Mixed Reality Media: Integration of live video feed in 3D environments

---

Martin Zier

July 24, 2017
Version: Initial Drafting

Beuth University of Applied Sciences

CleanThesis

Department VI: Computer Sciences and Media

Bachelor Thesis

Mixed Reality Media: Integration of live video feed in 3D environments

Martin Zier

1. Reviewer Kristian Hildebrand
Department VI: Computer Sciences and Media
Beuth University of Applied Sciences

2. Reviewer Prof. Dr.-Ing. René Görlich
Department VI: Computer Sciences and Media
Beuth University of Applied Sciences

Supervisors Kristian Hildebrand and Joachim Quantz

July 24, 2017

Martin Zier
Mixed Reality Media: Integration of live video feed in 3D environments
Bachelor Thesis, July 24, 2017
Reviewers: Kristian Hildebrand and Prof. Dr.-Ing. René Görlich
Supervisors: Kristian Hildebrand and Joachim Quantz

Beuth University of Applied Sciences
Department VI: Computer Sciences and Media
Luxemburger Straße 10
13353 Berlin

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Acknowledgement

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

This is the second paragraph. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Contents

1 Introduction

> *If a technological feat is possible, man will do it.*
> *Almost as if it's wired into the core of our being.*
>
> — **Motoko Kusanagi**
> (Ghost in the Shell)

Extending reality with the help of computer generated imagery is no new concept. Ever since real time 3D graphics was possible there was an attempt to extend the understanding of reality. Within the recent years there have been great successes in the industry, most notably in image augmentation was "Pokémon Go" with an estimated install base of 750 million downloads worldwide in June, 2017. [Ann17] Just before this thesis started, Apple and Google showed off their consumer-ready hard- and software for augmented reality experiences.
Virtual Reality Head Mounted Displays have had a similar push in sales with an approximate of 5.83 million sold devices, which range in a sales price between 80 - 900€ for a VR kit, ranging from the very simple Google Daydream View and the very sophisticated HTC Vive. [Erg17] And in these figures are the sales of Google Cardboards missing, which is approximated at around 80 Million.
This generation of computer systems, in which are PC workstations, game consoles and smartphones, is finally sophisticated enough in computation speed and sensor-sensitivity to allow low latency tracking, precise to just a few millimeters.

1.1 Overview

The idea of Virtual Reality (VR) and Head Mounted Displays (HMDs) stems from a cultural need to switch into roles of foreign worlds. Through the advancing development of hard- and software over the last decades emerges a medium which has unmatched immersion and creates an unique, transforming experience into any imaginable environment.

VR and HMDs are now advanced enough for consumer markets - but it stumbles at communicating the experience. Without having ever put on a VR-Headset it is nearly impossible to understand - or even imagine - what the virtual reality experience

means. Any observer of Virtual Reality, usually done by showing what the VR actor is seeing, will not be able to get an understand of the importance and shift of reality perception without wearing the headset himself.
Showing the video output from a HMD as marketing material is contradicting with classic motion video productions. There is even only one famous example where the perspective of a First Person Shooter is reenacted, which was in the overwhelmingly negatively received Doom (2005) movie.

The VR industry, including but not limited to game developers, exhibition creators and creative studios is in need of better communication of their products that includes more than the current headset wearer and allows for a similar, adapted and immersive experience.
The currently method is called "Mixed Reality" (MR) and uses an external camera with the same tracking hardware of the headset to produce a video signal that shows the real world actor with the environment around him. There are currently three main ways of producing MR footage - where as only one variant allows for live compositing with highly accurate imaging results.

1.2 Motivation

My early teenage years started around the time where digitalization and global interconnectivity begun and broadband Internet became commercially available. Suddenly remote multiplayer games, unlimited image sharing - and yes, music sharing, too -, Java-Applets, Flash, HTML framesets and "Marquee" CSS emerged in that medium. 3D Acceleration became a de-facto standard and even simple office PCs got weak, but dedicated graphics processing units built in. The mass of pixels by increasing the resolution of displays was basically a yearly iteration in greater, better, smaller and brighter.

I am personally very interested and invested in Virtual/Mixed/Augmented Reality to succeed and liked the idea to merge multiple forms of media into one - which is, in my personal opinion, a great summary of my studies and its contents. This thesis represents my interests and the reasons why I chose these studies.

1.3 Problem Statement

Initially I will research motion video productions, computer generated imagery and color theory. This leads to the knowledge to implement basic, interactive live motion video.

The core aspect will be integrating a multitude of Hardware in a software that allows for dynamic video compositing in 3D environments at runtime while a user is interacting with the virtual reality scene. This allows that the person using the Vive HMD to be composited into the scenery and it looks like he is in that scene standing. The essential difference between classic post production is, that this system is planned to operate on runtime, allow additional observers to get an interesting composited imagery of what the VR actor is experiencing.
An additional extension is to dynamically track the camer position, allowing for dynamic camera movement and a freely moving actor.

1.4 Challenges & Scope

This thesis discusses the development and usage of a mixed reality setup inside a single PC and a single application.
It will highlight the core motion video production for mixed reality, as it differs from other MR setups, by staying inside the programs boundaries, integrating natively with the engine and the ability to mitigate the effects on different round trip times from the video feed.
As core aspect will be chroma keying in real time rendering discussed, as well as image reproduction from the chroma result. In detail there will be a discourse of layered image composition as result of fore- and background of the VR actor.
Another throughout discussion will be over latency mitigation and alternative solutions to reproduce an accurate mixed reality image.

camera tracking solution is currently missing.

Outside of the scope of this thesis will be greenscreen compositing, as it is used as tool - the same goes for video matting, which would require its own research to allow for accurate realtime results with little to none user configuration.

add the paper about realtime video matting on Nvidia Quattros

1.5 Results

Result stuff

1.6 Thesis Structure

This thesis gets contemplated by digital, mostly motion video, material hosted on GitHub. Print is a great medium, but lacks the ability for short demonstrations of video imaging solutions, problems and edge cases. To visualize these problems properly, all video media will have an annotation for cross referencing on the website. It is strongly suggested to follow these links, they will be sorted by chapters.

2 Extending Reality

> *You are an aperture through which the universe is looking at and exploring itself.*
>
> — **Alan W. Watts**
> (Philosopher)

The well known urban legend of "L'Arrivée d'un train en gare de La Ciotat" in which a train arrives at the La Ciotat station, is, that "the audience was so overwhelmed by the moving image [...] coming directly at them that people screamed and ran to the back of the room". [Wika] With that a new medium was created, which matured into a new art form of film and movies.

This sounds more like prosa text.

2.1 Motion Video Production

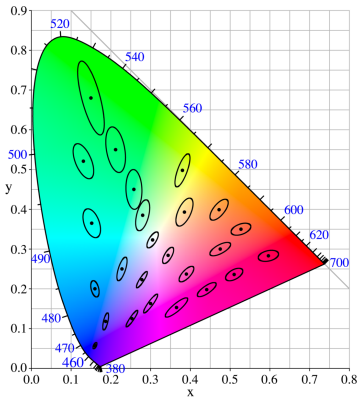
Producing motion video has come a long way and a sufficient history of it would be far out of scope of this thesis. Concentrating on key aspects of composition techniques might give an appropriate overview to range where Mixed Reality takes its inspiration from.

Way before digital imaging processing took over production sets similar problems as discussed in this thesis had to be solved, i. e. how an actor can be captured without a back- or foreground and how he would then be integrated into an imaginative set. Today, modern action movies don't even necessarily capture the actor but his movements, which then will be artificially rendered with computer generated imagery.

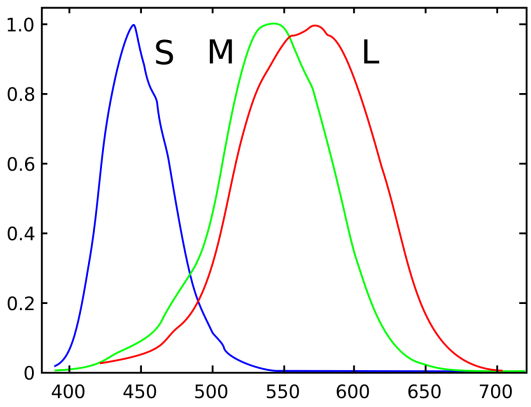
2.2 CGI & Video Composition

2.2.1 History of Green & Blue Screen Productions

Set theory is beyond the scope of this thesis, but green- and blue screen production has first and foremost a simple reasoning: Green and blue are two of the three color triplets that resemble a least amount of color of humans - and to a certain extent any flora and fauna. Since chroma keying (see Ch. 4.1) takes color distance as general basis, production environments generally use green screen keying.
Greenscreens abuse a correlating advantage that the human eye is most susceptible to green, allowing for a visual high color range and an ability to differentiate between many shades of green. Experiments to color range have been done since 1942, trying to understand color ranges and color difference of human vision. Experiments conclude that eye cone cells see a blending range of wavelengths to different intensities, giving the green vector space its highest range. [Mac42]



(a) MacAdam ellipses on 1931 standard chromaticity diagram [Wikb]



(b) Normalized responsivity spectra of human cone cells, S, M, and L types after Wyszecki et. al [Wikc]

The layout is a bit wonky here.

Most consumer cameras - and even production cameras - use dot-matrix sensors with a weighted ration of green (4), red (2) and blue (2) pixels, called Bayer pattern [Bay76]. Green is generally easier to light, illuminate and adjust over blue screens. Small irregularities, for example through uneven lightning or crinkles, can be adjusted easily by the user and allows for a relatively clean camera image.

2.3 What's VR - Differentiation of AR, VR & MR

In search of an appropriate abbreviation for computer-enhanced realtime imagery a recent addition is "XR", where *X* is a letter of your choice. Definitions are getting more diluted and generally describe a technique, rather than an apparent effect by now.

