker container and pulls the needed dependencies:

```
~/jit-webrtc$ docker/main/main.sh -v --threads 16 --port 8080 https://s3.eu-
  central-1.amazonaws.com/accurate-player-demo-assets/timecode/sintel-2048-
  timecode-stereo.mp4
```

The optional parameter `-v` (verbose) provides additional information or detailed output when executing the code. The threads and the port are set with the parameters `--threads` and `--port` and then the link to the test video file is added. This is the file that will then be played in the Accurate Player.

The test video file that is being used for this thesis project, is the animated short film *Sintel*, which was during its development known as *The Durian Open Movie Project*. It was created by the Blender Foundation using Blender, which is an open-source 3D animation software and released in 2010. It tells the story of a young girl, who is searching for her lost dragon companion. The film is allowed to be used for free distribution, sharing and adaptation, as long as the following credit is given to the original creators. This licensing approach promotes collaborations and progress of the FOSS community.

This copyright attribution is applied to the entire use of frames of the film in this thesis. It will not be added to each extracted frame. The *Sintel* video file that is being used in this thesis project is retrieved from the following source:

https://s3.eu-central-1.amazonaws.com/accurate-player-demo-assets/timecode/sintel-2048-timecode-stereo.mp4

5 Design and Implementation

In this Chapter, the system architecture and the implementation are explained. First a brief overview of the architecture is given in Section 5.1 and then the frontend and backend are described in more detail (Section 5.2 and Section 5.3). For the application of a MLT filter, different filters were compared to select the most suitable option in Section 5.4. Finally, the implementation is described with examples and code excerpts in Section 5.5.

5.1 Architecture Design

The system consists of four components: The frontend that is displayed in the browser, `main.py` as the Python backend, the MLT framework and a session service. The frontend is described in Section 5.2 and the backend components (`main.py`, the MLT framework and session service) will be described in more detail in Section 5.3. The structure and usage of MLT was introduced in Section 3.1.

The system architecture can be seen in Figure 13. A session is initiated through the browser with the session service, which initiates the `main.py` script. The Python backend communicates with the MLT framework to use its video editing, processing and transcoding functionalities. The real-time communication is implemented through a WebRTC channel.

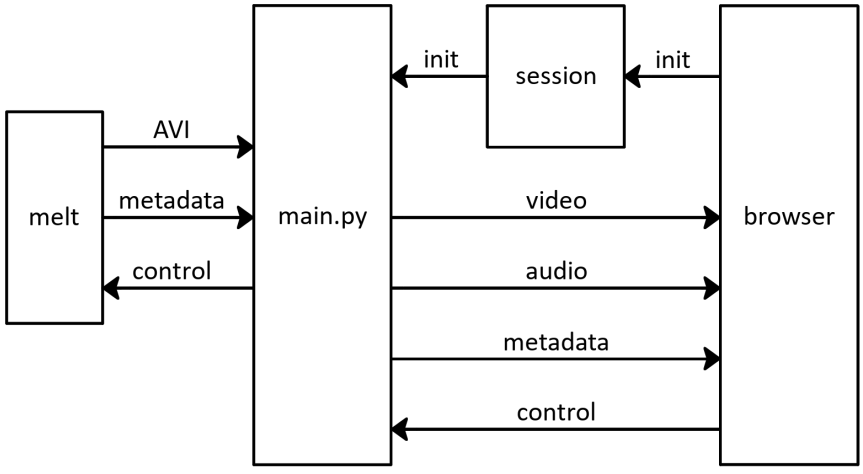


Figure 13: Architecture of the system that consists of four parts.

For development and testing without the session service, the backend can be started in a Docker container. This has been done for the implementation of this thesis project.

5.2 Accurate Video

The following description of the system is based on the code base and the information from the `README` file [9]. The frontend is displayed in the browser (highlightes in Figure 14) and uses Node.js as its runtime environment with Yarn to manage project dependencies and packages. It is developed using HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript to create the user interface (UI).

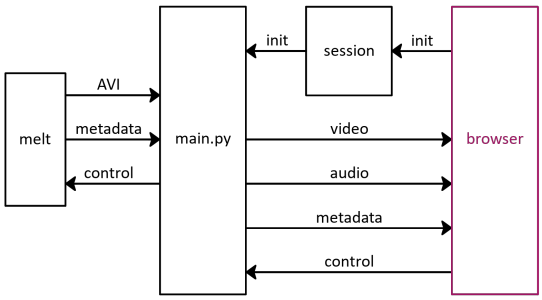


Figure 14: Accurate Video in the system architecture.

The user interface when playing a video in the Accurate Player can be seen in Figure 15. This can be found under `http://localhost:5000/controls/jit/index.html` when executing the code as described in Section 4.2. First, a yellow field can be seen (shown on the left in Figure 15). After clicking the field, a loading screen is displayed (in the middle in Figure 15) and then the video can be started by clicking on the play button. The playback of a video can be seen on the right in Figure 15.

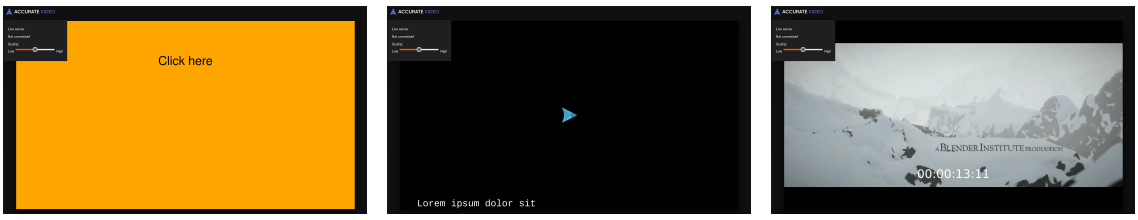


Figure 15: Screenshots of the Accurate Player when playing a video.

In addition to this, the user interface of the Accurate Player has multiple control fields, which can be seen in Figure 16, 17 and 18 and will be further described in the following.

Permanently fixed in the top left corner of the Accurate Player is a window that contains a slider to adjust the quality of the video in real time. Changes to the slider apply to both a currently playing video and a frame of a paused video. This can be seen in Figure 16. In this project, the sliders for the RGB colour grading will be added in this window, too. This is described in Section 5.5.

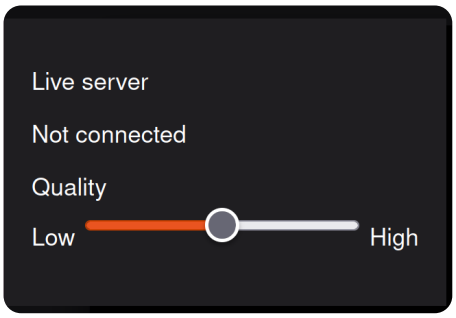


Figure 16: Quality slider in the frontend.

In the top right corner of the Accurate Player window that is displayed in the browser, two icons to expand further menus can be found. This includes a menu for subtitles and audio settings, which is divided into two parts for those settings. This control window can be seen in Figure 17. It enables the audio functionalities for muting the audio or changing to a different audio track, which can be used for example if a video is available in different languages to switch between the languages. These options can be seen in Figure 17 on the right part of the control menu. The left side of the window allows for enabling and disabling specific subtitles from a list of available subtitles.

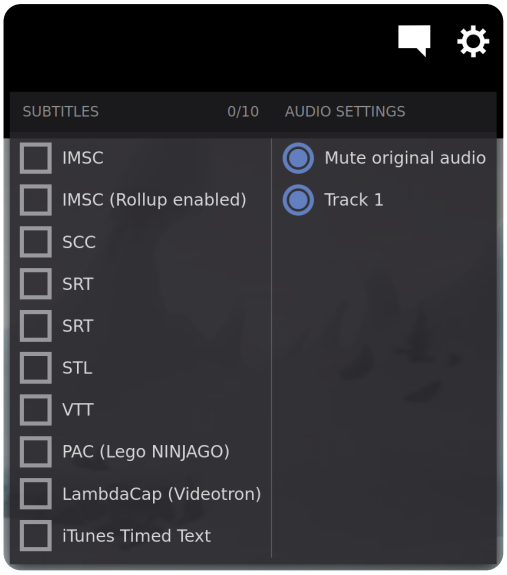


Figure 17: Subtitle and audio control field in the frontend.

Additionally, a menu with further settings can be found when expanding the menus. This menu can be seen in Figure 18 and expands when clicking on the *settings* symbol. Here, further settings, including the time format, the setting of keyboard shortcuts or the fading of the control options when playing the video can be adjusted. All of the above described menus for settings enhance the level of personalisation available to the user. The customisation options allow the user to adapt the Accurate Player to their preferences and expectations.

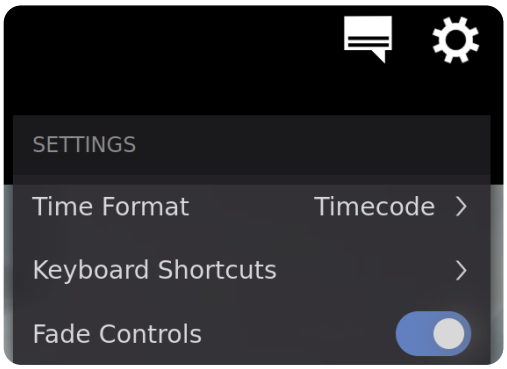


Figure 18: Window for settings in the frontend.

With JIT techniques in the Accurate Player it is possible to play different formats in the browser without the transcoding into a browser friendly format. This is possible, because the Accurate Player uses JIT techniques to dynamically handle the playback of different video formats on-the-fly, instead of relying on pre-transcoded video files. The video data is being processed in real-time, allowing for on-the-fly adjustment of options such as the quality slider or the RGB colour adjustment that is being implemented for this thesis project.

5.3 JIT-WebRTC

The following description of the system's backend is based on the code base and the information from the `README` file [12]. JIT-WebRTC is a live transcoder with a WebRTC output. As described in Section 3.2, transcoding is the conversion of one digital data format into another [2]. The backend consists of three components: The MLT framework, `main.py` and a session service.

The user initiates a session through the browser, which then initiates the `main.py` script. The system's frontend that is displayed in the browser was described in Section 5.2. Control commands are then sent from the browser to the Python backend with a WebRTC data channel. From the `main.py` file, the control commands get sent to MLT via `stdout` (standard output) and `stdin` (standard input). These are the default output and input channels that allow the Python script communication with other components. Data written to `stdout` by the Python code can be captured or

redirected and data from an external source can be read from `stdin` by the Python script [49]. The data flow of the control commands is highlighted in Figure 19.

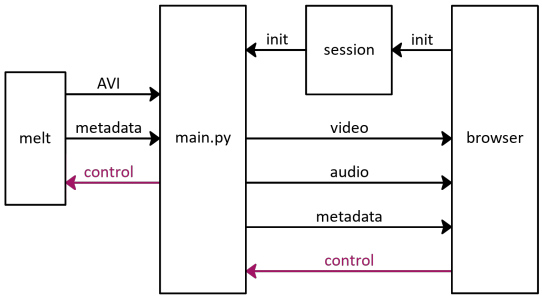


Figure 19: Control commands in the system architecture.

As a response, the MLT framework sends an AVI stream with the rendered video and audio, status messages and metadata to the Python backend. This is highlighted in Figure 20. AVI is a file format that can contain audio and video information to allow synchronised playback of audio and video components, as described in Section 3.2 [33].

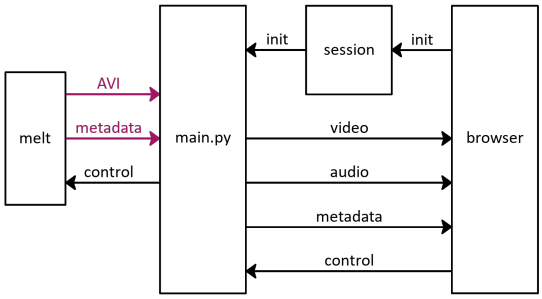


Figure 20: AVI stream in the system architecture.

The audio and video from the incoming AVI stream is extracted by the `main.py` script to encode it for sending it to the browser with WebRTC. The WebRTC connection serves as a direct communication channel between the browser and the backend, enabling real-time data exchange, involving audio, video or other data streams.

The JavaScript client of the browser receives the video and audio to play it in the Accurate Player, which was described in Section 5.2. In addition to this, the browser receives metadata from the `main.py` script and sends commands and responses over the control channel. This connection is highlighted in Figure 21.

