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

---

Martin Zier

September 3, 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

September 3, 2017

Martin Zier
Mixed Reality Media: Integration of live video feed in 3D environments
Bachelor Thesis, September 3, 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

This thesis discusses a general rendering pipeline for runtime 3D engines for Mixed Reality Media, a form of video composition that places a real person inside a virtual reality scenes and recreates the scene environment.

In practice a green screen will be used, the background matte will be chroma keyed and virtual image parameters will be recreated.

Contents

1 Introduction **1**

1.1 Overview 1
1.2 Motivation 2
1.3 Problem Statement 3
1.4 Challenges & Scope 3
1.5 Results 4
1.6 Thesis Structure 4

2 Extending Reality **5**

2.1 Motion Video Production 5
2.2 CGI & Video Composition 6
2.2.1 History of Green & Blue Screen Productions 6
2.3 What's VR - Differentiation of AR, VR & MR 7
2.4 Immersion vs. Communication 8
2.4.1 Evolution of Virtual Reality Footage 9
2.5 Mixed Reality and its use cases 10
2.6 Current state of Mixed Reality Production 10
2.6.1 SteamVR & Oculus SDK plugins 10
2.6.2 Fantastic Contraption 10
2.6.3 Owlchemy Labs Mixed Reality 11
2.6.4 Apple Keynote 11

3 System Setup **13**

3.1 Hardware Configuration 13
3.1.1 PC Workstation 14
3.1.2 Inogeni 4K2USB3 14
3.1.3 Panasonic GH2 Systemcamera 14
3.1.4 HTC Vive with Controllers and Lighthouses 15
3.1.5 Vive Controller Tripod Mount 15
3.2 Software 15

4 From Video to Mixed Reality **17**

4.1 Chroma Key 18
4.1.1 Initial Assumption 19

4.1.2 Euclidean RGB Distance . . . 20
4.1.3 Euclidean YCgCo Distance . . . 21
4.1.4 Euclidean Lab Difference . . . 22
4.1.5 Comparison between computational models . . . 25
4.2 Camera Input Lag . . . 27
4.2.1 Framebuffer Swapper implementation . . . 30
4.2.2 Double Access Ringbuffer . . . 31
4.3 Mitigating Frame Jitter . . . 32
4.4 Virtual projection parameters from real world camera . . . 33
4.5 Virtual Z Sorting . . . 34
4.6 Additional Camera Stencil . . . 38
4.7 Light Environment Reproduction . . . 39
4.8 Additional Coloring Operations . . . 41
4.8.1 Color spill removal & Recoloring . . . 42
4.8.2 Brightness, Contrast and Saturation . . . 43
4.9 Closing remark: order of calculation . . . 44

5 Evaluation & future work **45**
5.1 Hardware setup variations . . . 45
5.2 Rendering Setup Variations . . . 46
5.2.1 Single Camera - 3D plane in space . . . 46
5.2.2 Deferred shading Path . . . 47
5.2.3 Composition Workstation (4 patch) . . . 47
5.3 Rendering operational variations . . . 47
5.4 Edge Cases . . . 48
5.4.1 Image Clipping - incorrect Z calculation for hands . . . 48
5.4.2 Matting failures . . . 49
5.4.3 Calibration problems (wrong clipping, wrong reprojection, long setup times) . . . 50

6 Conclusion **51**
6.1 3D Environment and Composition Considerations . . . 51
6.2 Performance Considerations . . . 51

Appendices **53**

A Unitys' Monobehaviour Loop **55**

B Simple Green Screen Setup **56**

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. [**appannie:2017**] Just before this thesis started, Apple and Google showed off their consumer-ready hard- and software for augmented reality experiences.

urls to these devices/solutions

Virtual Reality Head Mounted Displays have had a similar push in sales with an approximate of 5.83 million sold devices, which range in sales price between 80 - 900€ for a VR kit, covering from the very simple Google Daydream View and the very sophisticated HTC Vive. [**erguerel:2017**] And in these figures are the sales of Google Cardboards missing, which is approximated at around 80 Million[**superdata:market-brief:2017**].

This generation of computer systems, which includes PC workstations, game consoles and smart phones, is finally sophisticated enough in computation speed and sensor-accuracy to allow low latency tracking, precise to just a few millimeters spanning over an recommended area of about $20m^2$[**htc:vive-manual:2016**].

1.1 Overview

The idea of Virtual Reality (VR) and Head Mounted Displays (HMDs) stems from a cultural need to slip 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 and feels like. Any observer of VR, usually done by showing what the VR actor is seeing, will not be able to get an understanding 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 devices for their products that includes more than the current headset wearer and allows for a similar immersive but adapted experience.
The currently next step 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, which will be discussed in this thesis.

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, allowing for more complex user interfaces, animations and graphical fidelity. The mass of pixels by increasing the resolution of displays was basically a yearly iteration in greater, better, smaller, denser 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 this field of studies.

1.3 Problem Statement

Initially I will research motion video productions, computer generated imagery and color theory. This leads to the knowledge of implementing basic, interactive live motion video feeds.
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 for an actor using the Vive HMD to be composited into the scenery from a third person view and it will look like he is surrounded by the virtual scenery. The essential difference between classic post production is, that this system is planned to operate on runtime, allowing additional observers to get an interesting composited imagery of the VR actors experience.
An additional extension is to dynamically track the cameras position, allowing for dynamic camera movement and a freely moving actor. Further work goes into increasing the resulting composited image quality by removing artifacts from the video feed and recreating the lightning environment and transferring it to the actors video feed.

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. Initially there is a need to solve for different input latencies, especially from the live video feed, which changes the rendering pipeline significantly.
Another main aspect will be chroma keying in real time rendering, as well as image reproduction from the chroma matting result. In detail there will be a discourse of layered image composition as result of fore- and background of the VR actor.
Lastly composition controls will be implemented, adding a color-correction layer to the input video, giving a more natural look to the actor inside a surrounding 3D environment.

Outside of the scope of this thesis will be green screen compositing, as it is used as basic tool - the same goes for high fidelity video matting, which would require its own research for accurate runtime results with little to none user guidance. [**gong:realtime-matting:2010**], [**gastal:shared-sampling:2010**]

1.5 Results

Not so sure what to write in here

1.6 Thesis Structure

This thesis gets completed by digital 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 issues properly, all video media will have an annotation for cross referencing on the website. It is strongly suggested to follow these links for better understanding and highlighting of certain problems. They are be sorted by chapters for easier navigation.

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". [**wiki:train:2017**] With that a new medium was born, which matured into a new art form of film and movies.
Most important takeaway is, that with this short clip alone, a door beyond still images and their limited depiction of motion has been opened. Video imagery changed our imagination and allowed for a new communication form, inviting into an animated, moving world. There have been great achievements in the last century in motion video production with a great amount of visual trickery for composing more realistic and imaginative video content. With the help of Computer Generated Imagery (CGI) blurs the boundaries between real acting and virtual recreation so far, that it is almost impossible to differentiate between real world video capture and 3D recreated imagery in high budget productions.

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 for this thesis. Concentrating on key aspects of composition techniques might give an appropriate overview to range where Mixed Reality takes its inspiration from.

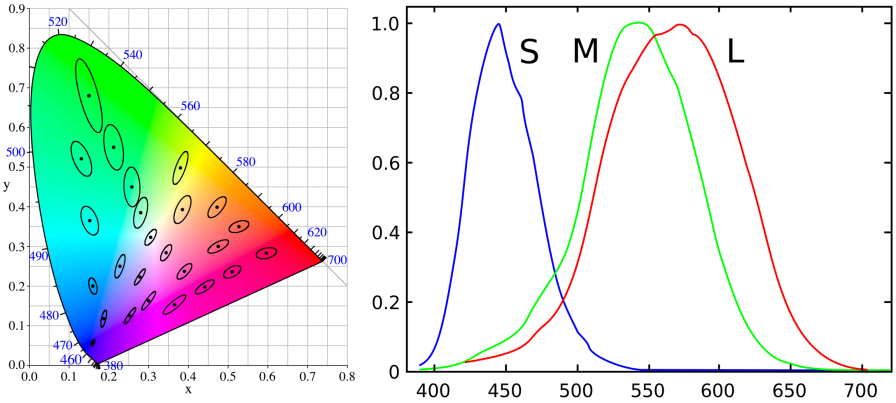
Way before digital imaging processing took over production sets similar problems as discussed in this thesis had to be solved, in example how an actor can be captured without a back- or foreground and how he would then be integrated into an imaginative set. Today, modern action movies don't even necessarily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

Missing: Camera tricks, computer aided visual effects, green screens

2.2 CGI & Video Composition

2.2.1 History of Green & Blue Screen Productions

Production 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 found in capturing of humans - and to a certain extent any flora or fauna. Since chroma keying (see Ch. 4.1) takes color distance as general basis, production environments use green screen keying in varying forms. Next to ΔE, which will be discussed further, another approach is the calculation of fore- and background color spaces and separation of them. The latter solution is highly computation intense and is not suitable for runtime applications. [**disney:unmixing:2017**]
Green boxes also abuse a correlating advantage that the human eye is most susceptible to green (cf. figure **??** b), 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 gambit ranges and color differentiation of human vision (figure **??** a). Experiments concluded that eye cone cells see a blending range of wavelengths to different intensities, giving the green vector space its highest perceptible range. [**MacAdam:1942**]



(a) MacAdam ellipses on 1931 standard chromaticity diagram [**wiki:macadam:2017**]

(b) Normalized responsivity spectra of human cone cells, S, M, and L types after Wyszecki et. al [**wiki:Wyszecki:2017**] . Notice the overlap between M and L cones.

The layout is a bit wonky here.

Most consumer cameras - and even most production cameras - use dot-matrix sensors with a weighted ration of green (4), red (2) and blue (2) pixels, called Bayer pattern [**kodak:bayer:1976**], giving it - similar to human vision - a high color vector space. Green is generally easier to light, illuminate and adjust over blue screens. Small

irregularities, for example through uneven lightning or crinkles in the material, can be adjusted easily by the user and allows for a relatively clean camera image generation.
The quality of input footage makes a big difference in separating back- and foreground, hence a well lit and adjusted set is a good backbone for well working chroma keying.