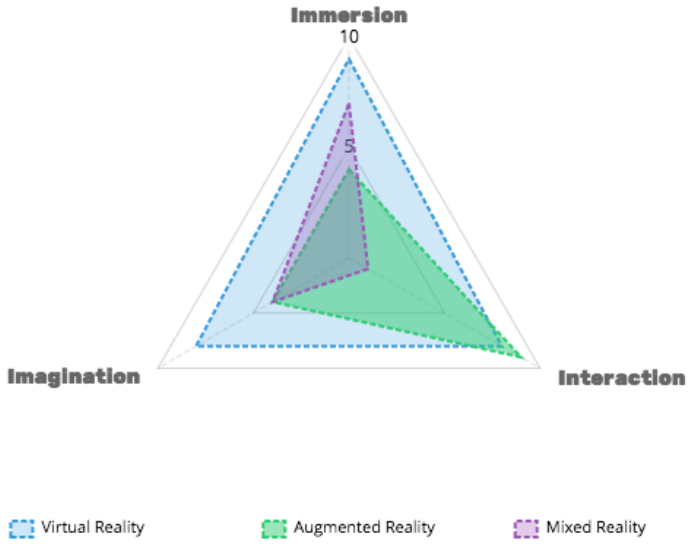


Fig. 2.2.: I^3 Triangle - figurative quantization of different reality extending methods

Augmented Reality is a concept to augment real world imagery with additional information. It ranges from very simple devices displaying data in the field of view of the user up to full augmentation, displaying 3D models overlayed of real world objects. This can be done ranging from Pepper's Ghost projections, to augmenting video - a famous example is the rather successful "Pokémon GO" -, up to the Microsoft HoloLens, that has sensors for a wide range of spatial mapping, spatial anchoring and distance calculation.

Virtual Reality is usually done by stereo projection of a 3D environment on a Head Mounted Display. It takes a user out of the current room and puts him into a complete new, virtual reality. Its hardware ranges from the simplistic Google Cardboard to the Samsung GearVR up to the Oculus Rift and HTC Vive. The latter two products offer room-scale experiences where a user is able to move freely in his play space (basically a bounding box) and allows for six degrees of freedom (6DOF) tracking.

Mixed Reality is an extension of Virtual Reality, allowing bystanders to get an impression of the virtual reality around an actor. By reproducing virtual projection parameters of a 3D environment, it is possible to place a real world camera feed at the right position inside the 3D application. This yields to a combined technique of Augmented and Virtual Reality. A production environment can be achieved with a Six Degrees of Freedom (6DOF) HMD and additional - either user- or tracking input of - positional parameters for the camera.

2.4 Immersion vs. Communication

maybe the overview does already a "good enough" job to bring this across.

Virtual Reality, as previously mentioned in 1.1, is very immersive but the experience is hard to imagine without wearing a HMD yourself. Additionally doesn't VR offer any ways to allow observers a similar experience as the VR actor.
A very obvious problem starts on interaction. A VR user doesn't always need to see his hands to interact with a scene, due to the natural way of holding these controllers in his hands and directly translating to interaction inside the virtual reality scene. An outside viewer however does not see hands and will not understand actions performed by the user. Any usage context that happens off-screen cannot be communicated and therefore will be lost.
A recent game example, Rick and Morty: Virtual Rick-ality, tries to mitigate this issue by placing virtual CCTV cameras into the scene, which can be controlled through bystanders - giving a neutral third-person view into the three-dimension scene. The VR actor is replaced as a loose avatar representing a figure (Morty) from the cartoons universe.
Mixed Reality merges the actors and virtual realities context, allowing outside viewers a comparable window into the actors experienced world. In fact, initial promotional material for the HTC Vive showed mixed reality footage, produced by one VR computer and a secondary composition PC [Lei]. It's setup is comparable to the one in this thesis and differs by using more than one software context.

2.4.1 Evolution of Virtual Reality Footage

2.5 Mixed Reality and its use cases

Summarize.

3 System Setup

The following section describes the hard- and software components used for the thesis and results. All demonstrations have been performed on that environment. All dependencies have been explicitly marked to allow a similar, but not exact, setup to reproduce these results.

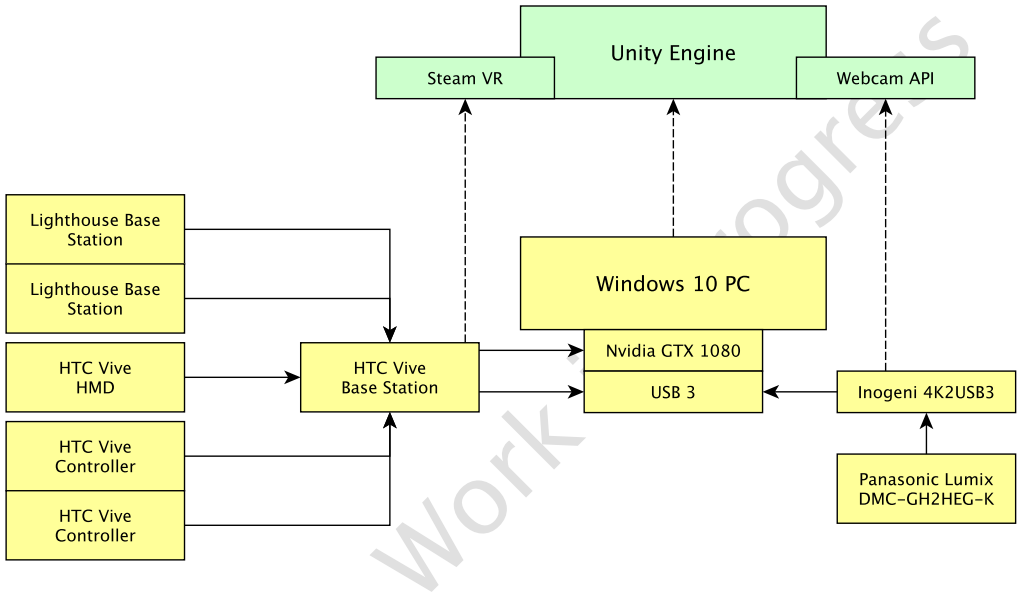


Fig. 3.1.: Diagram of hard- and software components.

3.1 Hardware Configuration

The hardware configuration is split in three main parts:

1. Windows PC Workstation
2. Virtual Reality Tracking Solution
3. Motion Video Input Feed

Each individual configuration is basically interchangeable with other systems, as long as predefined conditions are met. Each condition is listed first in each subsection.

3.1.1 PC Workstation

As the software is built in the Unity Engine, the workstation is limited to either Windows or Mac OS X systems the only requirement - besides being powerful enough to render the 3D scenes - is two USB3 ports to ensure enough data throughput for the video and virtual reality solution, as well as two video outputs for a monitor and its headset.
The configuration used here is:

```
CPU: Intel i7-4700K @ 4.00 GHz
RAM: 16GB DDR4
GPU: Nvidia GTX 1080
```

This system configuration is to date a high end workstation that has an abundance of render performance, allowing it to process and keep enough framebuffer for the operation described further in.

weak text.

3.1.2 Inogeni 4K2USB3

The Ingoeni 4K2USB3 converter is a standalone box that allows to receive any HDMI source and converts it as external webcam video feed. It's advantage is by the arbitrary choice of video cameras and a very simple integration with any software through the systems provided webcam API. With the help of the converter box it's possible to request a webcam as video resource and process that video feed as a texture on the GPU.

3.1.3 Panasonic GH2 Systemcamera

This camera provides a direct video feed via HDMI with low latency. It can directly feed into the Inogeni 4K2USB3 and produces a stable, high quality video feed with a low signal to noise ratio in well lit environments.

still unclear if this remains the target camera.

3.1.4 HTC Vive with Controllers and Lighthouses

The current best virtual reality and tracking device is the HTC Vive. It includes two infrared sending stations called "Lighthouse", two Vive Controllers and a Headset, both systems with 6 degrees of freedom (6DOF) tracking. The tracking system is a blackbox, in which only the transformation matrices for the hand controllers and the HMD can be accessed. By default this transformation has a normalized length of 1 unit to 1 meter. Designing scenery and sense of size is therefore rather easy. The data providing is done by a library called "SteamVR for Unity", which makes the usage in engine transparent.

3.1.5 Vive Controller Tripod Mount

Most cameras have a standardized way of mounting tripods. Since the Vive controllers have no reference plane and minuscule differences in mounting angles changes the projection parameters to noticeable effects, it was necessary to build a mount for the camera to keep controller and video equipment transformation in sync, I built a mount that fits on tripod attachment points and keeps the controller locked in the same position.

Fig. 3.2.: Camera mount for a HTC Vive controller

add example for incorrect projection and model of mount

3.2 Software

The software of choice is Unity3D, which is free for students, non-profit organizations and small studios. It provides an easy introduction to game / 3D engine programming and has a huge development community. While it is not the technologically most advanced engine, its fairly easy usage and fast development cycles make it a great tool for a bachelor thesis.
Thankfully, the high abstraction of system APIs means that cross-platform development only needs a single code base and makes excruciating tasks like webcam access simple - so much so that it boils down to one line of code.
It's weaknesses is usually API documentation and - on the downside, too - high abstraction levels from most APIs. In example, Unity relies on its own shading language which cross-compiles to HLSL, OpenGL and WebGL - this leads to problems in framebuffer management, which cannot be controlled well inside the engine.

The software discussed in this thesis integrates and depends additionally on SteamVR, which is a library integrated with Unity, providing the necessary tracking data in the engine. SteamVR is developed by Valve, the software is available for free on Steam, the complementary library is hosted on GitHub. As of writing this thesis, SteamVR is available for Windows and Mac OS X and this software works on both systems.
There are no further dependencies or external libraries used.

4 From Video to Mixed Reality

Needs better sourcing.

To achieve a real time rendering environment, as previously mentioned, there are two main production cycles. The one discussed in this thesis resolves this problem by staying inside one application with multiple render cycles per frame, the other will be briefly mentioned in related work.
The first and most important render cycle is the stereoscopic output of the Vive HMD, which has a set frame rate of either 45 or 90 frames per second - this is a hard limitation by the providing library, which stalls Unitys' rendering if a render cycle is above $11ms$. It is important to have consistent performance, otherwise the experience for an actor with the HMD will be terrible. This will influence a mixed reality composition, since frame rate targets should be evaluated early in production to fit the composition environment. Since the render pipeline is mostly affected by GPU performance, high fidelity graphics can be achieved with later iterations of graphical hardware. Also recent history has shown that different instancing and render methods can yield massive performance gains.
After rendering a stereoscopic image to the HMD another, secondary render cycle has to be done on the same frame, which is an in-engine camera inside the virtual scene and the relative position of the real world HMD and real world camera. Since the SteamVR library for the HTC Vive already exposes a normalized, synchronized tracking, it is easily possible to position the in-engine camera at an accurate location. This positional and rotational data is achieved by strapping a HTC Vive Controller (or HTC Vive Tracker) to the real world camera and adding another transform to the in-engine render camera to allow for an offset, which is yielded to the difference between sensor location of the video capturing device and the tracking anchor of a controller.