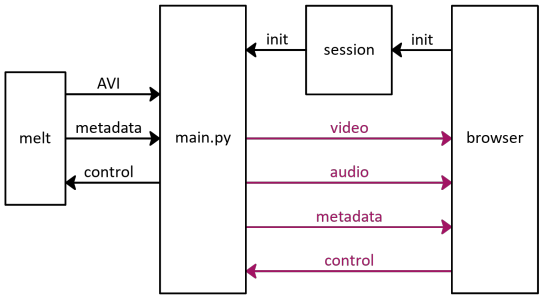


Figure 21: WebRTC connection and control commands in the system architecture.

The MLT framework and the `main.py` script are started once for each JIT session. The Java-based `Session` service is designed to support the machine-to-machine interaction between the browser and the Python backend. The `Session` service exposes a REST API (described in Section 3.2) for initiating a JIT session. For development and testing without the session service, JIT can be started in a Docker container. This has been done for the implementation in this project.

5.4 MLT Filter Comparison

The goal for this thesis project is the on-the-fly adjustment of the RGB values of videos with sliders in the frontend. For this purpose, various MLT filters are examined and compared to identify the most suitable one for implementing this functionality. RGB colour representation was described in Section 2.3 and the MLT framework was described in Section 3.1.

Preselected for further investigation are filters from the MLT web page that contain the word *color* or *RGB* in the filter name [37]. In Table 2, a selection of filters with their name, description and a first evaluation for the suitability for the RGB adjustment are listed. Filters that are not suitable for the RGB adjustment are marked with the word *Rejected* and a short reasoning for the rejection. This does only refer to the use case of RGB adjustment with individual parameters for adjusting the RGB values. Three of the listed filters in the table are considered suitable for further analysis and comparison.

Name	Description	Suitability
`avfilter.colorbalance`	Adjust the colour balance	Suitable for further investigation
`avfilter.colorchannelmixer`	Adjust colours by mixing colour channels	Suitable for further investigation
`avfilter.colorcontrast`	Adjust colour contrast between RGB components	Rejected: Only contrast can be adjusted, not the RGB values individually
`avfilter.colorcorrect`	Adjust colour white balance selectively for blacks and whites.	Rejected: RGB not adjustable
`avfilter.colorhold`	Turns a certain colour range into gray	Rejected: RGB not completely adjustable
`avfilter.colorize`	Overlay a solid colour on the video stream	Rejected: Input as hue, saturation and lightness instead of RGB values
`avfilter.colorkey`	Turns a certain colour into transparency	Rejected: RGB not completely adjustable
`avfilter.colorlevels`	Adjust the colour levels with black point and white point	Rejected: RGB not adjustable
`avfilter.colormatrix`	Convert colour matrix	Rejected: Input not as RGB values possible
`avfilter.colorspace`	Convert between colour spaces	Rejected: RGB not adjustable
`avfilter.colortemperature`	Adjust colour temperature	Rejected: RGB not completely adjustable
`frei0r.coloradj_RGB`	Simple colour adjustment	Suitable for further investigation
`frei0r.colorize`	Colourizes image to selected hue, saturation, and lightness	Rejected: Input as hue, saturation and lightness instead of RGB values

Table 2: List of different MLT filters that influence the colour of a video with their name, description and suitability for RGB adjustment [37].

In Table 3, three filters that are considered as suitable for further investigation from the table above are listed. This includes the filter name, description and possible parameters that are listed on the MLT website [37]. Only parameters relevant to this thesis are listed. Then, those filters were applied to compare the visual results of applying them. Other MLT filters that change the visual appearance of the video can be seen in Appendix C.

Name	Description	Parameters
`avfilter.colorbalance`	Adjust the colour balance	`av.rs`: set red shadows `av.gs`: set green shadows `av.bs`: set blue shadows `av.rm`: set red midtones `av.gm`: set green midtones `av.bm`: set blue midtones `av.rh`: set red highlights `av.gh`: set green highlights `av.bh`: set blue highlights
`avfilter.colorchannelmixer`	Adjust colours by mixing colour channels	`av.rr`: set red gain for red channel `av.rg`: set green gain for red channel `av.rb`: set blue gain for red channel `av.ra`: set alpha gain for red channel `av.gr`: set red gain for green channel `av.gg`: set green gain for green channel `av.gb`: set blue gain for green channel `av.ga`: set alpha gain for green channel `av.br`: set red gain for blue channel `av.bg`: set green gain for blue channel `av.bb`: set blue gain for blue channel `av.ba`: set red gain for alpha channel `av.ar`: set red gain for alpha channel `av.ag`: set green gain for alpha channel `av.ab`: set blue gain for alpha channel `av.aa`: set alpha gain for alpha channel
`frei0r.coloradj_RGB`	Simple colour adjustment	`R`: amount of red `G`: amount of green `B`: amount of blue

Table 3: List of preselected MLT filters that influence the colour of a video with their name, description and parameters [37].

To analyse the listed filters and their parameters two tests are conducted. In a first test, the parameter that adjusts the red tones is set to its maximum value and the filter is applied to a sample frame of the video. A second test evaluates whether several parameters of the same filter can be applied simultaneously. For this evaluation, the filters with their different parameters for green and red are applied to the same frame of the test video file. Regarding the RGB colour model, this should result in a yellow toned output frame. The functionality of the RGB colour model was explained in Section 2.3. In the evaluation of the comparison, the adjustment of all three colour channels (red, green and blue) will be presented for completeness.

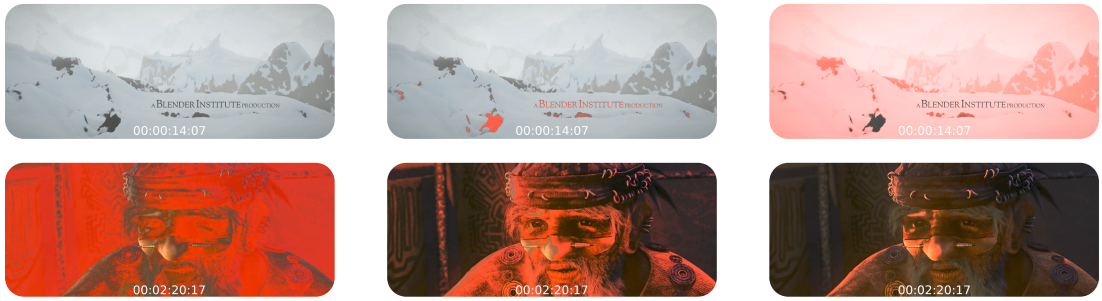
For the purpose of evaluating and comparing, two frames were selected from the video file of the short film *Sintel*. Each of those frames has a distinct colour scheme. One frame shows a snow landscape and the *Blender Institute Production* text, while the other frame shows an elderly man in a warm lighted indoor setting. The original frames 343 and 3377 are shown in Figure 22 for reference.

Figure 22: Frames 343 on the left and 3377 on the right, extracted from the test video file *Sintel* without a filter.

In the following, the filter application and parameters of each filter are introduced and described using the example of the adjustment of the colours red and yellow.

Filter `avfilter.colorbalance`

The parameters of the filter `avfilter.colorbalance` take input values as a float between −1 and 1. The filter has individual parameters to set the shadows, midtones and highlights per colour (red, green and blue). For comparison, each parameter for the red gain was set to its maximum value. The results of adjusting the red shadows with `av.rs=1`, the red midtones with `av.rm=1` and the red highlights with `av.rh=1` are shown in Figure 23.



`av.rs=1` `av.rm=1` `av.rh=1`

Figure 23: Different parameters from the filter `avfilter.colorbalance` applied: Left with `av.rs=1` for red shadows, middle with `av.rm=1` for red midtones and right with `av.rh=1` for red highlights.

To achieve an even colouring result of the frames, which would be applicable for a colour change with one slider per colour, the parameters for the shadows, midtones and highlights can be adjusted simultaneously when adjusting one colour. In Figure 24, the shadows, midtones and highlights for the red value are set to 1.

Figure 24: All parameters for the adjustment of the red tones with the value 1 from the filter `avfilter.colorbalance` applied: `av.rs=1`, `av.rm=1` and `av.rh=1`.

To assess the effects of the combined colour adjustment of multiple colours, the values for green and red (shadows, midtones and highlights) are set to the maximum of 1 and applied simultaneously. The parameters that are applied for this evaluation are `av.rs=1`, `av.rm=1`, `av.rh=1`, `av.gs=1`, `av.gm=1` and `av.gh=1`. This results in yellow toned pictures that can be seen in Figure 25, which shows that the combination of different RGB values into mixed colours is possible with the filter `avfilter.colorbalance`.

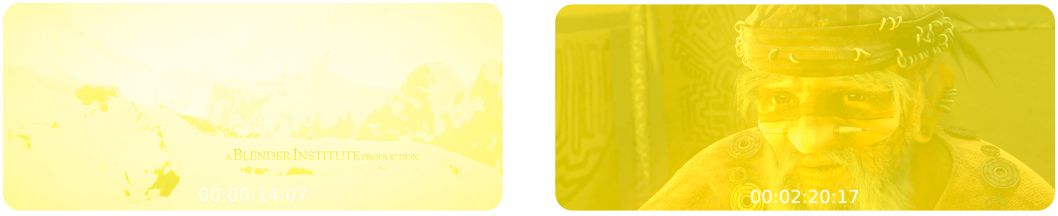


Figure 25: All parameters for red and green with the value 1 from the filter `avfilter.colorbalance` applied: `av.rs=1`, `av.rm=1`, `av.rh=1`, `av.gs=1`, `av.gm=1` and `av.gh=1`.

Filter `avfilter.colorchannelmixer`

The parameters of the filter `avfilter.colorchannelmixer` take input values as floats between -2 and 2. The filter has individual parameters to set the red, green, blue or alpha gain for the red, green, blue or alpha channel. The application of the maximum value of 2 for the red gain on the red channel, alpha gain on the red channel and red gain on the alpha channel can be seen in Figure 26. Only the left frames with the parameter `av.rr=2` for the maximum red gain on the red channel show a relevant result for the RGB adjustment, because the other frames do not show a visible increase in red tones or a red tint.

`av.rr=2` `av.ra=2` `av.ar=2`

Figure 26: Different parameters from the filter `avfilter.colorchannelmixer` applied: Left with `av.rr=2` for red gain on the red channel, middle with `av.ra=2` for alpha gain on the red channel and right with `av.ar=2` for red gain on the alpha channel.

For further investigation of the filter `avfilter.colorchannelmixer`, in Figure 27 the red gain on the red, blue and green channel is applied simultaneously. The parameter settings for this are `av.rr=2`, `av.gr=2` and `av.br=2`. This is not suitable for the RGB adjustment, because the red tones in those frames are not increased.

Figure 27: Red gain with the value 2 for the red, green and blue channel from the filter `avfilter.colorchannelmixer` applied: `av.rr=2`, `av.gr=2` and `av.br=2`.

To assess the effects of the combined colour adjustment of multiple colours for the `avfilter.colorchannelmixer` filter, the values for the green gain on the green channel and the red gain on the red channel are set to the maximum of 2 and applied simultaneously. The parameters that are applied are `av.rr=2` and `av.gg=2`. This results in yellow toned pictures that can be seen in Figure 28.

Figure 28: Red gain on red channel and green gain on green channel with the value 2 from the filter `avfilter.colorchannelmixer` applied: `av.rr=2` and `av.gg=2`.

Filter `frei0r.coloradj_RGB`

The parameters of the filter `frei0r.coloradj_RGB` take input values as floats between 0 and 1. The filter has individual parameters to set the red, green and blue value for RGB adjustment. The application of the red parameter with the maximum value of 1 can be seen in Figure 29.

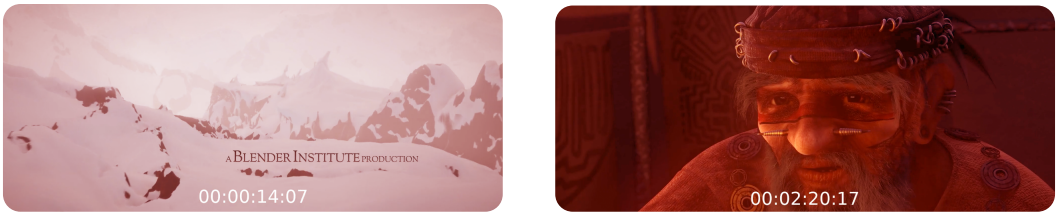


Figure 29: Red parameter with the value 1 from the filter `frei0r.coloradj_RGB` applied: `R=1`.

To assess the effects of the combined colour adjustment of multiple colours for the `frei0r.coloradj_RGB` filter, the values for the green and red parameter are set to the maximum value 1 and applied simultaneously. The applied parameters are `R=1` and `G=1`. This results in yellow toned pictures, which can be seen in Figure 30.