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

In search of an appropriate abbreviation for computer-enhanced real time 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.

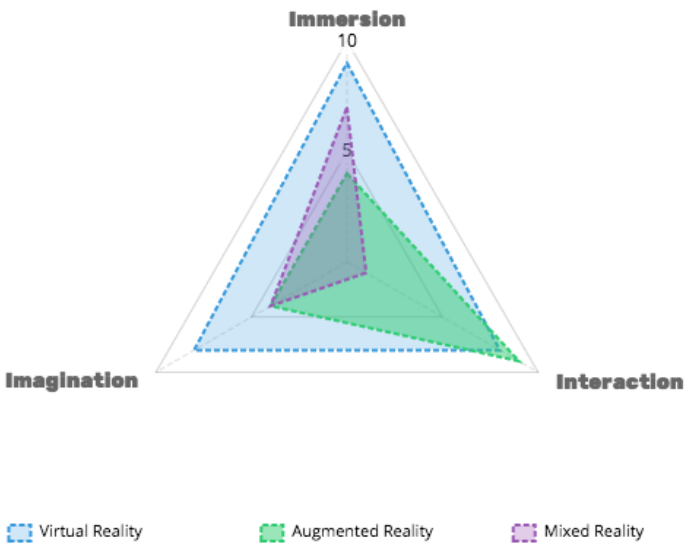


Fig. 2.2.: I^3 Triangle - figurative quantization of different reality extending methods

Augmented Reality (AR) is a concept by augmenting real world imagery with additional information and interactable objects. It used to be called Mixed Reality[**satoh:case:1998**][**tamura:mixed-reality:2001**], merging real world imagery with 3D objects but has slowly adapted to AR, since they both described the same concept. It ranges from very simple devices displaying data in the field of view of an user up to full augmentation, displaying 3D models overlaying on real world objects. This can be done ranging from Pepper's Ghost projections, 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 distant field calculation.

Virtual Reality is a concept usually done by stereo projection of a 3D environment inside a Head Mounted Display. It takes an user out of the current room and sets him into a complete new, virtual reality - hence its names origin. HMD hardware ranges from the simplistic Google Cardboard [1] to the Samsung GearVR [2] up to the Oculus Rift and HTC Vive[3]. The latter two products offer room-scale experiences where an user is able to move freely in his play space (basically a tacked bounding volume) and allows for six degrees of freedom (6DOF) tracking.

might want to remove footnotes and integrate into sentences

Mixed Reality is an extension of Virtual Reality, allowing bystanders to get an impression of the virtual 3D environment 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 scene. This yields a combined application of Augmented and Virtual Reality technique. A production environment can be achieved with a Six Degrees of Freedom (6DOF) HMD and additional - either user or tracking input for positional and rotational parameters for the real world 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 VR doesn't offer any way 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 VR controllers in his hands and directly translating controller interaction to the virtual environment. However an outside viewer does not see the actors hands and will not understand any actions performed by the user if he doesn't see the virtual hands. Any usage context that happens off-screen cannot be communicated and therefore will be lost.

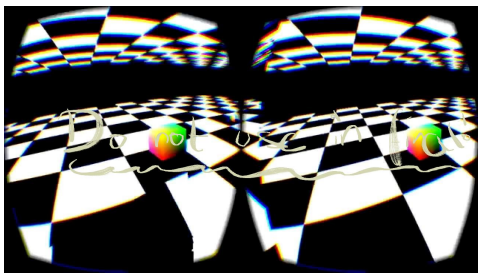
Mixed Reality merges the actors and virtual realities context, allowing outside bystanders a comparable window into the actors experienced world. In fact, initial promotional material for the HTC Vive showed mixed reality footage, produced by one VR computer and a secondary composition PC [**valve:mr-production:2016**]. It's setup is comparable to the one in this thesis and differs by composition techniques

[1] using a smart phone as display device
[2] similarly uses a smart phone as display driver
[3] which are driven by a mid- to high range PC

and by using more than one software context, done by outputting a rendered image of the virtual scene and compositing it on another system.

2.4.1 Evolution of Virtual Reality Footage

Originally VR footage consisted of the output that was sent to the headset, including image transformation that needs to be done to correct the headset lenses [4] - this distorted dual image is very hard to follow, as it adds two ambiguous images and has additional a high distortion. Since the direct feed to a HMD is used, any translation of the headsets position is visible, which results in a jittery and unnatural looking motion feed.



(a) Distorted direct dual output to a Oculus DK2.

(b) A first person, single screen output is an improvement, but is hard to follow in motion.

(c) "Rick and Morty: Virtual Rick-Ality" allows outside observers to control mounted CCTV cameras to observe the actors interaction.

(d) Currently a screenshot of The Lab MR - but actually will be one of my solution. Just gotta take some screenshots.

A next step was to render one eye in fullscreen of the attached monitor and do all image transformations (mainly crop and distort) after rendering it, allowing bystanders to get a clear first person image of the actors experience. Additionally, as a secondary camera, a dampened movement can be set to mitigate jittery motion. As previously pointed out, this works but sometimes loses interaction context, especially when no hands are visible.
Then there is currently only one game, Rick and Morty: Virtual Rick-ality, that has an optional CCTV feature enabled, in which third persons can control a variety of mounted, virtual cameras and follow the actors interaction with the 3D environment.

[4] this is referred to as "Barrel Distortion"

This VR actor is then replaced with an avatar to visualize what how he is interacting with the scenery.
Finally there is mixed reality, where the VR actor is placed in context of the virtual reality scene. This has been done previously by post-production for trailers and allows for a better understanding between virtual interaction and real actor motion. With its help it is possible to invite third person viewers into the VR experience in a natural way.

Maybe I should pin down the timeline - how long it took for a next step to emerge from the medium.

2.5 Mixed Reality and its use cases

There is a meeting planned to discuss A+Cs interest and use cases for MR - which then probably ends up in here in a summary.

2.6 Current state of Mixed Reality Production

There have been an increasing number of presentations for mixed reality setups. To sort this thesis into the current scope of production environments, it is necessary to look at other approaches and their differences to the proposed solution.

2.6.1 SteamVR & Oculus SDK plugins

Both SteamVR and Oculus SDKs supply plugins to enable mixed reality capturing. These system approach the problem similarly by compositing an incoming video stream directly at the current position of an registered camera tracker, thus failing to accommodate for different input latencies from the motion video feed, yielding an inconsistent visual performance. Oculus states on their manuals, that these are currently only intended "for proof-of-concept, troubleshooting, and hobby use." [**oculus:mr-setup:2017**]
In addition SteamVRs solution supports only video-output composition with a 4K HDMI output - which in turn means that the signal has to be captured on an external device and has to be composited on a secondary system. The configuration parameters are "barebones" at best and yielding good results is a matter of the best possible studio setup capture.

2.6.2 Fantastic Contraption

The game "Fantastic Contraption" from 2016 allows livestreamers to do a video composition for mixed reality. While this approach allows to mitigate video input delays it cannot have a free moving camera with an additional motion tracker.

"Fantastic Contraptions" trailers also show another approach by replacing the actor with an avatar by basically rendering a certain depth and then placing real world camera footage as background. With such a system any live video background removal is not necessary and real world footage of the actor is lost. The games trailer composition has been achieved in post production. [**gartner:cinematography:2017**]

2.6.3 Owlchemy Labs Mixed Reality

Owlchemy Labs is a VR game studio located in Austin, Texas. They're working on VR experiments and use similar techniques discussed in this paper. Their key difference in visual reproduction is by using a stereoscopic camera to record an actor and can reproduce the actors depth per pixel, allowing for more complex fore- and background separation for a more accurate visual reproduction.
While Owlchemy Labs announced real time mixed reality compositing, there hasn't been a shipped product or update for one of their games that allows for live mixed reality capture[**owlchemy:mr:3:2017**].

2.6.4 Apple Keynote

Apple presented a real time mixed reality showcase done on their computer systems on their annual keynote "WWDC17". It is driven by Unreal Engine and uses also similar techniques as discussed in this thesis. That presentation is currently the best live performance of mixed reality with a high quality chroma keying, depth reconstruction and high fidelity graphics. There is no technical information available yet[**ilmxlab:mr-demo:2017**].

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.

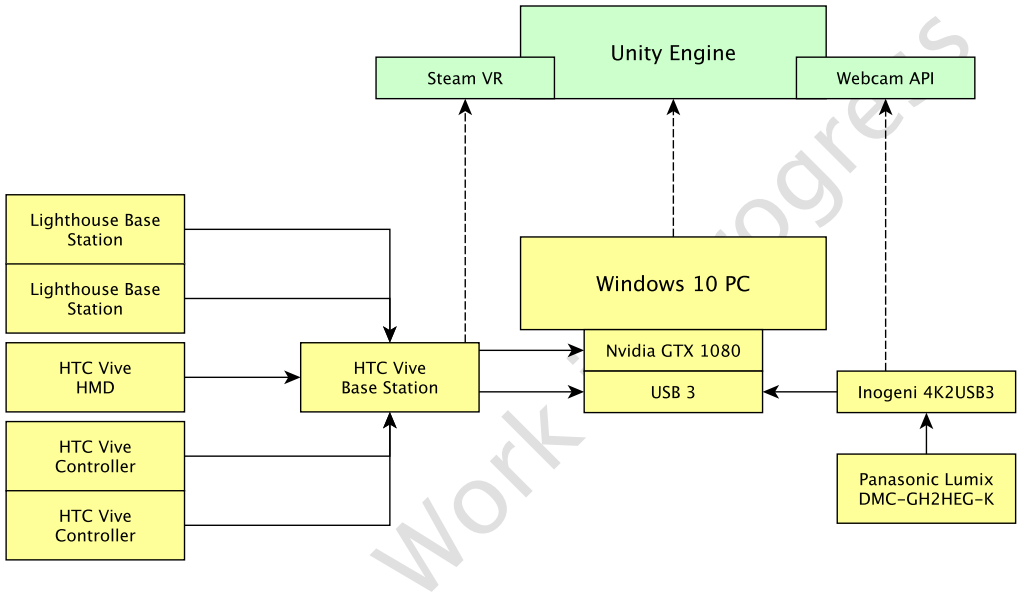


Fig. 3.1.: Diagram of hard- and software components.

