Technology
Arts Sciences
TH Köln

Technische Hochschule Köln
Fakultät für Informations-, Medien- und Elektrotechnik

MASTERPROJEKT

Dynamische 4D-Rekonstruktion mit Spacetime und HAC-Gaussians: Eine End-to-End Lösung für Human Performance Capture

TH Köln
Campus Deutz
Medientechnologie Master

vorgelegt von: Kai Altwicker
Dennis Luca Amuser
Matthias Bullert
David Martin Karg
David Mertens
Alisa Rüge
Steffen-Sascha Stein
Marvin Winkler

Prüfer: Prof. Dr.Ing. Arnulph Fuhrmann

Köln, 12.05.2025

Kurzfassung / Abstract

Das Projekt VolumanXR, entwickelt im Rahmen des Masterstudiengangs Medientechnologie an der TH Köln, präsentiert eine vollständige Pipeline zur Aufnahme und Verarbeitung volumetrischer Daten für 4D-Gaussian-Splatting. Die Kernkomponenten umfassen ein synchronisiertes Mehrkamera-Aufnahmesystem, eine robuste ML-basierte Trainingspipeline und eine immersive VR-Anwendung zur Wiedergabe auf Head-Mounted Displays. Hinzu kommt eine optimierte Pipeline für humanoide animierbare auf Gaussians basierende Avatar-Charaktere. Das Projekt gliedert sich in vier Untergruppen: Das Volumetric Capture System besteht aus einem oktogonalen Rig mit 68 synchronisierten Raspberry Pi-Kameras, welches eine hochwertige 360°-Erfassung von Personen ermöglicht. Die Trainingsgruppe entwickelte eine leistungsfähige Pipeline zur Verarbeitung der Aufnahmen mittels Spacetime Gaussians, optimiert für Szenen mit entferntem Hintergrund und variablen Bildauflösungen. Der VR-Viewer realisiert die Darstellung der volumetrischen Videos auf VR-Brillen, wobei Path-Through-Funktion sowie die Möglichkeit zur Navigation und Manipulation der 3D-Szene implementiert wurden. Die HAC-Gaussians-Komponente untersuchte fortschrittliche Methoden zur verbesserten Repräsentation von Bewegungsdaten auf fotorealistischen Avataren. Die Realisierung des Projekts erfolgte mit beachtlicher Kosteneffizienz, unterstützt durch ein Budget von 7.500€ von der Initiative Kickstart. Wesentliche Erfolge umfassen die Entwicklung eines modularen und erweiterbaren Aufnahmesystems mit professioneller Bildqualität, eine optimierte Trainingspipeline mit signifikanter Reduktion der Berechnungszeit für Gaussian Splats sowie ein interaktiver Browser-basierter Viewer für die immersive Darstellung der volumetrischen Inhalte. Das System bietet eine flexible Plattform für zukünftige Forschung an der Schnittstelle von Medientechnologie, Computergrafik und KI-gestützter Bildverarbeitung.

Inhaltsverzeichnis

1 Einleitung

Diese Dokumentation beschreibt das Projekt *VolumanXR*, das im Rahmen des Masterstudiengangs Medientechnologie an der TH Köln während des Projektsemesters entwickelt wurde. Ziel von VolumanXR war es, eine Pipeline zur Aufnahme und Verarbeitung für 4D-Gaussian-Splatting zu entwickeln. Diese umfasst die synchronisierte Videoaufnahme mit einem Mehrkamera-Rig, eine robuste Trainingspipeline sowie eine immersive XR-Anwendung zur Wiedergabe auf einem Head-Mounted Display (HMD). Zudem wurden weiterführende Techniken mittels HAC-Gaussians untersucht, um Bewegungsdaten präzise aus dem aufgenommenen Material extrahieren und darstellen zu können.

Das Projekt gliedert sich in vier Untergruppen:

- **Volumetric Capture System**: Zuständig für den Aufbau eines synchronisierten Mehrkamera-Aufnahmesystems zur Erstellung qualitativ hochwertiger volumetrischer Videoaufnahmen. (**Kai Altwicker, Dennis Amuser**)
- **Training**: Verantwortlich für die Verarbeitung der aufgenommenen Videos durch eine Machine-Learning-Pipeline, um hochwertige 4D-Gaussian-Repräsentationen zu erzeugen. (**Matthias Bullert, David Mertens**)
- **Viewer**: Erarbeitung eines intuitiven XR-Viewers für die immersive Visualisierung der volumetrischen Aufnahmen auf HMD-Geräten. (**David Martin Karg, Alisa Rüge, Steffen-Sascha Stein**)
- **HAC-Gaussians**: Untersuchung der Implementierung von HAC-Gaussian-Techniken zur verbesserten Extraktion und Repräsentation von Bewegungsdaten. (**Marvin Winkler**)

Steffen-Sascha Stein führte die Projektleitung bis kurz vor dem Abschluss des Projekts, in der Schlussphase übernahm Kai Altwicker aufgrund gesundheitlicher Gründe die stellvertretende Projektleitung. Unterstützt wurde das Projekt durch Kooperationen insbesondere mit dem Makerspace der TH Köln, der umfangreiche technische Unterstützung und Fertigungsmöglichkeiten bot. Weitere wertvolle Beiträge kamen von Prof. Fuhrmann sowie der Zentralwerkstatt Elektrotechnik der TH Köln.
Finanziell wurde das Projekt maßgeblich durch die Initiative Kickstart, ein Teil des StartUpLab@TH Köln, unterstützt, die eine Förderung in Höhe von 7.500€ bereitstellte. Diese Mittel waren entscheidend für die Entwicklung des Prototypen. Zusätzlich erfolgte ein Mentoring durch Kickstart, welches der Projektgruppe Möglichkeiten für eine potenzielle Ausgründung aufzeigen sollte. Weitere 1.000€ wurden aus dem Budget für Masterprojekte der Fakultät zur Verfügung gestellt.
Die weitere Dokumentation ist in die Kapitel 2 Grundlagen, 3 Vorbereitende Arbeiten, 4 Umsetzung sowie 5 Evaluation gegliedert. Jedes dieser Kapitel adressiert subteamspezifische technische und praxisrelevante Aspekte des Projekts. Dabei wird jeweils explizit aufgeführt, welche Teammitglieder an den entsprechenden Arbeitsschritten beteiligt waren. Die Reihenfolge der Nennung orientiert sich absteigend am Umfang der jeweiligen Mitwirkung.

2 Grundlagen

2.1 Volumetric Capture

Das Forschungsgebiet rund um Volumetric Capture wird bereits seit geraumer Zeit erforscht. Es befasst sich mich dem Ziel Dinge der reellen Welt, darunter Menschen, Tiere, Objekte oder auch Umgebungen, so zu erfassen, dass Sie räumlich in Form von 3D-Szenen dargestellt werden können und daraus neuartige Sichtweisen generiert werden können. Besonders interessant für die Forschung ist dabei die photorealisistische Darstellung von Menschen. Diese Darstellung soll jedoch nicht auf eine statische 3D-Szene begrenzt sein, sondern auch den zeitlichen Aspekt berücksichtigen und die Darstellung so zu einem 4D-Volumetrischen Video machen. Unterschiedliche Methoden dazu werden daher in vielen verscheidenden Forschungsprojekten der letzten Jahre untersucht. Einiger dieser Methoden und Ansätze stechen aufgrund verschiedener Aspekte heraus. Diese Aspekte sind zum einen Qualität, Kommerzialisierungsfähigkeit aber auch Kosteneffizienz. Das Projekt VolumanXR zielt darauf ab, eine kosteneffiziente Methode zu entwickeln, um Menschen mit einem speziellen Kamera-Rig aus einer Vielzahl von Perspektiven aufzunehmen, wobei der finanzielle Rahmen berücksichtigt wird. Im Anschluss werden die erfassten Daten unter Zuhilfenahme neuartiger Verfahren weiterverarbeitet. Ziel dieser Verfahren ist es, eine photorealistische und gleichzeitig recheneffiziente Darstellung zu unterstützen, um diese in einer Browseranwendung auf einem HMD darzustellen. Im Folgenden werden einige Lösungen bisheriger Forschungsprojekte erörtert, die Aspekte wie Qualität, Kommerzialisierungsfähigkeit oder Kosteneffizienz adressieren und die Umsetzung des Projektes VolumanXR inspiriert haben.

2.1.1 The Relightables

Ein herausragendes Beispiel für ein hochqualitatives volumetrisches Aufnahmesystem stellt das von Google entwickelte Projekt *The Relightables* dar (Guo u. a. 2019). Das System kombiniert klassische Verfahren der 3D-Rekonstruktion mit Fortschritten aus dem Bereich Deep Learning, um fotorealistische und unter beliebigen Lichtbedingungen relightbare 4D-Videos von Menschen zu erzeugen. Der Kern des Systems besteht aus einem Kamera-Rig mit 331 programmierbaren LED-Lichtquellen sowie 90 synchronisierten Kameras, bestehend aus 58 hochauflösenden RGB-Kameras und 32 Infrarotkameras. Ergänzt wird dies durch 16 strukturierte Lichtprojektoren, die Tiefeninformationen mit einer Auflösung von 4112×3008 Pixeln bei 60 Hz liefern. Zur Erfassung photometrischer Eigenschaften wird während der Aufnahme eine abwechselnde Beleuchtung mit farbigen Gradientenmustern eingesetzt, was die Berechnung von Albedo, Oberflächennormalen, Glanz und Umgebungsverdeckung direkt in UV-Texturräumen ermöglicht. Durch diese präzise Erfassung und Verarbeitung erreicht *The Relightables* eine herausragende Detailtreue — sichtbar unter anderem an der naturgetreuen Wiedergabe von Hautfalten und Kleidungstexturen — und eignet sich damit besonders für Anwendungen, bei denen fotorealistische Darstellungen von Menschen unter verschiedenen Lichtverhältnissen essenziell sind. (Guo u. a. 2019)
The Relightables ist ein Forschungsprojekt von Google AI. Es zeigt, wie mit traditionellen Rekonstruktionsmethoden hochqualitative Ergebnisse erziehlt werden können (Palladino 2019). Diese sind jedoch mit einem enormen Aufwand an Rechenressourcen sowie besonderen technische Anforderungen an den Kamera-Rig verbunden.

2.1.2 Kommerzialisierte Volumetric Capture Studios

In den letzten Jahren haben mehrere Unternehmen kommerzielle volumetrische Studios entwickelt, die sich in Ausstattung, Skalierbarkeit und Mobilität unterscheiden. Im Folgenden werden vier Beispiele vorgestellt.

Volucap (Potsdam-Babelsberg) Das Volumatric Capture Studio von *Volucap* in Potsdam-Babelsberg gilt als eines der fortschrittlichsten volumetrischen Studios Europas. Es ist mit 32 synchronisierten Kameras ausgestattet, die jeweils über einen 65-Megapixel-9K-Sensor mit Global Shutter verfügen. Dies ermöglicht eine hochpräzise Erfassung von Personen in Bewegung. (Schreer u. a. 2019) (befores und afters 2022)
Die Beleuchtung erfolgt durch 235 ARRI SkyPanels, die für eine gleichmäßige Ausleuchtung sorgen. Volucap bietet zudem portable Lösungen wie das „Volucap Max"-System an, das für mobile Anwendungen konzipiert ist.(Ehrenhöfer 2025)

Microsoft Mixed Reality Capture Studios Microsoft etablierte mehrere Mixed Reality Capture Studios, unter anderem in San Francisco, London und Tokio. Diese Studios nutzten 106 synchronisierte Kameras, um 360-Grad-Aufnahmen von Personen zu erstellen. Die erfassten Daten wurden über die Azure-Cloud-Plattform verarbeitet. (Collet u. a. 2015)
Im Jahr 2023 wurde jedoch bekannt, dass Microsoft das interne Team der Mixed Reality Capture Studios im Zuge von Umstrukturierungen aufgelöst hat. (Lowpass 2023)

Intel Studios (Los Angeles) Intel Studios betrieb eine der größten volumetrischen Aufnahmestudios weltweit mit einer Fläche von ungefähr 930 Quadratmetern. Das Studio war in der Lage, komplexe Szenen mit mehreren Darstellern gleichzeitig aufzunehmen. Im Jahr 2018 arbeitete Intel Studios beispielsweise mit Paramount zusammen, um eine volumetrische Performance des Musicals „Grease" mit 20 Tänzern aufzuzeichnen. (Intel 2020)
Aufgrund wirtschaftlicher Bedingungen und der Auswirkungen der COVID-19-Pandemie wurde das Studio im Jahr 2020 geschlossen (Farrell 2025).

Polymotion Stage (MRMC) Die Polymotion Stage von Mark Roberts Motion Control (MRMC) ist ein mobiles volumetrisches Aufnahmestudio, das in einem LKW-Anhänger untergebracht ist. Es verfügt über 106 Kameras, die in einer Kombination aus 96 horizontalen und 10 vertikalen Positionen angeordnet sind, um eine vollständige 360-Grad-Erfassung zu ermöglichen. (*Polymotion Stage* 2025)
Das Studio kann in kurzer Zeit aufgebaut werden und bietet eine flexible Lösung für On-Location-Drehs. Es wurde bereits bei verschiedenen Produktionen eingesetzt, darunter Sportveranstaltungen und Fernsehserien. (*Polymotion Stage Case Study* 2025)

2.1.3 Kosteneffiziente Volumetric-Capture-Systeme

Neben hochwertigen Studiolösungen existieren auch Forschungsansätze, die auf kostengünstiger Hardware basieren. Ein exemplarisches Projekt wurde von Bönsch u. a. 2019 vorgestellt,

in dem ein Volumetric Capture System auf Grundlage eines Photogrammetrie-Cages mit 104 unsynchronisierten Raspberry Pi Kameras entwickelt wurde.
Der Aufbau verwendet Raspberry Pi 2 B+ Einheiten in Verbindung mit Pi Camera Module v1.3. Um die fehlende Synchronisation unter den Raspberry Pis zu kompensieren, erfolgt die zeitliche Abstimmung der Aufnahmen mittels zweier ausgelöster optischer Blitze zum Beginn sowie zum Ende einer Aufnahme, die in den Videos automatisch über Luminanzdifferenzen erkannt werden. Eine mehrstufige Softwarepipeline übernimmt anschließend die Bildextraktion, Synchronisation und framebasierte Mesh-Rekonstruktion. Für die 3D-Rekonstruktion der einzelnen Frames wurde die kommerzielle Photogrammetriesoftware *Agisoft Photoscan* eingesetzt.
Trotz begrenzter Bildqualität und einer Framerate von 25 fps erreicht das System eine überraschend hohe Detailtreue — inklusive mimischer Merkmale, Kleidung und Haarbewegung. Die Meshes werden auf einem 50-Knoten-Cluster berechnet, wodurch die gesamte Verarbeitung eines 30-Sekunden-Clips mit 104 Kameras in etwa 4 Stunden möglich ist. Im Vergleich zu professionellen Studiosystemen stellt diese Lösung einen bemerkenswert günstigen Einstieg in die volumetrische Aufnahme dar, insbesondere für Forschungs- oder Bildungszwecke.