Figure 30: Red and green parameter with the value 1 from the filter `frei0r.coloradj_RGB` applied: `R=1` and `G=1`.

Evaluation

The MLT filters `avfilter.colorbalance`, `avfilter.colorchannelmixer` and `frei0r.coloradj_RGB` are compared regarding the colour vibrance and intensity of the result. This analysis of the filters above was conducted regarding the colours red and yellow in depth and a side-by-side comparison can be seen in Figure 31 and 34. The colour changes of the other two colour channels green and blue work analogically and are displayed in Figure 32 and 33.

Figure 31: Comparison of red filter option of all three filters.

Figure 32: Comparison of green filter option of all three filters.

Figure 33: Comparison of blue filter option of all three filters.

In the Figures above, it can be seen that the red, green and blue tones of the frames with the filter `avfilter.colorbalance` applied is the most intense and vibrant. This makes the filter `avfilter.colorbalance` a suitable choice regarding the adjustment of a single colour channel, because a broader range in the colour intensity leads to more options for personalisation.

In Figure 34, the results of the application of two mixed colours for each of the filters can be seen as a side-by-side comparison. The chosen colours are red and green, which leads to a yellow result.

Figure 34: Comparison of red filter option of all three filters.

It can be seen that the colour of the `avfilter.colorbalance` is the most intense and vibrant again, which confirms the results of the previous comparison regarding the single colour adjustment of red, green and blue. This makes the filter `avfilter.colorbalance` a suitable choice for the implementation of the RGB colour

adjustment with sliders in the frontend, because it leads to a broader colour range and more options for personalisation. The filter `avfilter.colorbalance` shows the most intense colour results for single colour adjustments as well as for mixed colours.

In the above presented tests, which are visualised in the middle in Figure 31, 32, 33 and 34, the results of the `avfilter.colorchannelmixer` filter appear to have more depth and look more dynamic, because the colour gain was only executed on the respective colour channels. This can be an aspect that lets the user achieve more aesthetically pleasing results but the overall intensity of the colours is lower. In addition to this, the expected behaviour of sliders for RGB colour adjustment might not be to only change the colour intensity on one of the respective colour channel. Based on this, the filter `avfilter.colorbalance` seems to deliver the most fitting results in those criteria.

The results of the filter `frei0r.coloradj_RGB` that can be seen on the right in Figure 31, 32, 33 and 34 seem to have darker and less vibrant results that do not present the expected colours as accurately as the other two filters. These filters were therefore excluded from further consideration.

In conclusion, the `avfilter.colorbalance` filter leads to the most vibrant and intense results of the above listed selection of filters. Additionally it has the option to adapt the shadows, midtones and highlights individually, which creates the option for more pre-adjustments of the results in the backend or for introducing more sliders or options for the user-sided adjustment as future work. But on the downside, to tone the whole frame evenly, three instead of one parameter have to be changed and applied – for the shadows, midtones and highlights for each colour slider. When applying this with and without JIT, no performance difference was noticeable but for an accurate evaluation, performance tests would have to be performed and the implementations of the filters would need to be examined, which exceeds the scope of this project. A deeper examination and evaluation of the different options of other MLT filters is an interesting option for future work or for Codemill to implement. For this thesis project, the filter `avfilter.colorbalance` will be used, because its results look the most vibrant and fitting for the expected outcome of RGB colour adjustment.

In Figure 35, the seven basic colour combinations of the subtractive colour model can be seen with the application of the filter `avfilter.colorbalance`. For this, the RGB values were set to either 1.0 or 0, which are the maximum and default values of this filter.

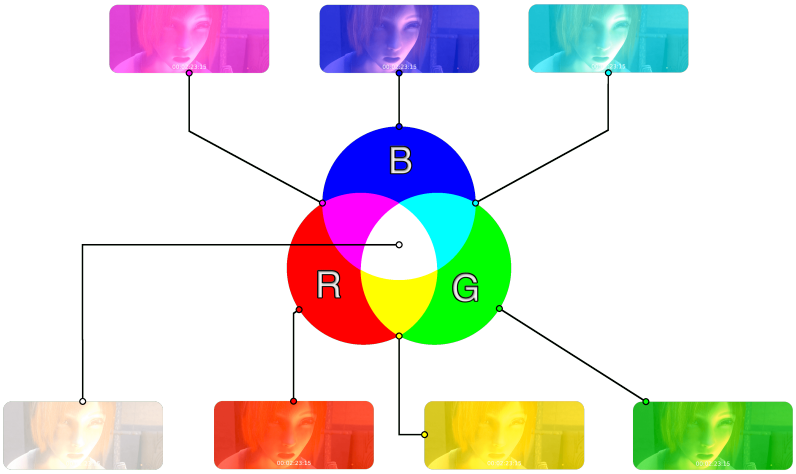


Figure 35: Basic colours of the subtractive colour model with the application of the filter `avfilter.colorbalance`.

For comparison, the original frame 3447 from the test video file can be seen in Figure 36 without applied filters.

Figure 36: Frame 3447 of the test video file without a filter.

5.5 Implementation

In this Section, the implementation is described. This is separated into three parts, that are described individually: The frontend, the protocol buffer and the application of the filter in the backend. For the implementation, the sliders for each colour (red, green and blue) are added in the frontend. The input value of those sliders are sent with a protocol buffer to the backend, where the values are used to set the parameters of the MLT filter `avfilter.colorbalance` before applying the filter. This results in the user-sided RGB adjustment of the video colouring, which is played back in the frontend.

Frontend

In the frontend, input sliders for the user-sided adjustment of the RGB values are added below the already existing slider for the quality adjustment. This is located in the overlay windows in the top left corner. The sliders are coloured according to their associated functions in red, green and blue. This can be seen in Figure 37.

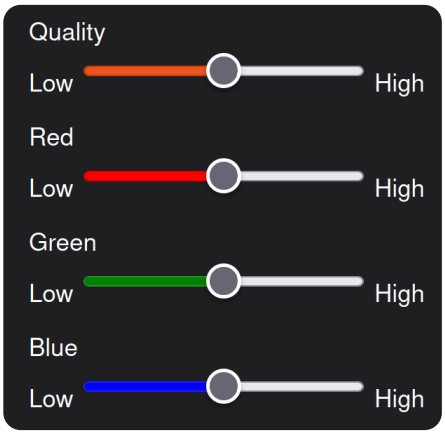


Figure 37: Sliders for RGB colour adjustment.

The HTML code for the red slider can be seen below. The other sliders for green and blue work analogically. The input range is set between -100 and 100, to divide it later by 100 to achieve a float value between -1 and 1 for the filter parameter. The default value is set to 0. The input value from the slider then gets read out in a TypeScript file.

```
<p>
  <label for="red-slider">Red</label><br />
  Low
  <input
    type="range"
    min="-100"
    max="100"
    value="0"
    class="slider"
    id="red-slider"
  />
  High
</p>
```

The design of the sliders is defined in a CSS file: The `class` attribute of the slider uses the design setting of the pre-existing quality slider and the `id` with the value `red-slider` sets the background colour of the slider to red.

Protocol Buffer

For the data transfer between frontend and backend for the colour grading, fields and enums are added to the `.proto` file. Protocol buffers are explained in Section 3.4. The values that are retrieved by the previously described sliders for the RGB value are transferred with this data structure. In the code, the message `JitControl` and the enum `ControlType` can be seen. In the message, three optional fields for the red, green and blue value are defined as `red_value`, `green_value` and `blue_value` to represent the RGB colour values. In the enum, the three constants (`RED_VALUE`, `GREEN_VALUE` and `BLUE_VALUE`) are added. Through the sliders in the frontend, users can manipulate these values for red, green and blue. The input values of those sliders are read out and assigned to the enum `JITAction` in the frontend. They are then transmitted to the backend with the proto2 data format. The application of the MLT filters with the retrieved values is described in the following part.

```
syntax = "proto2";

message JitControl {
  ...
  optional float red_value = 4;
  optional float green_value = 5;
  optional float blue_value = 6;
}

enum ControlType {
  ...
  RED_VALUE = 7;
  GREEN_VALUE = 8;
  BLUE_VALUE = 9;
}
```

Application of the MLT filter

The application of the MLT filter is implemented in the `jit.c` file. One characteristic in the application of the MLT filter `avfilter.colorbalance` is the overwriting of not specifically applied parameters with the default value of zero. The overwriting has to be considered when applying colours that have a non-zero RGB value for more than one colour channel. This refers to the achieving of mixed colour results, for example yellow, which can be created by adjusting the red and the green colour channel. To avoid the overwriting and being able to apply a colour that consist of two or more non-zero values in the RGB channels, the previous values for each colour channel need to be saved in the backend. Then they can be applied together with the recently changed value to prevent the overwriting of the already adjusted colours with the default value. For

```
float red_value;
float green_value;
float blue_value;
```

the implementation, global variables are initialized for each of the colours and one of them are set to the current value in each call, while the other two store the previous values for the other colours. This can be seen in the code snippet above and prevents the overwriting of the previously adjusted values with the default value of zero.

The application of the MLT filter with the current input values from the sliders is implemented in the function `jit_action()` that has a MLT producer as a parameter. In the code below, the beginning of the function can be seen. The MLT properties, consumer and producer are retrieved and set. Then the filter `avfilter.colorbalance` is set. The comparison of the different filter options can be seen in Section 5.4.

```
mlt_properties properties = MLT_PRODUCER_PROPERTIES(producer);
mlt_consumer consumer = mlt_properties_get_data(properties, "transport_consumer",
    NULL);
...
mlt_producer service = mlt_producer_service(producer);
mlt_filter filter = mlt_factory_filter(service, "avfilter.colorbalance", NULL);
```

In the following code excerpt, a pointer `jit_control` is declared, which casts a value into the type `JitControl`. Then, a switch statement is started, based on the value of `jit_control`. The `->` operator is used to access the type field of the `jit_control` structure and depending on the value, the execution will jump to the corresponding case within the switch statement.

```
JitControl *const jit_control = (JitControl*) value;
switch (jit_control->type) { ...
```

Three cases are implemented: One for each colour value (red, green, blue). In the following, the switch case for the adjustment of the red value is explained in more detail. The other cases work analogically.

In the code part below, the case for the red value can be seen. First, the red value that is retrieved from the slider in the frontend is assigned to the variable `red_value`. Then, an `mlt_properties` object is created for setting the filter parameters later on.

```
case CONTROL_TYPE__RED_VALUE:
  red_value = jit_control->red_value;
  {
    mlt_properties filter_props_red = mlt_filter_properties(filter);
```

In the first if statement of this case, the red value is added to the filter properties. For this, the value needs to be in the input range of the filter parameter, which has to be a float between -1 and 1. Then, a string containing the value is created, because the

function `mlt_properties_set` requires the value for the filter parameter in String format. After this, the filter properties are set to this value. Three parameters per colour have to be set, because the chosen filter has the possibility of adjusting the shadows, midtones and highlights individually. The comparison of the different filters is explained in Section 5.4 and the described code is shown below.

```
  if (red_value >= -1.0 && red_value <= 1.0) {
    char red_value_str[20];
    snprintf(red_value_str, sizeof(red_value_str), "%.2f", red_value);
    mlt_properties_set(filter_props_red, "av.rs", red_value_str);
    mlt_properties_set(filter_props_red, "av.rm", red_value_str);
    mlt_properties_set(filter_props_red, "av.rh", red_value_str);
  }
```

The previous values that are set on the sliders for the green and red values are saved in the variables that are declared outside of the function. In the code below, the values are set analogically to the above described red value. This is necessary to not overwrite the previously adjusted colour values.

```
  if (green_value >= -1.0 && green_value <= 1.0) {
    char green_value_str[20];
    snprintf(green_value_str, sizeof(green_value_str), "%.2f", green_value);
    mlt_properties_set(filter_props_red, "av.gs", green_value_str);
    mlt_properties_set(filter_props_red, "av.gm", green_value_str);
    mlt_properties_set(filter_props_red, "av.gh", green_value_str);
  }
  if (blue_value >= -1.0 && blue_value <= 1.0) {
    char blue_value_str[20];
    snprintf(blue_value_str, sizeof(blue_value_str), "%.2f", blue_value);
    mlt_properties_set(filter_props_red, "av.bs", blue_value_str);
    mlt_properties_set(filter_props_red, "av.bm", blue_value_str);
    mlt_properties_set(filter_props_red, "av.bh", blue_value_str);
  }
}
```

In the following code snippet, the filter and the producer get connected after which the consumer and the filter get connected aswell.

```
mlt_filter_connect(filter, service, 0);
mlt_consumer_connect(consumer, mlt_filter_service(filter));
break;
```

The above presented and described code is executed when the slider for the red value is changed in the frontend and the red value is applied to the video output in real time with the MLT framework.