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 for this thesis is:

```
CPU: Intel i7-6700 @ 3.40 GHz
RAM: 16GB DDR4
GPU: Nvidia Geforce GTX 1080
System: Windows 10 v. 1703
Engine: Unity 5.6f3
```

This system configuration is to date a high end workstation that has an abundance of render performance and memory for complex and taxing computations that have to be performed for a mixed reality composition.

maybe a bit short?

3.1.2 Inogeni 4K2USB3

The Ingoeni 4K2USB3 converter is a standalone box that allows to receive any HDMI source and converts it as a webcam video feed usable by "plug'n'play" device management of Windows. It's advantage is that it enables any HMDI source to be usable as system webcam, thus enabling and arbitrary choice of video cameras and a very simple integration with any software through the systems provided webcam API. With 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. Additionally it has a well sized photo sensor allowing the camera to capture singular frames with reduced motion blur.

3.1.4 HTC Vive with Controllers and Lighthouses

The current generation best virtual reality and tracking device is the HTC Vive. It includes two infrared sending stations called "Lighthouse", two Vive Controllers[1] and a headset. Both, headset and controller systems, are enabled with 6 degrees of freedom (6DOF)tracking. The tracking system is a black box, in which only the transformation matrices for the hand controllers and the HMD can be accessed[2], which already have been processed by the additional SteamVR software. By default this transformation has a normalized length of 1 unit to 1 meter. Designing scenery and sense of size is therefore rather imaginable. The data providing is done by a library called "SteamVR for Unity", which makes the usage of said matrices in engine fully transparent.

6DOF was defined earlier iirc

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, so I've built a mount that fits on tripod attachment points and keeps the controller locked in the same rotation and position (see figure 3.2).

This image renders on the wrong page

3.2 Software

The software of choice is Unity3D, which is a free game engine 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. However, this has drawbacks in overall performance, partially introduced by the engines garbage collection, which are no issue for the used system.
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

[1] often delightfully called "Wands" due to its controllers design
[2] as well as sensor input data from the controllers

Fig. 3.2.: Camera mount for a HTC Vive controller, mounted on the cameras tripod mount

language which cross-compiles to HLSL, OpenGL and WebGL - this leads to problems in buffer and data management, which cannot be controlled well inside its render loop.

The software discussed in this thesis integrates and depends additionally on SteamVR, a library for 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 the software of this thesis 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 operations 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 $11ms = 1s/90$. It is important to have consistent performance, otherwise the experience for an actor with the HMD will be terribly degraded. 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 and will likely improve in the future of VR[**oculus:improv-render:2016**].
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 device tracking, it is easily possible to position the in-engine camera at an accurate location of the real world capturing device. 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 center of a controller / tracker.

The following chapter describes the techniques used to transform motion video inside a green screen into a mixed reality image. As brief overview, the steps required are performed in order of biggest impact in graphical fidelity (Fig. 4.1) for the composition of a real world 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. A closing remark with the order of calculation has been added at the end of the chapter.

If it doesn't stay in the appendix

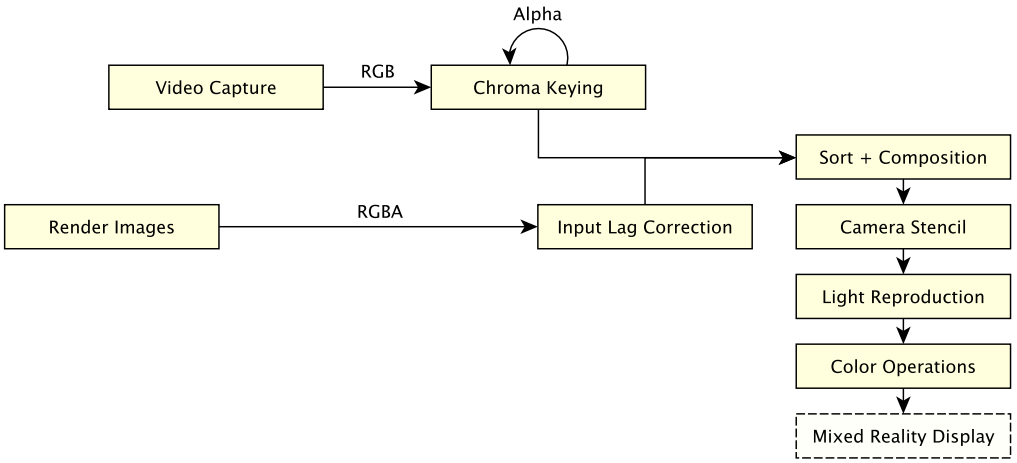


Fig. 4.1.: Full mixed reality graphics pipeline in order of graphical fidelity impact

4.1 Chroma Key

Beginning from a real world camera, the video signal travels through the Inogeni 4KUSB3 converter and is accessible with the systems API for webcams. (See figure 3.1)

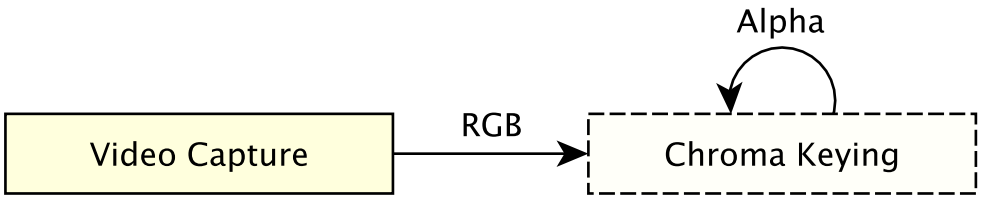


Fig. 4.2.: Initial step upon receiving the camera image

The initial step is to remove the green (or blue) background from the image, which should be live footage of a green (or blue) box. 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 by a checkbox to show raw output from the camera. Then an average green from the background box 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, thus making a recalibration necessary.

Image of the material editor

An extreme example case is used for comparing different chroma keying variants, which shows high motion blur due to a low shutter speed of the capturing camera and fast movement of the depicted actor:

Fig. 4.3.: Comparison Image [**vimeo:shia:2015**] - sRGB Output

4.1.1 Initial Assumption

Each RGB color can be represented as a discrete 3-Vector of (red, green and 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:

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

This does T as "Total"

thus:

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

$$I_{x,y} = (1 - \alpha_T)F_{x,y} + \alpha_T Bx, y \tag{4.3}$$

This equation can take $\alpha_T < 1$ into account, which is needed later when fore- and background of the virtual environment are merged with a layer in between that results from the real world capture.

4.1.2 Euclidean RGB Distance

Assuming a source pixel color C_S and a reference color C_R we can calculate the euclidean distance between these two color vectors:

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

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

Maybe put a highlight of the image together

This is computationally very low cost and works well enough for tell a difference between two distinct colors. It fails to accommodate for coloring that is perceived as different, but are tinted by the reference colors. Since the green screen 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.

4.1.3 Euclidean YCgCo Distance

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, thus splicing the color model into brightness and color appearance. 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.

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

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 better matte pulling, less green edges and a more continuous, smooth 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 inside the *Lab* color vector volume. The final distance calculation is, too, a linear euclidean scalar as with all other models, but accommodates for perceived color differences. It is a more expensive computation wise and needs in implementation code branching, which will evaluate all branches first by the GPU before using either code path.

A reference white has to be used to convert RGB to the Lab color model to accommodate for different RGB color spaces[2]. Luckily the given webcam signal defaults to sRGB D65 and does not need a configurable reference matrix based on a given color model to handle different RGB color standards.

sRGB conversion needs to be converted to linear RGB while respecting light energy per color channel, rather than relative, perceived color brightness:

this is very rough and only contains equations currently used - also could need a better explanation for energy leveling

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

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 an linear RGB color can be converted to XYZ by the following equation:

[2] which are usually only different on stored image material like photos or video

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

where:

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

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

The needed $\Big[M\Big]$ for RGB D65 is defined as:

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

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_S and C_R, which already have been converted to Lab:

$$\Delta E = \sqrt{(C_R L - C_S L)^2 + (C_R a - C_S a)^2 + (C_R b - C_S b)^2} \tag{4.18}$$

The resulting values are rated by their perceptive difference (after Mkrzycki et. al. [**mokrzycki:2012**]):

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 with a proposed "smooth step" algorithm:

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

$$g(f(\Delta E)) = y = \begin{cases} n & \text{if } x \leq n \\ x & \text{if } n \leq x \leq m \\ m & \text{if } m \leq x \end{cases} \tag{4.20}$$

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

With that we receive a more natural matte with nearly no green edges, continuous actor imagery and smooth alpha transition between actor and green screen. Motion blur is hard to account for and is even with professional video matting hardware,

extensive post production or intrinsic frame matting algorithms hard to remove without rough image results.

Could need an example case with real matting soft-/hardware

Fig. 4.6.: Chroma Keying by using ΔE distance

4.1.5 Comparison between computational models

To chose the best variant, balancing runtime and efficiency, there has to be a comparison between the presented methods. Again, the previous image will be used, since it has complicated color mixing between background and actor. First we pick scan line of the image, in this case the 301st row from top, and calculate the color distance from each pixel to a reference color $[R, G, B] = [0.341, 0.588, 0.42]$.