2.2 Gaussian Splatting

Beteiligte Teammitglieder: Matthias Bullert

Die Darstellung von 3D-Szenen für neuartige Sichtweisen (*novel view synthesis*) hat in den letzten Jahren große Fortschritte gemacht. Traditionelle Methoden wie Mesh-basierte Modelle Riegler und Koltun 2020 Yang u. a. 2013 oder explizite Punktwolken Zhou und Koltun 2013 stoßen oft an ihre Grenzen, insbesondere hinsichtlich Effizienz und Qualität. Neuartige Methoden wie Neural Radiance Fields (NeRF) Mildenhall u. a. 2021 bieten zwar exzellente Ergebnisse, sind jedoch rechenintensiv.
Ein vielversprechender Ansatz ist 3D Gaussian Splatting Kerbl u. a. 2023, eine Methode, die die Vorteile von expliziten Punkt-basierten und kontinuierlichen volumetrischen Darstellungen kombiniert. Dabei wird jede Szene als eine Menge von 3D-Gaussians beschrieben, die jeweils durch folgende Parameter definiert sind:

- **Position** $\mu \in \mathbb{R}^3$: Definiert den Mittelpunkt des Gaussians in der Szene.
- **Kovarianzmatrix** $\Sigma \in \mathbb{R}^{3\times3}$: Bestimmt die anisotrope Form der Gaussians und ermöglicht eine präzisere Anpassung an die Geometrie.
- **Opazität** α: Steuert, wie stark eine Gaussian zur Darstellung der Szene beiträgt.
- **Farbwerte** (RGB) und **sphärische Harmonische** (SH): Modellieren die Beleuchtung und Ansichtabhängigkeit der Darstellung.

zunächst wird die Szene durch eine initiale Punktwolke beschrieben, die aus einer *Structure-from-Motion (SfM)*-Rekonstruktion stammt. Diese initialen Punkte bilden die Basis für die 3D-Gaussians, deren Parameter im Trainingsprozess optimiert werden.
Der Optimierungsprozess umfasst mehrere Kernkomponenten:

- **Positionsoptimierung**: Die Positionen μ der Gaussians werden iterativ durch Stochastic Gradient Descent (SGD) aktualisiert, um die Szene möglichst genau zu rekonstruieren.
- **Optimierung der Kovarianz** Σ: Die Kovarianzmatrix steuert die Form und Orientierung der Gaussians. Eine anisotrope Kovarianz erlaubt eine genauere Anpassung an geometrische Details und wird durch eine spezielle Transformation

$$\Sigma = RSS^T R^T \tag{1}$$

  optimiert, wobei R die Rotation und S die Skalierung beschreibt.
- **Adaptive Dichtekontrolle**: Während der Optimierung können Gaussians hinzugefügt oder entfernt werden. In Regionen mit geringer Abdeckung werden Gaussians dupliziert und verschoben, während in überrepräsentierten Bereichen große Gaussians in kleinere aufgespalten werden.
- **Sphärische Harmonische für Farbmodellierung**: Um Ansichtabhängigkeit und Beleuchtungseffekte zu modellieren, werden Farbwerte nicht direkt gespeichert, sondern mit sphärischen Harmonischen (SH) bis zu einer bestimmten Ordnung dargestellt.

Spacetime Gaussians

Beteiligte Teammitglieder: David Mertens

In dem Paper SSpacetime Gaussian Splatting for Dynamic Scene Reconstruction"wurde das Konzept der Spacetime Gaussians vorgestellt. Diese erweitern das klassische 3D Gaussian Splatting, indem sie nicht nur die räumlichen, sondern auch die zeitlichen Veränderungen einer Szene erfassen. Sie bilden die Grundlage für die Szenenrekonstruktion im Rahmen des in dieser Dokumentation beschriebenen Projekts.
Während klassisches 3D Gaussian Splatting ausschließlich für die Darstellung statischer Szenen geeignet ist, integriert Spacetime Gaussian Splatting eine zeitliche Dimension, sodass auch dynamische Inhalte rekonstruiert werden können. Dies wird erreicht, indem sämtliche Eigenschaften eines Gaussians – darunter Position, Kovarianz und Farbe – als Funktionen der Zeit ttt modelliert werden.
Statt für jedes Frame eine vollständig neue Menge an Gaussians zu generieren, bleiben die einzelnen Gaussians über mehrere Frames hinweg erhalten. Innerhalb dieses Zeitraums passen sie dynamisch ihre Position, Orientierung, Farbe und Größe an, um Bewegungen innerhalb der Szene zu folgen. Dadurch wird eine effizientere und kohärentere Darstellung dynamischer Inhalte ermöglicht, ohne redundante Rekonstruktionen für jeden einzelnen Frame durchführen zu müssen.
Die zeitliche Entwicklung der einzelnen Parameter wird durch die folgenden Funktionen beschrieben:

- Position: Die 3D-Position eines Gaussians verändert sich über die Zeit und wird durch eine Funktion $\mathbf{x}(t)$ beschrieben:

$$\mathbf{x}(t) = \mathbf{x}_0 + \mathbf{v}t \tag{2}$$

wobei $\mathbf{x}_0$ die Ausgangsposition und $\mathbf{v}$ die geschätzte Geschwindigkeit des Gaussians ist.

- Kovarianz: Die Kovarianzmatrix Σ, welche die räumliche Ausdehnung des Gaussians beschreibt, kann ebenfalls zeitabhängig sein:

$$\Sigma(t) = R(t)\Lambda(t)R(t)^T \tag{3}$$

wobei $R(t)$ eine zeitabhängige Rotationsmatrix und $\Lambda(t)$ eine diagonale Skalierungsmatrix ist, die anisotrope Veränderungen beschreibt.

- Farbe: Die Farbwerte eines Gaussians ändern sich über die Zeit, um Beleuchtungseffekte oder Bewegungen zu reflektieren:

$$\mathbf{c}(t) = (c_r(t), c_g(t), c_b(t), \alpha(t)) \tag{4}$$

Hierbei sind c_r, c_g, c_b die Farbkanäle und α die Opazität. Diese Werte können durch Interpolation über die Zeit hinweg angepasst werden.

- Orientierung: Die Rotation eines Gaussians wird über eine zeitabhängige Rotationsmatrix $R(t)$ dargestellt:

$$R(t) = R_0 + \omega t \tag{5}$$

wobei R_0 die initiale Rotation und ω die Winkelgeschwindigkeit ist.

- Skalierung: Die Größe eines Gaussians kann ebenfalls variieren:

$$s(t) = s_0 + \dot{s}t \tag{6}$$

wobei s_0 die anfängliche Skalierung und $\dot{s}$ die Änderungsrate über die Zeit ist.

2.3 HAC

Beteiligte Teammitglieder: Marvin Winkler

HAC basierend auf Gaussians (HAC-Gaussians) sind ein innovativer Ansatz zur Erzeugung und Animation fotorealistischer virtueller Menschen, der die Rendereffizienz von *3D Gaussian Splatting* mit fortschrittlichen Techniken zur Körperanimation kombiniert. Diese Technologie ermöglicht die Erstellung detailreicher, ausdrucksstarker digitaler Avatare mit komplexer Deformationen von Körper und Kleidung. Im Gegensatz zu traditionellen Techniken bieten HAC-Gaussians eine verbesserte Balance zwischen visueller Qualität und Berechnungseffizienz. Diese Arbeit basiert auf aktuellen Entwicklungen im Bereich der dreidimensionalen Avatarerzeugung und -animation mittels 3D Gaussian Splatting. In diesem Abschnitt werden die relevanten Grundlagen und der aktuelle Forschungsstand erläutert.

2.3.1 3D Gaussian Splatting für Avatar-Modellierung

3D Gaussian Splatting ist eine effiziente Methode zur expliziten Darstellung dreidimensionaler Szenen, die im Vergleich zu impliziten *Neural Radiance Fields (NeRF)* eine höhere Rendergeschwindigkeit bei gleichzeitig hoher visueller Qualität bietet. Die grundlegende Idee besteht darin, eine Szene durch eine Sammlung von 3D Gaussians zu repräsentieren, welche durch ihre Position, Rotation, Skalierung, Farbe und Opazität definiert sind. Diese Primitiven können effizient auf die Bildebene projiziert werden, was ein schnelles Rendering ermöglicht. Die Anwendung dieser Technik auf menschliche Avatare stellt aufgrund der Komplexität menschlicher Bewegungen besondere Herausforderungen dar. Verschiedene Forschungsgruppen haben in den letzten Jahren unterschiedliche Ansätze entwickelt, um dieses Problem zu lösen. Liu et al. Liu u. a. 2024 demonstrieren die Effizienz dieser Methode, indem sie hochwertige Avatar-Rekonstruktionen in nur 5-30 Sekunden Training erreichen und Echtzeit-Rendering mit 40-170 Frames Per Second (FPS) ermöglichen. Moreau et al. Moreau u. a. 2024 stellen mit *HuGS* eine Methode vor, die photorealistisches Rendering mit etwa 80 FPS bei 512×512 Auflösung ermöglicht und dabei Animationsfähigkeiten für neue Posen bietet.

2.3.2 Deformationstechniken für 3D Gaussians

Eine zentrale Herausforderung bei animierbaren Avataren ist die Deformation der 3D Gaussians entsprechend menschlicher Bewegungen. Hierfür wurden verschiedene Ansätze entwickelt.

Linear Blend Skinning und seine Erweiterungen Traditionelle Ansätze zur Charakter-Animation verwenden oft *Linear Blend Skinning (LBS)*, bei dem Vertices durch gewichtete Transformationen von Knochen bewegt werden. Z. Qian u. a. 2024 kombinieren in ihrem 3DGS-Avatar-Ansatz 3D Gaussian-Primitive mit dem parametrischen *SMPL*-Körpermodell und lernen ein Netzwerk für nicht-rigide Deformationen, um komplexe posenabhängige Verformungen zu handhaben. Moreau u. a. 2024 verwenden in ihrem HuGS-System einen *Coarse-to-Fine*-Ansatz, der Forward Skinning für skelettbasierte Bewegungen mit einem neuronalen Netzwerk für lokale Kleidungsdeformationen kombiniert.

Alternative Deformationstechniken Zielonka u. a. 2025 überwinden die Grenzen von LBS mit ihrem *D3GA*-Ansatz (*Drivable 3D Gaussian Avatars*), indem sie 3D Gaussians in tetraedrische Käfige einbetten. Diese käfigbasierte Deformation ermöglicht eine natürlichere Reorientierung und Streckung der Gaussians während der Deformation. Jung u. a. 2023 präsentieren mit *ParDy-Human* einen alternativen Ansatz mit deformierbarem 3D Gaussian Splatting. Ihr System verwendet ein zweistufiges Design, bestehend aus einem SMPL-basierten Vertex-Deformationsmodul und einem Deformations-Verfeinerungsmodul. Shao u. a. 2024 führen mit *SplattingAvatar* einen hybriden Ansatz ein, der Gaussian Splatting in ein Dreiecksnetz einbettet. Die Gaussians werden mit baryzentrischen Koordinaten auf Mesh-Dreiecken definiert und können durch eine spezielle Optimierungstechnik über Dreiecksgrenzen hinweg "wandern".

2.3.3 Detailmodellierung und Erscheinungsqualität

Die realistische Darstellung von Kleidungsdetails, Falten und Texturen stellt eine besondere Herausforderung dar. Zhe Li u. a. 2024 konzentrieren sich auf hochdetaillierte dynamische Avatare mittels pose-abhängiger *Gaussian Maps*. Ihr Ansatz parametrisiert eine Vorlage auf kanonischen Gaussian Maps und verwendet ein *StyleGAN*-basiertes Convolutional Neural Network (CNN) zur Vorhersage pose-abhängiger Details. Diese Methode eignet sich besonders für Kleidung mit loser Dynamik wie Kleider. Hu u. a. 2024 verbessern mit *GaussianAvatar* die Detailgenauigkeit durch ein dynamisches Erscheinungsnetzwerk mit optimierbarem Feature-Tensor. Durch die gemeinsame Optimierung von Bewegung und Erscheinung werden nicht nur bessere visuelle Ergebnisse erzielt, sondern auch fehlausgerichtete Bewegungen korrigiert. Saito u. a. 2024 fokussieren sich auf realistische Beleuchtungseffekte mit ihren *Relightable Gaussian Codec Avatars*. Ihr Ansatz zerlegt die Erscheinung in diffuse und spiegelnde Komponenten und verwendet sphärische Harmonics für diffuse und sphärische Gaussians für spiegelnde Reflexionen.

2.3.4 Kopf- und Gesichtsmodellierung

Für viele Anwendungen ist die detaillierte Modellierung von Kopf und Gesicht besonders wichtig. S. Qian u. a. 2024 stellen mit *GaussianAvatars* eine Methode für photorealistische Kopf-Avatare mit gerippten 3D Gaussians vor. Ihr Ansatz kombiniert die Flexibilität von 3D Gaussian Splatting mit der Steuerbarkeit parametrischer Gesichtsmodelle, indem 3D Gaussians an Dreiecke eines parametrischen Gesichtsmodells (*FLAME*) angeheftet werden. Xiang u. a. 2024 präsentieren *FlashAvatar*, eine leichtgewichtige 3D-animierbare Kopf-Avatar-Darstellung, die in Minuten aus einem kurzen monokularen Video rekonstruiert werden kann und mit 300 FPS auf handelsüblichen Graphics Processing Unit (GPU)s bei 512×512 Auflösung rendert. Ihr Ansatz erhält ein einheitliches 3D-Gaussian-Feld, das in die Oberfläche eines parametrischen Gesichtsmodells eingebettet ist. Moon, Shiratori und Saito 2024 integrieren in *ExAvatar* die Gesichtsmodellierung in einen Ganzkörper-Avatar, der mit dem SMPL-X-Ausdrucksraum kompatibel ist und neuartige Gesichtsausdrucksanimationen ermöglicht.

2.3.5 Ganzheitliche Avatar-Modellierung

Die vollständige Modellierung eines Avatars umfasst neben dem Körper auch Gesicht und Hände, die für expressive Darstellungen unerlässlich sind. Moon, Shiratori und Saito 2024 adressieren

diese Herausforderung mit ExAvatar, einem expressiven Ganzkörper-3D-Avatar, der Gesichts-, Körper- und Handanimationen unterstützt. Ihr Ansatz kombiniert das Ganzkörper-parametrische Modell SMPL-X mit 3D Gaussian Splatting und führt eine hybride Darstellung ein, die 3D Gaussians als Vertices mit Mesh-Konnektivität behandelt. Zielonka u. a. 2025 bieten mit ihrem schichtbasierten Ansatz ebenfalls eine Lösung für die ganzheitliche Avatar-Modellierung, indem sie separate Steuerungsmechanismen für Körper, Gesicht und Hände ermöglichen. Diese Modularität erlaubt eine präzisere Kontrolle über verschiedene Aspekte des Avatars. Yuan u. a. 2024 präsentieren mit *GAvatar* eine Methode zur Synthese hochwertiger 3D-animierbarer Avatare aus Textaufforderungen. Ihr primitiv-basierter impliziter Gaussian-Ansatz definiert 3D Gaussians innerhalb posengesteuerter Primitive, um die Animation zu erleichtern.

2.3.6 Lernen aus begrenzten Daten

Die Erstellung hochwertiger Avatare aus begrenzten Trainingsdaten ist ein zentrales Forschungsthema. Jung u. a. 2023 und Hu u. a. 2024 zeigen, dass qualitativ hochwertige Avatare bereits aus einem einzigen monokularen Video erzeugt werden können. Dies wird durch intelligente Regularisierungstechniken und die explizite Natur von 3D Gaussians ermöglicht. Z. Qian u. a. 2024 demonstrieren mit *3DGS-Avatar* eine 400-fache Beschleunigung des Trainings (30 Minuten auf einer einzelnen GPU) im Vergleich zu bisherigen Methoden, was die praktische Anwendbarkeit deutlich verbessert. Zhe Li u. a. 2024 verbessern die Generalisierung auf neue Posen durch eine *PCA*-basierte Pose-Projektionsstrategie, was die Erstellung realistischer Avatare auch bei begrenzter Posendiversität im Trainingsdatensatz ermöglicht.

2.3.7 Implementierungsaspekte und Optimierungen

Die praktische Implementierung animierbarer 3D Gaussian Avatare erfordert verschiedene technische Optimierungen, die in der aktuellen Forschung adressiert werden:

Effiziente Rendering-Techniken Ein Hauptvorteil der 3D Gaussian Splatting-Technologie ist ihre Rendering-Effizienz. Shao u. a. 2024 erreichen mit SplattingAvatar extreme Rendering-Geschwindigkeiten von über 300 FPS auf modernen GPUs und 30 FPS auf mobilen Geräten. Diese Effizienz wird durch die Einbettung von Gaussians in Dreiecksmeshs erreicht, was die Renderingkosten erheblich reduziert. Z. Qian u. a. 2024 berichten von einer 250-fachen Beschleunigung der Inferenz im Vergleich zu bisherigen Ansätzen, mit effizientem Rendering von über 50 FPS auf Consumer-Hardware und 30 FPS auf mobilen Geräten. Dies macht die Technologie praktisch einsetzbar auch auf ressourcenbeschränkten Geräten. Xiang u. a. 2024 demonstrieren mit FlashAvatar die Möglichkeit, hochdetaillierte Kopf-Avatare mit 300 FPS bei 512×512 Auflösung zu rendern, während Moreau u. a. 2024 mit HuGS etwa 80 FPS bei gleicher Auflösung für Ganzkörper-Avatare erreichen.