The following chapter describes the techniques used to transform motion video inside a greenscreen into a mixed reality image. As brief overview, the steps required are performed in sequential order from the motion video feed. This is different to the actual render order but gives a better understanding of the techniques used to achieve a mixed reality imagery.

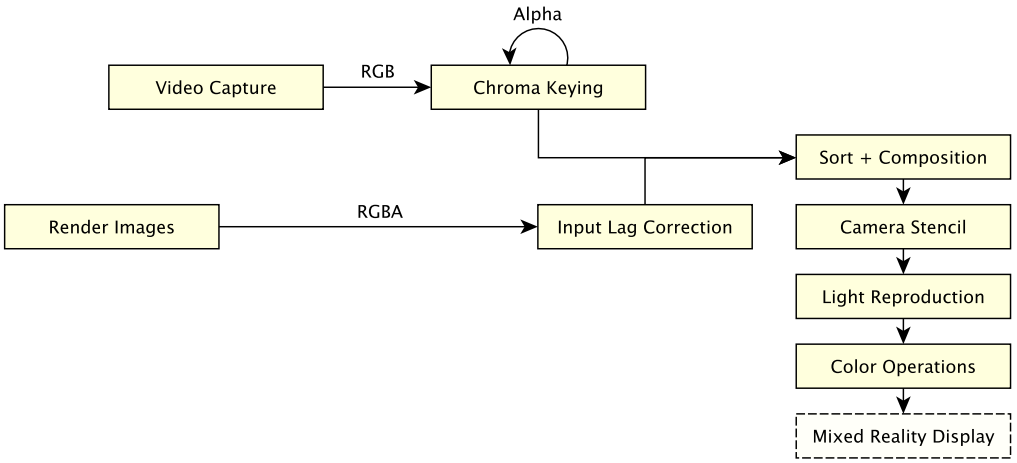


Fig. 4.1.: Full mixed reality graphics pipeline

4.1 Chroma Key

Beginning from the camera, the video signal travels through the Inogeni converter and is accessible with the systems API for webcams. (See figure 3.1)

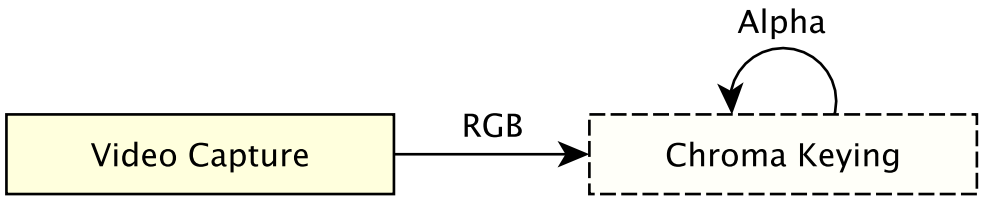


Fig. 4.2.: Initial step upon receiving the camera image

The initial step is to remove the green (or blue) background from the image, which should be either a green (or blue) box, in this case it is a greenscreen. Other literature usually refers to it as "pulling a video matte" or "chroma keying". For a reference green, there has to be a color picked manually in the material editor of Unity - this was made easy by a checkbox to show raw output from the camera. Then a middle-ground green can be picked. This is an important setup step, since lightning situations can vary greatly and minor differences in light setups can have a great effect on the outcome of visible green background captured by the camera.
An extreme example case is used for comparing these chroma keying variants, which shows high motion blue due to a low shutter speed and fast movement of the depicted actor:

Fig. 4.3.: Comparison Image [Vim] - sRGB Output

4.1.1 Initial Assumption

Each RGB color can be represented as a discrete 3-Vector of (red, green, blue) values in range of $[0, 1]$[1]. An interpolation between two colors can be summarized as matting equation as following, where a foreground image C_F and a background image C_B - α_B is assumed to be 1:

$$I(x, y) = \alpha(x, y)C_F(x, y) + (1 - \alpha(x, y))C_B(x, y) \tag{4.1}$$

This matting equation has to be generalized for a later step, where α is a value between [0, 1] on fore- and background, yielding a total Alpha of α_T as following:

$$\alpha_T = \alpha_B * (1 - \alpha_F) \tag{4.2}$$

thus:

[1] Software RGBA color representations usually take 8bit per color channel and map between $[0, 255]$ - graphic computing usually maps between $[0, 1]$

$$I(x,y) = (1 - \alpha_T)F(x,y) + \alpha_T B(x,y) \tag{4.3}$$

4.1.2 Euclidean RGB Difference

Assuming a source pixel color C_S and a reference color C_R we can calculate the euclidean distance between these colors.

$$\alpha = \sqrt{(C_R R - C_S R)^2 + (C_R G - C_S G)^2 + (C_R B - C_S B)^2} \tag{4.4}$$

This is computationally very low cost and works well enough for tell a difference between two separate colors. It fails to accommodate for colors that are perceived as different, but are tinted by the reference colors. Since the greenscreen material will never achieve 0% reflectivity, some color will spill onto the filmed actor and will therefore generate unwanted chroma-keying artifacts, most noticeable on semi-glossy reflection of skin or color tints on white clothing.

Fig. 4.4.: Chroma Keying by using euclidean RGB distance

4.1.3 Euclidean YCgCo Difference

YCgCo gets its name for Luminance (Y), chrominance green (Cg) and chrominance orange (Co) and helps decorrelating color spaces by splitting color-lightness from color chrominances. Since it is a fast, lossless color transformation it is used in example for H.264 video encoding and other image compression techniques. The two chrominance channels are then split into green to magenta and orange to blue color channels and allow for a more accurate distance calculation between two colors.
Transforming any RGB color to YCgCo can done with a single matrix multiplication, which is - again - a very low-cost computation on a GPU:

$$\begin{bmatrix} Y \\ Cg \\ Co \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} \\ \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.5}$$

Given two colors, one from the video source C_S and a reference color C_R it is now possible to calculate the euclidean distance on the two chrominance channels:

$$\alpha = \sqrt{(C_R Cg - C_S Cg)^2 + (C_R Co - C_S Co)^2} \tag{4.6}$$

Since the increased decorrelation, the result is more accurate and shows less artifacting on target pixels, allowing for a more accurate matte pulling, less green edges and a more continuous image of an actor.

4.1.4 Euclidean Lab Difference

The International Color Consortium (ICC) defined 1976 *Lab* ΔE as a standard model of calculating color differences with *Lab* colors. The final distance calculation is, too, a linear euclidean distance as with all other models, but accommodates for perceived color differences. It is a more expensive computation wise and needs in implementation code branches, which will both be evaluated by the GPU before using either branching code path.

Since any given RGB color has to be converted to the Lab color model a reference white has to be used to accommodate for different color models. Luckily the given webcam signal defaults to sRGB D65 and does not need a configurable reference matrix based on a given color model.

Fig. 4.5.: Chroma Keying by using euclidean YCgCo distance

this is very rough and only contains equations currently used

sRGB conversion to linear RGB in respect of energy per channel:

$$v \in \{r, g, b\} \wedge V \in \{R, G, B\} \tag{4.7}$$

where:

$$v = \begin{cases} V/12.92 & \text{if } V \leq 0.0405 \\ ((V + 0.055)/1.055)^{2.4} & \text{otherwise} \end{cases} \tag{4.8}$$

from there and RGB to XYZ conversion can be done by:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} M \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.9}$$

where:

$$\left[M\right] = \begin{bmatrix} RX_r & GX_g & BX_b \\ RY_r & GY_g & BY_b \\ RZ_r & GZ_g & BZ_b \end{bmatrix} \tag{4.10}$$

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \left[M\right] = \begin{pmatrix} X_r/Y_r & X_g/Y_g & X_b/Y_g \\ 1 & 1 & 1 \\ \frac{1-X_r-Y_r}{Y_r} & \frac{1-X_g-Y_g}{Y_g} & \frac{1-X_b-Y_b}{Y_b} \end{pmatrix} \tag{4.11}$$

Where $\left[M\right]$ for RGB D65 is:

$$\begin{bmatrix} 0.4124564 & 0.3575761 & 0.1804375 \\ 0.2126729 & 0.7151522 & 0.0721750 \\ 0.0193339 & 0.1191920 & 0.9503041 \end{bmatrix} \tag{4.12}$$

Based on a reference white $U_r \in \{X_r, Y_r, Z_r\}$:

$$U \in \{X, Y, Z\} \wedge W \in \{L, a, b\} \tag{4.13}$$

$$\epsilon = 0.008856 \wedge \kappa = 903.3 \tag{4.14}$$

where:

$$w_r = \frac{U}{U_r} \tag{4.15}$$

$$f(w) = \begin{cases} \sqrt[3]{w_r} & \text{if } U > \epsilon \\ \frac{\kappa w_r + 16}{116} & \text{otherwise} \end{cases} \tag{4.16}$$

$$\begin{bmatrix} L \\ a \\ b \end{bmatrix} = \begin{bmatrix} 116 f_y - 16 \\ 500(f_x - f_y) \\ 200(f_y - f_z) \end{bmatrix} \tag{4.17}$$

With this conversion from sRGB to linear RGB to XYZ to Lab we can now calculate the euclidian linear distance between two colors C_1 and C_2, which already have been converted to Lab:

$$\Delta E = \sqrt{(C_2L - C_1L)^2 + (C_2a - C_1a)^2 + (C_2b - C_1b)^2} \tag{4.18}$$

These values are rated by their perceptive difference [MW]:

0.0 . . . 0.5	the difference is unnoticeable
0.5 . . . 1.0	the difference is only noticed by an experienced observer
1.0 . . . 2.0	the difference is also noticed by an unexperienced observer
2.0 . . . 4.0	the difference is clearly noticeable
4.0 . . . 5.0	fundamental color difference
> 5.0	gives the impression that these are two different colors

Now it's possible to map alpha values for each pixel based on ΔE distances between m, n by clamping and biasing ΔE:

$$f(\Delta E) = x = \frac{\Delta E - n}{m - n} \tag{4.19}$$

$$\alpha_{\Delta E} = \begin{cases} n & \text{if } x \leq n \\ x & \text{if } n \leq x \leq m \\ m & \text{if } m \leq x \end{cases} \tag{4.20}$$

$$\alpha(I(x, y)) = 3\Delta E^2 - 2\Delta E^3 \tag{4.21}$$

With that we receive a more natural matte with nearly no green edges, continuous actor imagery. Motion blur is hard to account for and is even with professional video matting hardware, extensive post production or intrinsic frame matting hard to remove without very rough image results.