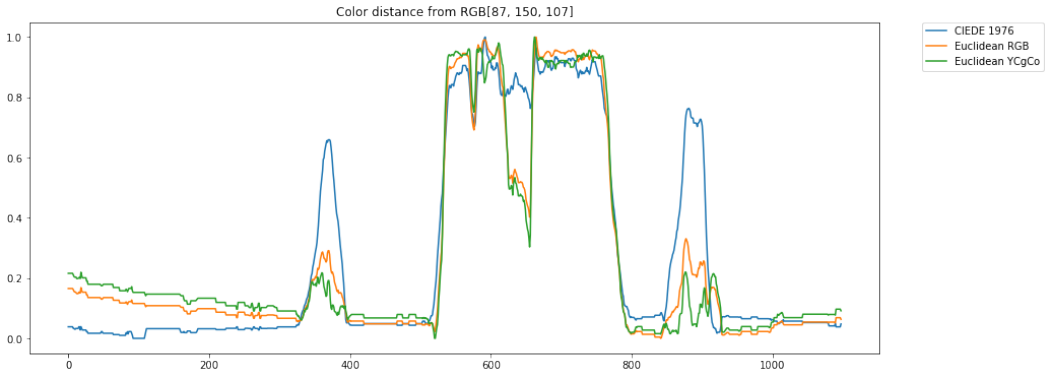
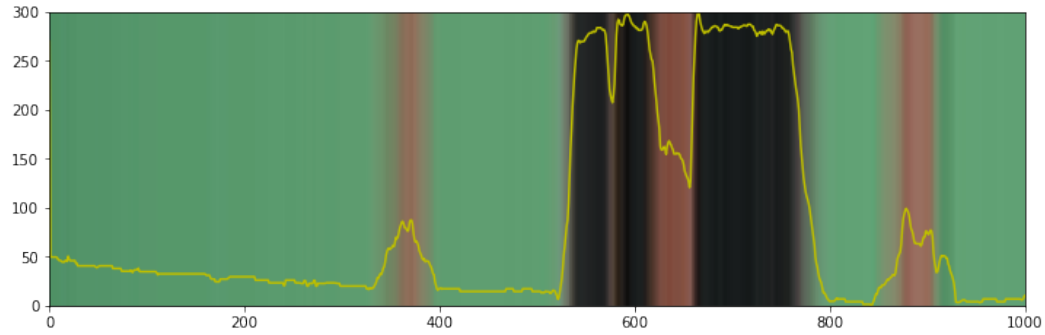


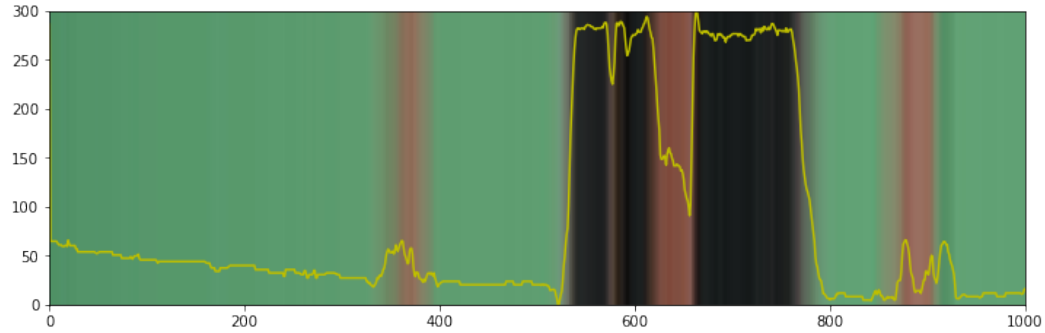
Fig. 4.7.: Normalized graph comparing color difference methods

As seen in figure 4.7, CIEDE76s performance is significantly better in separating colors and has a broader margin between green background and other foreground. This can be observed in figure 4.8, where the scan line is set as background and the values are normalized again:

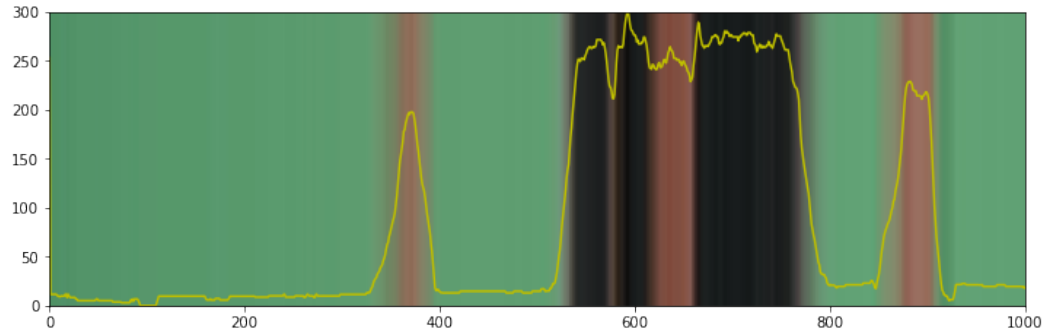
Fig. 4.8.: Comparison between different color distance methods



(a) RGB color distance



(b) YCgCo color distance



(c) CIEDE76 color distance

Finally, comparing CIEDE76, CIEDE94 and CIEDE2000 reveals, that CIEDE76 performs well enough while having the lowest performance overhead. The similarities between the 1976 and 2000 method are very apparent, while CIEDE2000 has a significantly more complex algorithm [**sharma:ciede2000:2005**].
Additionally CIEDE76 shows less variance on similar colored spaces to the target color as well as high color difference peaks on all other pixels. This yields an accurate upper and lower limit, in which the green screen can be calibrated in.
Additionally, performing the same operation on the same data with different CIEDE

color distance revisions (figure 4.10), that there is little difference between CIEDE76 and CIEDE2000.

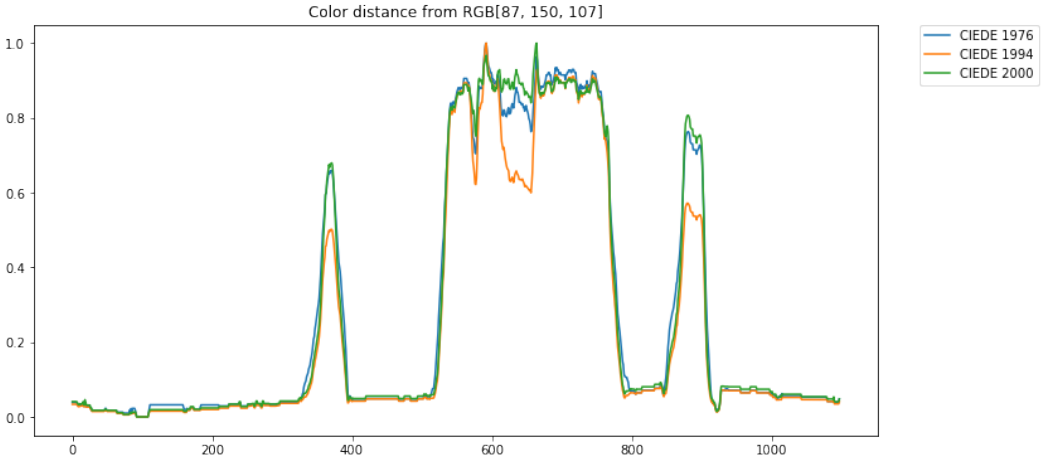


Fig. 4.10.: Comparison with different CIEDE ΔE variants

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

Fig. 4.11.: Initial step upon receiving the camera image is to adjust the time drift between engine renderings and video capture

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:

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

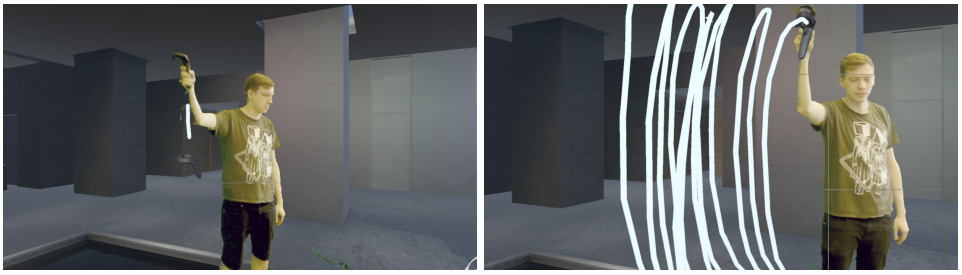
Assuming 25 frames per second, that is $\frac{1}{30}s$:

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

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

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 **??**.

Fig. 4.12.: Visual comparison between unaligned and aligned frames[3]



(a) Before video and engine frames have been aligned, there is a noticable difference in motion

(b) After aligning frames the motion in engine and video capture are in sync

To mitigate this offset there are two options:

1. Change the camera setup - in example for a direct 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 - assuming that the time drift is constant. 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 for this thesis uses the secondary option, since it is able to minimize any kind of constant offset between render image and a video stream, the hardware setup can stay dynamic[4] 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

[3] a video example is provided on the additional storage device
[4] as long as it is provided by the systems webcam API

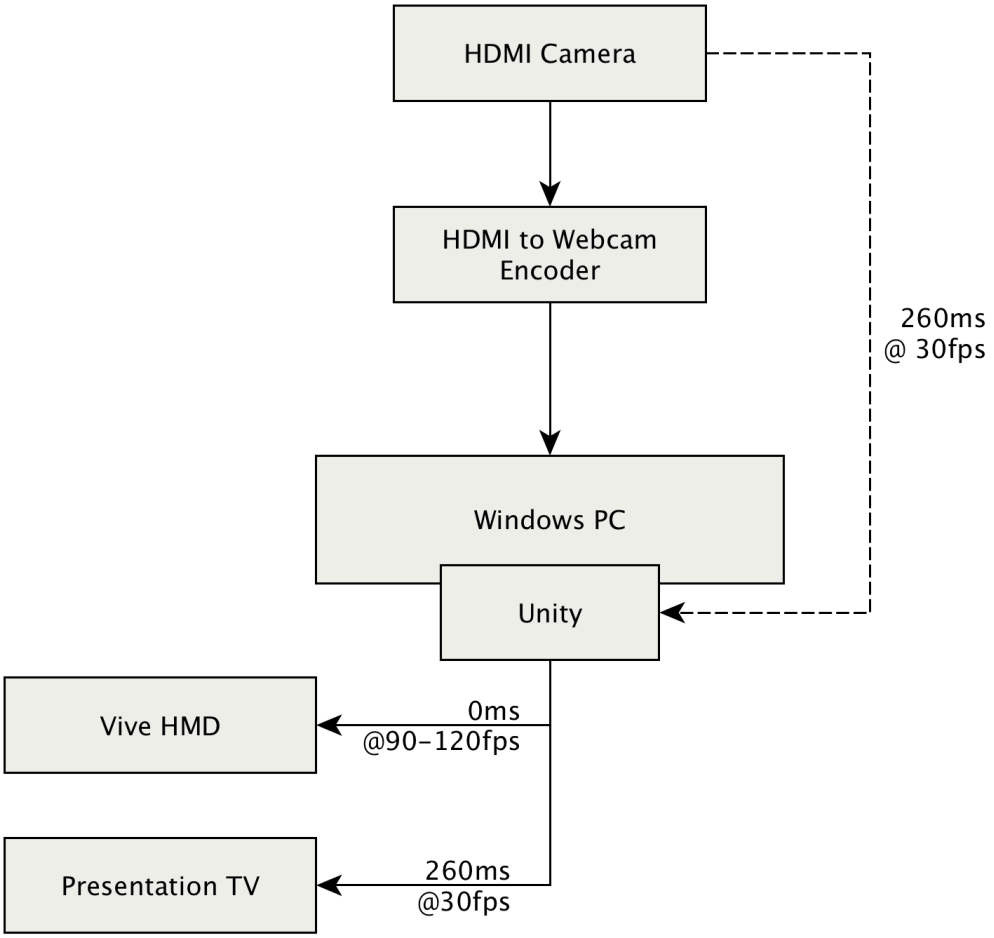


Fig. 4.14.: Components in considering video input lag and frame rates. While latencies between each components cannot measured, it is observed with help of an interactive VR object.