Optimierte Datenstrukturen Liu u. a. 2024 verwenden einen *Multi-Head-Hash-Encoder* für pose-abhängige Form und Erscheinung, was eine kompakte Speicherung bei gleichzeitiger detaillierter Darstellung ermöglicht. Dieser Ansatz reduziert den GPU-Speicherbedarf deutlich im Vergleich zu früheren Methoden. S. Qian u. a. 2024 stellen eine *Binding-Inheritance*-Strategie

für adaptive Dichtekontrolle vor, die es ermöglicht, die Anzahl der 3D Gaussians je nach Bedarf anzupassen und so die Recheneffizienz zu steigern.

Beschleunigtes Training Z. Qian u. a. 2024 erreichen eine 400-fache Beschleunigung des Trainings (30 Minuten auf einer einzelnen GPU) im Vergleich zu bisherigen Ansätzen. Dies wird durch eine effiziente Kombination von 3D Gaussian-Primitiven mit dem parametrischen SMPL-Körpermodell erreicht. Liu u. a. 2024 demonstrieren die Möglichkeit, hochwertige Avatar-Rekonstruktionen in nur 5-30 Sekunden Training zu erreichen, was eine dramatische Verbesserung gegenüber traditionellen Methoden darstellt, die oft Stunden oder Tage benötigen. Yuan u. a. 2024 verwenden einen primitiv-basierten impliziten Gaussian-Ansatz, der die Trainingszeit durch die Definition von 3D Gaussians innerhalb posengesteuerter Primitive reduziert.

2.3.8 Offene Herausforderungen

Trotz der beeindruckenden Fortschritte in der 3D Gaussian Avatar-Technologie bleiben einige Herausforderungen bestehen: Die realistische Darstellung komplexer Kleidungsstücke und lose Kleidung bei extremen Bewegungen. Die präzise Steuerung feiner Gesichtsausdrücke und Handbewegungen. Die Echtzeit-Rekonstruktion von Avataren aus unstrukturierten *In-the-Wild*-Videos. Die Integration physikbasierter Simulationen für realistischere Bewegungsdynamik. Die aktuellen Forschungsarbeiten, insbesondere die von Moon, Shiratori und Saito 2024, Saito u. a. 2024 und Yuan u. a. 2024, bieten vielversprechende Ansätze zur Bewältigung dieser Herausforderungen und zeigen das Potenzial für zukünftige Entwicklungen in diesem Bereich.

3 Vorbereitende Arbeiten

3.1 Volumetric Capture System

3.1.1 Benchmarking Raspberry Pi

Beteiligte Teammitglieder: Kai Altwicker

Basierend auf Tests mit synthetischen Datensätzen (siehe **??**) wurde festgestellt, dass das geplante Rig in seinen Dimensionen mindestens 40 Kameras benötigt, um Objekte überhaupt als Gaussian Splats reproduzieren zu können. Ab etwa 60 Kameras erzielte das System bereits eine zufriedenstellende Reproduktionsqualität, wobei sich ab circa 90 Kameras lediglich noch marginale Verbesserungen zeigten. Aus diesem Grund wurde der Bedarf auf 70 Kameras festgelegt.
Aus Kostengründen sollte die Kameraeinheit aus einem Raspberry Pi in Kombination mit dem zugehörigen Raspberry Pi Camera Modul realisiert werden. Ziel des Systems ist es, Videoaufnahmen in 1080p mit einer Mindestbildrate von 25 bis 30 Bildern pro Sekunde (fps) zu ermöglichen. Das vorgegebene Gesamtbudget beträgt etwa 7.500€.
In der nachfolgenden Analyse werden ausschließlich die Kosten der Kernkomponenten betrachtet: Die Raspberry Pi Module, die Kameramodule sowie gegebenenfalls zusätzliche Objektive. Kosten für ergänzende Komponenten wie Speicherkarten, Netzteile oder Verkabelung werden aus der Analyse ausgeklammert, müssen jedoch ebenfalls innerhalb des Budgetrahmens realisiert werden.

Für die Evaluierung kommen folgende Komponenten in Betracht:

- **Raspberry Pi Modelle:** Pi Zero 2 W, Pi 4 B, Pi 5.
- **Kamera-Module:** Camera Module V2, Camera Module V3, High Quality (HQ) Camera, Global Shutter Camera.

Im weiteren Verlauf werden zunächst die Preise inklusive üblicher Objektivkosten miteinander verglichen. Anschließend erfolgt eine Analyse der Leistungsfähigkeit und optischen Qualität der einzelnen Module, mit dem Ziel, unter Einbeziehung einer groben Kostenübersicht die optimale Raspberry Pi-Kamera-Kombination für das System zu ermitteln.

Preisvergleich
Die Gesamtkosten einer einzelnen Kamera-Pi-Kombination ergeben sich aus dem Preis des Raspberry Pi und dem des jeweiligen Kameramoduls. Für die HQ- und Global Shutter-Module fallen zusätzlich Kosten für Objektive an.
Wie Tabelle 1 verdeutlicht, bleiben Kombinationen unter Verwendung der Standard-Kameramodule (V2 und V3) deutlich innerhalb des vorgesehenen Budgets. Im Gegensatz dazu tendieren Systeme mit HQ- oder Global Shutter-Modulen dazu, die Budgetgrenze zu überschreiten, da hier zusätzliche Objektivkosten, beispielsweise ca. 25€ pro Standardobjektiv, anfallen. Somit kommen bei der gewünschten Anzahl an Kameras im Wesentlichen nur Kombinationen mit den Standard-Kameramodulen in Frage.

Konfiguration	**Preis pro Einheit (€)**	**Gesamtkosten (70 Einheiten, €)**
Pi Zero 2 W + Camera Module V2	30	ca. 2.100
Pi Zero 2 W + Camera Module V3	50	ca. 3.500
Pi 4 B + Camera Module V3	70	ca. 4.900
Pi 5 + Camera Module V3	85	ca. 5.950
Pi 4 B + HQ Camera (+ Objektiv)	120	ca. 8.400
Pi 4 B + Global Shutter Camera (+ Objektiv)	120	ca. 8.400

Tabelle 1: Preisvergleich der Kamera-Pi-Kombinationen (ungefähre Preise)

Leistungsfähigkeit

Die Leistungsfähigkeit des Systems wird primär an der Fähigkeit gemessen, 1080p-Videomaterial mit mindestens 25 bis 30 fps aufzunehmen.

- **Der Raspberry Pi Zero 2 W** basiert auf einem 1GHz Quad-Core Cortex-A53 mit 512MB RAM und ist mit einem Hardware-Encoder (VideoCore IV) ausgestattet. Obwohl der Encoder offiziell 1080p bei 30fps unterstützt, ergaben Praxistests in Kombination mit dem Camera Module V2 lediglich etwa 18fps. Diese Diskrepanz lässt sich vermutlich auf die reduzierte Prozessorleistung oder auf thermisches Throttling zurückführen. (Raspberry Pi Foundation 2025a)
- **Der Raspberry Pi 4 B** verfügt über einen 1,5GHz Quad-Core Cortex-A72 und eine VideoCore VI GPU mit einem dedizierten H.264-Hardware-Encoder. Diese Architektur ermöglicht stabile 1080p-Aufnahmen bei 30fps. Darüber hinaus bieten Gigabit Ethernet und USB 3.0 eine verbesserte I/O-Leistung, was zusätzliche Freiheiten in der Entwicklung des Synchronisationskonzepts eröffnet. (Raspberry Pi Foundation 2025b)
- **Der Raspberry Pi 5** ist mit einem 2,4-GHz-Quad-Core Cortex-A76 ausgestattet, verzichtet jedoch auf einen dedizierten Hardware-Encoder. Stattdessen erfolgt die Videokodierung softwareseitig, was einen höheren CPU-Ressourcenbedarf zur Folge hat. Trotz dieser Einschränkung ermöglicht die gesteigerte Rechenleistung des Pi 5 stabile 1080p-Aufnahmen bei 30fps. Darüber hinaus erlaubt die erhöhte Prozessorleistung eine umfangreiche Bildverarbeitung bereits auf dem Gerät, ein Aspekt, der für zukünftige Erweiterungen des Systems von Relevanz sein könnte. (Raspberry Pi Foundation 2025c)

Optische Güte

Die optische Güte der eingesetzten Kameramodule beeinflusst maßgeblich die Bildqualität und Detailgenauigkeit der Aufnahmen.

- **Das Camera Module V2** basiert auf dem Sony IMX219 Sensor mit 8MP und ist offiziell für 1080p bei 30fps ausgelegt. Das Modul arbeitet mit einem Rolling-Shutter, der bei schnellen Bewegungen zu Verzerrungen führen kann. Zudem setzt die hohe Pixeldichte in Kombination mit dem verbauten Objektiv dem optischen Auflösungsvermögen klare Grenzen. Es eignet sich daher vor allem für kostengünstige Anwendungen, bei denen die maximale Bildrate und volle Auflösung nicht zwingend erforderlich ist. (Raspberry Pi Foundation 2025d)

- **Das Camera Module V3** basiert auf dem Sony IMX708 Sensor mit 12MP, verfügt über Autofokus sowie HDR-Funktionalitäten und ermöglicht Aufnahmen in 1080p bei bis zu 50fps, zudem sind auch UHD-Aufnahmen möglich. Das im Modul verwendete Objektiv liefert im Vergleich zum V2-Modul eine signifikant verbesserte Abbildungsleistung. Diese Eigenschaften resultieren in einer deutlich gesteigerten Bildqualität und höheren Bildraten, was das V3-Modul besonders attraktiv für den Einsatz in einem Multi-Kamera-Rig macht. (Raspberry Pi Foundation 2025d)

- **Die High Quality (HQ) Camera** basiert auf dem Sony IMX477 Sensor (12,3MP) und bietet höchste Bildqualität sowie Flexibilität durch den Einsatz austauschbarer C/CS-Mount-Objektive. Dank des größeren Bildsensors und der freien Wahl der Objektive verspricht dieses Modul eine hervorragende Detailtreue sowie eine verbesserte Low-Light-Performance. Allerdings gehen diese Vorteile mit höheren Kosten und einem erhöhten Montageaufwand einher, was bei einem großflächigen Einsatz problematisch sein kann. Zudem hängt die tatsächliche Bildqualität maßgeblich von der Qualität der verwendeten Objektive ab, sodass bei kostengünstigeren Optionen die Ergebnisse in etwa auf dem Niveau des V3-Moduls liegen können. (Raspberry Pi Foundation 2025d)

- **Die Global Shutter Camera** basiert auf dem Sony IMX296 Sensor mit 1,6MP und einem Global Shutter, wodurch typische Rolling-Shutter-Artefakte vermieden werden, ein wesentlicher Vorteil bei der Aufnahme schneller Bewegungen im Kamerarig. Ein weiterer bedeutender Aspekt dieses Moduls ist die Möglichkeit, es über einen externen Triggerimpuls auszulösen, was den Synchronisationsprozess erheblich vereinfacht. Als Nachteile ergeben sich jedoch die geringere Auflösung im Vergleich zu anderen Modulen sowie, analog zur HQ-Kamera, die zusätzlichen Kosten für kompatible Objektive. (Raspberry Pi Foundation 2025d)

Evaluation

Die Evaluation zeigt, dass ein Kompromiss zwischen Kosten, Leistungsfähigkeit und optischer Güte gefunden werden muss. Die wesentlichen Ergebnisse sind:

- **Preis:** Systeme mit Camera Module V2 oder V3 in Kombination mit dem Pi Zero 2 W sind sehr kostengünstig, erreichen jedoch mit dem Pi Zero 2 nur ca. 18 fps. Der Raspberry Pi 4 B in Verbindung mit dem Camera Module V3 bietet ein gutes Preis-Leistungs-Verhältnis und bleibt deutlich unter dem Budget von 7.500 €. Systeme mit HQ- oder Global Shutter-Kameras überschreiten aufgrund der zusätzlichen Objektivkosten tendenziell das Budget.

- **Leistungsfähigkeit:** Der Pi Zero 2 W liefert nicht die erforderliche Bildrate. Der Pi 4 B sowie der Pi 5 ermöglichen stabile 1080p-Aufnahmen bei 30 fps. Zwar bietet der Pi 5 höhere Rechenleistung, jedoch erfolgt hier die Videokodierung softwareseitig, was in diesem Anwendungsfall den Vorteil relativiert.

- **Optische Güte:** Das Camera Module V3 überzeugt durch höhere Auflösung, Autofokus und HDR-Funktionalitäten. HQ und Global Shutter bieten spezielle Vorteile (höhere De-

tailtreue bzw. verzerrungsfreie Aufnahme), sind jedoch im großflächigen Einsatz aufgrund höherer Kosten und größerer Montageaufwände weniger praktikabel.

Basierend auf dieser Untersuchung wurde entschieden, den **Raspberry Pi 4B 1GB** in Kombination mit dem **Camera Module V3** anzuschaffen. Der Raspberry Pi 4B bietet ausreichend Leistungsreserven, um 1080p-Videomaterial zuverlässig aufzuzeichnen, wobei der geringere Arbeitsspeicher für Videoencoding keine wesentlichen Einschränkungen mit sich bringt. Die Wahl der 1GB-Version ermöglicht somit zusätzliche Budgeteinsparungen.
Das Kameramodul V3 überzeugt durch eine optisch ausreichende Qualität und stellt als Komplettpaket eine kosteneffizientere Alternative zur Anschaffung einer HQ- oder Global Shutter-Kamera dar. Bei den geschätzten Gesamtkosten von ca. 5.000€ für 70 Pis und Kameras verbleibt zudem ein finanzieller Spielraum, der eine flexible Planung für weitere erforderliche Komponenten ermöglicht.

3.2 Training

Beteiligte Teammitglieder: Matthias Bullert,David Mertens

3.2.1 Aufsetzen des Repository

Beteiligte Teammitglieder: David Mertens

Die Einrichtung des ursprünglichen Repositorys stellte eine größere Herausforderung dar als zunächst angenommen, da die im zugehörigen README bereitgestellte Anleitung fehlerhaft und unvollständig ist. Die Funktionsfähigkeit des Repositorys hängt von einer spezifischen Kombination bestimmter Versionen einzelner Python-Module ab. Da kein Installations- oder Setup-Programm vorhanden ist, erwies sich der Einrichtungsprozess als kleinschrittig und zeitintensiv. Um die Verwendung der im Rahmen dieses Projekts entwickelten Erweiterungen zu erleichtern, wurde im GitHub-Ordner des Projekts unter training/SpacetimeGaussians/readme.md eine Installationsanleitung hinterlegt. Diese stellt sicher, dass das Projekt sowohl auf Windows- als auch auf Linux-Systemen korrekt eingerichtet werden kann.