4.2 Camera Input Lag

After rather simple integration of the keyed video signal into the scene, where the 3D environment is used as background, an offset between the render image from the

Fig. 4.6.: Chroma Keying by using ΔE distance

scene and the captured footage from the camera can be observed due to the pipeline from camera through video converting of the Inogeni 4KUSB3 bound to the systems webcam API.

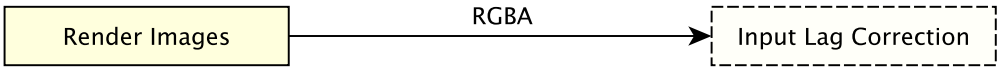


Fig. 4.7.: Initial step upon receiving the camera image

After consulting the specification of the Inogeni 4K2USB3 it states that a conversion from any HDMI video source takes two intrinsic frames for encoding. The camera framerate is $F_C = 25\frac{frames}{second}$. This would mean, in theory:

$$t = \frac{2}{F_C} \tag{4.22}$$

Assuming 25 frames per second, that is $\frac{1}{30}s$:

$$t = \frac{2}{25\frac{frame}{second}} \tag{4.23}$$

$$t = 80ms \tag{4.24}$$

The observed offset from camera to engine is far longer in reality and remains at about 260ms after testing this setup and observing the drift, therefore noticeable in any motion video as shown in figure 4.8.

Fig. 4.8.: Example of the visual difference between the (left) real world capture and (right) the received imagery.

To mitigate this offset there are two options:

1. Change the camera setup - in example for a webcam, which usually has a lower input latency, ranging between 5 - 250ms. That would degrade the image quality significantly but would enable better rendering conditions inside the engine.
2. Capture virtual images of the 3D environment and keep them on the GPU until a real world video frame is loaded onto the GPU for usage. This keeps the image quality but needs significant effort to reproduce the rendering conditions at the engine time when a video frame was taken.

The proposed solution uses the second option, since it is able to minimize any kind of offset between render image and a video stream, the hardware setup can stay dynamic and it is little to no difference by switching to a webcam-integrated solution, than an encoding box and the resulting displayed image has imperceptible differences to the former variant. It is therefore more user friendly and can accommodate for a wide range of video devices.

This graphic is outdated. FPS and such - also a timeline is missing, which has to be thrown in there by hand

Based on the component diagram 4.9 there are two important takeaways:

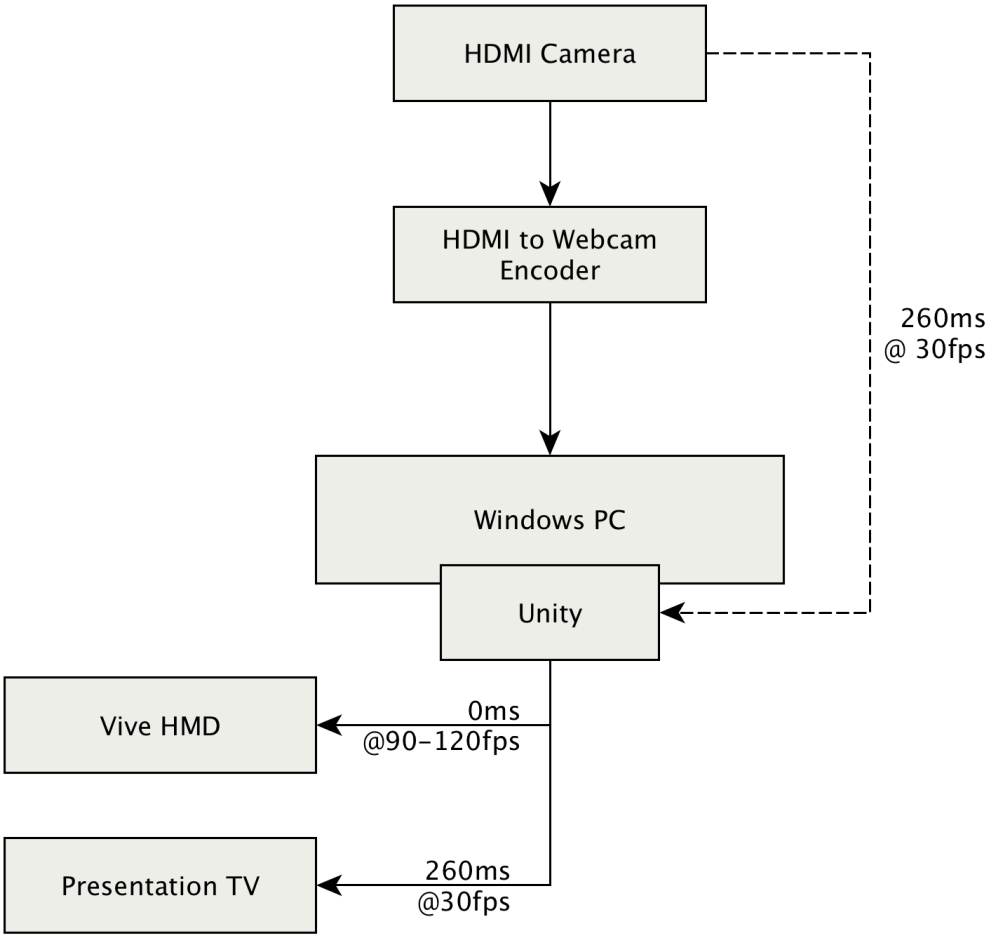


Fig. 4.9.: Components in considering video input lag and frame rates. While latencies between each components cannot measured, it is observed with help of an interactive VR object.

1. There is a input latency between the production camera and the Unity Engine, which in turn need to be reproduced on the mixed reality presenting device
2. Frame rates between the Vive HMD and the presentation TV differ because the TVs frame rate should be matched to the input video feed from the camera

At time of writing Unity does not support dynamic frame rates on multiple viewports[2].
It is possible to manually initiate a render, however, this causes the render loop to mistime and yields to inconsistent frame timings inside the HMD and is no different between the average GPU load to display the presentation TVs viewport. It makes no real difference if a scene renders 11ms twice and then falls down to 22ms once - the observed stuttering is a major degradation of the actors experience.
To conclude: The software has to store a set amount of framebuffers and cycle them

[2] A viewport can be either a D3D, OpenGL or Metal context

at the right frame to guarantee minimal delays between camera and 3D environment. Noteworthy is that the render loop can be 45 or 90 fps, depending on scene complexity, overall system performance or - especially in case of Unitys outdated C# version - garbage collection, that could halt the engine for a significant amount of time. To account for this a strategy is needed in which Unity's `Time.deltaTime` property is used, which describes the time between last and current frame.

4.2.1 Framebuffer Swapper implementation

Unity has a well engineered engine loop, where it can perform different operations in specific steps of engine execution. For more detail about Unitys core loop[3]. As initial data needed is `cameraFPS` , `cameraOffset` and `Time.deltaTime` . From there it is possible to calculate the remaining data for this algorithm:

```
frameWindow = 1.0 / cameraFPS;
delayCnt = cameraOffset / frameWindow;
frameDelay = (int) delay * frameWindow;
fractionDelay = delay % (1 * frameWindow);
innerTimer = 0.0;
absoluteTimer = 0.0;
while(true) {
        innerTime += Time.deltaTime;
        absoluteTimer += Time.deltaTime;
        localTime = innerTimer - fractionDelay;
        if(localTime < frameWindow ||
           absoluteTimer < initialDelay) {
                continue;
        }

        innerTimer %= frameWindow;
        absoluteTimer %= (1f + initialDelay);
}
```

this code listing is the wrong one :(- also I am missing representative material why we need to mitigate the latency mitigation

In Unitys' case a framebuffer is called `RenderTexture` and allows for either data or depth buffer access, where a framebuffer would allow for simultaneous access.

[3] there is a flowchart in the Appendix B, depicting MonoBehaviors life cycle.

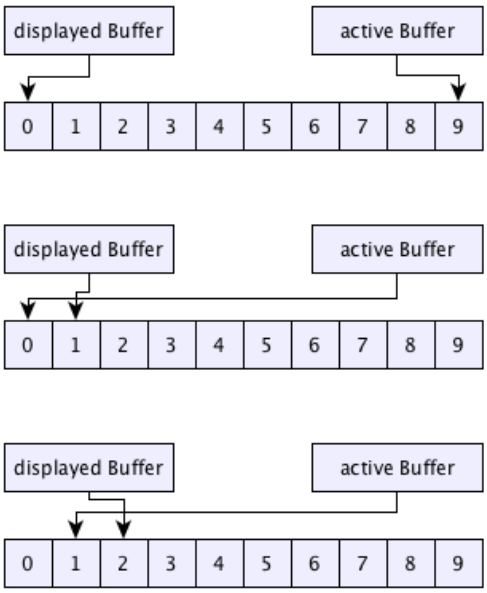


Fig. 4.10.: Schema of an the ringbuffer

4.2.2 Double Access Ringbuffer

To spare memory overhead it is possible to reuse previously allocated `RenderTextures` by overwriting the second oldest `RenderTexture` and compositing a mixed reality image with the oldest `RenderTexture` (see Figure 4.11).

To accommodate for that behavior we have to simply write frame data into the current index and display the frame on its next index. After that step taken we increment by one. This way we're overwriting the oldest seen RenderTexture and show the one written after it.

```
class DoubleAccessRingBuffer<T> {
        public int bufferSize;
        private List<T> buffers;
        private int index;

        public DelayedRingBuffer(int bufferSize) {
                this.bufferSize = bufferSize;
                index = 0;
                buffers = new List<long>();
                RebuildBuffers();
        }

        public T[] Next(T writeTo, T display) {
                index %= buffers.Count;
```

```
                T writeTo = buffers[index];
                T display = buffers[
                        (index + 1) % buffers.Count
                ];
                if(bufferSize != buffers.Count) {
                        RebuildBuffers();
                }
                index++;
                return new T[]{writeTo, display};
        }

        void RebuildBuffers() {
                buffers.RemoveAll(_ => true);
                for(int i = 0; i < bufferSize; i++) {
                        buffers.Add(new T());
                }
        }
}
```

Needs "real" pseudo code, rather than implementation

4.3 Mitigating Frame Jitter

An additional step that can be handled by the same algorithm is the mitigation of frame or time jittering, a term describing an effect of different running framerates of an actors captured video footage and rate of 3D environment rendering. Since the native framerate of the HMD is higher than the frames produced by the input video feed, there will be noticeable small jitters of virtual camera movement, which are not present on the source video material. The HTC Vive Controllers are very good at picking up minuscule changes in motion, which are instantly translated to the transform of its camera rig and then yield minor motion of the 3D environment projection. Visually this shows in a shaking virtual world while the real world video stands still.