The implementation of the switch case for the application of the filter for the colour adjustment of green and blue can be seen in the code below:

```
case CONTROL_TYPE__GREEN_VALUE:
green_value = jit_control->green_value;
{
  mlt_properties filter_props_green = mlt_filter_properties(filter);

  if (red_value >= -1.0 && red_value <= 1.0) {
    char red_value_str[20];
    snprintf(red_value_str, sizeof(red_value_str), "%.2f", red_value);
    mlt_properties_set(filter_props_green, "av.rs", red_value_str);
    mlt_properties_set(filter_props_green, "av.rm", red_value_str);
    mlt_properties_set(filter_props_green, "av.rh", red_value_str);
  }
  if (green_value >= -1.0 && green_value <= 1.0) {
    char green_value_str[20];
    snprintf(green_value_str, sizeof(green_value_str), "%.2f", green_value);
    mlt_properties_set(filter_props_green, "av.gs", green_value_str);
    mlt_properties_set(filter_props_green, "av.gm", green_value_str);
    mlt_properties_set(filter_props_green, "av.gh", green_value_str);
  }
  if (blue_value >= -1.0 && blue_value <= 1.0) {
    char blue_value_str[20];
    snprintf(blue_value_str, sizeof(blue_value_str), "%.2f", blue_value);
    mlt_properties_set(filter_props_green, "av.bs", blue_value_str);
    mlt_properties_set(filter_props_green, "av.bm", blue_value_str);
    mlt_properties_set(filter_props_green, "av.bh", blue_value_str);
  }
}
  mlt_filter_connect(filter, service, 0);
  mlt_consumer_connect(consumer, mlt_filter_service(filter));
  break;
```

```
case CONTROL_TYPE__BLUE_VALUE:
  blue_value = jit_control->blue_value;
  {
    mlt_properties filter_props_blue = mlt_filter_properties(filter);

    if (red_value >= -1.0 && red_value <= 1.0) {
      char red_value_str[20];
      snprintf(red_value_str, sizeof(red_value_str), "%.2f", red_value);
      mlt_properties_set(filter_props_blue, "av.rs", red_value_str);
      mlt_properties_set(filter_props_blue, "av.rm", red_value_str);
      mlt_properties_set(filter_props_blue, "av.rh", red_value_str);
    }
```

```
  if (green_value >= -1.0 && green_value <= 1.0) {
    char green_value_str[20];
    snprintf(green_value_str, sizeof(green_value_str), "%.2f", green_value);
    mlt_properties_set(filter_props_blue, "av.gs", green_value_str);
    mlt_properties_set(filter_props_blue, "av.gm", green_value_str);
    mlt_properties_set(filter_props_blue, "av.gh", green_value_str);
  }
  if (blue_value >= -1.0 && blue_value <= 1.0) {
    char blue_value_str[20];
    snprintf(blue_value_str, sizeof(blue_value_str), "%.2f", blue_value);
    mlt_properties_set(filter_props_blue, "av.bs", blue_value_str);
    mlt_properties_set(filter_props_blue, "av.bm", blue_value_str);
    mlt_properties_set(filter_props_blue, "av.bh", blue_value_str);
  }
}
mlt_filter_connect(filter, service, 0);
mlt_consumer_connect(consumer, mlt_filter_service(filter));
break;
```

6 Comparison of Video Colour Grading Results

In this Chapter, the results of the user-sided colour grading with JIT are compared to the results of the colour grading with the Melt CLI tool and KDEN Live, which uses the MLT framework as a backend. For the comparison, frames of the original video file, the frames created by the JIT implementation, Melt and KDEN Live are compared with and without the application of filters.

To evaluate the differences in the frames and the filter application between the tested components, the frames are extracted. Those tests were run on the same computer to ensure consistency and the frames were extracted by taking screenshots of the output or pre-existing frame extraction functions. Then, the colours of the images have been subtracted from each other using the *grain extract* function in GIMP, as described in Section 2.4.

In the Figures 38 and 39, two different results of the colour subtraction can be seen. Figure 38 shows the subtraction of an image from itself. This results in a uniformly grey image, because the images are identical. Figure 39 in comparison shows the subtractions of two separate screenshots of the same frame. Small distortions can be seen, which is caused by slightly different scaling or other inconsistencies in the screenshot process. Those small differences have to be considered in the comparison, because the images can not be expected to be exactly the same, when extracting them from different sources. Two subtracted frames that have a similar result to Figure 39 can be considered as the same frames.

Figure 38: Colour subtraction of the same file from itself results in a uniformly grey image.

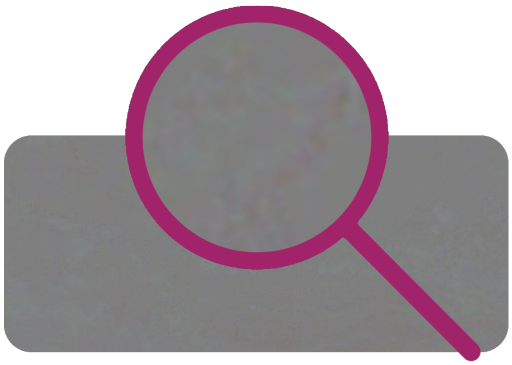


Figure 39: Colour subtraction of the same frame but a different screenshot, which results in slight distortions.

6.1 Comparison with Melt

In this Section, the results of video colour grading using the CLI tool Melt and the in Section 5.5 described implementation with JIT are compared.

For the evaluation, a comparison between the results of Melt with the `avfilter.colorbalance` filter applied and the RGB adjustment in the Accurate Player with the sliders is made. For the tests, the red value is set to the maximum with the respective method and the results will be compared and analysed below. The results and comparisons of the colours green and blue perform analogical, which can be seen in Appendix B. The filter is applied and frame 343 of the short film *Sintel* is extracted from both components. The results of Melt can be seen in Figure 40, with the `avfilter.colorbalance` filter applied and the parameters for shadows, midtones and highlights set to the maximum of 1 (`av.rs=1`, `av.rm=1` and `av.rh=1`). In comparison, the extracted frame from the Accurate Player frontend can be seen in Figure 41. This result is obtained by setting the red slider in the frontend onto the maximum value and leaving the sliders for green and blue on the default value, which corresponds to 0 in the filter application in the backend. The maximum value of the red slider gets sent to the backend and is being applied for the parameters of the filter `avfilter.colorbalance`, setting the parameters for shadows, midtones and highlights to the maximum of 1 (`av.rs=1`, `av.rm=1` and `av.rh=1`) as well.

Figure 40: Frame 343 after the application of the maximum value for red with Melt.

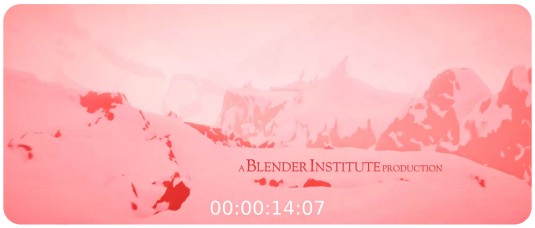


Figure 41: Frame 343 after the application of the maximum value for red in the Accurate Player.

In these Figures it can be seen that the red tone of the two frames visually differs. In Figure 41, the colours seem to be darker or more saturated. This is an unexpected result, because both components use the same MLT filter with the same parameters.

Considering the visual difference in the frame extractions from the two different approaches of applying the filter `avfilter.colorbalance`, the subtraction of those two frames shows more differences than the expected distortions that were described above in Figure 39.

In Figure 42, the extracted frame of Melt is subtracted from the frame of the Accurate Player. In this case, the result has a red tone itself, because the seemingly less red saturated frame (Figure 40) is subtracted from the seemingly more red saturated frame (Figure 41). The pixels in the Accurate Player frame have higher RGB values on the red colour channel than the corresponding pixels in the Melt extraction. This causes the red tone in the resulting image in Figure 42. This shows the relative differences in the red value between the two frames. The result image from the subtraction is red tinted and has no visibly green or blue areas, which reveals that the frame of the Accurate Player has an overall more saturated red tone than the extracted frame from Melt.

Figure 42: Subtraction of the same frame with filter application: Melt frame is subtracted from the frame of the Accurate Player.

To evaluate, if this difference is caused by the application of the filter or by other aspects, such as video compression or differences in video players, the frames from the original video file of the short film *Sintel* are compared with the frames from both methods without a filter applied to them. For this, frame 343 of the original video file is compared to the same frame extraction of Melt and the Accurate Player without the application of the filter. The frame from the original video file can be seen in Figure 43.

Figure 43: Frame 343 without filter application in the original video file.

In Figure 44, the Melt extraction of frame 343 without a filter application can be seen. In comparison to this, the extracted frame from the Accurate Player without an applied filter is shown in Figure 45, which seems to be more saturated.

Figure 44: Frame 343 without filter application in Melt.

Figure 45: Frame 343 without filter application in the Accurate Player.

To analyse the differences between the two frames without a filter applied further, another frame subtraction is conducted. Each of the frames are compared to the extraction of the original video file, to identify, which method is altering the results. In Figure 46, the subtraction of the Melt frame from the original frame is visualised using the *grain extract* function. In this image, the outlines of the motive of the frame can be seen in different grey tones, which reveals that the subtracted frames do not have the same colour values. In Figure 47 in comparison, the subtraction is done with the Accurate Player frame from the original frame. Figure 47 shows the expected result for a same output, with some differences in the grey tones but no clear areas or fragments from the original frame motive.

Figure 46: Subtraction of the Melt frame from the original frame.

Figure 47: Subtraction of Accurate Player frame from the original frame.

The extracted frames from the different approaches display significant differences, even when no filter is applied. The results show that the Accurate Player plays back the video more accurately than the Melt command line tool. These differences in the video representation without a filter application make it difficult to evaluate additional data regarding the application of filters using this method. The cause

for the different display of the video frames by Melt remains unclear. One possible explanation for these differences in the extracted frames could be rooted in the system environment potentially influencing the evaluation of the comparison on different platforms. The differences could be caused by software dependencies imposed by Melt or its video player.

6.2 Comparison with KDEN Live

In this Section, the results of video colour grading using KDEN Live and the in Section 5.5 described implementation of the Accurate Player are compared. In Figure 48, the list of categories for the available effects in KDEN Live can be seen. This includes the *Color and Image correction* option, which will be examined in the following. The options for the RGB colour adjustment in KDEN Live are examined and the results are compared to the frames of the Accurate Player. With those results it can be seen, if KDEN Live uses the `avfilter.colorbalance` filter that is being used in the Accurate Player or a different filter for the adjustment of the RGB values. The comparison of the results of platforms that use different filters and techniques for the achievment of RGB adjustment is a difficult task, because it cannot be evaluated which aspects of the results differ due to the differences in the filters and which aspects are influenced by other aspects.

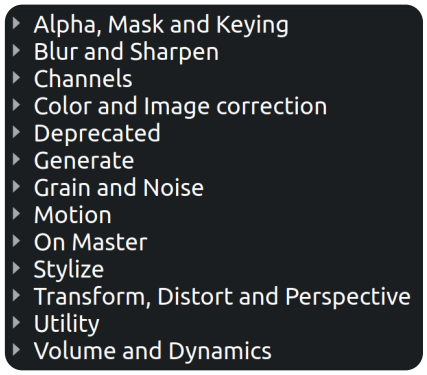


Figure 48: List of effect menus in KDEN Live.

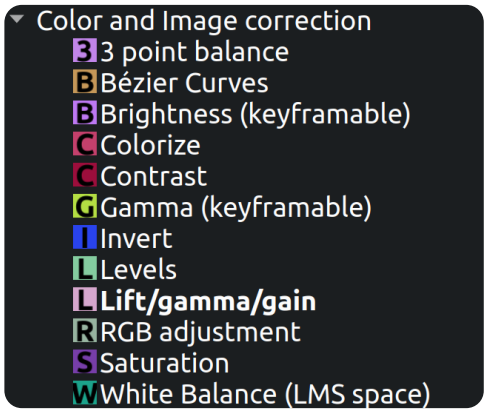


Figure 49: List of colour effects (KDEN Live).

In the effect category *Color and Image correction* in KDEN Live, different effects for colours can be found. The list of effects in this category can be seen in Figure 49. Those effect options were examined and *RGB adjustment* is the only effect that is adjusting the colour of the produced video with RGB values. This adjustment is implemented with sliders for red, green and blue, similar to the user-sided RGB adjustment in the Accurate Player using JIT. The other effects

in the list that can be seen in Figure 49, are not adjusting the RGB colouring of the video. For example the effect *3 point balance* might sound promising as well but refers to the adjustment of the values for black, gray and white.