3.2.2 Synthetischer Datensatz und Kamerasimulation

Beteiligte Teammitglieder: David Mertens, Matthias Bullert

Für die Erstellung eines Gaussian Splat wird ein Datensatz aus Videos benötigt, die aus unterschiedlichen Perspektiven aufgenommen wurden. Da bereits existierende Datensätze stark von der jeweiligen Kamerakonstellation abhängig sind, wurde ein synthetischer Datensatz entwickelt, um gezielt verschiedene Parameter wie Kameraanzahl und Auflösung zu evaluieren.
Hierzu wurde in Blender eine Szene aufgebaut, die ein reales Kamera-Rig möglichst genau nachbildet. Ein 3D-Modell des Rigs wurde erstellt und die intrinsischen Kameraparameter – wie

Brennweite und Auflösung – der tatsächlich verwendeten Kameras übernommen. In die Mitte des Rigs wurde eine virtuelle, sich schnell bewegende Person platziert. Der Hintergrund besteht aus dem 3D-Modell eines Raumes.
Ein zentrales Element des synthetischen Setups war ein eigens entwickeltes Skript, das für ein gegebenes Mesh an jedem Vertex eine Kamera positioniert. Diese Kameras blicken ins Innere des Meshes und sind auf die virtuelle Person ausgerichtet. Das Skript erlaubt das automatische Rendern der Szene und legt die Bilder kamerabezogen in einer strukturierten Ordnerstruktur ab. Dadurch konnten verschiedene Kamerasetups systematisch getestet werden.
Parallel zur Entwicklung des physischen Rigs war es notwendig, bereits mit realistischen Trainingsdaten zu arbeiten. Der synthetische Datensatz diente hierbei als vorbereitendes Werkzeug zur Evaluation und Optimierung des Kameradesigns.
Ziel der Kamerasimulation war es, die optimale Kameraanzahl und -auflösung zu bestimmen. Kriterien waren dabei die Schärfe und Detailgenauigkeit der rekonstruierten Szene, das Auftreten von Artefakten sowie die Trainingsdauer. Es zeigte sich, dass eine Kameraanzahl zwischen 60 und 70 ein gutes Gleichgewicht zwischen Qualität und Ressourcenverbrauch bietet. Geringere Kameraanzahlen führten zu sichtbaren Artefakten, insbesondere in verdeckten Bereichen wie unter den Armen oder zwischen den Beinen. Auch die Qualität der COLMAP-Rekonstruktion verschlechterte sich deutlich bei zu wenigen Kameras.
Eine weitere Erhöhung über etwa 90 Kameras hinaus brachte keine signifikanten visuellen Verbesserungen, führte jedoch zu gesteigertem VRAM-Verbrauch und erhöhter Trainingszeit. Auch die Auflösung der Eingangsvideos hatte einen klaren Einfluss auf das Ergebnis: Eine Halbierung der Auflösung reduzierte Details und führte zu weicheren Rekonstruktionen.
Die Simulation bestätigte zudem, dass eine orthogonal zur Trägerstange ausgerichtete Kamera – wie in Abschnitt 4.1.1 beschrieben – keine negativen Auswirkungen auf das Ergebnis hat.

Abbildung 1: Blender Rekonstruktion des Rigs

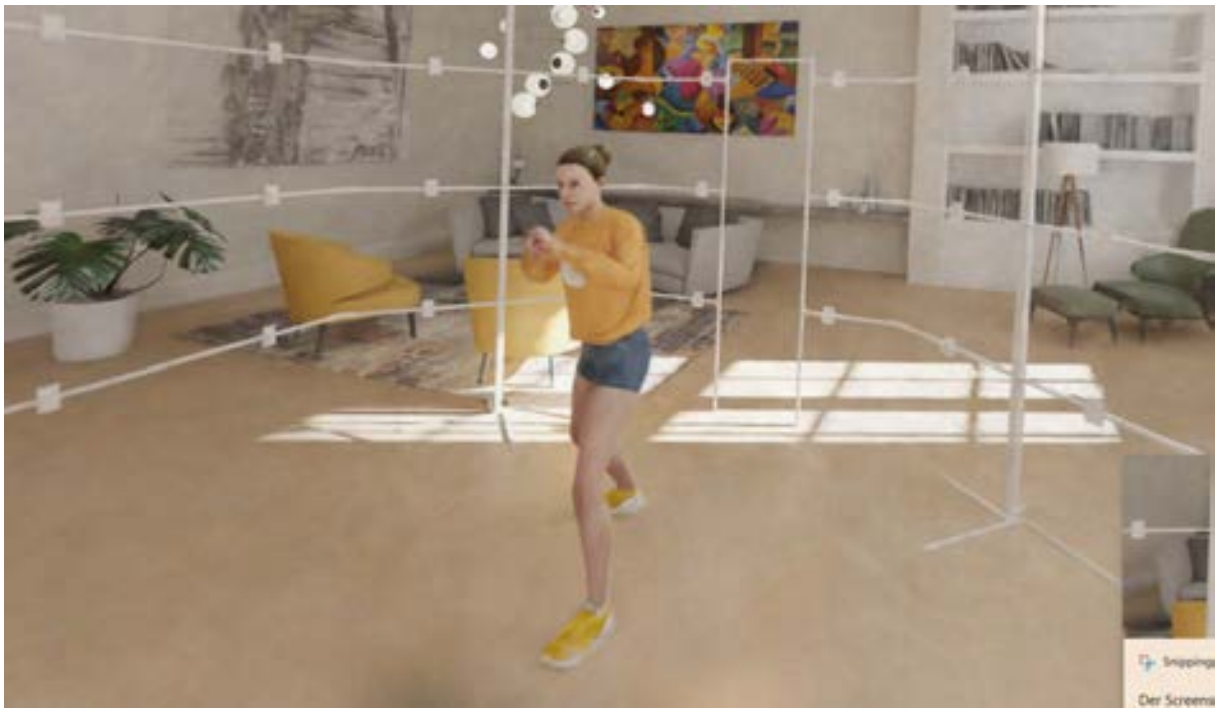


Abbildung 2: Screenshot der Szene, die mit Spacetime Gaussian auf Grundlage des synthetischen Datensatzes rekonstruiert wurde

(a) Halbe Anzahl an Kamerawinkeln

(b) Halbe Auflösung der Input-Videos

Abbildung 3: Verschiedene Ergebnisse bei Variationen des Datensatzes

3.3 Viewer

Beteiligte Teammitglieder: David Martin Karg, Alisa Rüge, Steffen-Sascha Stein

3.3.1 Zielsetzung Viewer

Für den Viewer wurden folgende Ziele vorgesetzt: In einer XR-Umgebung soll eine einzelne, sich bewegende Person mittels Space-Time-Gaussians in Echtzeit dargestellt werden. Im Idealfall soll die XR-Umgebung durch Passthrough der echten Umgebung entsprechen, und die mittels des Kamera-Rigs gescannte Person soll sich möglichst gut in die Umgebung einfügen. Die virtuelle Person soll durch die Steuerung mit den Controllern platzierbar, skalierbar und rotierbar sein. Die Viewer-Anwendung soll auf aktuellen VR-Brillen funktionieren, in diesem Fall der Metaquest II und Metaquest III.

3.3.2 Kickstart Prototyp

Um das Projekt zu finanzieren, wurde an dem Förderprogramm Kickstart@TH Köln teilgenommen. Im Rahmen dieses Förderprogramms sollte ein Prototyp unseres Viewers vorgestellt werden, damit Nutzer ein Gefühl davon bekommen sollten, was mit Space-Time-Gaussians und mit dem Scannen von Personen im Kamera-Rig möglich ist. Da die Zeit vom Start des Projekts bis zum ImpacTH-Tag begrenzt war, mussten die Anforderungen des Viewers auf die wesentlichen Aspekte reduziert werden. Dementsprechend wurde die Arbeit auf den Kickstart-Prototypen in kleinere Arbeitspakete unterteilt. Hierzu wurde vorerst eine Liste an Anforderungen erstellt, welche Funktionalitäten der Kickstart-Prototyp können muss, soll und was vielleicht möglich ist, falls das Wesentliche schon vor dem Ende des Projektes implementiert und erreicht wird.

Anforderungen: Die Mindestanforderung an den Kickstart-Prototypen ist das Abspielen der volumetrischen Videos im Loop. Im Idealfall bestehen die Videos aus einer einzelnen Person, gegebenenfalls mit Hintergrund. Die Anwendung sollte zumindest vorerst auf der Metaquest III in Echtzeit und VR funktionieren, anfangs ohne Passthrough. Sind diese Anforderungen erfüllt, können weitere in Betracht gezogen werden. So sollte die Performance nicht dadurch beeinträchtigt werden, sollte auch Passthrough in der Anwendung aktiviert werden, sodass die Person sich in die Umgebung eingliedert. Außerdem sollte die virtuelle Person ohne Hintergrund in der Gaussian-Splatting-Szene gerendert werden. Zudem kann die Steuerung angegangen werden, indem die virtuelle Person durch die Controller initial platzierbar ist.

Arbeitspakete: Das wichtigste und erste Arbeitspaket besteht aus dem Aufsetzen der Viewer-Umgebung. Dabei wurde die offene Code-Basis von Kevin Kwok verwendet, auf der aufgebaut werden sollte (Kwok 26.03.2024). Da es sich um eine Webanwendung handelt, wurde ein lokaler Server aufgesetzt, auf dem die Anwendung ausgeführt werden soll.
Die Code-Basis ist sehr unübersichtlich, da sich alle Funktionen und Klassen in einer Javascript-Datei befinden. Dadurch ergab sich ein weiteres Arbeitspaket, welches die Strukturierung des

Codes beinhaltete. Die einzelnen Klassen und Funktionalitäten müssen in neue Dateien geschrieben werden, sodass sich klare Unterteilungen der Aufgabenbereiche ergeben.
Ein weiteres Arbeitspaket ist die Recherche und das Einarbeiten in WebGPU und WebGL, da diese für das Verständnis des Renderers und der Shader wichtig ist. Außerdem wurde in Erwägung gezogen, für das anstehende Optimieren des Bottleneck-Prototypen, manche parallelisierbare Operationen an die Grafikkarte zu übergeben.

Ergebnis Kickstart Prototyp: Da es zu dem Zeitpunkt des Kickstart-Prototypen noch keine trainierten Modelle und Kamera-Rigs gab, musste auf bereits existierende Modelle aus dem Paper (Zhan Li u. a. 2024) zurückgegriffen werden.
Das gewählte Modell zeigt einen Mann mit Flammenwerfer in seiner Garage. Ein Problem dabei ist, dass der Hintergrund – die Garage – sehr dominant wirkt. Zudem wird die Person je nach Blickwinkel lückenhaft dargestellt. Ein weiteres Problem betrifft die Performance, da das volumetrische Video mit Passthrough nicht flüssig läuft.
Diese drei Probleme wurden wie folgt gelöst: Der Hintergrund des Modells wurde weitestgehend entfernt, indem innerhalb des Shaders eine Bounding-Box definiert wurde, in der nur die Punkte der Pointcloud gerendert werden, welche sich innerhalb dieser zum Zeitpunkt des Renderings auch befanden. Hierbei ist zu beachten, dass diese Bounding-Box sich nicht mit der Kamera des Betrachters mitbewegen darf, sondern immer an der Position der virtuellen Person platziert ist.
Es wurde in Erwägung gezogen, den Hintergrund auf Dateiebene zu entfernen. Hierzu können alle Punkte der Pointcloud entfernt werden, die dem Hintergrund angehören. Dieser Ansatz ist jedoch verlustbehaftet und wurde aus diesem Grund verworfen.
Für die Ausstellung des Prototypen am ImpacTH, wurde das Modell des Mannes mit dem Flammenwerfer verwendet. Da der Platz hierfür und dadurch die Bewegungsfreiheit des Nutzers begrenzt war, konnten die Artefakte umgangen werden. Das Modell wurde im Vorhinein manuell so platziert und ausgerichtet, dass es mit ein wenig Abstand vor dem Nutzer steht. Dadurch sind die Artefakte kaum bemerkbar.

3.3.3 Bottleneck Prototyp

Nach den Kickstart-Prototypen, der zum Vorstellen unsere Projektes gedacht war, war die Idee, an diesem Prototypen weiterzuarbeiten und die Funktionen zu implementieren, die wir uns zu Anfang des Projektes vorgesetzt hatten. Außerdem sollte es in diesem Arbeitspaket auch darum gehen, mögliche Bottlenecks ausfindig zu machen und nach Möglichkeit zu beheben.

Anforderungen: Die Anforderungen, die für den Bottleneck-Prototypen gestellt wurden, waren folgende:
Das volumetrische Video soll eine einzelne Person, sich bewegende Person darstellen, und der Koordinaten-Ursprung soll unter den Füßen der Person liegen.
Die Person soll mit den Controllern verschiebbar und rotierbar sein.
Es gab auch die Idee, mehrere Sequenzen eines volumetrischen Videos, also mehrere .ply-Dateien beim Konvertieren zu .splatv-Dateien, in eine einzige .splatv-Datei zusammenzupacken, sodass man nicht das Problem hat, jede Sequenz hintereinander zu laden.

Außerdem, um zu analysieren, wo eventuell Bottlenecks liegen, sollte die Performance messbar sein.

Arbeitspakete: Eine Reihenfolge zum Abarbeiten der Arbeitspakete wurde nicht überlegt, da diese unabhängig voneinander bearbeitet werden konnten.
Zum einen sollten die WebXR- und WebGPU-Bindings getestet werden, um zu prüfen, wie realistisch die Integration von WebGPU in den Viewer ist.
Was auch implementiert werden sollte, war die Steuerung mit dem Controllern. Vornehmlich das Rotieren und Verschieben der Modelle im virtuellen Raum.
Da schon das Problem des begrenzten RAMs beim Training bekannt war, sollte getestet werden, wie schnell mehrere Modelle hintereinander geladen werden können, und wie flüssig der Übergang ist.
Ein weiteres Arbeitspaket war die Bottleneck-Analyse. In diesem Arbeitspaket ging es darum die Performance des Viewers zu messen, und zu schauen, welche Teile des Viewers verantwortlich für Performance-Probleme sind.

Ergebnis Bottleneck Prototyp: Im Verlauf des Bottleneck-Prototypen wurde das Rotieren und Verschieben der Modelle mittels Controllern implementiert. Die Geschwindigkeit des Rotierens und Verschiebens hängt davon ab, wie stark man mit den Joysticks in die entsprechende Richtung drückt. Allerdings gab es zu dem Zeitpunkt noch das typische Problem, dass, wenn das Modell verschoben wurde und danach rotiert werden sollte, das Modell sich um die Ursprungsachse drehte.
Die Idee mehrere .ply-Dateien in eine .Splatv-Datei zu packen wurde verworfen. Zum einen ist das Format .splatv nicht dafür ausgelegt mehrere Splats zu halten, die zu unterschiedlichen Zeitpunkten angezeigt und abgespielt werden sollen. Stattdessen wurde entschieden, dass man versucht mehrere .splatv-Dateien möglichst flüssig hintereinander abzuspielen. Die Implementierung mehrere .splatv-Dateien hintereinander abzuspielen hat gezeigt, dass der Übergang möglichst schnell passieren muss, und dass die Splats vorher sortiert werden müssen, bevor sie an den Renderer übergeben werden. Außerdem wurde klar, dass man eine saubere Codearchitektur braucht, wenn man problemlos mehrere Videos, die wiederum aus mehreren Sequenzen von .splatv-Dateien bestehen, verwalten möchte. Daraus entstand die Idee der Volumetric-Video-Architektur, auf die später näher eingegangen wird.

3.4 HAC

Beteiligte Teammitglieder: Marvin Winkler

Die Erstellung fotorealistischer, animierbarer menschlicher Charaktere mit Echtzeit-Renderleistung stellt nach wie vor eine komplexe Herausforderung im Bereich der Computergrafik dar. Mit der stetigen Weiterentwicklung eröffnen sich jedoch neue Möglichkeiten, die visuelle Qualität und Rendereffizienz signifikant zu verbessern. Dieses Teilprojekt untersucht den Einsatz dieser fortschrittlichen Rendering-Verfahren zur Generierung und Animation menschlicher Avatare, mit dem Ziel, ein optimales Gleichgewicht zwischen Darstellungsqualität und Performanz zu erreichen.