To minimize the effect, it is possible to overwrite framebuffers for as long as one duration of a video frame and then swapping out these buffers. The headset recommends to run it at 90fps and miss timings are no issue either, since it results in non noticeable errors in virtual projections while the displayed frame is stable, thus visual performance is unaffected. It is possible to write less than three times into a framebuffer, which is mitigated again by displaying the final result later - the

composed image is therefore unaffected besides imperceptible differences between the real world camera and its virtual transformation matrix.

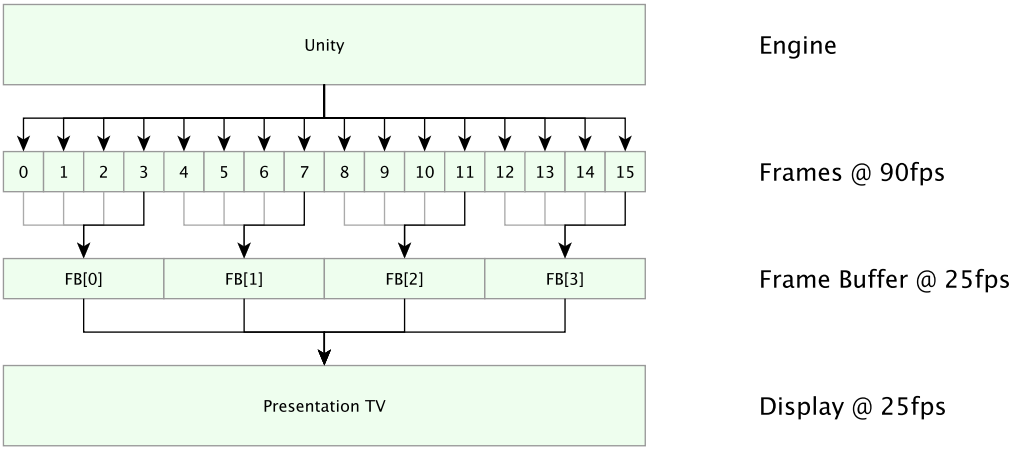


Fig. 4.11.: Workflow of the render swapper, in which rendered frames will be overwritten as long as it is needed - and then another frame buffer will be written into.

needs then lstlisting when the code further up is modified

4.4 Virtual projection parameters from real world camera

To produce a virtual projection there are four unknowns to solve for:

1. Position of the real world camera
2. Rotation of the real world camera
3. Field of View (FoV) of the camera
4. Distance between the HMD and the real world camera

Luckily the former two are solved by the tracking solution, thus can be used directly as transformation matrix for the virtual camera - ignoring an additional offset from the actual controller to the camera, which is accounted for inside the software.
The calculation of a corresponding distance between a camera and the Vive HMD to control the virtual projection parameters can be solved, too: Since both devices are tracked, one natively and the other with a controller as tracking anchor, it can be calculated with the same euclidean distance as discussed earlier, with C as camera position and H as HMD position:

$$Z = \sqrt{(C_X + H_X)^2 + (C_Y + H_Y)^2 + (C_Z + H_Z)^2} \tag{4.25}$$

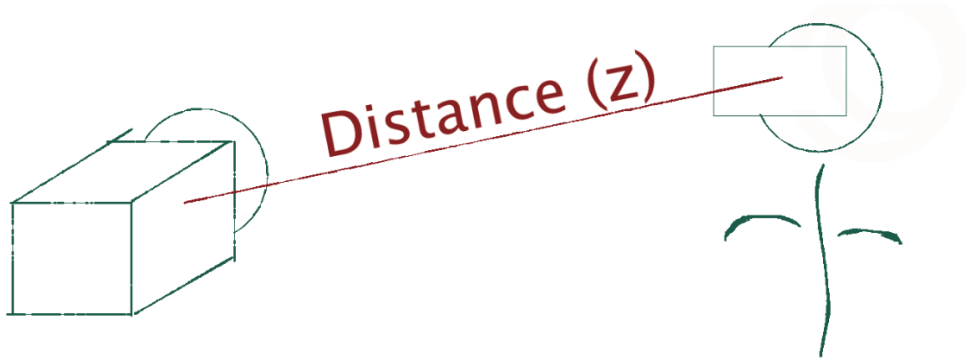


Fig. 4.12.: Distance correlation

Another important projection parameter is field of view. Most production cameras only declare a focal length on lenses - which makes sense in that context, since field of view is a constraint between sensor size and focal length. Through the specification sheet of the camera the current field of view can be calculated inside Unity, with the sensor height as $S_h = 17.3mm$ and focal length as $F_l = 14mm$:

$$FoV = 2 * tan^{-1} \frac{S_h}{2 * F_l} \tag{4.26}$$

$$FoV = 63.42028° \tag{4.27}$$

With that we have now all projection parameters for the virtual environment to generate an image that matches in relative transformation parameters as the real world camera would look at.

4.5 Virtual Z Sorting

We have now a properly keyed video feed and synchronous motion between a VR actor and the 3D environment. To increase the immersion of that composition the next important step is to sort the scene on a tri-graph of planes.

There are multiple ways to achieve proper segmentation and composition of all three layers, depending on the rendering method. Deferred shading allows for better lightning simulation in engine but changes alpha- and depth maps of a rendered scenery - this yields incorrect layer blending and results into an incorrectly displayed image. This software takes account for this and lets creators decide between two render modes:

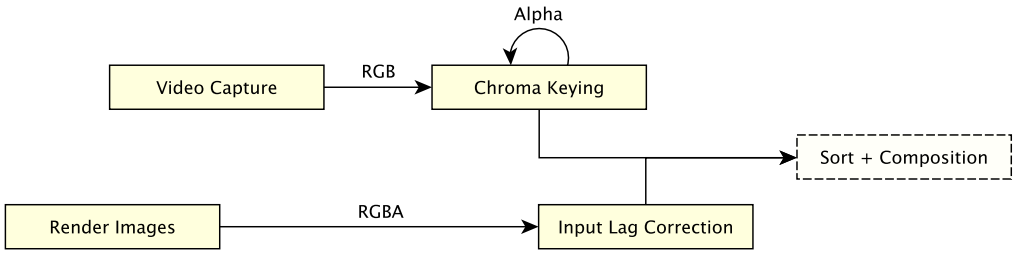


Fig. 4.13.: Current steps taken through the graphics pipeline

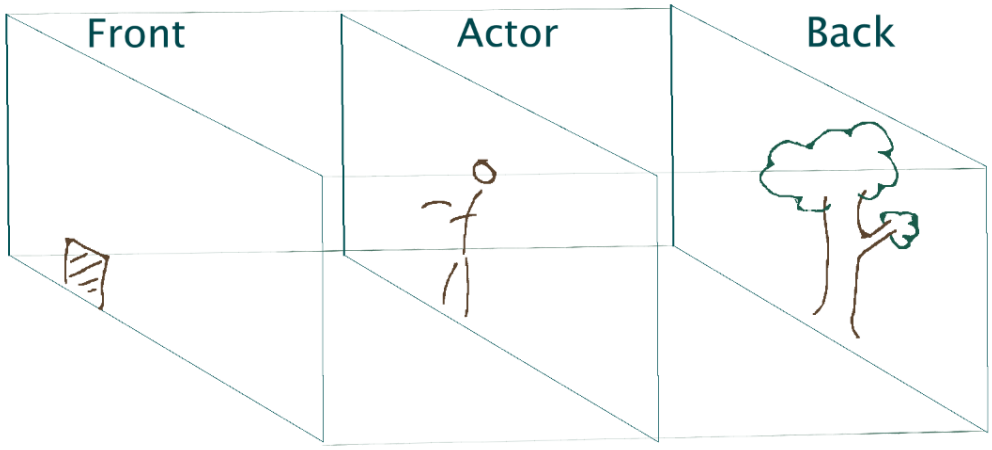


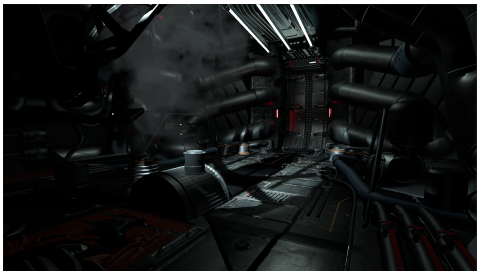
Fig. 4.14.: A sketch of the video composition with three layers of projection

1. Replace Masking: A front plane is displayed, after it follows a chroma-keyed video and then the background. This is the most accurate image generation, if the front image is generated correctly
2. Alpha Masking: A front alpha mask of the geometry is being generated, then the actor is mixed with a full render image of the background. The resulting image has inconsistencies with alpha-blending but this method works with all rendering setups.

The decision is between accuracy and presentation. Many post processing effects or deferred shading paths are not able - or simply do not respect - the alpha matte, which is usually no issue, since these steps are taken after rendering is complete, thus causing no unwanted side-effects in regular rendering setups. Other post effects need certain projection requirements and / or get discarded through culling, like in Figure 4.16e where the volumetric lightning is culled out of the virtual projection and therefore gets completely ignored by the attached compute shader, resulting in a black front geometry.

Fig. 4.15.: A comparison of different composition methods in engine

(a) The proposed composition, simulated, with an arbitrary depth of the video feed



(b) The full length rendering

(c) A composition by rendering a front, followed by the video and then the background

(d) A composition with the front geometry as mask, and then a mixing of the video and a full length render

(e) The virtual projection of the front camera - volumetric lightning does not work due to the short projection length

(f) The chroma keying result

Mixing these three layers are similarly effortless, using the previously established matting equation (4.2) and (4.3). Assuming an ARGB front render image C_F, a RGB video feed C_V and the RGB background C_B, we can mix all three layers:

Replace Masking:

$$\alpha_{C_T} = \alpha_{C_V} * (1 - \alpha_{C_F}) \tag{4.28}$$

$$I(x,y) = (1 - \alpha_{C_T}) * C_F(x,y) + \alpha_{C_T} * V(x,y) \tag{4.29}$$

$$\alpha_{C_S} = \alpha_{C_B} * (1 - \alpha_{I(x,y)}) \tag{4.30}$$

$$J(x,y) = (1 - \alpha_{C_S}) * I(x,y) + \alpha_{C_S} * C_B(x,y) \tag{4.31}$$

Alpha Masking is very similar by transforming the alpha-mask of the webcam footage after chroma-keying it. It is masking the video further, after the video matte is pulled already:

$$\alpha_{V_T} = \begin{cases} 1 - \alpha_{C_F} & \text{if } \alpha_{C_V} > 0 \\ \alpha_{C_V} & \text{otherwise} \end{cases} \tag{4.32}$$

This is a bit incorrect as what is currently used in the shader.

$$\alpha_{C_T} = \alpha_{C_B} * (1 - \alpha_{C_{V_T}}) \tag{4.33}$$

$$I(x,y) = (1 - \alpha_{C_T}) * C_V(x,y) + \alpha_{C_T} * C_B(x,y) \tag{4.34}$$

remindme: there is the depth-offset missing currently.