In Figure 50, the RGB colour sliders of the effect *RGB adjustment* can be seen. As an additional parameter, the following different actions can be chosen: Change gamma, add constant and multiply. For each of those actions a comparison through colour subtraction with the *grain extract* function in GIMP is performed. Those frames from KDEN Live are compared with the extracted frames from the Accurate Player, with the application of the filter `avfilter.colorbalance` in the backend. The frames were extracted from the Accurate Player through taking screenshots of the output and in KDEN Live the *extract frame* function was used.

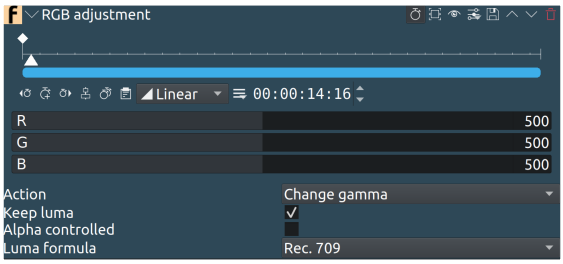


Figure 50: RGB colour adjustment in KDEN Live.

For the other additional parameters (*keep luma*, *alpha controlled* and *luma formula*), the default values were used for this comparison. In the following, the RGB adjustment with the different actions are shown and analysed on the example of the colour red. The colours green and blue work analogical and can be found in Appendix B.

Change gamma

Change gamma refers to adjusting the luminance or brightness of an image or video by applying a gamma correction factor. This can be used to adapt the output to different devices [41].

In Figure 52, the extracted frame from KDEN Live with the application of the maximum red value and the *Change gamma* action is shown. The slider for the adjustment of the colour red is set to the maximum while the other sliders remain unchanged. In comparison, the extracted frame from the Accurate Player with the maximum red value applied can be seen in Figure 51. This result is obtained by setting the red slider in the frontend onto the maximum value and leaving the sliders for green and blue on the default value. In the backend, the filter `avfilter.colorbalance`

with the parameters for red shadows, midtones and highlights to the maximum of 1 is applied (`av.rs=1`, `av.rm=1` and `av.rh=1`). This frame from the Accurate Player was shown before in Figure 41 and is displayed here again for a better visualisation of the frame comparison.

The frame extraction from KDEN Live (Figure 52) is visually darker than the frame extraction from the Accurate Player (Figure 51). This leads to the assumption that the KDEN Live effect *RGB adjusment* is not based on the functionalities of the MLT filter `avfilter.colorbalance`.

Figure 51: Frame 343 after the application of the maximum value for red in the Accurate Player.

Figure 52: Frame 343 from KDEN Live with the action *change gamma* in the red filter application.

The result of the subtraction can be seen Figure 53. The Accurate Player frame was subtracted from the KDEN Live frame. The result does show green tones and no red tones, which demonstrates that the Accurate Player frame has more saturated red values than the KDEN Live frame in all areas. Because the frames differ, the outlines and fragments of the original motive of the frame can be seen.

Figure 53: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame (with the action *change gamma*).

Add constant

Adding a constant value involves increasing or decreasing the intensity of a colour channel in an image or video by a fixed amount. Assuming, the MLT filter `avfil-`

`ter.colorbalance` adjusts the RGB colour channels by a fixed amount, which is set by the filter parameters, this action could lead to more similar results to the Accurate Player frame. The other actions in KDEN Live imply that multiple filters or parameters are used to manipulate the content.

In Figure 55, the extracted frame from KDEN Live with the application of the maximum red value for the *Add constant* action is displayed. In Figure 54 in comparison, the extracted frame from the Accurate Player with the maximum red value applied can be seen. This frame was shown before in Figure 41 and 51 and is displayed here again to allow for a direct comparison. The adjustment of the red slider to the maximum value in the frontend leads to the application of the filter `avfilter.colorbalance` with the red parameters set to the maximum value of 1 in the backend (`av.rs=1`, `av.rm=1` and `av.rh=1`).

The KDEN Live frame extraction (Figure 55) is visually darker than the frame extraction from the Accurate Player (Figure 54) and looks similar to the frame that was created with the *Change gamma* action that was shown in Figure 52.

Figure 54: Frame 343 after the application of the maximum value for red in the Accurate Player.

Figure 55: Frame 343 from KDEN Live with the action *add constant* in the red filter application.

The subtraction of those frames can be seen in Figure 56. It reveals that the images that were created with the *Change gamma* and the *Add constant* action differ more than visually noticeable at first glance. The result of the subtraction of the Accurate Player frame from the KDEN Live frame extraction in Figure 56 shows that the Accurate Player frame has a more saturated red value in some areas, while in other areas the KDEN Live frame extraction has a more saturated red value. The resulting image contains red and green pixels after applying the *grain extract* function for the subtraction in GIMP.

Figure 56: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *add constant*).

Multiply

Multiplying involves changing the intensity of each colour channel (red, green, blue) by a specified factor. This adjustment allows for precise control over colour balance and saturation [32].

In Figure 58, the extracted frame from KDEN Live with the application of the maximum red value for the *Multiply* action is displayed. In comparison in Figure 54, the extracted frame from the Accurate Player with the maximum red value applied can be seen. This frame was shown and explained before and is displayed here again to allow for a direct visual comparison of the frames.

The frame extraction from KDEN Live (Figure 58) is visually more saturated and red toned than the frame extraction from the Accurate Player (Figure 54).

Figure 57: Frame 343 after the application of the maximum value for red in the Accurate Player.

Figure 58: Frame 343 from KDEN Live with the action *Multiply* in the red filter application.

In Figure 59, the subtraction of the Accurate Player frame extration from the KDEN Live frame can be seen. This subtraction result shows a mostly red picture, caused by the high red values in the KDEN Live frame. In the green areas on the other hand, the Accurate Player frame had a more saturated red value than the KDEN

Live frame. This is presumably caused by the darkness of those areas in the frame that was extracted from KDEN Live.

Figure 59: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *Multiply*).

Evaluation KDEN Live

When observing the results of the KDEN Live RGB adjustment, visual similarities to the results of the `frei0r.coloradj_RGB` filter are noticeable. A further inspection of the filter parameters on the MLT website reveals that the RGB adjustment parameters in KDEN Live are the same as those of the `frei0r.coloradj_RGB` filter. Both RGB adjustment processes have the adjustable input parameters `R`, `G`, `B`, `Action`, `Keep luma`, `Alpha controlled` and `Luma formula` [37].

To confirm the usage of the filter `frei0r.coloradj_RGB` in KDEN Live's RGB adjustment, the code base of KDEN Live is consulted. This is possible, because KDEN Live is a open source software and the code is publicly available. The following code excerpt shows parts of the XML file of the `frei0r.coloradj_RGB` in the KDEN Live code base [15].

```
<?xml version="1.0"?>
<!DOCTYPE kpartgui>
  <effect LC_NUMERIC="C" tag="frei0r.coloradj_RGB" id="frei0r.coloradj_RGB">
    <name>RGB adjustment</name>
...
```

This confirms the usage of the `frei0r.coloradj_RGB` filter in the KDEN Live RGB adjustment.

6.3 Conclusion of the Filter Comparison

In this Chapter, the results of the user-sided colour grading with JIT were compared to the results of the colour grading with the Melt CLI tool and KDEN Live, which uses the MLT framework as a backend.

In Section 6.1, the results of video colour grading using Melt and the implementation with JIT were compared. The results in this comparison display significant differences, even when no filter is applied. The results of the comparison with the original video frame and without a filter application reveal that the Accurate Player plays back the original video more accurately than the Melt video player. These differences in the video representation without a filter application make it difficult to evaluate additional data regarding the application of filters using this method. The cause for the differences between the video frames by Melt remains unclear. One possible explanation for these differences in the extracted frames could be rooted in the system environment potentially influencing the evaluation of the comparison on different platforms. The differences could be caused by software dependencies influencing Melt or its video player.

Additionally, the comparison with KDEN Live did not show the same results as the filter application in the Accurate Player either. The results of the visual comparisons, functionalities and colour subtractions lead to the assumption that the KDEN Live effect *RGB adjustment* is not using the MLT filter `avfilter.colorbalance` that is being used in this thesis project to adjust the RGB values but the `frei0r.coloradj_RGB` filter. This information gives insight of the implementation and backend of KDEN Live. The comparison revealed the different options and functionalities that arise with the use of different MLT filters and techniques, but those techniques make KDEN Live unsuitable for further comparisons with the implementation of this thesis project.

Those significant differences in the compared frames show the complexity of the topic of colour adjustment and representation. For this thesis project, all tests were run on the same device to ensure consistency. Ensuring a consistent output across various platforms and devices adds even more complexity to the task of colour representation and video streaming. Additionally it was revealed that there are many different ways to achieve similar results or functionalities in the topic of video colour grading. Many parameters and aspects have to be taken into consideration. This confirms the necessity for user-sided colour adjustment for a more customisable video streaming experience and satisfying results.

7 Conclusion

In this Chapter, a summary of this thesis project, its contributions as well as its limitations are described. Afterwards, opportunities for future work are introduced and examined.

7.1 Summary

The topic of this thesis project is on multimedia processing in real-time, regarding the adjustment of the RGB values of a video with JIT using the MLT framework as a backend. This solution was implemented in Codemill's Accurate Player and using WebRTC as a data channel.

In this thesis, background knowledge on colour theory and representation (Chapter 2) and the technical background, especially the structure and usage of the MLT framework were given. Additionally protocol buffers and other streaming components were introduced in Chapter 3.

This thesis project explored two research questions. The first question aimed to evaluate the feasibility of the real-time colour grading with JIT and the MLT framework. The implementation as described in Chapter 5 confirms that it was achievable. To select the most suitable MLT filter for the RGB adjustment, a comparison of different MLT filters was conducted. The selected filter is the `avfilter.colorbalance` filter, which can be used to adjust the RGB values of the highlights, midtones and shadows of a video individually and achieves the most vibrant and saturated results.

The second research question considers the comparison of video colour grading results with MLT filters that were applied on different platforms. The compared platforms are the Accurate Player that was implemented in this thesis project, the Melt CLI tool and KDEN Live (Chapter 6). The comparison with Melt revealed an interesting result: The extracted frames display significant differences, even without the application of filters. This showed that the Accurate Player is presenting the original videos files more accurately than Melt does. The comparison with KDEN Live did not show the same results as the filter application in the Accurate Player either. The results of the colour subtractions lead to the presumption that KDEN Live is using the MLT filter `frei0r.coloradj_RGB` for the adjustment of the RGB values. Those significant differences in the compared platforms and filters emphasise the complexity of the topic of colour adjustment and representation. Furthermore, insights about the backend of KDEN Live are gained. Additionally it was revealed that there are many different ways to achieve similar results or functionalities in the

topic of video colour grading. Many parameters and aspects have to be taken into consideration. This confirms the necessity for user-sided colour adjustment for a more customisable video streaming experience to achieve satisfying results.

7.2 Contributions and Limitations

Regarding the contributions, the implementation of the RGB adjustment shows the feasibility of applying filters in real-time using the MLT framework. This contributes to the advancement of the field of real-time multimedia processing.

Through the comparison of different MLT filters for RGB adjustment, this thesis project provides insights into the suitability of various MLT filters. Additionally, different options for MLT filters that exceed the area of RGB adjustment can be seen in Appendix C. This contributes to the knowledge regarding the selection of MLT filters and gives an overview of different tasks using the MLT framework.

By examining discrepancies in the colour representation and RGB adjustment across different platforms, it is shown that there are various options to implement a solution for user-sided RGB adjustment. It is also revealed that the Accurate Player displays video files more precisely than Melt's video player.

Regarding the limitations, the results are limited to a specific implementation of real-time RGB adjustment within the MLT framework. The findings and conclusions can not be generalised to other streaming scenarios or platforms. Additionally, the application of different MLT filters that do not focus on the adjustment of the RGB values, including audio filters, might not be comparable. Another aspect that needs to be considered is that the system's environment might have influenced the evaluation of the comparison on different platforms. The frames that were extracted from Melt and the Accurate Video frontend showed visible differences. The cause of those differences remains unclear, but it could be caused by software dependencies imposed by Melt or its video player.

7.3 Future Work

In this Section, different opportunities for future work are discussed. This includes the implementation of video presets, options for advanced video processing improvements in the user interface and the application of audio filters.