Based on the component diagram 4.14 there are two important takeaways:

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 (see Section 4.3)

At time of writing Unity does not support dynamic frame rates on multiple viewports[5].
It is possible to manually initiate a rendering, 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

unsure if I should elaborate

[5] A viewport can be either a D3D, OpenGL or Metal 3D context

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 at the right frame to guarantee minimal delays between camera and 3D environment, thus realigning the observed time drift between the engine and captured video frames.
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, allowing for accurate timing how framebuffers have to be handled and swapped.

4.2.1 Framebuffer Swapper implementation

Unity has a well engineered engine loop, where it can perform different operations at specific steps of the main engine execution. For more detail about Unitys core loop[6].
As initial data needed is `cameraFPS` , `cameraOffset` and `Time.deltaTime` .
With that data it is possible to calculate the remaining variables 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);
}
```

[6]there is a flowchart in the Appendix A, depicting MonoBehaviors life cycle.

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 regular framebuffer would allow for simultaneous access, thus two separate buffers are needed.

4.2.2 Double Access Ringbuffer

To spare memory and bandwidth 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 4.16).

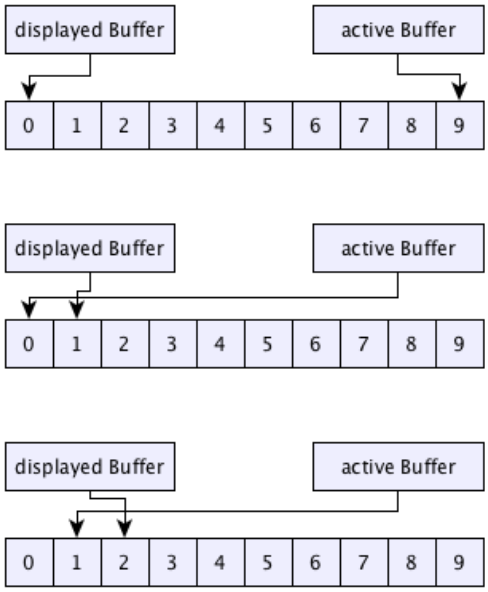


Fig. 4.15.: Schema of an the ringbuffer

To accommodate for that behavior we have to 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 to the next index.

```
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 `Render Textures` for as long as one duration of a video frame and then forward 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 of the presentation device is unaffected. It is possible to write less than three times into a `Render Textures` , 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.

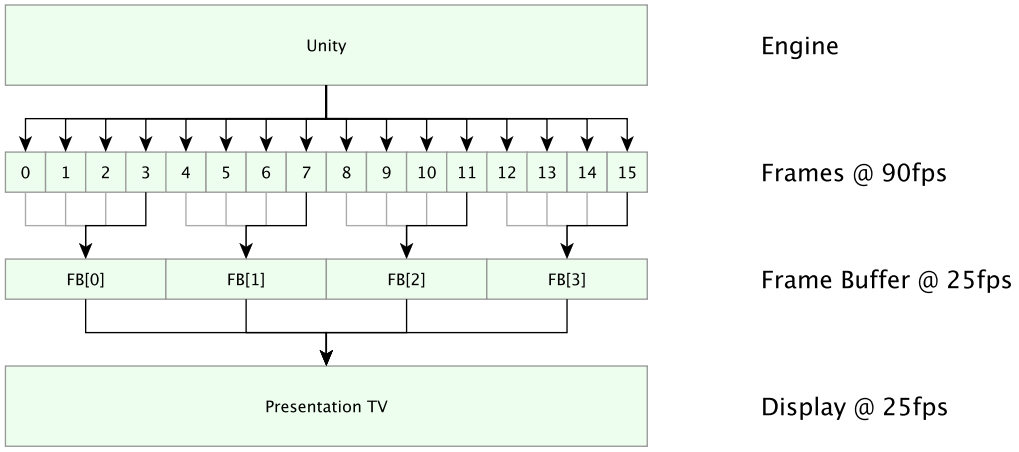


Fig. 4.16.: Workflow of the render swapper, in which rendered frames will be overwritten as long as it is needed - and then another frame buffer will be written into.

needs then lstlisting when the code further up is modified

4.4 Virtual projection parameters from real world camera

To reproduce a virtual projection for the real world camera 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 casmera - ignoring an additional fixed 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:

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

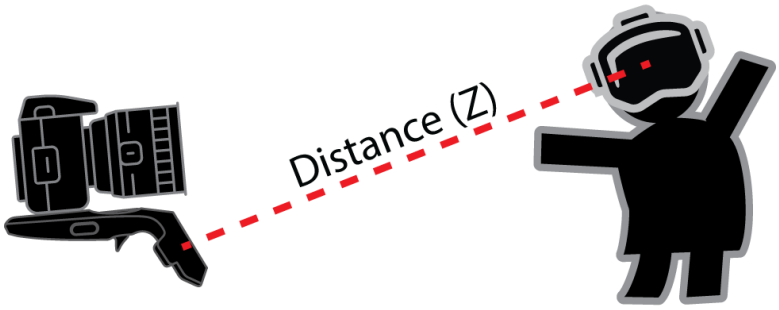


Fig. 4.17.: Distance correlation between real world camera and the VR actor

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, varying from camera object to another. 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 $F_l = 14mm$:

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

$$FoV = 63.42028^\circ \tag{4.27}$$

With that we have now all projection parameters reconstructed 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 with a fore- and background - in between are frames from the motion video feed.

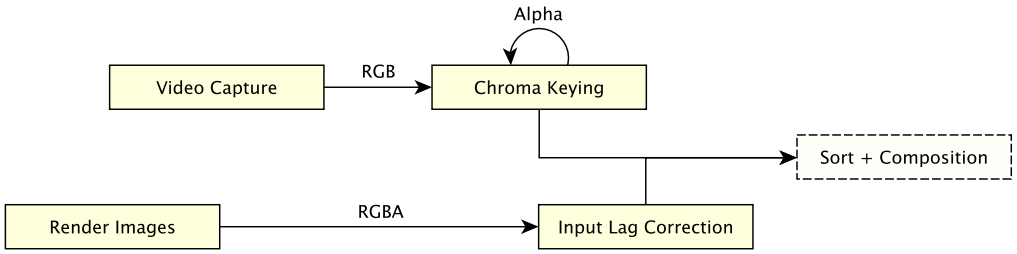


Fig. 4.18.: Current steps taken through the graphics pipeline

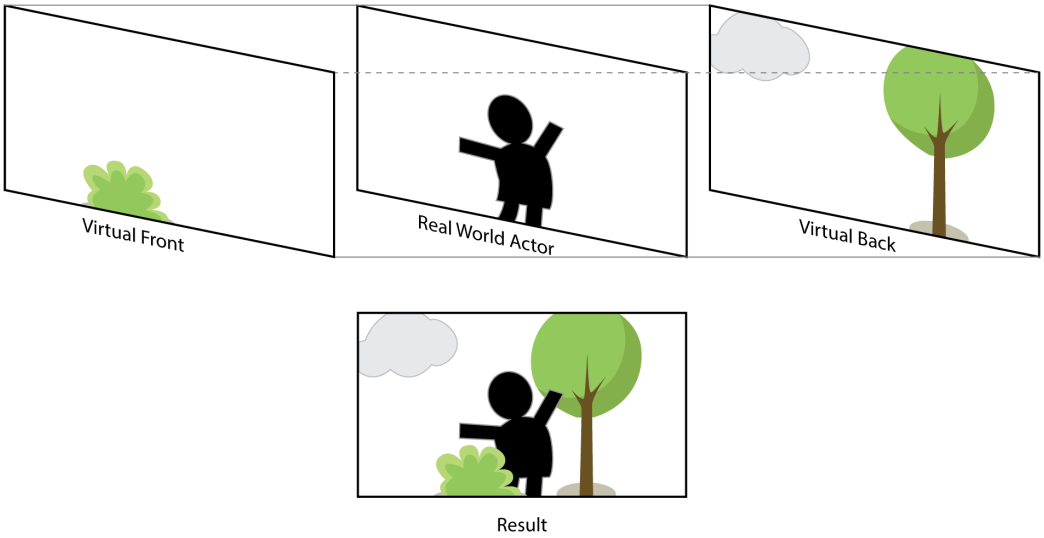


Fig. 4.19.: A sketch of the video composition with three layers of projection

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 visual artists decide between two render modes:

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 and not modified by post effects.
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 sceneries background. The resulting image has inconsistencies with alpha-blending but this method works with all rendering setups.

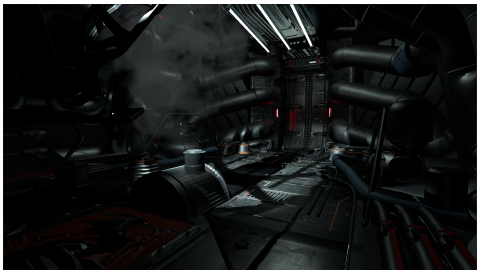
Needs an example of that behavior

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 pipelines. Other post effects need certain projection requirements and / or get discarded through culling, like in Figure 4.21e 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, unlit front geometry because the light source is outside of the virtual cameras frustum.