3.4.1 Ursprünglicher Ansatz: HAC-TRIPS

Der ursprüngliche Plan dieses Teilprojekts war die Entwicklung von "Humanoid Animated Characters based on TRIPS" (HAC-TRIPS). *TRIPS* (*Trilinear Point Splatting for Real-Time Radiance Field Rendering*) Franke u. a. 2024 stellt einen innovativen Ansatz für point-basiertes Radiance Field Rendering dar. Das Verfahren vereint die Vorteile von 3D Gaussian Splatting Kerbl u. a. 2023 und *ADOP* (*Approximate Differentiable One-Pixel Point Rendering*) Rückert, Franke und Stamminger 2022, indem es Punkte mittels Trilinear-Splatting in eine Bildpyramide im Bildraum rastert.
Die Besonderheit von TRIPS liegt in der Verwendung einer Bildpyramide, bei der die Auswahl der entsprechenden Pyramidenebene durch die projizierte Punktgröße bestimmt wird. Dieser Ansatz ermöglicht das Rendern beliebig großer Punkte mittels einer einzigen trilinearen Operation. Ein leichtgewichtiges neuronales Netzwerk rekonstruiert anschließend ein lückenloses Bild mit Details, die über die Auflösung der Splats hinausgehen. Der bedeutendste Vorteil dieses Verfahrens ist die Kombination aus hochqualitativen Renderings bei gleichzeitiger Echtzeitfähigkeit von etwa 60 Bildern pro Sekunde auf handelsüblicher Hardware.
Trotz intensiver Bemühungen war es jedoch nicht möglich, das TRIPS-Repository zum Laufen zu bringen. Dies lag nicht an fehlenden Systemvoraussetzungen, die vollständig erfüllt waren:

- Unterstützte Betriebssysteme: Ubuntu 22.04, Windows
- NVIDIA GPU (getestet wurde auf verschiedenen Systemen mit mindestens einer RTX 2070)
- Unterstützte Compiler: `g++-9` (Linux), `MSVC` (Windows, Version 19.31.31105.0)
- Software-Anforderung: `Conda` (Anaconda/Miniconda)

Die Versuche wurden sowohl unter Windows als auch unter Linux durchgeführt, führten jedoch nicht zum gewünschten Erfolg.

3.5 Neuorientierung und Technologieauswahl

Nach der Erkenntnis, dass TRIPS für dieses Projekt nicht verwendet werden konnte, erfolgte eine gründliche Untersuchung alternativer Technologien. Der Fokus lag dabei auf Avatarsystemen,

die auf Gaussian Splatting basieren. Verschiedene Implementierungen und Forschungsarbeiten wurden analysiert und gegeneinander abgewogen.
Die Evaluierung umfasste mehrere aktuelle Ansätze, darunter:

- *Deformable 3D Gaussian Splatting for Animatable Human Avatars* Jung u. a. 2023
- *GaussianAvatar* Hu u. a. 2024
- *Expressive Whole-Body 3D Gaussian Avatar* Moon, Shiratori und Saito 2024
- *Drivable 3D Gaussian Avatars* Zielonka u. a. 2025
- *3DGS-Avatar* Z. Qian u. a. 2024
- *Animatable Gaussians* Zhe Li u. a. 2024

Nach sorgfältiger Abwägung fiel die Entscheidung auf "Animatable Gaussians" Zhe Li u. a. 2024. Ausschlaggebend für diese Wahl waren mehrere Faktoren:

1. Erfüllung der projektspezifischen Anforderungen
2. Kompatibilität mit der verfügbaren Hardware
3. Verfügbarkeit des Quellcodes
4. Implementierung in `Python`, was eine einfache Bearbeitung ohne aufwendige Build-Prozesse ermöglicht

3.5.1 Animatable Gaussians

Animatable Gaussians stellt eine neuartige Avatar-Repräsentation dar, die 3D Gaussian Splatting mit leistungsstarken 2D CNNs (Convolutional Neural Networks) kombiniert. Der Ansatz lernt zunächst ein parametrisches Template aus Multi-View-Videos und parametrisiert dieses in Front- und Back-Canonical Gaussian Maps. Für die Modellierung der pose-abhängigen Dynamik kommt ein auf *StyleGAN* basierendes CNN (*StyleUNet*) zum Einsatz.
Die Stärke dieses Ansatzes liegt in der effizienten Kombination aus:

1. Einem charakterspezifischen Template, das auch für komplexe Kleidung wie lange Kleider geeignet ist
2. Der template-gestützten Parametrisierung, die 3D Gaussians auf 2D-Karten abbildet und so die Verwendung leistungsfähiger 2D-Netzwerke ermöglicht
3. Einer *Pose-Projektionsstrategie*, die bessere Generalisierung auf neue Posen erlaubt

Nach erfolgreicher Installation und ersten Tests zeigte sich jedoch, dass die Performance, wie im Paper beschrieben, problematisch war. Während die visuelle Qualität der erstellten Avatare beeindruckend war, erwies sich die Trainingszeit selbst auf leistungsstarker Hardware als unverhältnismäßig lang. Auf der für das Projekt verfügbaren RTX 3060 stellte dies eine erhebliche Einschränkung dar.

Diese Performanceprobleme bildeten den Ausgangspunkt für den Hauptteil der durchgeführten Arbeit: die Implementierung von Performanceoptimierungen unter bestmöglicher Beibehaltung der visuellen Qualität. Die detaillierte Beschreibung dieser Optimierungen folgt im nachfolgenden Abschnitt.

4 Umsetzung

4.1 Volumetric Capture System

Das *Volumetric Capture System* umfasst das entwickelte Konzept zur Aufnahme und Speicherung synchronisierter Multikameraaufnahmen – sowohl in Form von Fotos als auch als Videoaufnahmen. Das System gliedert sich in zwei wesentliche Komponenten: Zum einen das *Capture-Rig*, welches die Hardwareelemente wie beispielsweise den Gerüstbau und die eingesetzten Kameras umfasst, und zum anderen die *Capture-Software*, die zur Steuerung des Capture-Rigs benötigt wird. Im Folgenden werden beide Komponenten getrennt voneinander detailliert dokumentiert.

Capture Rig

4.1.1 Konstruktion und Planung

Im Kontext Computer Aided Design (CAD) bezeichnet der Begriff *Konstruktion* die systematische Entwicklung und Gestaltung technischer Produkte oder Bauteile unter Berücksichtigung funktionaler, fertigungstechnischer und ästhetischer Anforderungen (Ponn und Lindemann 2011). Dieser Prozess umfasst die Erstellung von 2D-Zeichnungen und 3D-Modellen mittels CAD-Software, die Simulation und Analyse technischer Eigenschaften sowie die Vorbereitung der Bauteile für die Fertigung. CAD ermöglicht dabei eine präzise und effiziente Umsetzung des gesamten Konstruktionsprozesses, wie beispielsweise im vorliegenden Projekt, in dem Autodesk Inventor Professional für die Planung des Kamerarigs, der Kamerahalter sowie für Teile der Stromversorgung eingesetzt wurde.

Kamerarig

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Das Kamerarig umfasst alle zugehörigen Bauteile des Trägergerüsts, an welchem die Kameras, die Stromversorgung und weitere Komponenten montiert werden. Die Entwicklung dieses Systems erfolgte in einem fließenden Prozess, der sich in drei Iterationen gliedern lässt:

Das initiale Mockup diente der Bewerbung zur Kickstart-Förderung und hatte primär den Zweck, eine grobe Kostenabschätzung und eine Visualisierung der Konzeptidee zu ermöglichen. Bereits in dieser Phase wurden erste Überlegungen zu allgemeinen Dimensionen angestellt, die auf Erfahrungen mit zuvor realisierten Versuchseinrichtungen basierten. Es wurde beschlossen, das Rig aus Aluminium-Rundrohr zu fertigen, welches in einem definierten Radius gebogen wird, um eine gleichmäßige visuelle Abdeckung zu erzielen. Aluminium-Rundrohr ist dabei kostengünstig beschaffbar und die entsprechende Biegemaschine ist in der Zen-

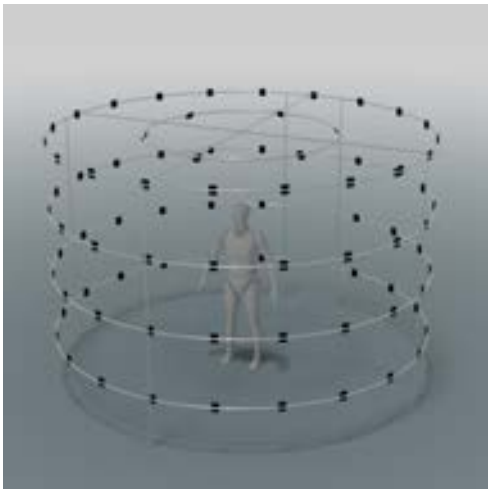
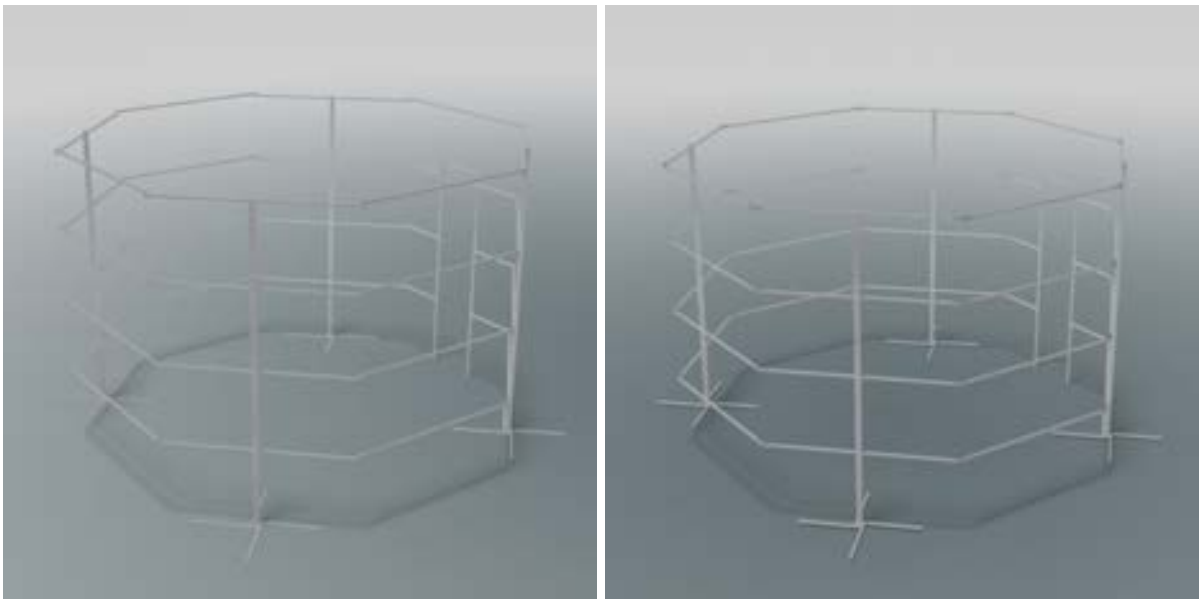


Abbildung 4: Render Mockup

tralwerkstatt Maschinenbau der TH Köln vorhanden. Der gewählte Radius von 4m sorgt dafür, dass sich auch große Personen komfortabel im Rig bewegen können, so bleibt bei einer Körpergröße von 2m und ausgestreckten Armen immer noch ein Abstand von ca. 1m zu den Kameras bestehen. Zur Realisierung aufsichtiger Perspektiven wurde die Höhe des Rigs auf 2,6m festgelegt. Die Kameras werden an vier Ringen in gleichmäßigem Abstand montiert, um eine kontinuierliche vertikale Abdeckung zu garantieren, während an der "Deckeëin zusätzlicher, kleinerer Ring eingeplant wurde, der eine vertikale Erfassung von oben ermöglicht. Zum Zeitpunkt der Planung wurde ein Bedarf von 86 Kameras ermittelt.

Kamerarig V1 beinhaltet den Großteil der Planungsarbeiten und stellte die erste Version dar, in der das komplette Rig mithilfe von CAD erbaut wurde. Aufgrund der Vorgaben der Kickstart-Förderung, die enge Grenzen an Zulieferer und Dienstleistungen setzten, war die Nutzung der Zentralwerkstätten der TH Köln zur Fertigung nicht mehr möglich. Somit erwies sich die Realisierung der im Mockup verwendeten gebogenen Rundrohre als nicht mehr kostenoptimal, weshalb entschieden wurde, das gesamte Rig aus sogenannten *Item-Profilen* der Firma SMT zu konstruieren. Diese Profile zeichnen sich durch ihre modulare Bauweise, hohe Stabilität und einfache Verbindungstechnik aus, da sie auf Nutenprofilen basieren, die eine flexible Konstruktion ohne Schweißen ermöglichen. Die Profile finden als „Quasistandard“ breite Anwendung in der Industrie, da eine große Anzahl an Verbindermöglichkeiten zur Auswahl steht. Für dieses Projekt wurden Profile der Baureihe 5 gewählt, deren Basisraster 20x20mm beträgt. Dieses Raster, das sowohl die Nutbreite als auch den Abstand der Löcher definiert, gewährleistet, dass alle Profile miteinander kombinierbar sind und Standard-Zubehörteile wie Verbinder, Riegel und Füße problemlos eingesetzt werden können.
Die Höhe des Rigs wurde durch die maximal per Postweg lieferbare Profillänge von 2,5m festgelegt. Um eine möglichst gleichmäßige Abdeckung der Kameras zu gewährleisten und der Kreisform des ursprünglichen Mockups nahe zu kommen, wurde eine Oktagonform gewählt. Bei ei-



V1 V2

Abbildung 5: Render: Rig V1 vs. Rig V2

nem Durchmesser von 4m und Eck-auf-Eck angelegten Profilen ergab sich eine Kantenlänge von 1,53m, die auf einen runden Wert von exakt 1,5m reduziert wurde, um mit einfacheren Zahlen zu rechnen. Daraus resultiert ein Außenradius von 3,89m sowie ein Innendurchmesser (Seitenabstand) von 3,55m. Vertikal wurden die Profile analog zum Mockup in vier Ringen mit einem gleichmäßigen Abstand von 0,625m angeordnet, wobei auf einen zweiten, kleineren Kopfkreis verzichtet wurde, da bei Bedarf Top-Kameras direkt an der Raumdecke montiert werden können. Zusätzlich wurde an einer Seite ein Durchgang mit einer Höhe von 2m eingeplant. Die Durchgangsbreite von 0,5m stellt einen Kompromiss zwischen Passierbarkeit und der Einschränkung der Kamerapositionierung dar. Obwohl der Durchgang dadurch nicht mehr barrierefrei ist, besteht die Möglichkeit, diesen Bereich mit entsprechendem Aufwand zu erweitern, sodass auch mobilitätseingeschränkte Personen oder größere Objekte das Rig betreten können.
Zur Vereinfachung des späteren Aufbaus wurden für die Vertikalprofile einfache Kreuzfüße konstruiert, die die einzelnen Segmente während des Aufbaus aufrecht halten, solange das Rig noch nicht verschraubt ist. Nach vollständigem Aufbau stabilisiert sich das Rig durch seine Form selbst, sodass die Füße zur Vermeidung von Stolpergefahren demontiert werden können. Die Eckverbindung der horizontalen Profile erfolgt durch eine Durchgangsbohrung und anschließende Verschraubung beider Profile, während alle anderen Verbindungen mittels Winkelverbindern realisiert werden, die mit entsprechenden Nutensteinen verschraubt sind. Aus Kostengründen wurden nur die vertikalen Profile als 40x40-Profile ausgelegt, während die horizontalen Profile aus Gewichtsgründen als 20x20mm geplant wurden. Da hier nur geringe, symmetrische Lasten auftreten kann eine Durchbiegung der Profile vernachlässigt werden.
Alle Maße wurden im Vorhinein in einem sogenannten *Calc-Sheet* erfasst und anschließend als adaptive Parameter in Autodesk Inventor hinterlegt. Dieser Ansatz ermöglicht es, nachträgliche Änderungen der Maße vorzunehmen, wobei sich die gesamte Konstruktion dynamisch an diese Anpassungen anpasst.