Video Presets

Video presets, also known as video filters, video effects or Look-Up-Tables (LUT), are pre-configured settings or adjustments that can be applied to videos to achieve specific visual styles or effects. These presets often include templates for colour grading. Video presets are commonly used in video editing software and social media platforms. They provide the user with options to customise their videos easily [25]. One example of video presets can be seen in Figure 60, where the presets in Adobe Premiere Pro with the *Magic Bullet Looks* plug-in are shown [1, 42].

Figure 60: Preset overview in Adobe Premiere Pro with the plug-in *Magic Bullet Looks* [1, 42].

Implementing options for the application of those presets is an interesting opportunity for future work. To implement this, different MLT filters can be used or combined. Different options for those MLT filters as video preset filters can be seen in Appendix C.

Advanced Video Processing

With the implementation of options for advanced video processing tools, other opportunities arise. An online video editing tool with a MLT backend could be implemented. The functionalities could include the cutting of video clips, the application of above described video presets and the manual adjustment of individual video and audio parameters.

Improvements of the User Interface

Improvements of the user interface can be implemented to make the adjustment of the RGB colours more intuitive. This could involve a preview of the colour that is being created by adjusting the RGB sliders and that is then applied to the video. An example for this preview can be seen in Figure 61. To decide, which changes in the user interface enhance the usability of the colour adjustment and feel intuitive for the user, extensive testing and user studies could be conducted.

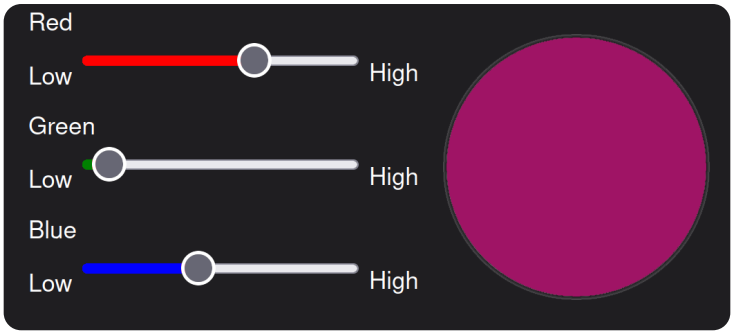


Figure 61: Example for the preview of the colour that is being created by adjusting the RGB sliders and then applied to the video.

In addition to this, the above described future work opportunities of implementing video presets or an online video editing tool need a design for the user interface as well, to guarantee the usability.

Performance Evaluation

In this thesis project, the results of different methods of applying MLT filters were visually compared. Another interesting aspect is the comparison of the performance of those different methods. For this, the runtime as well as the memory usage could be interesting parameters to observe. To ensure accuracy, a separate system or server would be needed, which should be isolated from other processes to prevent interference. Monitoring tools can be used to track the resource usage. Those tests can be run on test cases that were developed to represent typical scenarios or more specialised test cases for information with specific restrictions and conditions. It would be interesting to vary parameters such as the video file size or the amount of applied filters. This data can then be statistically analysed.

Audio Filter

This thesis project focuses on the application of visual filters to a video, especially the adjustment of the RGB values. Aside from this, audio processing plays a role in the processing of videos, too. Similar to the visual content, the audio can enhance the experience and evoke emotions in the viewer. Audio filters can also be used to improve the quality of the recorded audio for example by applying noise suppression or to add context to a scene by adding fitting surrounding noises. In addition to this, simple aspects including the volume or tone can be modified. The list of filters on the MLT website contains the available audio filters [36]. The implementation of the audio filter integration is an interesting option for future work. To compare different audio filters, the according sound waves of a track could be read out and compared.

In this thesis project, insights on multimedia processing, colour theory and the MLT framework have been gained and contributions have been made to the field. This lead to various options for future work projects.

List of Figures

List of Tables

References

[1] Adobe Systems. *Adobe Premiere Pro*. Computer software. 2020.

[2] I. Ahmad, Xiaohui Wei, Yu Sun, and Ya-Qin Zhang. "Video transcoding: an overview of various techniques and research issues". In: *IEEE Transactions on Multimedia* 7.5 (2005), pp. 793–804. DOI: 10.1109/TMM.2005.854472.

[3] D Alamsyah, N Othman, and H Mohammed. "The awareness of environmentally friendly products: The impact of green advertising and green brand image". In: *Management Science Letters* 10.9 (2020), pp. 1961–1968.

[4] Valentin Bojinov. *RESTful Web API Design with Node. js 10: Learn to create robust RESTful web services with Node. js, MongoDB, and Express. js*. Packt Publishing Ltd, 2018.

[5] Bolagsverket. *CodeMill AB (publ)*. Accessed 06.02.2024. URL: https://foretagsinfo.bolagsverket.se/sok-foretagsinformation-web/foretag/556762-3839/foretagsform/AB.

[6] Bolagsverket. *CodeMill Handelsbolag*. Accessed 06.02.2024. URL: https://foretagsinfo.bolagsverket.se/sok-foretagsinformation-web/foretag/969729-6854/foretagsform/HB.

[7] Peter Brophy and Jenny Craven. "Web accessibility". In: *Library trends* 55.4 (2007), pp. 950–972.

[8] Robert H Chen. *Liquid crystal displays: fundamental physics and technology*. John Wiley & Sons, 2011.

[9] Codemill. *Accurate-Player-3-Core README*. Retrieved from company's codebase. Accessed 07.02.2024. File attached in Appendix A.1.

[10] Codemill. *Accurate.Video - Deliver with confidence*. Accessed 06.02.2024. URL: https://www.codemill.se/accuratevideo.

[11] Codemill. *Accurate.Video Documentation*. Accessed 09.02.2024. 2024. URL: https://docs.accurate.video/docs/.

[12] Codemill. *Jit-WebRTC README*. Retrieved from company's codebase. Accessed 07.02.2024. File attached in Appendix A.2.

[13] Codemill. *LinkedIn profile of Codemill*. Accessed 06.02.2024. URL: https://se.linkedin.com/company/codemill.

[14] Codemill. *This is Codemill*. Accessed 02.02.2024. URL: https://www.codemill.se/about-us.

[15] KDE Community. *KDENlive*. https://invent.kde.org/multimedia/kdenlive. Accessed 15.05.2024. 2024.

[16] Chris Currier. "Protocol buffers". In: *Mobile Forensics–The File Format Handbook: Common File Formats and File Systems Used in Mobile Devices*. Springer, 2022, pp. 223–260.

[17] *Dolby Vision: Content Creation Guidelines*. Tech. rep. White Paper. Dolby Laboratories, 2020. URL: https://web.archive.org/web/20200520033844/https://www.dolby.com/us/en/technologies/dolby-vision/dolby-vision-white-paper.pdf.

[18] Fabio Duarte. *Video Streaming Services Stats (2023)*. Accessed 02.02.2024. Sept. 2023. URL: https://explodingtopics.com/blog/video-streaming-stats.

[19] Andrew J Elliot. "Color and psychological functioning: a review of theoretical and empirical work". In: *Frontiers in psychology* 6 (2015), p. 127893.

[20] Andrew J Elliot and Daniela Niesta. "Romantic red: red enhances men's attraction to women." In: *Journal of personality and social psychology* 95.5 (2008), p. 1150.
[21] Filmbaker. *The Role of Color in Video Production: How to Evoke Emotions and Convey Meaning*. Accessed 22.03.2024. Aug. 2023. URL: https://www.filmbaker.com/blog/the-role-of-color-in-video-production-how-to-evoke-emotions-and-convey-meaning.
[22] OpenJS Foundation. *About Node.js*. Accessed 09.02.2024. Jan. 2024. URL: https://nodejs.org/en/about.
[23] Google Developers. *Protocol Buffers - Proto2*. Accessed 06.04.24. URL: https://protobuf.dev/programming-guides/proto2/.
[24] Russell A Hill and Robert A Barton. "Red enhances human performance in contests". In: *Nature* 435.7040 (2005), pp. 293–293.
[25] Steve Hullfish. *The art and technique of digital color correction*. Routledge, 2013.
[26] "IEEE Standard for Learning Technology–JavaScript Object Notation (JSON) Data Model Format and Representational State Transfer (RESTful) Web Service for Learner Experience Data Tracking and Access". In: *IEEE Std 9274.1.1-2023* (2023), pp. 1–103. DOI: 10.1109/IEEESTD.2023.10273185.
[27] Docker Inc. *Docker Documentation*. Accessed 09.02.2024. Jan. 2024. URL: https://docs.docker.com/.
[28] Alexander Jacobs. "Comparison of JavaScript package managers". In: (2019).
[29] Rachel Kaplan and Stephen Kaplan. *The experience of nature: A psychological perspective*. Cambridge university press, 1989.
[30] Kdenlive. *Kdenlive*. Accessed 09.04.24. URL: https://kdenlive.org/en/.
[31] Rolf G Kuehni. *Color: An introduction to practice and principles*. John Wiley & Sons, 2012.
[32] Olivier Lecarme and Karine Delvare. *The book of GIMP: A complete guide to nearly everything*. No Starch Press, 2013.
[33] Library of Congress. *AVI (Audio Video Interleaved) File Format*. Tech. rep. Library of Congress Preservation, Jan. 2022.
[34] Anthony C Little and Russell A Hill. "Attribution to red suggests special role in dominance signalling". In: *Journal of evolutionary psychology* 5.1 (2007), pp. 161–168.
[35] MasterClass. *Color Correcting vs. Color Grading: Understanding Film Coloring*. Accessed 23.03.2024. June 2021. URL: https://www.masterclass.com/articles/color-correcting-vs-color-grading.
[36] LLC Meltytech. *Melt*. Accessed 08.02.2024. URL: https://www.mltframework.org/docs/melt/.
[37] Meltytech, LLC. *Filter Plugins*. https://www.mltframework.org/plugins/PluginsFilters/. Accessed 25.03.2024.
[38] MLT Framework. *Melt Framework*. https://github.com/mltframework/mlt. Accessed 25.03.2024. 2023.
[39] Nielsen. *Streaming services remain most popular destination for TV viewing in December*. Accessed 02.02.2024. Jan. 2023. URL: https://www.nielsen.com/

insights/2023/streaming-services-remain-most-popular-destination-for-tv-viewing-in-december/.

[40] A Cubed Productions. *The Importance of Color Grading in Videos: Elevating Visual Impact and Storytelling*. Accessed 22.03.2024. June 2023. URL: https://acubedproductions.com/2023/07/the-importance-of-color-grading-in-videos-elevating-visual-impact-and-storytelling/.

[41] Shanto Rahman, Md Mostafijur Rahman, Mohammad Abdullah-Al-Wadud, Golam Dastegir Al-Quaderi, and Mohammad Shoyaib. "An adaptive gamma correction for image enhancement". In: *EURASIP Journal on Image and Video Processing* 2016 (2016), pp. 1–13.

[42] Red Giant. *Magic Bullet Looks*. Adobe Premiere Pro plugin. 2013.

[43] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly Media, Inc., 2008.

[44] Thomas Rusert, Kenneth Andersson, Ruoyang Yu, and Harald Nordgren. "Guided just-in-time transcoding for cloud-based video platforms". In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 1489–1493. DOI: 10.1109/ICIP.2016.7532606.

[45] Kong-King Shieh and Chin-Chiuan Lin. "Effects of screen type, ambient illumination, and color combination on VDT visual performance and subjective preference". In: *International journal of industrial ergonomics* 26.5 (2000), pp. 527–536.

[46] Diomidis Spinellis. "Git". In: *IEEE software* 29.3 (2012), pp. 100–101.

[47] Branislav Sredojev, Dragan Samardzija, and Dragan Posarac. "WebRTC technology overview and signaling solution design and implementation". In: *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*. IEEE. 2015, pp. 1006–1009.

[48] Patricia Valdez and Albert Mehrabian. "Effects of color on emotions." In: *Journal of experimental psychology: General* 123.4 (1994), p. 394.

[49] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[50] Vegas. *Understanding What Color Correction And Color Grading Are – And Aren't*. Accessed 23.03.2024. URL: https://www.vegascreativesoftware.com/ca/video-editing/color-grading-vs-color-correction-process/.

[51] T. Wiegand, G.J. Sullivan, G. Bjontegaard, and A. Luthra. "Overview of the H.264/AVC video coding standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (2003), pp. 560–576. DOI: 10.1109/TCSVT.2003.815165.

[52] Atila Yüksel. "Exterior color and perceived retail crowding: Effects on tourists' shopping quality inferences and approach behaviors". In: *Journal of Quality Assurance in Hospitality & Tourism* 10.4 (2009), pp. 233–254.