Now we have a well mixed image composition where the actor is placed in between two projections and thus can have an interactive fore- and background. The initial assumption is, that the actors depth is flattened, based on the distance between a real world camera and the Vive Head Mounted Display.

4.6 Additional Camera Stencil

When producing on small and / or amateur sets, there are usually constraints to size and proportions of the greenscreen production, thus limiting the record-able space. Since a calibrated play space can be fetched from the SteamVR API to receive a proper-sized bounding box, it is possible to do a projection of the greenscreens real size inside a virtual scene and use this as a stencil for the incoming video feed, effectively cropping off around all edges outside of a calibrated greenscreen. Effectively this means that a real world camera can film outside the edges of a green screen and it will not degrade the visual performance of the image composition.

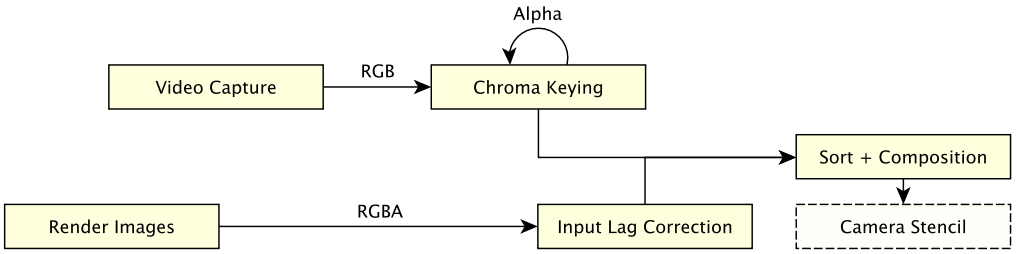
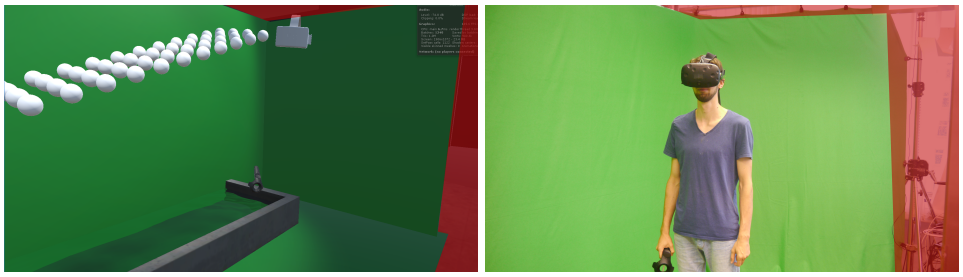


Fig. 4.17.: After sorting the scene additional stenciling is necessary

A virtual projection can be seen in figure 4.18 with reconstructed camera parameters. With help of the engine-editor a greenscreen can be calibrated and should match very closely to all real world parameters, allowing a real world camera to film over the edges of a greenscreen without destroying the image composition. If a VR actor is outside of the chroma key planes, the resulting image composition wouldn't be usable, thus this solution enhances a resulting image without major drawbacks.

In-engine this setup is a simple addition: After registering another camera to the camera manager, it simply renders a green box. Taken from previous projection parameters in 4.4 this virtual camera receives a dedicated culling mask which only contain all green box projection planes. All other cameras ignore this layer. After each drawn frame, Unity is able to clear the framebuffer with a color alpha as 0 - all remaining fragments from the green box write any color with an alpha as 1. This creates a lookup texture where an alpha-mask is created which will be transferred to the video feeds alpha.

Fig. 4.18.: Virtual projection and photo of VR actor - note: in-engine screenshot and photos were taken shortly apart and therefore don't fit exactly



(a) Virtual reprojection of valid green screen - red colored areas will be cut off

(b) Masking what would be remaining video content - red colored areas will be cut off

Now we achieved a green box projection inside the same scene without using complex management processes with Unitys' scene manager. Due to the quick setup it works well for fast calibration and allows for a broad camera angle without degrading the mixed reality performance.
Unfortunately stencil-opertions are poorly documented in Unity and add an overhead which cannot be measured from builtin profiler tools - as such it adds unknown performance costs and have not been used in this thesis.

4.7 Light Environment Reproduction

To conclude by now: We have successfully created an image composition with correct projection parameters, recalculated a planar depth, delayed in-engine frame times with camera frame timings and realigned the frame rate between both image generating systems.

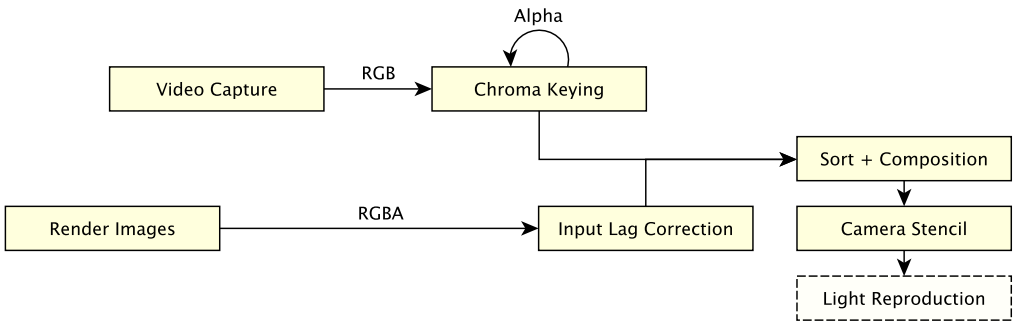


Fig. 4.20.: Following in this pipeline is lights reproduction

A minor last step is light-reproduction, in which an approximate lightning setting will be transferred from 3D environment to the video feed of a VR actor. Assuming that the video footage contains a natural lit, tint-free and calibrated video signal, it

is possible to approximate how a VR actor would be lit like if he truly was inside a virtual environment.

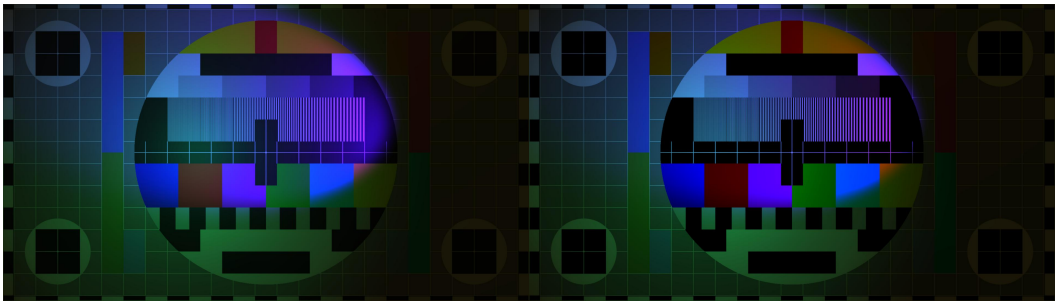


Fig. 4.21.: Left: original, Right: reconstruction

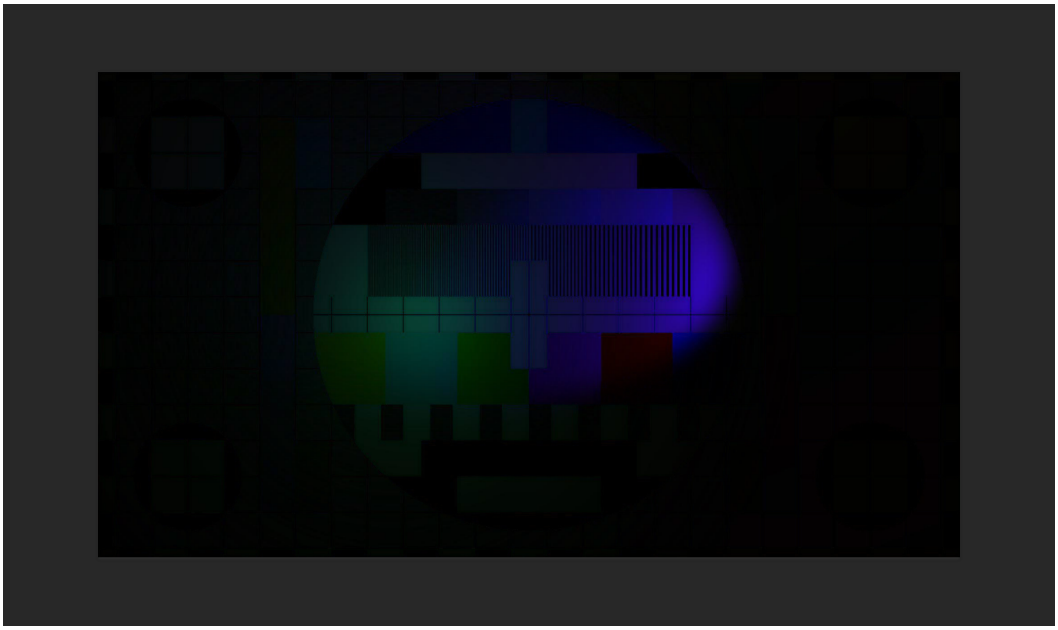


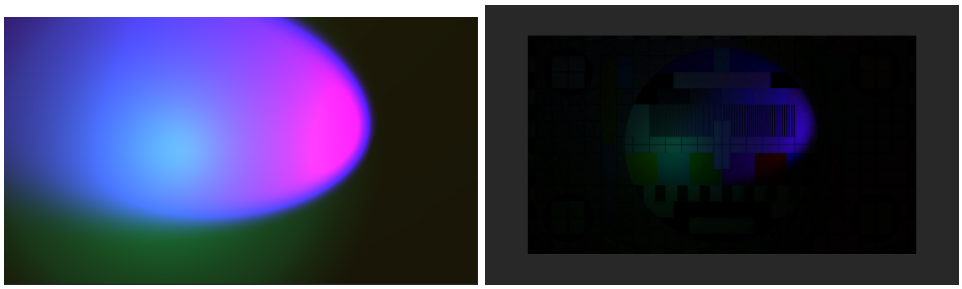
Fig. 4.22.: Color difference by substracting a reconstruction from the original.