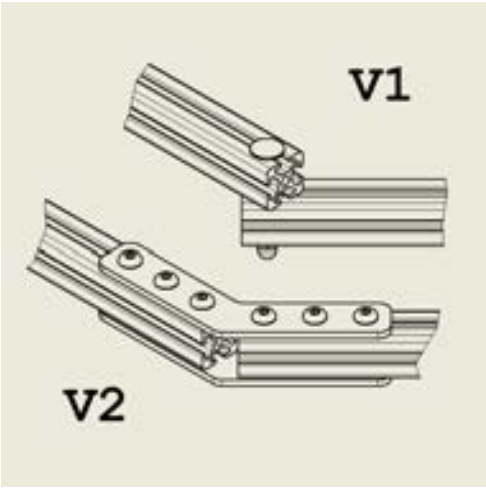


Abbildung 6: Eckverbinder V1 vs V2

Kamerarig V2 unterscheidet sich von V1 ausschließlich in der Gestaltung der Eckverbinder zwischen den horizontalen Profilen. In Rücksprache mit den Verantwortlichen des Makerspace wurden Bedenken hinsichtlich der Stabilität geäußert, sodass die ursprünglichen Verbinder überarbeitet wurden. Die neuen Verbinder bestehen aus zwei separaten Stahlplatten mit einer Dicke von 3mm, die jeweils so lasergeschnitten wurden, dass sie einen 120°-Winkel aufweisen. Sie werden mithilfe von Nutensteinen an den Profilen befestigt und von beiden Seiten montiert, sodass jegliche Biegekräfte zuverlässig abgefangen werden. Gleichzeitig stellt die Form der Verbinder eine korrekte Winkellage zwischen den Profilen sicher. Mit dieser Lösung wurde das Rig als ausreichend stabil eingestuft, um in den Bestellprozess und die Fertigung zu gehen.

Die Behebung weiterer potentieller Instabilitäten wurde bewusst auf einen späteren Zeitpunkt verschoben, da diese schwer vorhersehbar und simulierbar sind und das Konzept zunächst in der

Realität getestet werden muss. Dennoch wurden bereits Ansätze erarbeitet, wie beispielsweise die Umsetzung einer Verstrebung durch zusätzliche Profile oder ein Versteifen mittels des Einziehens von Stahlseilen. Da das gesamte Rig inklusive aller Verbinder, Schrauben und weiteren Bauteilen im CAD-Programm vollständig modelliert wurde, konnte abschließend eine Stückliste für die Anschaffung aller Komponenten erstellt werden. Dabei wurde insbesondere bei den Nutensteinen und Schrauben ein erhöhter Bedarf eingeplant, um auch spätere Anpassungen und zusätzliche Anbauteile problemlos realisieren zu können.

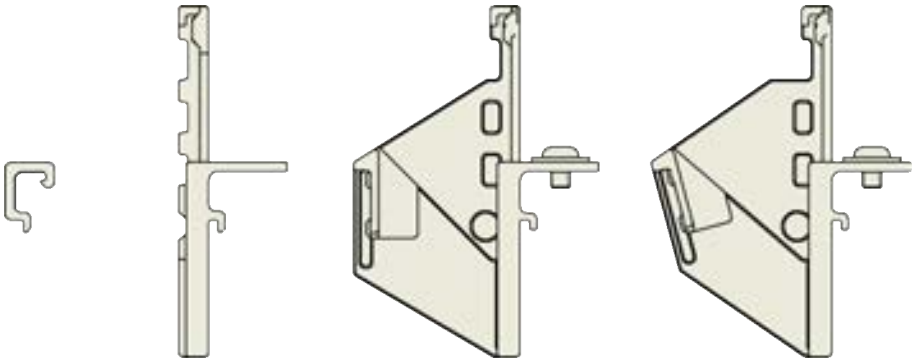


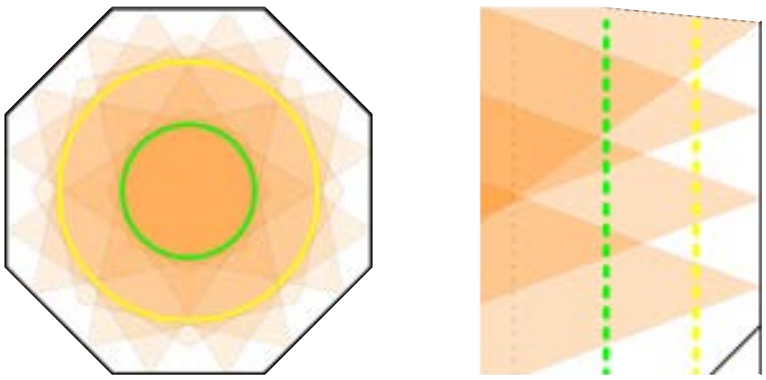
Abbildung 7: Iterationen Kamerahalter

Kamerahalter

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Im Zuge der verlängerten Lieferzeiten wurden parallel zur Firmware auch die Halterungen für die Raspberry Pis und Kameras konstruiert. Zur Kostenersparnis war von Beginn an geplant, diese später auf 3D-Druckern zu fertigen, was spezielle Anforderungen an das Design stellte. 3D-gedruckte Teile sind nicht *isotrop*, da Kräfte, die parallel zu den Druckebenen wirken, weniger effizient aufgenommen werden als Kräfte, die senkrecht dazu einwirken. Zudem sind Polylactide (PLA), der kostengünstigste und am einfachsten zu druckende Werkstoff, relativ spröde, was in Kombination mit dünnen Wandstärken zu einem leichten Brechen der Teile führen kann, wenn ungünstige Kräfte auftreten. Für maximale Stabilität war es daher erforderlich, die richtige Druckausrichtung zu wählen und starke Überhänge in dieser Ausrichtung zu vermeiden, um ein optimales Druckergebnis zu erzielen.
Im Rahmen eines iterativen Verfahrens wurde zunächst die Befestigung an den Profilen optimiert. Der erste Ansatz sah eine komplett werkzeuglose Konstruktion vor, bei der die Halterungen lediglich auf die Profile geklemmt wurden. Diese Lösung funktionierte bei geringen Klemmenbreiten bis etwa 10mm sehr gut, doch bei größeren Breiten traten Probleme auf, da sich PLA, bedingt durch seine Sprödigkeit, nicht ausreichend verformen ließ. Insbesondere mit der geplanten Breite des Raspberry Pis war es nicht möglich, eine Halterung zu entwerfen, die einerseits einfach auf das Profil gesteckt werden konnte und andererseits fest genug saß, um ein Verrutschen zu verhindern. Daher wurde die Konstruktion so angepasst, dass die Halterung zwar weiterhin in der vertikalen Nut sitzt, jedoch zusätzlich durch eine Schraube fixiert wird, um ein Verrutschen zu unterbinden.
Nachdem diese Lösung ausreichend getestet wurde, erfolgte die Entwicklung einer Methode zur Montage des Raspberry Pi im Halter. Die technischen Dokumentationen geben vor, dass die

Montagelöcher des Pis für M2,5-Schrauben vorgesehen sind. Erfahrungen aus vergangenen Projekten zeigten jedoch, dass Gewinde kleiner als M4 in Kunststoff äußerst schwer zu schneiden sind, da sich die Gewinde aufgrund der Reibung des Schneidwerkzeugs schnell erhitzen und schmelzen können. Eine bewährte Alternative sind *Heat-Set-Inserts*, also Messingteile, in denen das entsprechende Gewinde bereits vorhanden ist und die in das Werkstück eingeschmolzen werden. Diese Methode bietet zuverlässige und dauerhafte Verbindungen, die auch mehrmaliges Ein- und Ausschrauben gut aushalten. Da jedoch die Inserts in Kleinmengen bei deutschen Händlern nur zu einem deutlich höheren Preis erhältlich waren und der Arbeitsaufwand in Anbetracht der Menge der Raspberry Pis nicht unterschätzt werden durfte, entschied man sich letztlich für einen gänzlich werkzeuglosen Ansatz. In dieser Lösung wird der Raspberry Pi mittels Heißkleber fixiert, wobei die Platine auf der einen Seite durch eine speziell geformte Nut gehalten wird und auf der anderen Seite durch eine Heißklebenaht gesichert ist. Tests zeigten, dass diese Konstruktion eine ausreichende Festigkeit gewährleistet und gleichzeitig ermöglicht, den Kleber bei Bedarf mit leichtem Kraftaufwand zerstörungsfrei zu lösen, ein wichtiger Aspekt, um einen einfachen Austausch im Schadensfall zu gewährleisten. Zudem wurden die Halterungen so entworfen, dass sämtliche Anschlüsse des Pis zugänglich bleiben, was zukünftige Ausbaustufen erleichtert.
Das Design der Halterung für das Kameramodul wurde nach intensiven Diskussionen bezüglich der Ausrichtung entwickelt. Ursprünglich wurde überlegt, die einzelnen Kameras stets exakt auf die Mitte des Kamerarigs auszurichten, was jedoch einen erheblichen Mehraufwand in der Konstruktion bedeutet hätte. Tests mit synthetischen Datensätzen (**??**) ergaben, dass die exakte Ausrichtung der Kameras relativ unkritisch ist, da die hohe Anzahl an Kameras eine ausreichende Abdeckung sicherstellt. Daher wurde entschieden, einen universellen Halter zu konstruieren, der die Kamera orthogonal zum Aluminium-Profil ausrichtet. Lediglich im obersten Ring wurden die Kameras leicht vertikal angewinkelt, um eine verbesserte Aufsichtsperspektive zu ermöglichen. Dabei wurde das Design so angepasst, dass dieser Winkel leicht im CAD-Programm angepasst werden konnte, falls sich zukünftig weitere Anforderungen ergeben sollten. Ähnlich wie bei den Raspberry Pis wird auch das Kameramodul auf der einen Seite durch eine Nut und auf der anderen Seite durch einen Heißklebepunkt fixiert, wodurch eine stabile und dennoch bei Bedarf lösbare Montage erreicht wird.



Horizontal Vertikal

Abbildung 8: Interpolationsvolumen Horizontal & Vertikal

Das Interpolationsvolumen beschreibt den Bereich, in dem sich das Sichtfeld mehrerer Kameras überlappt. In der geplanten Konfiguration und Positionierung der Kameras ergibt sich laut Abbildung 8 ein entsprechendes Volumen: Horizontal betrachtet entsteht in der Mitte ein Kreis mit einem Durchmesser von etwa 1,34m (grün), in dem sich alle Kameras überlappen. Darüber hinaus existiert ein größerer Kreis mit 2,6m Durchmesser (gelb), in dem in Nahbereichen an jeder Strebe jeweils eine Kamera verloren geht, das Objekt jedoch weiterhin von mindestens einer Kamera erfasst wird. Vertikal zeigt sich, dass innerhalb des grünen Kreises immer eine Überlappung von mindestens zwei Kameras gegeben ist, während dies innerhalb des gelben Kreises nicht in jedem Fall garantiert werden kann. Diese Visualisierung macht deutlich, dass das Rig innerhalb eines Bewegungsradius von knapp unter einem Meter gut in der Lage sein sollte, Objekte zu erfassen und zu reproduzieren.

Stromversorgung

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Aufgrund der Vielzahl angeschlossener Geräte und der erheblichen Kabellängen bei niedriger Betriebsspannung mussten besondere Anforderungen an die Stromversorgung berücksichtigt werden. Die technischen Spezifikationen der Raspberry Pis verlangen eine stabile Versorgungsspannung im Bereich von 4,75V bis 5,25V (Raspberry Pi Foundation 2024). Um den Spannungsabfall bei langen Kabelstrecken zu minimieren, war in der Planungsphase des Prototyps ursprünglich der Einsatz von *Hutschienennetzteilen* vorgesehen, die eine Ausgangsspannung von 24V liefern. Diese Netzteile sollten in Schaltkästen integriert werden, die jeweils einen Quadranten des Rigs mit Strom versorgen. Die höhere Spannung von 24V diente dabei sowohl zur Speisung von LED-Lampen für die Beleuchtung als auch zur Versorgung der Raspberry Pis über geeignete Spannungswandler. Jeder Pi sollte dabei einen eigenen *Step-Down-Converter* erhalten, der in unmittelbarer Nähe installiert wird und die 24V in die erforderlichen 5V umwandelt. Durch die geringe Länge der 5V-Leitung würde der Spannungsabfall entsprechend stark minimiert werden. Da die Hutschienennetzteile nicht vorkonfektioniert sind, müssen die Anschlussarbeiten an die 230V-Netzspannung durch eine qualifizierte Elektrofachkraft erfolgen, zu diesem Zeitpunkt wurde angenommen, dass diese Arbeiten kostengünstig in der Zentralwerkstatt Elektrotechnik der TH Köln durchgeführt werden könnten.

Abbildung 9:
Render Stromversorgung

Wie bereits in **Kamerarig V1** erwähnt, war dies nach Bewilligung der Kickstart-Förderung nicht mehr möglich. Zudem waren die ursprünglich vorgesehenen Spannungswandler bei den autorisierten Händlern nur zu deutlich höheren Preisen erhältlich, sodass das gesamte Versorgungskonzept neu bewertet werden musste. Die Zentralwerkstatt Elektrotechnik stand hierbei weiter beratend zur Seite, auch wenn sie selbst keine Fertigungsarbeiten durchführen durfte. Nach weiterer Abstimmung wurde entschieden, die Stromversorgung über USB-Netzteile umzusetzen. In diesem Konzept erhält jeder Raspberry Pi ein separates Netzteil, das über den USB-C-Anschluss mit der erforderlichen Spannung

versorgt wird. Da die Raspberry Pis das USB Power Delivery (USB-PD) Protokoll nicht unterstützen, können Standard-5V-Netzteile verwendet werden. Entsprechend der Spezifikation des offiziellen Raspberry Pi Netzteils wurden Geräte ausgewählt, die einen Maximalstrom von 3A liefern. Laut Datenblatt beträgt der maximale Eingangsstrom dieser Netzteile 0,4A bei 230V, was bei 64 Raspberry Pis einer Gesamtleistung von 5888W entspricht. Auch wenn diese Spitzenleistung nur kurzzeitig beim Einschalten der Netzteile auftritt, kann sie nicht über eine einzelne Sicherung bezogen werden, da herkömmliche Sicherungen meist mit 16A (entsprechend 3680W) ausgelegt sind. Glücklicherweise sind im Raum des Versuchsaufbau drei separat abgesicherte Stromkreise vorhanden, sodass die notwendige Versorgung gewährleistet ist. Die Netzteile werden über 10-fach-Steckdosen versorgt, die direkt am Rig montiert werden. Eine 10-fach-Steckdose versorgt dabei acht Raspberry Pis sowie zwei zusätzliche Steckplätze für zukünftige Erweiterungen. Je zwei dieser Mehrfachsteckdosen werden zusammen geschaltet, um einen Quadranten des Rigs zu versorgen. Zur Abschätzung der Kabellängen wurde der Kabelbaum eines Quadranten ebenfalls im CAD-Modell erfasst, wobei sich herausstellte, dass die standardmäßig konfektionierten Kabel der Netzteile aller Wahrscheinlichkeit nach ohne zusätzliche Verlängerungen für jede Kameraposition ausreichen.

4.1.2 Fertigung und Montage

Im Gegensatz zur Konstruktionsphase, in der die theoretischen Planungen erstellt werden, beschreibt die *Fertigung* die tatsächliche Realisierung und Montage gemäß diesen Plänen. Da es sich bei diesem Projekt um einen Prototypenbau handelt, wurde während der Fertigung an einigen Stellen von der ursprünglichen Planung abgewichen, sobald sich herausstellte, dass die Konstruktion in der Praxis nicht umsetzbar war. Zudem traten erst nach der Fertigung Optimierungsmöglichkeiten zutage, die während der Konstruktionsphase entweder nicht erkennbar oder bewusst offen gelassen wurden, um sie während der Montage zu berücksichtigen. Daher weicht der finale Aufbau in einigen Bereichen deutlich vom ursprünglichen CAD-Modell ab.
Das zum Aufbau verwendete Werkzeug stammte größtenteils aus dem Privatbesitz der Teammitglieder, da so weitere externen Abhängigkeiten vermieden werden konnten.