A README

A.1 Accurate-Player-3

The README-file of the Accurate-Player-3 code was referenced as [9] and is included for transparency in the following.

```
# Accurate Player 3 Monorepo

This monorepo contains all packages related to Accurate Player 3. Packages are located in the `packages` folder:

- `core` - Core player functionality used by all players
- `progressive` - Player that plays videos using [progressive download](https://en.wikipedia.org/wiki/Progressive_download)
- `hls` - Player that plays videos using [HTTP Live Streaming](https://en.wikipedia.org/wiki/HTTP_Live_Streaming)
- `dash` - Player that plays videos using [Dynamic Adaptive Streaming over HTTP](https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP)
- `abr` - Player that plays both HLS and DASH videos
- `cutlist` - Player that plays videos from a cut list using progressive download
- `plugins` - Player plugins
- `timecode` - Utilities for parsing/manipulating video timecodes
- `license-client` - License validation client
- `demo` - Player and plugin demos

`core` contains the basic player logic, interfaces and playback controllers.
It relies heavily on logic from `timecode`.

`core` itself doesn't contain much logic, playback is handled by playback controllers and `plugins` empower the player.

## Commands

- `yarn` - Installs dependencies.
- `yarn build` - Builds all packages. Production build. Minified source code and no source maps.
- `yarn build:dev` - Builds all packages. Development build. No minification of source code and includes source maps.
- `yarn build:watch` - Builds all packages but `demo` in watch mode (rebuilds on file changes). Development build.
- `yarn build:docs` - Builds documentation.
- `yarn build:demo` - Builds demos.
- `yarn test` - Runs all tests
- `yarn test:coverage` - Runs all tests and generate [istanbul](https://istanbul.js.org/) coverage reports.
- `yarn lint` - Checks all source for lint errors
- `yarn lint:fix` - Checks all source for lint errors and tries to fix all errors that can automatically be fixed.
- `yarn start` - Starts builds in watch mode in all packages and starts the `demo` -package on http://localhost:5000
- `yarn clean` - Clears all build folders and removes all installed packages.

See complete list of commands in `package.json`

**Note** IDE will not find related packages before they've been built.

# Development

To start only a subset of the packages with `yarn start` (or any other command) you can leverage `lerna`s
filter options, full documentation [here](https://github.com/lerna/lerna/tree/main/core/filter-options).

Some examples:

Start plugins and the demo package:

```
yarn start --scope "*/*plugins" --scope "*/*demo"
```

Start the progressive player package with all it's dependencies:

```
yarn start --scope "*/*progressive" --include-dependencies
```

This feature is useful to not start every package during development to speed things
```

```
up.

## JIT Demo

There are some environment variables used for the Jit demo

- `JIT_BACKEND`: Specify backend for Jit, can either be a complete url (eg `http://127.0.0.1:8080`) or
  or just an ip (eg `127.0.0.1`), in which case `http://` and `:8080` is added
  automatically.
- `AP_KEY`: Specify ap license key to use

Example command to start: `env JIT_BACKEND=127.0.01 AP_KEY=ABCD yarn start`

## Test without publishing

We can test changes locally without the need of publishing changes to nexus npm
repository.

### Using npm pack

Run `npm pack` inside the appropriate package (e.g. packages/core) to create a local
archive `codemill-accurate-player-core-x.y.z.tgz`.
Install it where needed by running `npm install path/to/*.tgz`.

## Release

### Test release

To create a prerelease unique to your current branch, run:

```
yarn release:pre
```

Normal release

Release using
`yarn increment -- pre{major|minor|patch}`
or just
`yarn increment`
selecting version manually

See RELEASE.md
```

A.2 JIT-WebRTC

The README-file of the JIT-WebRTC code was referenced as [12] and is included for transparency in the following.

```
# jit-webrtc

Live Transcoder with WebRTC output

## Design

The system consists of four "moving parts": three on the backend (`melt`, `main.py` and `session-service`)
and one on the frontend (website with a JavaScript/WebRTC client).

```
                  +---------+      +---------+      +---------+
                  |         | init |         | init |         |
                  |         |<-----| session |<-----|         |
+------+   AVI    |         |      |         |      |         |
|      |--------->|         |      +---------+      |         |
|      | metadata |         |         video         |         |
| melt |--------->| main.py |---------------------->| browser |
|      | control  |         |         audio         |         |
|      |<---------|         |---------------------->|         |
+------+          |         |        metadata       |         |
                  |         |---------------------->|         |
                  |         |        control        |         |
                  |         |<----------------------|         |
                  +---------+                       +---------+
```

`melt` is used to decode input files and perform rendering.
It is the command line interface (CLI) for the [MLT framework](https://mltframework.org/),
a popular framework for many free software non-linear editors (NLEs).

Control commands are sent from the browser to `main.py` via a WebRTC data channel,
and then from `main.py` to `melt` via a stdout/stdin.
`melt` in turn sends an AVI stream with rendered video and audio to `main.py` via a named pipe,
and status messages and metadata is sent to `main.py` via another named pipe.

`main.py` will extract audio and video from the incoming AVI stream and encode them for sending to the browser via WebRTC.
It has the ability to extract specific channels from the rendered audio, to be rendered to an stereo stream for WebRTC.
It does not appear to be possible to encode surround sound for WebRTC, despite WebRTC using the Opus codec which is surround capable.
This may be a limitation in `aiortc`, in WebRTC or in the browser, or all three.

The JavaScript client receives metadata from `main.py` and sends commands and ping replies in return over the control channel.
It also receives and plays audio and video.

`main.py` and `melt` is started once for each JIT session, and media is presented to it on startup.
`session` is a Java-based service that presents a REST api that can be used to start a new JIT session given a description of
the media to use. It also has support for managing a full ASG cluster on AWS, and distribute new JIT sessions on the instances in
the ASG.

For more information on each component see:
* [`melt`](https://github.com/sirf/mlt.git)
* [`main.py`](jit/README.md)
* [`session`](session/README.md)

Development

For development and quick testing without the session service, on can either start JIT as a docker, or run JIT directly on your computer.
The latter has some problems if you're on an Apple ARM.

Run the docker:

`docker/main/main.sh --threads 16 --port 8080 $VIDEOFILE`
```

```
62
63  Where `$VIDEOFILE` is either a local path or a URL. `threads` and `port` are both
    optional, with current default values set to 72 threads and port 8080.
64
65  To run JIT directly, see [`jit/README.md`](jit/README.md).
66
67  ### Open Docker IPs
68
69  Your Docker container will get an IP in the style of 172.17.0.\*. Using e.g. Docker CE
     on Linux this IP will be reachable from outside the container.
70  However, on Docker Desktop for Mac, this IP will not be opened.
71  This will cause problems. There is a service that can fix this. You **install and
    activate it once** by doing this:
72
73  ```
74  # Install via Homebrew
75  brew install chipmk/tap/docker-mac-net-connect
76
77  # Run the service and register it to launch at boot
78  sudo brew services start chipmk/tap/docker-mac-net-connect
79  ```
80
81
82  ## Deployment
83
84  Currently the only way supported for deployment is by using EC2 instances on AWS. An
    AMI is created by CI on release, containing a
85  local instance of the `session-service`, together with `main.py`, `melt` and anything
    else needed to start JIT processes on the EC2
86  instance. For more information about what the AMI contains, and how configuration can
    be fed to it see [`ami/README.md`](ami/README.md).
87
88  ### Terraform
89
90  Terraform modules are supplied that simplify setup of either single EC2 instances or a
     full ASG cluster of instances, including
91  `session-service` running as an orchestrator on ECS. See [`terraform/README.md`]
    (terraform/README.md) for more information on
92  available modules, their parameters and outputs.
93
```

B Comparison on all Colour Channels

Comparison with Melt

Red

Figure 62: Frame 343 after the application of the maximum value for red in Melt.

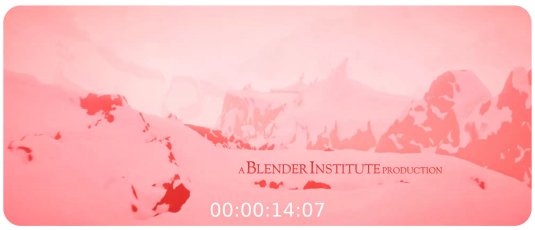


Figure 63: Frame 343 after the application of the maximum value for red in the Accurate Player.

Figure 64: Subtraction of the same frame with filter application: Melt frame is subtracted from the frame of the Accurate Player.

Green

Figure 65: Frame 343 after the application of the maximum value for green in Melt.

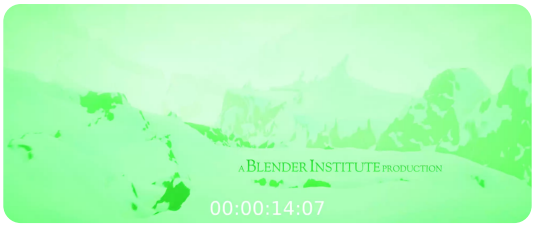


Figure 66: Frame 343 after the application of the maximum value for green in the Accurate Player.

Figure 67: Subtraction of the same frame with filter application: Melt frame is subtracted from the frame of the Accurate Player.

Blue

Figure 68: Frame 343 after the application of the maximum value for blue in Melt.

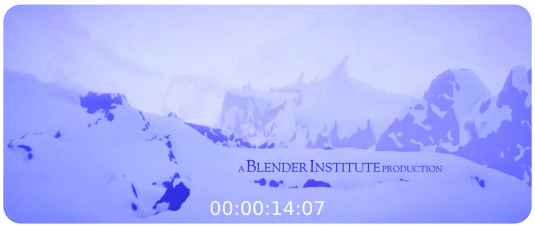


Figure 69: Frame 343 after the application of the maximum value for blue in the Accurate Player.

Figure 70: Subtraction of the same frame with filter application: Melt frame is subtracted from the frame of the Accurate Player.

Comparison with KDEN Live

Change gamma

Red

Figure 71: Frame 343 after the application of the maximum value for red in the Accurate Player.

Figure 72: Frame 343 from KDEN Live with the action *change gamma* in the red filter application.

Figure 73: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame (with the action *change gamma*).

Green

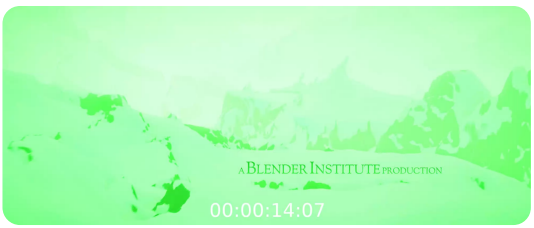


Figure 74: Frame 343 after the application of the maximum value for green in the Accurate Player.

Figure 75: Frame 343 from KDEN Live with the action *change gamma* in the green filter application.

Figure 76: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame (with the action *change gamma*).

Blue

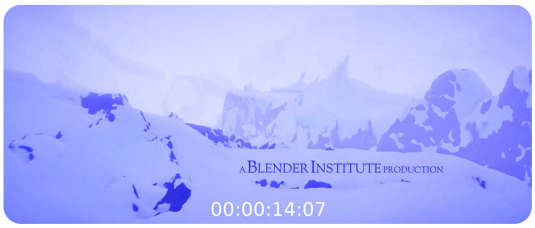


Figure 77: Frame 343 after the application of the maximum value for blue in the Accurate Player.

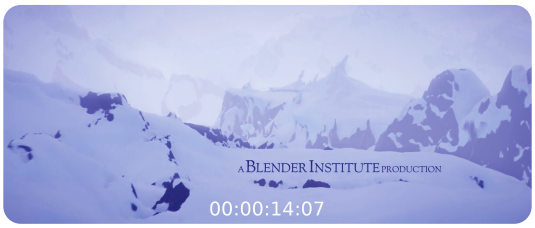


Figure 78: Frame 343 from KDEN Live with the action *change gamma* in the blue filter application.

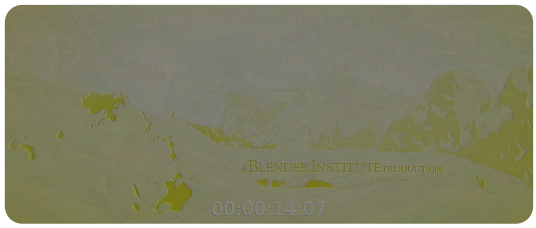


Figure 79: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame (with the action *change gamma*).

Add constant

Red

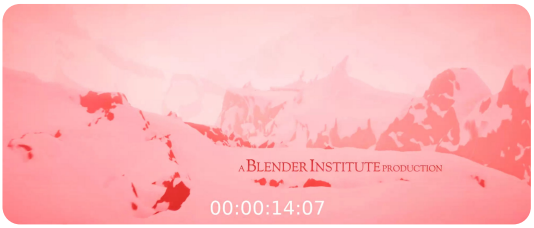


Figure 80: Frame 343 after the application of the maximum value for red in the Accurate Player.