Frustum - Glossary!

Fig. 4.20.: 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 transferring the alpha mask on 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 projection planes and thus can have an interactive fore- and background. The initial assumption is that the actors depth is flattened to a plane, 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 green screen 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 green screens 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 green screen. This means that a real world camera can film outside the edges of a green screen, recorded pixels beyond the green box are getting discarded and it will not degrade the visual performance of the image composition.

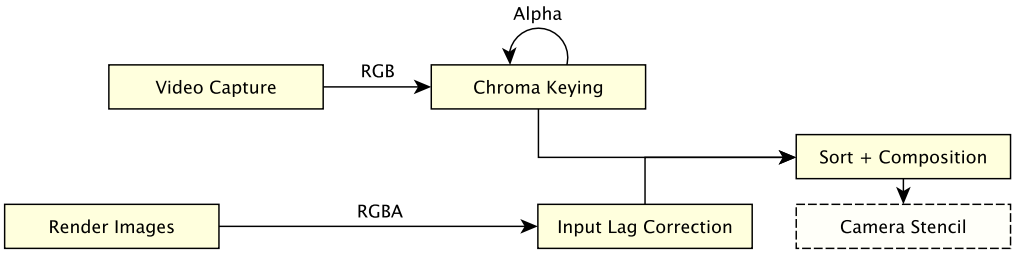


Fig. 4.22.: After sorting the scene additional stenciling of the video feed is necessary

A virtual projection can be seen in figure 4.23 with reconstructed camera parameters. With help of Unitys' engine-editor a green screen 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 green screen 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 composed image without major drawbacks.

Might need additional text for clarifying

Fig. 4.23.: Virtual projection and photo of VR actor - red colored areas will be cut off

(a) Virtual reprojection of a calibrated green screen

(b) Masking what would be remaining video content

(c) Composition after stencil has been applied

In-engine this setup is a simple addition: After registering another virtual camera to the camera manager it is used to simply render green planes where the real world green screen is placed. Taken from previous projection parameters in 4.4 this

virtual camera receives a dedicated culling mask which only contains all green box projection planes. All other cameras ignore this layer and are therefore invisible for any composing camera. After each drawn frame, Unity is able to clear the frame buffer with an alpha color as 0 - all remaining fragments from the green box write any color with an alpha as 1; the color compartment is discarded and will not be used further. This creates a lookup texture where an alpha-mask is rendered which will be transferred to the video feeds alpha (see figure 4.23).

We have now achieved a green box projection inside the same scene without any complex management processes needed in Unity. Due to the quick setup it works well for fast calibration and allows for a broad camera angle up to the green screens edges without degrading the mixed reality performance. Recording beyond these edges will yield into a composition of front and background renderings since all camera pixels will be completely discarded.
Unfortunately stencil-operations are poorly documented in Unity and add an overhead which cannot be measured from built in profiler tools - as such it adds unknown performance costs and have not been used in this thesis, hence the fall back of rendering an alpha map.

4.7 Light Environment Reproduction

To conclude all taken rendering steps by now: We have successfully created an image composition with correct projection parameters, recalculated a planar depth, synchronized in-engine frames with the camera input lag and realigned the frame rate between both image generating systems and cut off all overshot pixels that were recorded outside of the green screen.

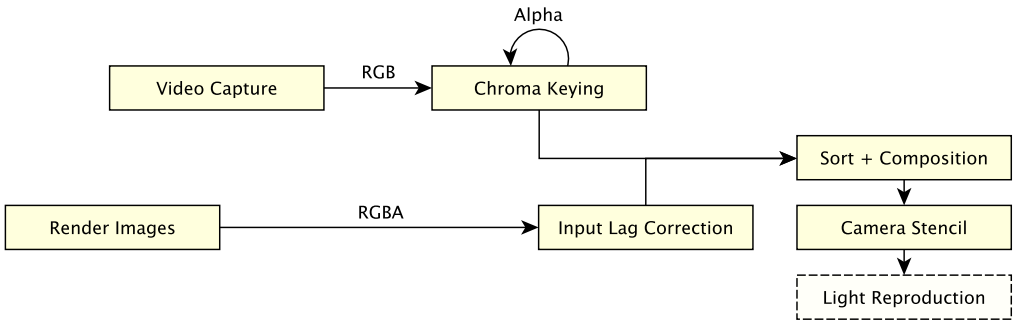


Fig. 4.25.: Following in this pipeline is lights reproduction

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.

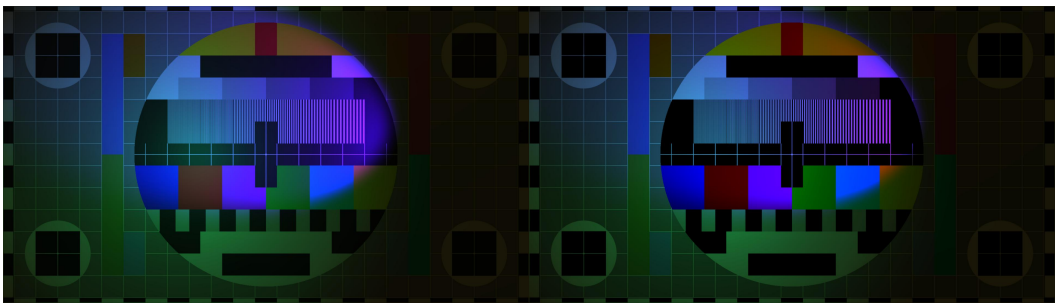


Fig. 4.26.: Left: original, Right: reconstruction

Ambient light reproduction hinges on two assumptions: An actors recorded video is flat for this purpose *and* he receives clean and consistent light from all angles that has no additional glossiness. With that we can 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 4.27 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:

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

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

These parameters can directly be applied to the transformation of said white plane which then captures the lightning environment inside the virtual scenes. As with all other composition-renderings, this step will be stored as a separate `RenderTexture` too and can operate on lower resolutions to speed up color and light sampling for performance gains.
The resulting frame buffer will be multiplied later onto the camera feed with a video color C_V and the light plane C_L:

$$C_T = \begin{bmatrix} C_{V_R} * C_{L_R} \\ C_{V_G} * C_{L_G} \\ C_{V_B} * C_{L_B} \end{bmatrix} \tag{4.37}$$

Lastly, a directional light with the same culling mask can be applied, to improve the overall brightness of this light mask to allow for more natural tint than a linear color operation.

Fig. 4.27.: A comparison of different composition methods in engine

(a) captured, lit plane **(b)** Light plane applied to the actor

margin of error of light reproduction

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 better into the environment.

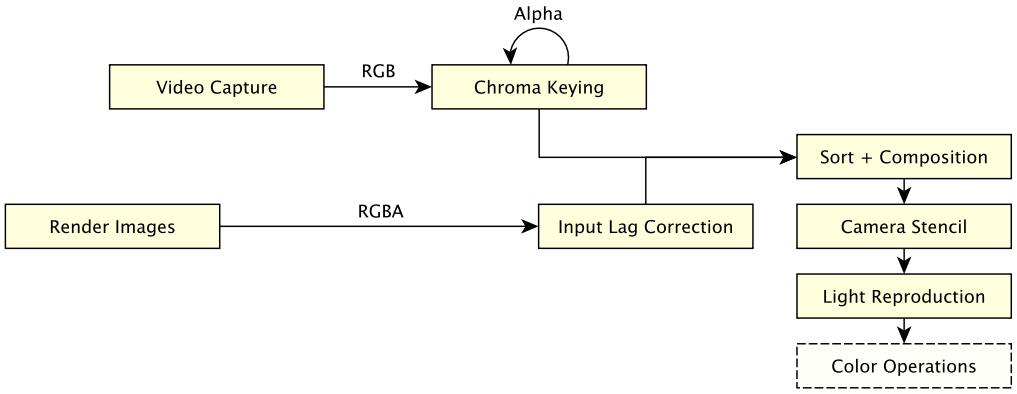


Fig. 4.29.: Initial step upon receiving the camera image

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 usually given by Unitys' post effects stack, which are unavailable due to the nature of this render pipeline.

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

$$R = \frac{C_{KeyCg,Co} \cdot C_{InputCg,Co}}{C_{KeyCg,Co} \cdot C_{KeyCg,Co}} \tag{4.38}$$

$$\begin{bmatrix} Y \\ Cg \\ Co \end{bmatrix} = \begin{bmatrix} Y \\ C_{KeyCg} * (R + 0.5) * W \\ C_{KeyCo} * (R + 0.5) * W \end{bmatrix} \tag{4.39}$$

"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 the imagery look more natural and fitting into a 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 for a maximum of 180 degrees forwards or backwards with a factor $H \in (-\pi, \pi]$ to yield a color C_F:

$$C_F = C_I \cos(H) + (0.57735 \times C_I) \sin(H) + 0.57735(k \cdot C_I)(1 - \cos(H)) \tag{4.40}$$

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 - if needed - 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:

Brightness increases all color channels of a given color C_I for a brightness factor of F_B:

$$C_T = \begin{bmatrix} C_{I_R} + F_B \\ C_{I_G} + F_B \\ C_{I_B} + F_B \end{bmatrix} \tag{4.41}$$

Contrast is a color multiplication in which the input color C_I will be decreased by half of a channels maximum value, multiplied by a contrast factor F_C and increased by half a maximum channels color again:

$$C_T = \begin{bmatrix} (C_{I_R} - 0.5) * F_B + 0.5 \\ (C_{I_G} - 0.5) * F_B + 0.5 \\ (C_{I_B} - 0.5) * F_B + 0.5 \end{bmatrix} \tag{4.42}$$

Saturation is done by calculating the color intensity I and then mixing these both colors for a saturation factor F_S:

$$I_{R,G,B} = \begin{bmatrix} C_{I_R} * 0.2126 \\ C_{I_G} * 0.7152 \\ C_{I_B} * 0.0722 \end{bmatrix} \tag{4.43}$$

And $Lerp(x, y, t)$ being defined as:

$$Lerp(a, b, t) = a(1 - t) + bt \tag{4.44}$$

$$C_T = \begin{bmatrix} Lerp(C_{I_R}, I_R, F_S) \\ Lerp(C_{I_G}, I_G, F_S) \\ Lerp(C_{I_B}, I_B, F_S) \end{bmatrix} \tag{4.45}$$

4.9 Closing remark: order of calculation

The discussed order is written in order of how impactful this operation is on image quality, 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 (4.30) demonstrates the actual calculation sequence. Also shown is a separation between engine scripting and shader programming.