Allgemeiner Aufbau

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Da das gesamte Rig bereits im CAD-Modell zusammengebaut war, konnte daraus auch eine Reihenfolge für den Zusammenbau abgeleitet werden. Zunächst wurden die Dimensionen mithilfe von Klebeband auf dem Boden markiert, um eine allgemeine Orientierung im Raum zu schaffen. Dabei wurde darauf geachtet, das Rig in möglichst gleichmäßigem Abstand zu den Wänden aufzubauen, an allen Wänden wurde mindestens ein Abstand von 0,75m eingehalten, sodass ein komfortabler Zugang rund um das gesamte Rig gewährleistet ist. Als Referenz wurde zusätzlich der Mittelpunkt (*Origin*) sowohl auf dem Boden als auch an der Decke mit farbigen Klebeband fixiert. Die zwei Referenzkreise des Interpolationsvolumens wurden zur Orientierung ebenfalls auf dem Boden markiert.
Der Aufbau des Rigs umfasste die Montage der vier vertikalen Streben, an denen jeweils vier horizontale Träger befestigt wurden, sowie die anschließende Fixierung der freischwebenden

Profile mittels 120°-Verbindern. Aufgrund erheblicher Lieferverzögerungen bei diesen Verbindern – bedingt durch die längere Nichtverfügbarkeit der zuständigen Kontaktperson und die Unklarheit, ob die Bestellung überhaupt ausgelöst worden war, musste kurzfristig eine alternative Lösung gefunden werden. Im Makerspace stand ein Lasercutter zur Verfügung, der zwar kein Metall, dafür aber Holz und Acrylplatten schneiden konnte, sodass beschlossen wurde, vorübergehend die Verbinder aus Acryl anzufertigen, um den weiteren Aufbau fortzusetzen. Die in der Konstruktionsphase befürchteten Stabilitätsprobleme traten dabei anders als erwartet auf: Die provisorischen Füße der vertikalen Streben erwiesen sich als stabil genug, um die einzelnen Segmente zu tragen, jedoch führten die Acrylverbinder zu einer gewissen Flexibilität des gesamten Aufbaus. Dies bewirkte ein Dehnen und Zusammenziehen der Oktagonform, sobald ein Kreis geschlossen wurde, wodurch Torsionskräfte über die Streben verteilt wurden. An zufälligen Stellen resultierten daraus Scherkräfte, die zum Versagen der Acrylverbinder führten. Um keine weiteren Komponenten zu gefährden, wurde entschieden, die Montage der Kameras bis zum Eintreffen der Stahlverbinder auszusetzen. Zur weiteren Stabilisierung des Rigs wurden an allen vier vertikalen Streben Traversen aus Holzbalken eingezogen, was die Stabilität erheblich verbesserte. Zusammen mit den zwischenzeitlich gelieferten Stahlverbinder wurde das Rig dadurch robust genug, um auch externe Stöße abzufangen.

Abbildung 10: Holztraverse

Parallel zum Aufbau des Rigs wurden zudem die Kamerahalter gefertigt, wobei drei verschiedene 3D-Drucker zum Einsatz kamen. Die Druckgeschwindigkeiten variierten dabei beträchtlich, von etwa 0,5h bis über einer Stunde pro Halter. Die Montage der Raspberry Pis in den Haltern sowie die Befestigung der Halter an den Profilen gestaltete sich problemlos, wobei auf jedem Raspberry Pi eine gut sichtbare Nummerierung angebracht wurde, um die spätere Fehlersuche zu erleichtern.

Abbildung 11: Fertig montierter Raspberry Pi

Während des Aufbaus stellte sich heraus, dass lediglich zwei der drei im Raum vorhandenen Stromkreise aktiv waren, was zunächst die Befürchtung aufkommen ließ, dass die Stromversorgung unzureichend sein könnte. Daher wurde in den anfänglichen Tests vorsichtig mit einem Leistungsmesser der Stromverbrauch des Rigs überwacht, um eine Überlastung der Sicherungen zu vermeiden. Die Messungen zeigten jedoch, dass die zuvor errechneten Leistungsangaben deutlich überzogen waren, da das gesamte Rig, selbst unter Last, weniger als 200W verbrauchte. Es sollte beim Einschalten dennoch streng darauf geachtet werden, dass alle verfügbaren Leisten nacheinander eingeschaltet werden, da das gleichzeitige Aktivieren eines Quadranten doch zu einem Auslösen der Sicherung führt, bedingt durch den Einschaltstrom der Netzteile.

Firmware

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Im folgenden Abschnitt wird die Konfiguration der Firmware auf den Raspberry Pis erläutert. Für detaillierte Informationen zu den einzelnen Softwarekomponenten wird auf das Kapitel Capture Software verwiesen.
Da auf allen Pis identische Firmware zum Einsatz kommen soll, wurde zunächst ein Raspberry Pi als Referenzsystem eingerichtet. Da diese Pis lediglich remote angesteuert werden, wurde hierfür Raspberry Pi OS Lite gewählt, um durch den Wegfall der grafischen Benutzeroberfläche Ressourcen für andere Aufgaben freizugeben. Die Formatierung erfolgte mithilfe des Raspberry Pi Imager, wobei neben der Konfiguration der WLAN-Zugangsdaten auch die Aktivierung von Secure Shell (SSH) sichergestellt wurde. Diese SSH-Funktion bildet das Fundament der Kommunikation mit den Pis, da ohne sie weder Ressourcen aktualisiert noch die Skripte in ihrer aktuellen Funktion genutzt werden können.
Zur Vereinfachung des Umgangs mit zahlreichen Geräten erhalten alle Raspberry Pis denselben Benutzernamen „voluman“ und das Passwort „pi“. Dies gewährleistet, dass beim automatisierten Fernzugriff keine unterschiedlichen Logindaten berücksichtigt werden müssen. Anschließend wurden die für die Aufnahme benötigten Python-Skripte im Nutzerordner abgelegt. Von diesem Setup wurde ein erstes *Image* der Speicherkarte erstellt, welches anschließend auf alle Geräte übertragen wurde. Da alle Speicherkarten demselben Modell entsprachen, wurde eine einheitliche Schreibgeschwindigkeit erwartet. Das Aufspielen der Images nahm allerdings zwischen 10min und einer Stunde pro Karte in Anspruch und variierte ohne erkennbares Muster von Karte zu Karte. Zudem wurden von insgesamt 80 Karten mindestens vier während des Flashvorgangs irreparabel beschädigt. Diese Beobachtungen deuten auf erhebliche Schwankungen in der Fertigungsqualität während des Produktionsprozesses hin.
Ein weiterer wichtiger Bestandteil des Systems ist die Datei `camera_list.json`, in der alle verfügbaren Raspberry Pis samt Hostnamen und IP-Adressen hinterlegt sind. Jeder Pi erhält eine feste IP-Adresse. Der Adressbereich für die Kameras beginnt bei .100, wobei die Adressen fortlaufend zugewiesen werden und sich an der Nummerierung der jeweiligen Pis orientieren. So weist beispielsweise der Pi mit dem Hostnamen CAM00 die IP-Adresse .100 auf, CAM01 erhält .101 usw. In früheren Versionen von Raspberry Pi OS wurde die Festlegung der IP-Adressen im Headless-Betrieb durch eine Bearbeitung der Datei `dhcpcd.conf` vorgenommen. Da jedoch in neueren Versionen standardmäßig der NetworkManager anstelle des Dynamic Host Configura-

tion Protocol Client Daemon (dhcpcd) aktiviert ist, kann die Konfiguration fester IP-Adressen ausschließlich über die Kommandozeile erfolgen. Da die Standardeinstellung die Zuweisung der IP-Adresse per DHCP vorsieht, mussten die Einrichtung der fixen IP-Adressen und die Umbenennung der Hostnamen initial manuell auf allen Pis durchgeführt werden.
Nach Abschluss dieser Einrichtung wurde ein Update-Skript entwickelt, das basierend auf der `camera_list.json` zukünftige Updates der Remote-Skripte per Secure Copy Protocol (SCP) ermöglicht und zusätzlich die Installation weiterer Pakete und Module über SSH realisiert. Dank der zentral gepflegten Kameraliste werden diese Updates automatisch auf alle in der Liste aufgeführten Kameras angewendet, sodass alle Raspberry Pis stets auf dem gleichen Stand gehalten werden.
Im Rahmen dieser Projektdokumentation wird ein Master-Image bereitgestellt, das den finalen Firmware-Stand vollständig abbildet. Dieses Image ermöglicht es, das bestehende System flexibel zu erweitern, indem neue Kameras hinzugefügt oder ausgefallene Speicherkarten ersetzt werden können. Dafür muss lediglich einmalig die entsprechende Kameranummerierung konfiguriert werden, was den administrativen Aufwand erheblich reduziert und eine nahtlose Integration gewährleistet.

Anpassung Kamerahalter

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

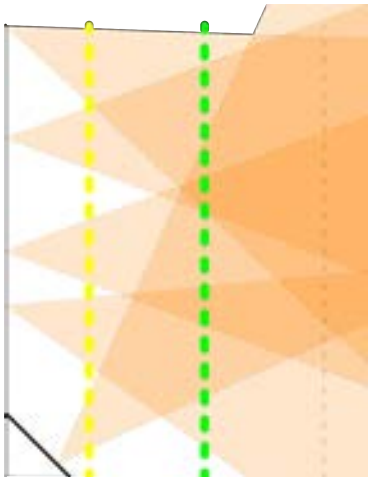


Abbildung 12: Interpolationsvolumen mit überarbeiteten Kamerapositionen

Nach der erstmaligen Einrichtung der Pis war es möglich, erste Aufnahmen mit dem gesamten System zu realisieren. Um sowohl die optimale Ausrichtung als auch die Reproduzierbarkeit des Rigs zu evaluieren, wurden regelmäßig Testaufnahmen unter Verwendung des Programms Jawset Postshot durchgeführt. Dieses Tool bietet einen schnellen und unkomplizierten Workflow zur Erstellung von Gaussian Splats aus Multikamera-Aufnahmen. Für die Ermittlung der Kameraposen in Postshot kommt, wie auch im eigens vom Trainingsteam entwickelten Tool, das Colmap-Verfahren zum Einsatz. Zusätzlich erfolgten sporadische Tests mit Reality Capture, einem in der Industrie etablierten Programm.
Die umfangreichen Testaufnahmen ergaben, dass auf Aufnahmen des oberen Rings unnötig viel ungenutzter Raum nach oben verloren geht. Dies deutete darauf hin, dass die entsprechenden Kameras weiter nach unten geneigt werden könnten, ohne negative Auswirkungen auf das Endergebnis zu befürchten. Gleichzeitig würde eine solche Anpassung zu einer noch besseren Abdeckung innerhalb des Kamera-Rigs führen. Daher wurden für den oberen Ring angepasste Kamerahalterungen mit modifizierten Neigungswinkeln neu gefertigt.
Des Weiteren wurde die Ausrichtung der Kameras im untersten Ring optimiert. Diese Kameras sollten ebenfalls nach unten geneigt werden, um eine verbesserte Aufsicht über Füße und Boden zu gewährleisten. Da die bestehenden Tophalter nahezu optimal vom Winkel her passten, wurde im Sinne der Ressourcenschonung beschlossen, diese wiederzunutzen. Um in der Rig-Mitte einen vergleichbaren Bildausschnitt pro Kamera zu erzielen, wurde der gesamte Ring um ca. 40cm

angehoben. Ergänzend dazu wurden vier Bodenkameras installiert, um das Volumen um stark untersichtige Perspektiven zu erweitern. Somit beläuft sich die Gesamtzahl der Kameras im Rig auf 68, woraus sich das in Abbildung 12 dargestellte Interpolationsvolumen ergibt.

Ausrichtung gerade Ausrichtung angepasst

Abbildung 13: Vergleich Kameraausrichtung CAM00

ArUco-Marker und Tracking mit Colmap

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Neben dem Optimierungspotenzial durch die Anpassung der Kameraausrichtung zeigten die fortlaufenden Tests zunächst eine erhebliche Inkonsistenz in der Tracking-Qualität von Colmap. Dieser Umstand war vor allem auf den symmetrischen Aufbau und die Positionierung der Kameras sowie zweier weißer Raumwände zurückzuführen. Da Colmap auf dem Prinzip *Structure from Motion* basiert und auf eindeutig bestimmbare optische Feature-Punkte angewiesen ist (Schönberger und Frahm 2025), führte die hohe Symmetrie zu einer Verwechslung zahlreicher dieser Merkmale, da sich die Analyse infolge der glatten Wandoberflächen primär auf die Form der Raspberry Pis konzentrierte.

Um die Symmetrie im System aufzubrechen, wurden zunächst temporär verschiedene Objekte, wie beispielsweise Akustikwände mit Noppenschaum, um das Rig herum angebracht. Diese Maßnahme steigerte die Zuverlässigkeit von Colmap bereits deutlich, allerdings blieb die Qualität der Analyse maßgeblich abhängig von der Struktur der im Rig enthaltenen Subjekte. So konnten beispielsweise Personen, die Hemden mit Knopfleiste oder bedruckte Oberteile trugen, aufgrund der klar unterscheidbaren Vorder- und Rückseite gut getrackt werden, während bei Kleidungsstücken ohne klar erkennbare Struktur, wie zum Beispiel Fleecepullovern, häufig Probleme auftraten.

Vor diesem Hintergrund wurden verschiedene Konzepte zur weiteren Verbesserung des Trackings und zur Steigerung der Konsistenz erarbeitet. Eine Idee sah vor, die Positionen der Kameras geringfügig von ihrer ursprünglichen Anordnung zu randomisieren, um die erwünschte Aufbrechung der Symmetrie zu erzielen und Colmap differenzierbarere Referenzpunkte zu liefern. Da dieser Ansatz jedoch mit erheblichem Konstruktionsaufwand verbunden gewesen wäre und zudem die Vergleichbarkeit mit bisherigen Aufnahmen erschwert hätte, wurde zunächst der Einsatz von ArUco-Markern getestet. Um eine zufällige Verteilung der Marker im Rig zu gewährleisten und zu verhindern, dass durch menschliche Gewohnheitseffekte erneut symmetrische Muster entstehen, wurden die Marker nach einem zuvor definierten System angebracht: Das Rig wurde in vier Quadranten unterteilt, wobei in jedem dieser Quadranten pro Kreis fünf mögliche Befesti-

gungspunkte zur Verfügung standen. Aus diesen fünf Positionen wurden mittels eines Skripts zufällig zwei Positionen bestimmt, wobei darauf geachtet wurde, dass sich die resultierenden Muster in den Quadranten nicht wiederholen. Insgesamt wurden auf diese Weise 24 Marker im Rig platziert.

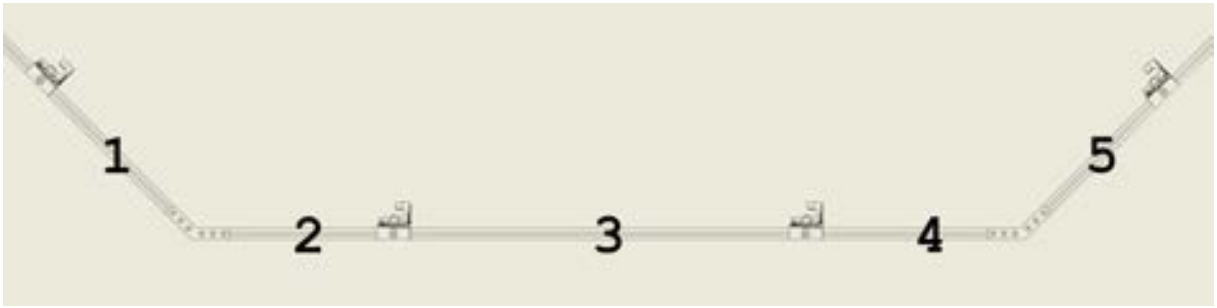


Abbildung 14: Positionen pro Ringsegment

Der Einsatz der ArUco-Marker führte schließlich zu den gewünschten Ergebnissen, allerdings erst, nachdem die Anzahl der Features pro Kamera für die Analyse reduziert wurde. Je höher die Anzahl der Features, desto stärker kam es zu Duplizierungen von Kamera-Positionen in der Analyse. Da die Marker relativ große Muster im Bild erzeugen, deutet dieses Verhalten darauf hin, dass eine erhöhte Feature-Anzahl zu einer Berücksichtigung zusätzlicher Details und feinerer Strukturen führt. Da jedoch deutlich mehr Raspberry Pis als Marker im Rig vorhanden sind, bewirkt eine höhere Anzahl von Kamerafeatures, dass diese übermäßig gewichtet werden und die symmetrische Positionierung erneut problematisch wird. Während der voreingestellte Wert in Postshot bei 8.000 Features liegt, liefert die Analyse bereits bei lediglich 2.000 Features ein zuverlässiges und konstantes Ergebnis. Ein höherer Wert führt zu schlechteren Ergebnis. Ein zusätzlicher positiver Nebeneffekt der reduzierten Feature-Anzahl ist die beschleunigte Analyse, da weniger Komponenten berechnet werden müssen.