Figure 81: Frame 343 from KDEN Live with the action *add constant* in the red filter application.

Figure 82: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *add constant*).

Green

Figure 83: Frame 343 after the application of the maximum value for green in the Accurate Player.

Figure 84: Frame 343 from KDEN Live with the action *add constant* in the green filter application.

Figure 85: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *add constant*).

Blue

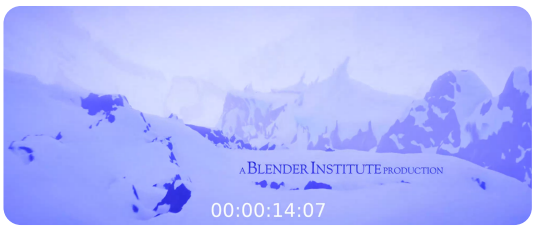


Figure 86: Frame 343 after the application of the maximum value for blue in the Accurate Player.

Figure 87: Frame 343 from KDEN Live with the action *add constant* in the green filter application.

Figure 88: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *add constant*).

Multiply

Red

Figure 89: Frame 343 after the application of the maximum value for red in the Accurate Player.

Figure 90: Frame 343 from KDEN Live with the action *Multiply* in the red filter application.

Figure 91: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *Multiply*).

Green

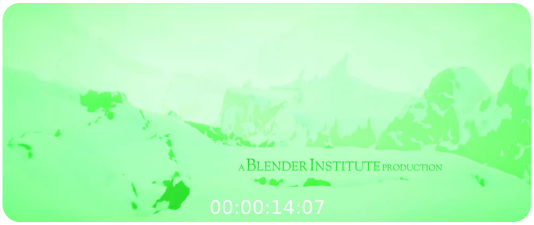


Figure 92: Frame 343 after the application of the maximum value for green in the Accurate Player.

Figure 93: Frame 343 from KDEN Live with the action *Multiply* in the green filter application.

Figure 94: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *Multiply*).

Blue

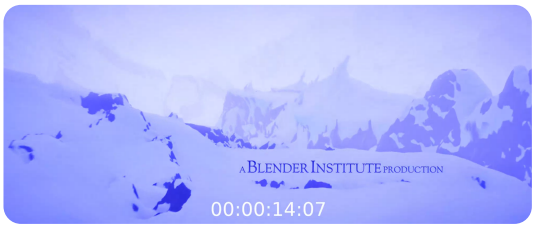


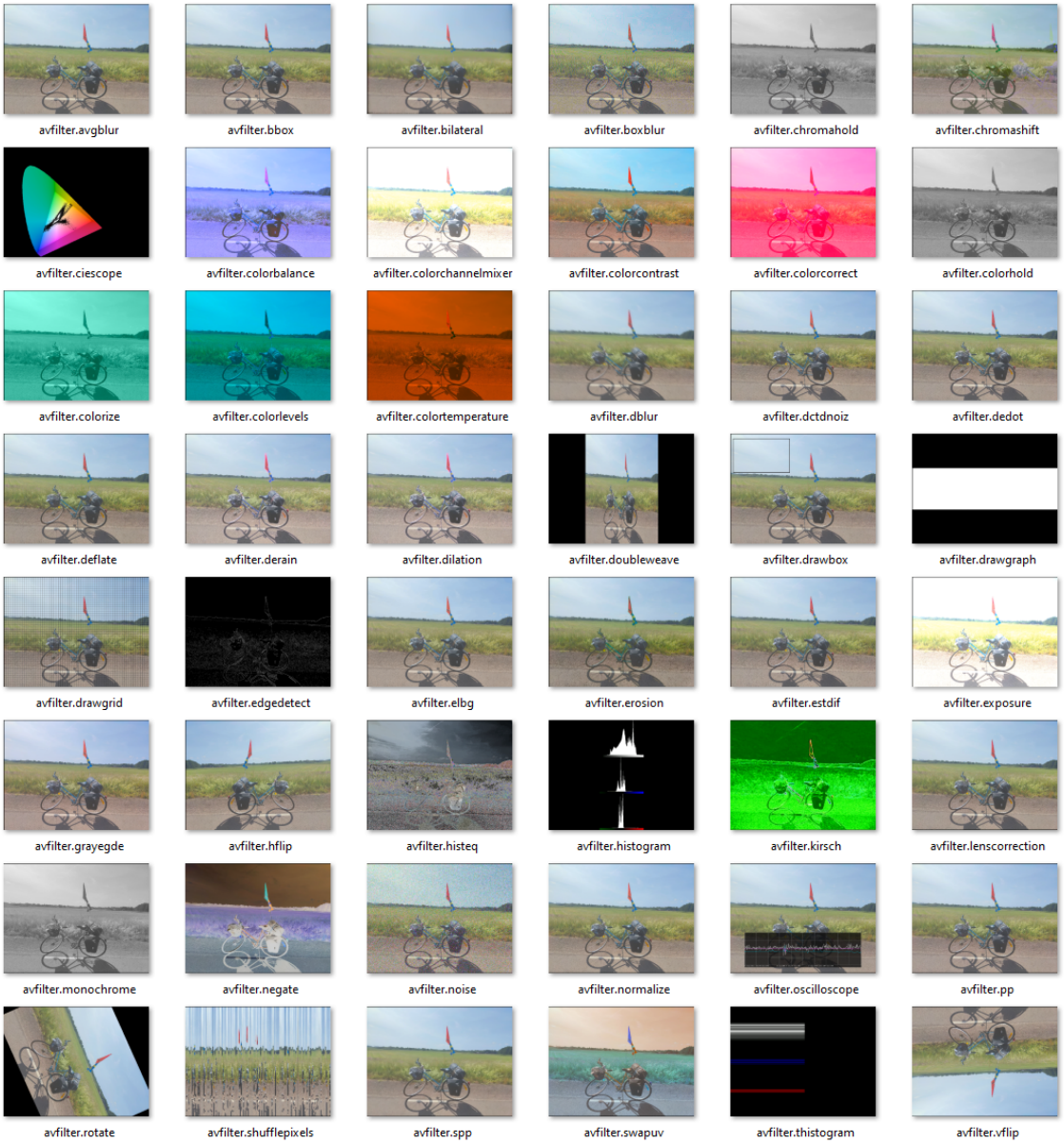
Figure 95: Frame 343 after the application of the maximum value for blue in the Accurate Player.

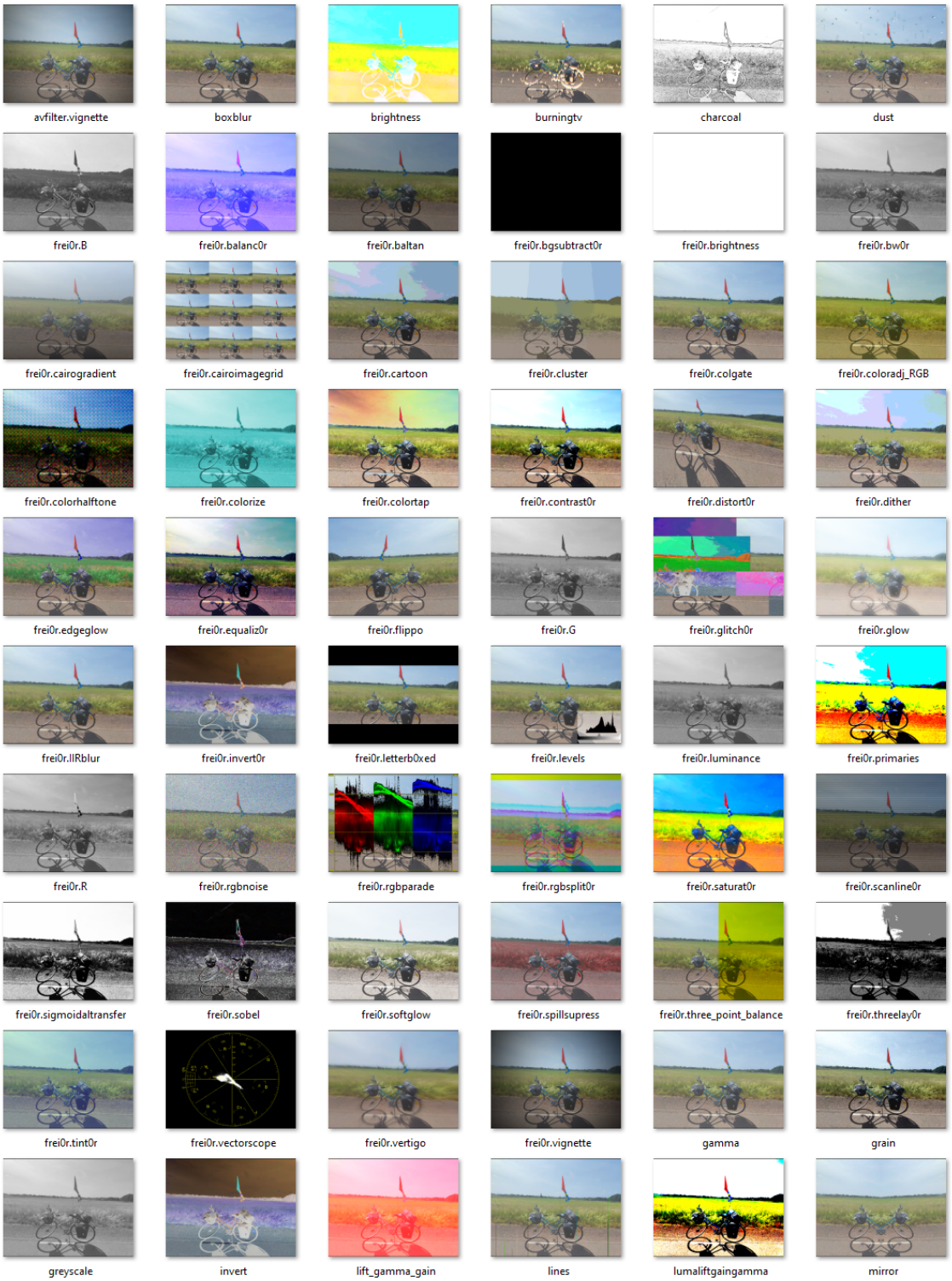
Figure 96: Frame 343 from KDEN Live with the action *Multiply* in the blue filter application.

Figure 97: Subtraction of the two frames: Accurate Player frame is subtracted from the KDEN Live frame extraction (with the action *Multiply*).

C Different MLT Filters

In the following, different MLT filters that influence the visual appearance of a video are listed. This list does not contain every filter and the applied parameters for the different filter results vary. This is not a complete listing or analysis of the MLT filters but an overview over many of the filters and the options that arise from combining them or adapting the parameters. The complete list of filters without visual examples can be found on the MLT web site [37].




avfilter.vignette
boxblur
brightness
burningtv
charcoal
dust
frei0r.B
frei0r.balanc0r
frei0r.baltan
frei0r.bgsubtract0r
frei0r.brightness
frei0r.bw0r
frei0r.cairogradient
frei0r.cairoimagegrid
frei0r.cartoon
frei0r.cluster
frei0r.colgate
frei0r.coloradj_RGB
frei0r.colorhalftone
frei0r.colorize
frei0r.colortap
frei0r.contrast0r
frei0r.distort0r
frei0r.dither
frei0r.edgeglow
frei0r.equaliz0r
frei0r.flippo
frei0r.G
frei0r.glitch0r
frei0r.glow
frei0r.IIRblur
frei0r.invert0r
frei0r.letterb0xed
frei0r.levels
frei0r.luminance
frei0r.primaries
frei0r.R
frei0r.rgbnoise
frei0r.rgbparade
frei0r.rgbsplit0r
frei0r.saturat0r
frei0r.scanline0r
frei0r.sigmoidaltransfer
frei0r.sobel
frei0r.softglow
frei0r.spillsupress
frei0r.three_point_balance
frei0r.threelay0r
frei0r.tint0r
frei0r.vectorscope
frei0r.vertigo
frei0r.vignette
gamma
grain
greyscale
invert
lift_gamma_gain
lines
lumaliftgaingamma
mirror


greyscale
invert
lift_gamma_gain
lines
lumaliftgaingamma
mirror
movit.blur
movit.crop
movit.diffusion
movit.flip
movit.glow
movit.lift_gamma_gain
movit.mirror
movit.saturation
movit.vignette
no filter
oldfilm
qtcrop
sepia
tcolor
vignette