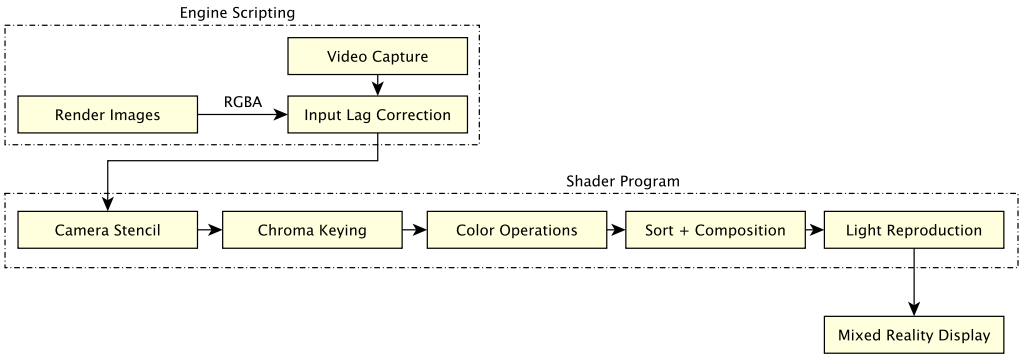


Fig. 4.30.: Actual order of computation

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

5 Evaluation & future work

5.1 Hardware setup variations

Due to the nature of this setup and fine-tweaking options inside the engine, this approach discussed can have a wide variety of operational setups. By lose coupling of these hardware factors there are only a few limitations that can either solved by better, future hardware or a different approach that are outside of the scope.

As integral part of this setup is the motion tracking solution, it is possible to hook up an Oculus Rift[1] instead of the HTC Vive and set one controller as camera-attachment point. For that there would be another 3D print needed to attach the controller to a camera solution.
Other third market VR-HMDs usually follow the Vives specifications closely and integrate natively with SteamVR, thus no modification on the original instalment has to be done.
Through Virtual Reality Peripheral Network or OpenVR simliar solutions can be developed, since none of the software explicitly depends on SteamVR features and allow a wide variety of systems and sensors.

The video capturing side can be varied greatly too, since the software only allows for webcam-compatible devices, it would be possible to remove the Inogeni encoder and replace it with cheap and simple webcams. Similarly can a camera replaced along with other recording solutions, as long as it outputs an HDMI (or HDMI-convertible) video stream. This allows recording to be as complex as needed but in general a DSLR camera will suffice.

Finally, since the full pipeline is rather complex, a good desktop PC will be needed. The CPU overhead is minimal but the graphical complexity is based on limitations of the GPU. With future iteration of this hardware it will be possible to create more complex scenery, output higher resolution video and better frame rates if the video capture device allows for it.
In theory a low-poly environment could be render-able on mobile, combined with a low-sample rate on the camera image, to produce a "window" into VR for multiple users at the same time, thus enabling an indefinite view into virtual reality. This

[1] with its room-scale setup

would remove the MR-pipeline from an actors system. Further work could tap into a Microsoft HoloLens solution, which could enable a direct contextualization of an actor into a VR scenery without a green screen at all. With fast approximate algorithms, like YCgCo-Keying, this could produce a high-framerate transparent overlay, so that the virtual scene does not clip the actor.

While working with this setup another possible use-case showed up: On June 2017 Apple presented their native Augmented Reality kit integrated in their consumer devices. Thus a similar system could be used to send all tracking parameters from the HTC Vive headset to these devices and have a calibrated north-wall with additional feature markers. This would potentially allow to have an augmented reality view around the actors world with a fast approximated actor position. It could then be possible for multiple users to use these devices as window into the virtual reality experience without a green screen at all.

5.2 Rendering Setup Variations

There are a few approaches that work similarly, but are different in complexity, their capabilities and hardware requirements. They have been already explored by producing companies and are covered here for completeness.

5.2.1 Single Camera - 3D plane in space

A novel approach is rendering the camera image onto a 2D plane positioned inside the 3D scene. This gives high control over projection parameters inside the engine without a running software and has good visual feedback for artists. All additional steps previously discussed are invalid, since this render can take full advantage of all runtime rendering parameters - which in turn means for lower render overhead and better graphical performance. It fails however on delay mitigation, which makes time drift between engine and camera visible, thus only usable with low latency video capturing devices. (Figure 5.1 for reference)

Fig. 5.1.: Demonstrative scene with a plane as camera rendering position

5.2.2 Deferred shading Path

Similarly to the single camera approach, a deferred shader could be used, in which the plane is projected and texturized after the scene has rendered and the graphics buffer is still present - this way the total time taken for rendering can be calculated and the chroma keying step can be chosen for a faster variant if needed. This gives generally better control and could yield higher performance, since all projected fragments have an assigned depth - the camera image only has to be calculated where the actors depth is smaller as the sceneries depth. This method is also lacking a way of adjusting for time drift. Additionally calculating lightning is relatively expensive, due to the reprojection of lighting parameters inside the graphics buffer. A snapshot of this buffer cannot be stored - which is a limitation of Unitys render pipeline - and is lost after the rendering loop completed.

5.2.3 Composition Workstation (4 patch)

Lastly, for full video production setups another rendering approach has been suggested, which was used for the initial promo material [**valve:vive-trailer:2016**] includes rendering a 4K video signal, outputting it to a composition PC which then takes care about managing time drift and video input from the camera.
The general concept involves a production of four 1920x1080 video signals of the virtual environment:

1. Foreground
2. Video matte of foreground
3. Background
4. First person view

Due to the system separation it is now possible to use green screen hardware compositors which have a visually higher quality in pulling the green background matte and then compositing it similarly into a mixed reality image. While this lifts some rendering overhead from the host PC, it fails in recreating a lightning environment for the VR actor and relies on two systems. While engine programming complexity decreases, operational setup complexity increases.

5.3 Rendering operational variations

This setup can handle another operational context by leaving out background-sorting and only rendering a virtual "front", by which a high quality augmented camera

Fig. 5.2.: Actual order of computation

system can be achieved. Since time drift between camera and engine is already handled, it is possible to render an augmented image. Since depth information is lost, it is not possible to handle obstructions - in example by an interacting user that is standing in front of the augmented object. However, with some composition and choreography, AR footage can be showed and captured in live production for further use. A reference plane has to be used, either by a Vive Tracker[2] or with feature markers.
This thesis assumes that the motion video feed is calibrated for a D65 white and augmented reality scenarios usually do not take real world lightning into account, it would give a good natural and high quality look into augmented reality use cases.

5.4 Edge Cases

The proposed method has edge cases which could be further improved by other approaches in rendering or capturing actor video. These highlight issues observed with the rendering operations discussed in this thesis.

5.4.1 Image Clipping - incorrect Z calculation for hands

Due to the planar projection of the real world feed inside the engine, any Z-information of the actor is squashed to a fixed depth. This means that hands

[2] like a controller

are on the same plane. In cases with high z-difference between actor and actors hands, it is possible that hand motion look unnatural and does not seems like it is to be supposed - in figure 5.3 is an actor depicted, that wraps his arms around a virtual cube. The produced mixed reality image shows his arms only behind the cube.

Fig. 5.3.: The actors hands do not visibly reach through the blue cube and the actor shears it because of the planar projection

One solution for future research could be acquiring an actors depth by either using Time of Flight cameras and using a resulting point cloud or by calculating each camera pixels depth with a stereoscopic solution. Thus each pixel could have a quantifiable depth with additional calibration parameters from the virtual reality tracking solution.

5.4.2 Matting failures

There are multiple problems while green screening, which are as old as green screens are used in video production. First and foremost is a partial transparent actor if his clothing consists of similarly green shaded material.
Additional green screen spill causes artifacts while pulling the matte which then clips the actor off. This can be mitigated by better production environments, i. e. with higher quality cloth and a generally well lit set[3].
Sometimes, for example in low light environments or folded green screen material, the background covers a wide color range, thus calibrating a good green with a single color value is nearly impossible. This could be solved by smaller selection margins inside the shader while assigning an array of colors for the ΔE calculation. In turn, this would be even more taxing on the GPU, since the calculation has be

[3]The appendix features a basic schematic of a small green screen setup.

done per selected color, multiplying the effort by the number of colors.
Lastly, real time chroma keying has problems with motion blur of the source video material - causing background mixing and invalid matting. This is a complex problem that is far beyond the scope of this thesis. One of the best solutions is a color unmixing approach researched by Disney and "typically requires 10s for local color estimation (assuming 8 dominant colors), another second to propagate the local color model to the following frame, and approximately 3s for color unmixing." [**disney:unmixing:2017**]

Add graphics depicting each issue. Also cross reference to chapter 4

5.4.3 Calibration problems (wrong clipping, wrong reprojection, long setup times)

The biggest margin of error is calibration of all projection parameters.
Beginning from field of view calculation, most DSLR cameras don't have specifications for their output feeds, where, for example, scaling factors could change the field of view angle. If these are not given or seem to be unfitting in production it is necessary to measure these parameters by hand, fixing the cameras position and calculating the spanning angle. Another factor are offset-parameters between controllers and camera sensor, which have been minimized by the 3D printed attachment. However, minimal differences in this transformation matrix have tremendous impact on miscalculated projections, visible by wrongly placed objects and a disconnect between virtual interaction and actor. Lastly, the most time spent after adding all mixed reality components to the scene is calibrating these parameters. A possible improvement could be done by fixing the cameras position, showing an overlay on the camera, where the secondary controller has to be placed and confirming it. This way the user can calibrate all projection parameters by himself with the help of a RANSAC / Lagrange Polynomial.

maybe add the ez calculation, too

Here comes a graphical representation of it

find out how this calibration is called:
https://www.youtube.com/watch?v=c_An0vxvPnk

6 Conclusion

I really, really need help here, Kristian. :<

6.1 3D Environment and Composition Considerations

While working with this pipeline there have been some problematic cases discovered, which generally do not work well with mixed reality. Very cluttered environment can easily have obstructing meshes that clip - partially or completely - through the virtual camera frustum and obstruct 3D projection. Minimizing this issue can be done by either disallowing a user to move the SteamVR bounding box at all, only allow for certain teleporting positions or do permit teleportations altogether that potentially clip through the SteamVR bounding box.
A green screen is crucial for a live performance and recent papers focus only on increased accuracy with unacceptable computation times in this context. Due to the needs of a studio-like setup it is additional overhead for having a mobile, transportable experience. However the gains of Mixed Reality might be very well worth it.
By placing a virtual monitor showing the final mixed reality composition it is possible to communicate and contextualize the real world actors performance back to himself, allowing him to have a third person perception of his actions.