Ambient light reproduction hinges on two assumptions: An actors recorded video is flat for this purpose and he receives a clean and consistent light from all angles that has no additional glossiness. With that we can have project a plane at the actors position, filling the frustum edges of another virtual camera with the same camera projection parameters from section 4.4. This plane contains a simple lit material with white albedo coloring, which captures the lightning situation at this given point (figure 4.23 a).

To calculate the position P_{pos} and size $\vec{P_{x,y}}$ with a given forward vector $\vec{C_{forward}}$ and position C_{pos} of the camera, as well as a distance Z between camera and actor and a current Field of View FoV in radians by assuming a 16:9 video feed:

$$P_{pos} = C_{pos} + Z * \vec{C_{forward}} \tag{4.35}$$

Fig. 4.23.: A comparison of different composition methods in engine



(a) captured, lit plane

(b) Color difference by substracting a reconstruction from the original.

$$P_{x,y} = \begin{bmatrix} 2 * \tan(FoV/2) * Z \\ P_x * \frac{16}{9} \end{bmatrix} \tag{4.36}$$

4.8 Additional Coloring Operations

Finally we have created the best possible recreation of a VR actor inside the virtual reality scene. Now we can follow up with post effects on the video feed to fit it to a better degree into the environment. This can be done with regular coloring operations, like hue rotation, brightness, contrast and saturation procedures on the video alone. It gives a content producer direct enhancing tools which would be given by Unitys' post effects stack but are unavailable due to the nature of this render pipeline.

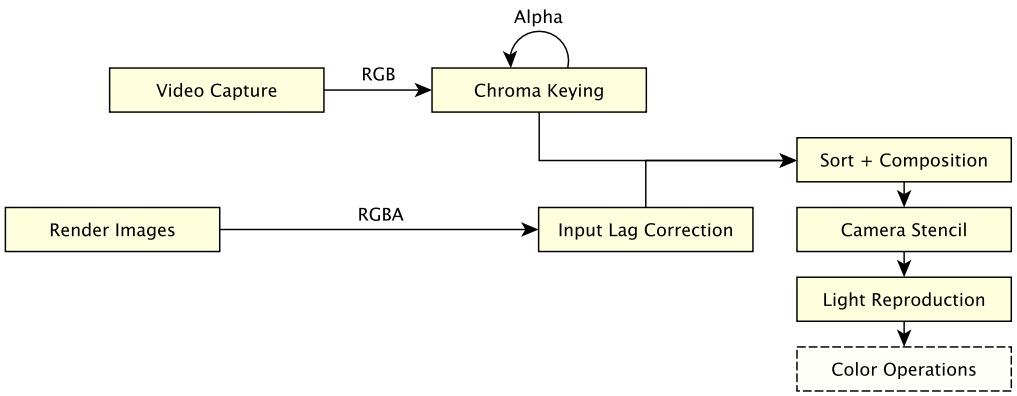


Fig. 4.25.: Initial step upon receiving the camera image

4.8.1 Color spill removal & Recoloring

The green box as background spills - so to say - its green color on the actor to a certain degree. With proper lightning setups it is possible to mitigate its effects but

color retouching is always a necessity. **YCgCo** is, again, good enough to perform this color operation, thanks to its color decoupling properties. By splitting a RGB image into YCgCo C_{Input} (see equation (4.5)) and then shifting towards the anti-color of the key color C_{Key} for a factor weight $W \in [0, 1]$:

$$R = \frac{C_{KeyCg,Co} \cdot C_{InputCg,Co}}{C_{KeyCg,Co} \cdot C_{KeyCg,Co}} \tag{4.37}$$

$$\begin{bmatrix} Y \\ Cg \\ Co \end{bmatrix} = \begin{bmatrix} Y \\ C_{KeyCg} * (R + 0.5) * W \\ C_{KeyCo} * (R + 0.5) * W \end{bmatrix} \tag{4.38}$$

"proper lightning setup" would be something for the appendix

Since this is a linear operation, it does not consider more apparent color spill around an actors edges - it slightly removes a green undertone to make it look more natural and fitting in this scene.

Additionally we can apply a hue color rotation by using Rodrigues' rotation formula to make changes to the overall tint of an image, shifting a color C_I 180 degrees forwards or backwards with a factor $H \in (-\pi, \pi]$ to yield a color C_F:

$$C_F = C_I \cos(H) + (0.57735 \times C_I) \sin(H) + 0.57735(k \cdot C_I)(1 - \cos(H)) \tag{4.39}$$

sourcing on 0.57735 needed

With that we can achieve a more natural looking video feed that integrates well into any given scene. Allowing for color-shifting degrades the signal but allows for a more fitting composition between an actor and his surrounding virtual reality scenery.

4.8.2 Brightness, Contrast and Saturation

Additionally, to give a user full control over image composition, we have a brief look at other linear image transformations to give good control over the video feed, which are brightness, contrast and saturation operations:

```
# aint nobody got time for that - here's HLSL
```

```
# brightness:
rgb = saturate(rgb + Brightness)
# Contrast
rgb = saturate((rgb - 0.5f) * Contrast + 0.5)
# saturation:
# - calc the most color-intense point
# - then simply mix both colors
half l = dot(rgb, half4(0.2126, 0.7152, 0.0722))
rgb = saturate(lerp(l, rgb, _Saturation));
```

4.9 Closing remark: order of calculation

The discussed order is written in respect of transformation operations from an incoming feed, in reality an optimized pipeline is changed slightly. In example, post processing the video feed has to be done before it is composited into the mixed reality image. This flowchart (4.26) demonstrates the actual calculation sequence. Also shown is a separation between engine scripting and shader programming.

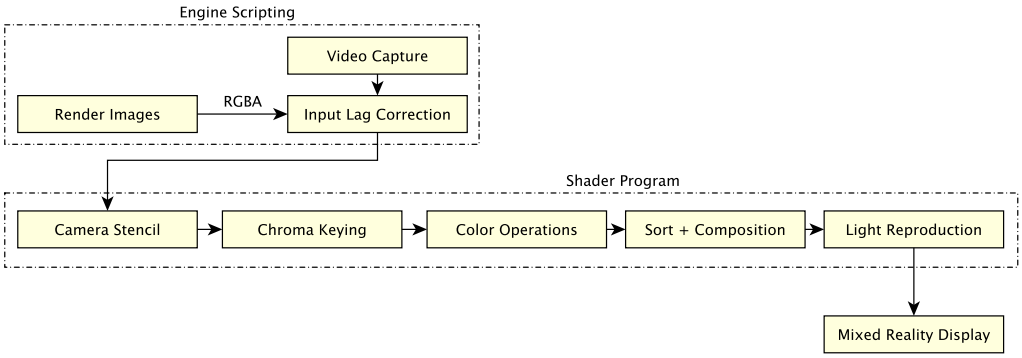


Fig. 4.26.: Actual order of computation

This might be appendix-worthy - then move a note to the chapters' beginning.

5 Evaluation

5.1 Selling VR Spaces

5.1.1 3rd Person Impressions

5.1.2 merging VR Interactivity with audience

5.1.3 managing VR shyness

5.2 Hardware Setup Variations

5.3 Rendering Setup Variations

5.3.1 Single Camera - 3D plane in space

5.3.2 Deferred shading Path

5.3.3 Composition Workstation (4 patch)

5.4 Edge Cases

5.4.1 Image Clipping - incorrect Z calculation for hands

5.4.2 Shadow Artifacts in multiple camera slices

5.4.3 Culling Artifacts

5.4.4 Stencil Clipping by faulty setup

6 Conclusion

6.0.1 3D Environment and Composition Considerations

6.0.2 Performance Considerations

7 Related Work

Glossary

6DOF Six Degrees of Freedom. 7

6DOF Describes free movement of a rigid body in three-dimensional space - specifically forward/backward (surge), up/down (heave), left/right (sway) translations, combined with changed of rotations around all three axis. **may need wikipedia cite** . 31

6DOF Six Degrees of Freedom (6DOF) . 7, *Glossary:* 6DOF

framebuffer It usually contains ARGB data of each remaining fragment. It can, however, contain any form of data, including fragment vector depth, depth-normals, normals and so on.. 21

A HLSL / GLSL Implementation of a Mixed Reality Shader

idk if this is needed or nah, but here it is:

To conclude: after building up a render pipeline that synchronizes engine frames with camera frame times most of the complicated work can be done in one single fragment shader step. It is an example of a masked Composition with blurry chroma key lookup. This shader is written in HLSL but is convertible to GLSL and its language subsets like WebGL and OpenGL ES - assuming a variety of shader constraints given before:

reference mask composition blur lookup

1. _BackTexture: Back (Full-Length) Camera Render Texture
2. _FrontTexture: Front Camera Render Texture
3. _WebcamTexture: Video Feed Texture
4. _WebcamMask: Stencil Mask Texture, default white 1x1
5. _LightTexture: Light Reproduction Texture, default white 1x1
6. _TargetColor: RGBA Color
7. _Threshold: Lower Chroma Key Cutoff, factor $\in [0, 5]$
8. _Tolerance: Upper Chroma Key Cutoff, factor $\in [0, 5]$
9. _SpillRemoval: Factor $\in [0, 1]$
10. _Hue: Factor $\in (-\pi, \pi]$
11. _Saturation: Factor $\in [0, 1]$
12. _Brightness: Factor $\in [0, 1]$
13. _Contrast: Factor $\in [0, 1]$

```
fixed4 frag(v2f i){
        # Sampling:
        fixed4 webCol = tex2D(_WebcamTexture, i.uv);
        fixed4 back = tex2D(_BackTexture, i.uv);
        fixed4 front = tex2D(_FrontTexture, i.uv);
        fixed4 webMask = tex2D(_WebcamMask, i.uv);
        fixed4 light = tex2D(_LightTex, i.uv);
```

```
            back.a = front.a;
            webCol.a = 1 - webMask.a;
            webCol.a = ChromaMin(
                    i.uv,
                    _BackTexture_TexelSize,
                    _WebcamTex,
                    _TargetColor
            );
            // final color touch ups
            webCol.rgb = spillRemoval(
                    webCol.rgb,
                    _TargetColor.rgb,
                    _SpillRemoval
            );
            // brightness:
            webCol.rgb = saturate(
                    webCol.rgb + _Brightness
            );
            webCol.rgb = satureate(
                    (webCol.rgb - 0.5f) *
                    _Contrast + 0.5f
            );
            half l = dot(
                    rgb, half4(0.2126, 0.7152, 0.0722)
            );
            webCol.rgb = saturate(
                    lerp(l, rgb, _Saturation)
            );
            return mixCol(back, webCol * light);
    }
    float ChromaMin(
            float2 uv,
            float4 texelSize,
            sampler2D tex,
            float4 targetColor) {
            float4 delta =
                    texelSize.xyxy *
                    float4(-0.5, -0.5, 0.5, 0.5);
            float alpha =
                    chromaKey(
                            tex2D(tex, uv + delta.xy),
                            targetColor
```

```
                );
        alpha = min(
                alpha,
                chromaKey(tex2D(tex, uv + delta.zy),
                targetColor)
        );
        alpha = min(
                alpha,
                chromaKey(tex2D(tex, uv + delta.xw),
                targetColor)
        );
        alpha = min(
                alpha,
                chromaKey(tex2D(tex, uv + delta.zw),
                targetColor)
        );
        return alpha;
}

float chromaKey(float4 col, float4 targetColor) {
        if(col.a == 0) {
                return 0;
        }
        float d2 = deltaE_CIE76_sRGB(
                col.xyz,
                targetColor.xyz
        ) / 100;
        d2 = smoothstep(
                _Threshold,
                (_Threshold + _Tolerance),
                d2
        ); // blend in min/max range
        return col.a * d2;
}
float deltaE_CIE76_sRGB(float3 srgb, float3 ref) {
        float3 refCol = float3(
                0.95047f,
                1.00f,
                1.08883f
        );
        srgb = XYZ2LAB(sRGB2XYZ(srgb), refCol);
        ref = XYZ2LAB(sRGB2XYZ(ref), refCol);
```

```
        return deltaE_CIE76(srgb, ref);
}
float cnRGB2XYZ(float val) {
        if(val > 0.04045) {
                return pow(
                        (val + 0.055) / 1.055, 2.4
                );
        }
        return val / 12.92;
}
float3 cnRGB2XYZ(float3 rgb) {
        return float3(
                cnRGB2XYZ(rgb.r),
                cnRGB2XYZ(rgb.g),
                cnRGB2XYZ(rgb.b));
}
float3 sRGB2XYZ(float3 rgb) {
        rgb = cnRGB2XYZ(rgb);
        float3x3 mat = float3x3(
                0.4124564, 0.3575761, 0.1804375,
                0.2126729, 0.7151522, 0.0721750,
                0.0193339, 0.1191920, 0.9503041
        );
        return mul(mat, rgb);
}
float cnXYZ2LAB(float val) {
        if(val > 0.008856f) {
                return pow(val, 1.0f / 3.0f);
        }
        return 7.787f * val + 0.137931f;
}
float3 cnXYZ2LAB(float3 rgb) {
        return float3(
                cnXYZ2LAB(rgb.r),
                cnXYZ2LAB(rgb.g),
                cnXYZ2LAB(rgb.b)
        );
}
float3 XYZ2LAB(float3 xyz, float3 refCol) {
        xyz = xyz / refCol;
        xyz = cnXYZ2LAB(xyz);
        return float3(
```

```
                (116.0f *  xyz.y) - 16.0f,
                500.0f * (xyz.x  - xyz.y),
                200.0f * (xyz.y  - xyz.z)
        );
}
float deltaE_CIE76(float3 lab1, float3 lab2) {
        return sqrt(
                pow(lab2.x - lab1.x, 2) +
                pow(lab2.y - lab1.y, 2) +
                pow(lab2.z - lab1.z, 2)
        );
}
float3 spillRemoval(
        float3 rgb,
        float3 targetColor,
        float weight) {
        float2 target =
                rgb2ycgco(targetColor.rgb).yz;
        float3 ycgco = rgb2ycgco(rgb);
        float remainder =
                dot(target, ycgco.yz) /
                dot(target, target);
        ycgco.yz -=
                target * (remainder + 0.5) * weight;
        return ycgco2rgb(ycgco);
}
float3 rgb2ycgco(float3 col) {
        return float3(
        0.25 * col.r + 0.50 * col.g + 0.25 * col.b,
        -0.25 * col.r + 0.50 * col.g - 0.25 * col.b,
        0.50 * col.r - 0.50 * col.b
        );
}
float3 ycgco2rgb(float3 col) {
        return float3(
        col.x - col.y + col.z,
        col.x + col.y,
        col.x - col.y - col.z
        );
}
```

Unitys' Monobehaviour Loop

B

The behavior of a Unity-initiated object is outlined by the following flowchart in **??**, taken from Unitys' manual.

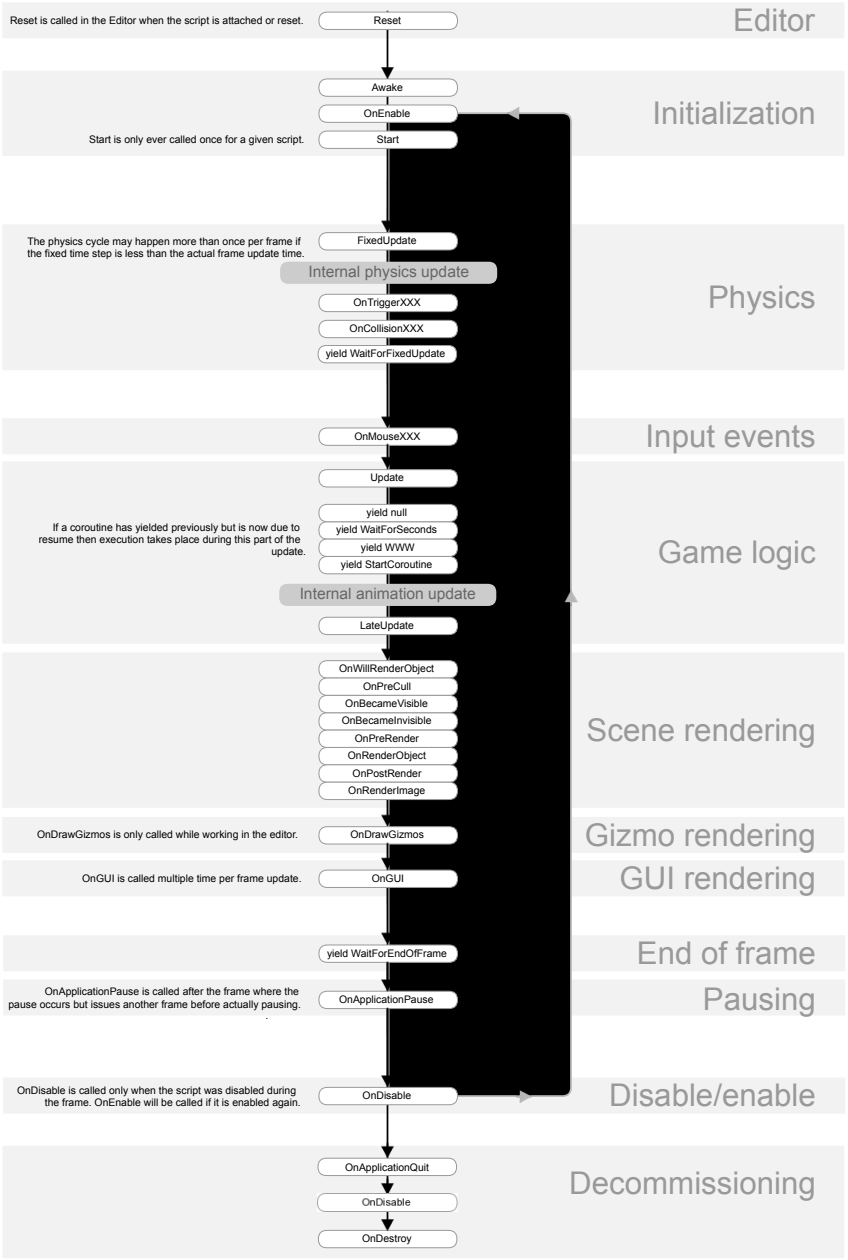


Fig. B.1.: Monobehaviour Flowchart

Bibliography

[Ann17] App Annie. *App Annie 2016 Retrospective*. Retrospective Report. 2017, pp. 2, 25 (cit. on p. 1).

[Bay76] B.E. Bayer. *Color imaging array*. US Patent 3,971,065. 1976 (cit. on p. 6).

[Mac42] David L. MacAdam. „Visual Sensitivities to Color Differences in Daylight∗“. In: *J. Opt. Soc. Am.* 32.5 (1942), pp. 247–274 (cit. on p. 6).

[MW] Tatol M. Mokrzycki W.S. *Colour difference ΔE - A survey*. Survey. Faculty of Mathematics, Informatics, University of Warmia, and Mazury, p. 20 (cit. on p. 20).

Websites

[Erg17] Deniz Ergürel. *The latest virtual reality headset sales numbers we know so far. As of March 2017*. 2017. URL: https://haptic.al/latest-virtual-reality-headset-sales-so-far-9553e42f60b5 (cit. on p. 1).

[Lei] Aaron Leiby. *Mixed Reality Videos*. URL: https://steamcommunity.com/app/358720/discussions/0/405694031549662100/ (cit. on p. 8).

[Vim] *#INTRODUCTIONS (2015)*. Vimeo. 2015. URL: https://vimeo.com/125095515 (cit. on p. 15).

[Wika] *L'Arrivée d'un train en gare de La Ciotat*. URL: https://en.wikipedia.org/wiki/L%27Arriv%C3%A9e_d%27un_train_en_gare_de_La_Ciotat (cit. on p. 5).

[Wikb] *MacAdam ellipses*. URL: https://en.wikipedia.org/wiki/File:CIExy1931_MacAdam.png (cit. on p. 6).

[Wikc] *Normalized responsivity spectra of human cone cells, S, M, and L types after Wyszecki et. al*. URL: https://en.wikipedia.org/wiki/File:Cones_SMJ2_E.svg (cit. on p. 6).

List of Figures

List of Tables

Colophon

This thesis was typeset with LaTeX 2$_\varepsilon$. It uses the *Clean Thesis* style developed by Ricardo Langner. The design of the *Clean Thesis* style is inspired by user guide documents from Apple Inc.

Download the *Clean Thesis* style at `http://cleanthesis.der-ric.de/`.

Declaration

You can put your declaration here, to declare that you have completed your work solely and only with the help of the references you mentioned.

Berlin, July 24, 2017

Martin Zier