6.2 Performance Considerations

Due to the nature of this rendering pipeline there is a significant performance overhead caused - since multiple virtual cameras are capturing the scenery at a high frame rate - and is overall very taxing on the GPU. Each camera change causes, including the two VR projections, multiple context changes, which is additionally drags down performance. The ΔE chroma keying operation runs per pixel twice through an if/else case which are likely[1] evaluated on both cases and later one of

[1] Unfortunately the Unity GPU profiler does not provide that data and external GPU debuggers seem to simply crash while attaching to Unitys rendering context

those is used, adding another layer of computation complexity on top.
The pipeline is already built in such a way to reduce the amount of cameras, but needs at least 4 virtual projections. Each feature, but fore- and background, can be disabled to ease the performance hit. With that in place it is possible to balance between accuracy and computation speed - with further considerations by doing full mixed reality on mobile phones or different performing devices, a dynamic rendering pipeline could be created that manages composition modes on runtime. Further exploration could be done to include Valves "The Lab Renderer" which also manages asset fidelity and rendering operations dynamically to hit an optimal frame rate. Future work could involve a reduced sampling rate on the chroma key operations for further performance improvement.
This pipeline is able to render about 120.000 triangles with pre-baked lightning and disabled post-fx on the set PC. This works well for small room-sized experiences but likely needs more performance improvement for more ambitious looking scenery.

Unitys' Monobehaviour Loop

A

The behavior of a Unity-initiated object is outlined by the following flowchart in A.1, taken from Unitys' manual.

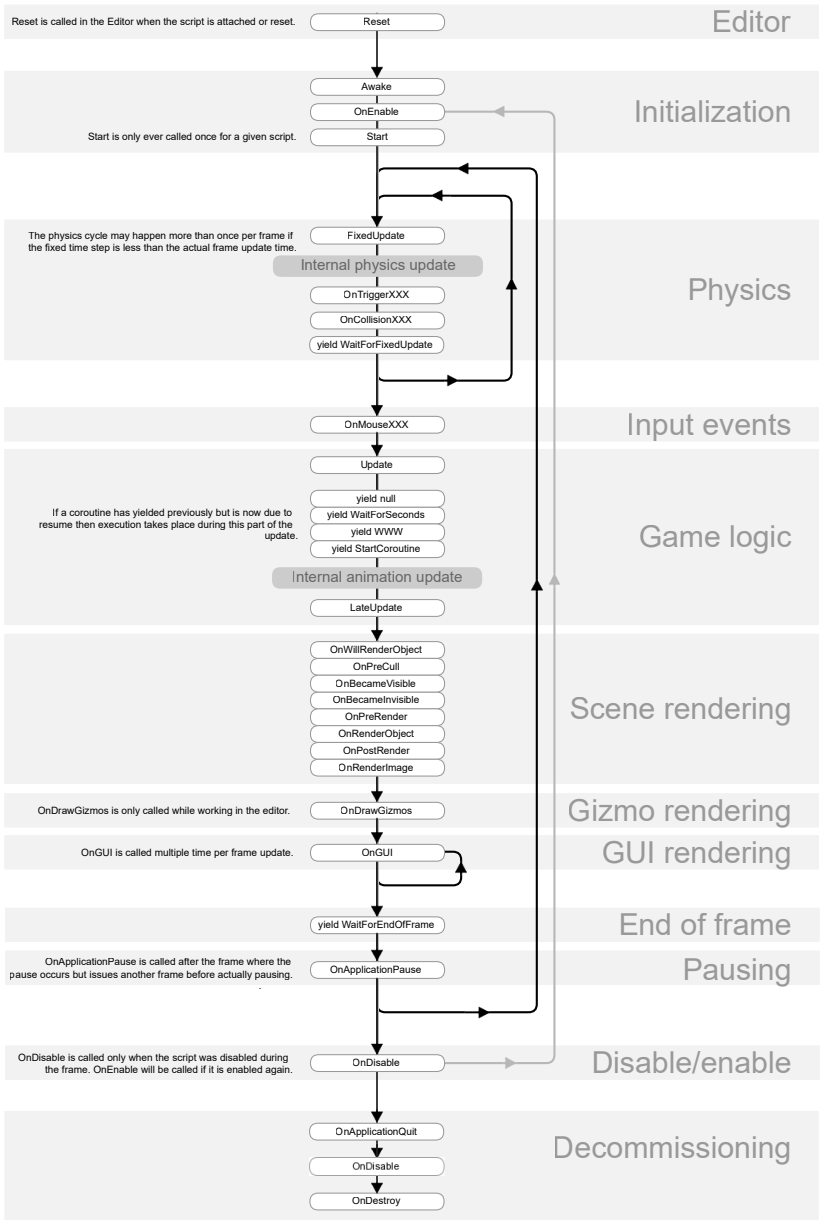


Fig. A.1.: Monobehaviour Flowchart

Simple Green Screen Setup

B

Building a green screen set is no easy task and takes a lot of careful consideration in light setup, background coloring and material used on set.
Figure B.1 shows a very simple and low-cost green screen setup after Foster et al. [**foster:greenscreen:2010**]

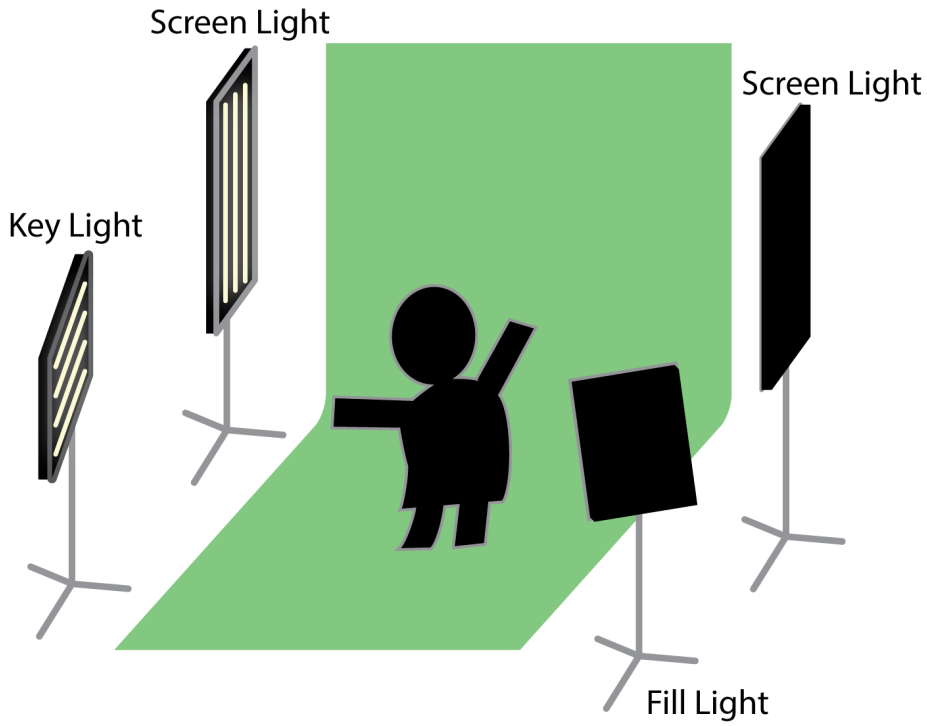


Fig. B.1.: Basic green screen setup

List of Figures

2.2 I^3 Triangle - figurative quantization of different reality extending methods 7

3.1 Diagram of hard- and software components. 13
3.2 Camera mount for a HTC Vive controller, mounted on the cameras tripod mount 16

4.1 Full mixed reality graphics pipeline in order of graphical fidelity impact 18
4.2 Initial step upon receiving the camera image 18
4.3 Comparison Image [**vimeo:shia:2015**] - sRGB Output 19
4.4 Chroma Keying by using euclidean RGB distance 20
4.5 Chroma Keying by using euclidean YCgCo distance 21
4.6 Chroma Keying by using ΔE distance 25
4.7 Normalized graph comparing color difference methods 25
4.8 Comparison between different color distance methods 26
4.10 Comparison with different CIEDE ΔE variants 27
4.11 Initial step upon receiving the camera image is to adjust the time drift between engine renderings and video capture 27
4.12 Some Argument 28
4.14 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. 29
4.15 Schema of an the ringbuffer 31
4.16 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. 33
4.17 Distance correlation between real world camera and the VR actor . . . 34
4.18 Current steps taken through the graphics pipeline 35
4.19 A sketch of the video composition with three layers of projection 35
4.20 A comparison of different composition methods in engine 36
4.22 After sorting the scene additional stenciling of the video feed is necessary 38
4.23 Some Argument 38
4.25 Following in this pipeline is lights reproduction 39
4.26 Left: original, Right: reconstruction 40
4.27 A comparison of different composition methods in engine 41
4.29 Initial step upon receiving the camera image 41

4.30 Actual order of computation 44

5.1 Demonstrative scene with a plane as camera rendering position 46
5.2 Actual order of computation 48
5.3 The actors hands do not visibly reach through the blue cube and the actor shears it because of the planar projection 49

A.1 Monobehaviour Flowchart 55

B.1 Basic green screen setup 56

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, September 3, 2017

Martin Zier