Beleuchtung

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Um eine gleichmäßige Ausleuchtung des Rigs zu gewährleisten und eine möglichst präzise Farbtreue sicherzustellen, wurde die Verwendung von mindestens vier LED-Panels vorgesehen. Im Vergleich zu Einzellampen bieten LED-Panels den Vorteil, dass sie von vornherein diffuses Licht erzeugen, was zu einer homogenen Ausleuchtung ohne signifikante Schattenbildung führt. Zudem weisen LED-Lampen im Gegensatz zu den im Raum vorhandenen Neonröhren den Vorteil auf, dass sie nahezu keiner Alterung unterliegen und ihre Farbtemperatur über die gesamte Lebensdauer konstant bleibt. Der Einsatz von Lampen aus dem Pro-Video-Bereich garantiert darüber hinaus eine hohe Farbtreue, die bei kostengünstigeren LED-Lampen oder Neonröhren häufig nicht gegeben ist.
Aufgrund des verbleibenden Restbudgets war es nicht möglich, RGB- oder Bicolor-Leuchten anzuschaffen. Folglich kann die Beleuchtung des Rigs nicht in Kombination mit dem Raumlicht betrieben werden, da die unterschiedlichen Farbtemperaturen ansonsten zu einer unerwünschten Mischlicht-Situation führen würden. Letztendlich wurden vier Walimex Pro LED-Lampen mit einer Leistung von je 65W und einer Farbtemperatur von 5600K ausgewählt. Dank der geringen

Leistungsaufnahme lassen sich diese Lampen problemlos in die bestehende Stromversorgung des Rigs integrieren. Die Lampen wurden jeweils in Augenhöhe an einer der vier vertikalen Streben montiert, um eine optimale Ausleuchtung zu erzielen.

Posten	**Kosten (EUR)**
Kamerakomponenten (Raspberry Pi, Kamera-Module, SD-Karten)	5 714,89
Kamerarig (Aluminiumprofile, Befestigungsschrauben, Halterungen)	952,11
Netzwerkrouter	132,36
Stromversorgung (Netzteile und Steckdosenleisten)	692,95
Beleuchtung	692,51
Gesamtsumme	**8 184,82**

Tabelle 2: Kostenaufstellung

Finale Arbeiten

Nachdem alle kritischen Punkte behoben und das Rig in seiner Funktion umfassend getestet wurde, erfolgten abschließend kleinere Optimierungsarbeiten. Für die Erstellung der Gaussian Splats sowie weitere Verarbeitungsschritte ist ein neutraler und detailarmer Hintergrund von Vorteil. Aus diesem Grund wurde im vorderen Segment des Rigs ein Vorhang installiert, der eine visuelle Abtrennung vom übrigen Raum ermöglicht und gleichzeitig verhindert, dass Personen außerhalb des Rigs auf den Aufnahmen erscheinen.

Zusätzlich wurden nach Projektende weitere Lampen zur verbesserten Ausleuchtung sowie vier Deckenkameras angebracht, um insbesondere die Reproduktionsqualität aus der oberen Perspektive weiter zu optimieren. Tabelle 2 zeigt die finale Kostenaufstellung für das Volumetric Capture System.

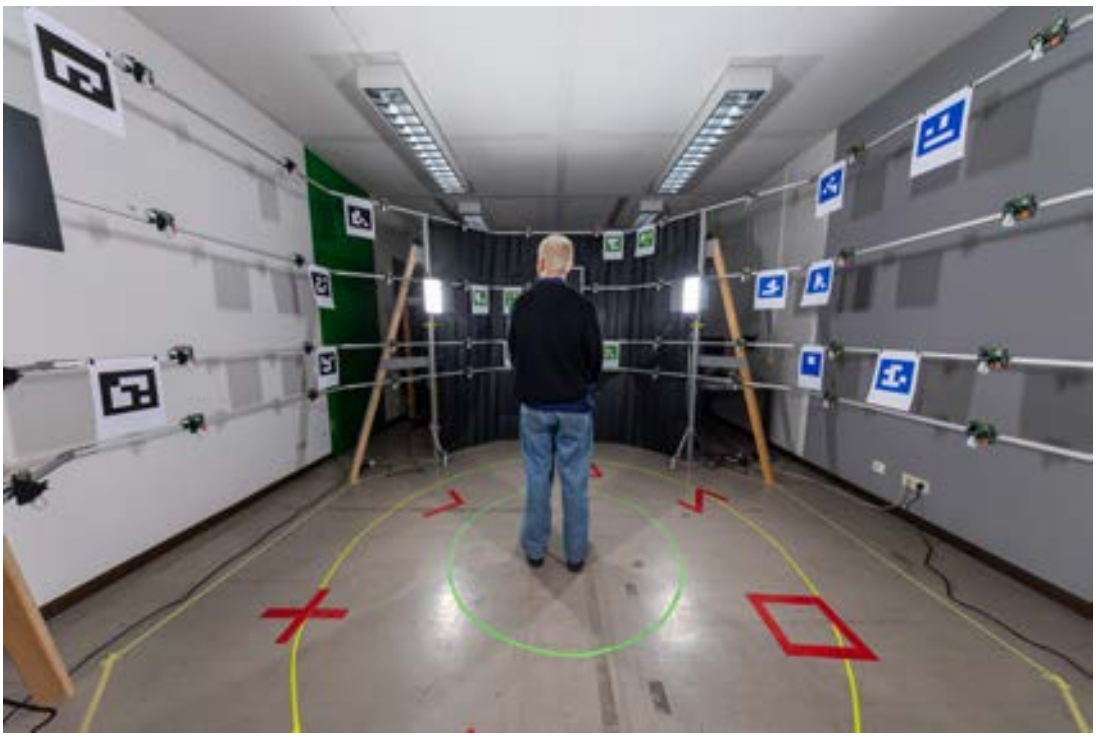


Abbildung 15: Finales Capture Rig

Capture Software

Die in diesem Projekt eingesetzten 68 Raspberry Pis, die jeweils mit einem Kameramodul ausgestattet sind und eigenständig Aufnahmen erstellen sollen, benötigen eine entsprechende Software, die in der Lage ist, diese Aufnahmen zu erzeugen. Die insgesamt drei Kernelemente der Software sind neben der Aufzeichnung aber auch eine einheitliche Kalibrierung der Kameras sowie der Datentransfer. Im Folgenden wird die Umsetzung dieser drei Kernelemente der Software genauer beschrieben. Aus teils technischen Gründen sind diese auf drei einzelne Programme aufgeteilt, die jeweils aus einem Master- und einem Remote-Teil bestehen.

4.1.3 Capture Controller

Der *Capture Controller* ist für die Aufnahmesteuerung zuständig. Er besteht aus einem Master-Skript zum Senden von Steuerbefehlen und einem Remote-Skript, welches die Steuerbefehle des Master-Skripts umsetzt. Beide Teile ermöglichen zusammen eine synchrone und zentrale Verwaltung von Aufnahmen. In diesem Abschnitt werden die besonderen Funktionsweisen von Master- und Remote-Teil erläutert.

Steuerungskonzept

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Bei dem Master-Skript des Capture Controllers handelt es sich um ein Python-Skript, welches auf einem Host-Computer mit Windows oder Unix (macOS oder Linux) Betriebssystem ausgeführt wird. Es bietet eine grafische Benutzeroberfläche zum Steuern von Aufnahmen und visualisiert Rückmeldungen zum momentanen Status und Fortschritt der 68 Raspberry Pis im Folgenden Agents genannt. Im Kern steht jedoch die Remote-Statemachine, welche ebenfalls in Form eines Python-Skripts auf allen Agents im Betrieb ausgeführt wird. Die Remote-Statemachine empfängt und verarbeitet die Befehle des Master-Skripts und setzt diese lokal auf den Agents um. Darüber hinaus sendet sie einmal pro Sekunde ein Feedback über ihren aktuellen Status und weiteren Informationen an das Master-Skript zurück. Auf diese Weise lassen sich Sessions (Videoaufnahmen) und Stills (Einzelbilder) synchron und dennoch dezentral aufnehmen.

Benutzeroberfläche

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Die in Abbildung 16 dagestellte Benutzeroberfläche des Capture Controllers ist mithilfe von Tkinter realisiert und bietet verschiedene Optionen zur Steuerung der Aufnahmen sowie Übersichten über den Systemstatus. Eine Session, also eine Videoaufzeichnung mit allen Agents, kann komfortabel über das UI gestartet werden. Dabei kann ein frei wählbarer Sessionname und eine gewünschte Datenrate vom Benutzer festgelegt werden. Die Aufnahme wird über einen Button gestartet und kann über einen weiteren Button gestoppt werden. Für einen Still, also eine Standbildaufnahme mit allen Agents, stehen ebenfalls Eingabemöglichkeiten zur Verfügung, bei denen der Nutzer einen Namen vergeben und die Zielauflösung in SD, HD, FullHD oder UHD auswählen kann. Darüber hinaus zeigt das UI einen umfassenden Systemstatus an. Dieser ist

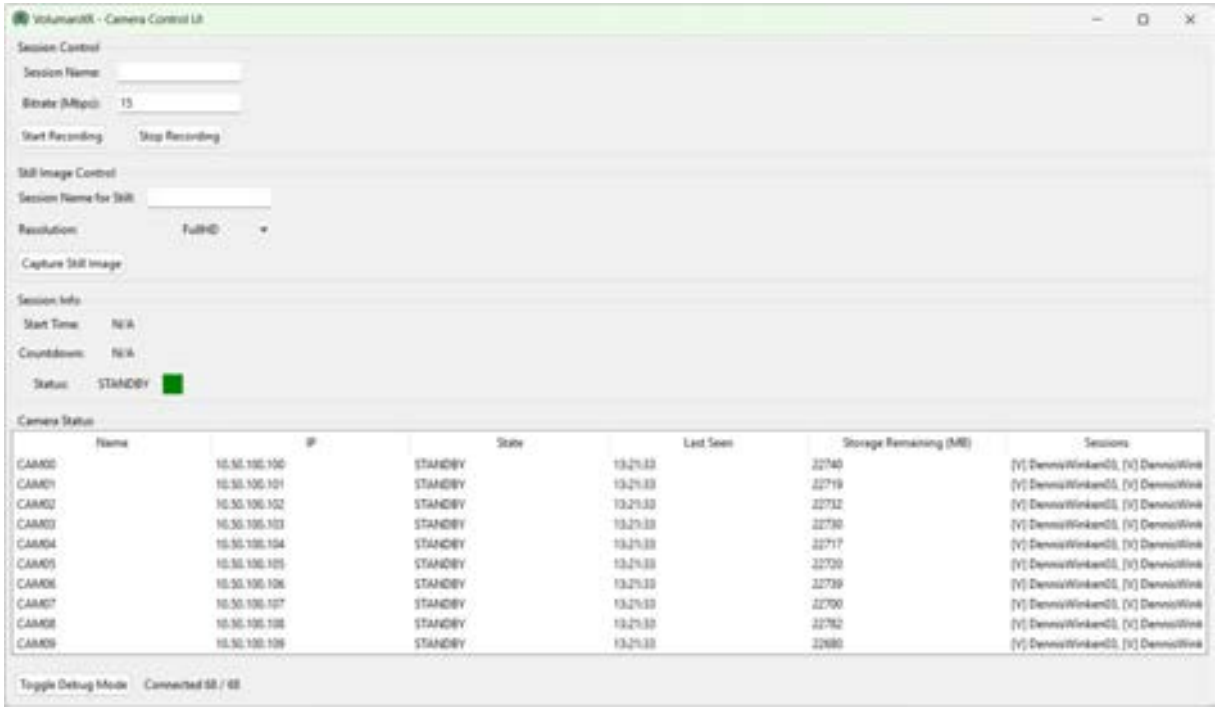


Abbildung 16: Die Benutzeroberfläche des Capture Controllers im aktuellen Status `STANDBY`

in allgemein zusammengefasste Systeminformationen und einer detaillierten Listenansicht aufgeteilt. In den allgemeinen Systeminformationen wird der Gesamtstatus des Systems angezeigt. Dieser kann sich im Status `STANDBY`, `PREPARING` oder `RECORDING` befinden. Zusätzlich informiert das UI über den absoluten Startzeitpunkt einer Aufnahme, wenn eine Aufnahme initiiert wurde. Ein Countdown zeigt an, in wie vielen Sekunden die nächste Aufnahme beginnt bzw. nach Ablauf des Countdowns wird angezeigt, wie lange eine laufende Sitzung bereits aufgezeichnet wird. Die Aufnahme startet aus Gründen der Synchronisation nämlich nicht sofort, sondern zu einem festgelegten, absoluten Zeitpunkt. Die Listenansicht im UI bietet eine detaillierte Ansicht aller Agents, welche in der IP-Liste definiert sind, inklusive ihres Hostnamens, ihrer IP-Adresse, ihres aktuellen Status, des letzten Meldezeitpunkts, des freien Speicherplatzes auf der MicroSD-Karte sowie die Namen der letzten drei aufgezeichneten Sessions. Zusätzlich zeigt die UI an, wie viele Agents momentan mit dem System verbunden sind.
Da alle Agents einmal pro Sekunde ihren Status an den Capture Controller übermitteln, können eventuelle System- und Verbindungsprobleme schnell erkannt werden. Schickt ein Agent länger als fünf Sekunden keine Statusmeldung, wechselt sein Status automatisch auf `NO RESPONSE`. Um zu verdeutlichen, ob es sich bei einer gespeicherten Aufnahme um eine Session (Video) oder ein Still (Bild) handelt, werden diese im Treeview mit einem `V` (Video) oder einem `I` (Image) gekennzeichnet. Letztlich kann der Debug Mode über ein zusätzliches Fenster Einblick darin geben, welche Nachrichten von den einzelnen Agents empfangen werden.

Aufzeichnung von Sessions und Stills

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Die Kernaufgabe des Capture Controllers besteht darin, verschiedene Arten von Aufnahmen zu ermöglichen. Dabei wird zwischen kontinuierlichen Videoaufnahmen (Sessions) und statischen Einzelaufnahmen (Stills) unterschieden. Auf diese Weise kann das System sowohl längere Szenarien durchgängig festhalten als auch statische Aufnahmen zu Referenz- oder Kalibrierungszwecken erstellen.

Alle auszuführenden Skripte, Konfigurationsdateien sowie der Ablageordner für aufgezeichnete Dateien befinden sich im Verzeichnis `/home/voluman` des Benutzers `voluman` auf den Raspberry Pis. Die neu erstellten Aufnahmen werden im dort befindlichen Unterverzeichnis `Recordings` gespeichert.

Abbildung 17 visualisiert den im Folgenden beschriebenen Ablauf einer Session-Aufnahme. Die Aufnahme auf den Agents beginnt mit einer Initialisierung durch den Capture Controller, welche Initialisierungsnachrichten an alle Agents sendet. Vor dem eigentlichen Aufnahmestart werden grundlegende Kameraparameter aus der Datei `camera_settings.json` eingelesen. Diese Konfigurationsdatei befindet sich lokal auf jedem Agent und ist auf allen Agents systemweit identisch. Dadurch wird sichergestellt, dass alle relevanten Kameraeinstellungen einheitlich gesetzt sind, um konsistente Aufnahmen über alle Agents hinweg zu gewährleisten. Details zur Konfiguration der Kameraparameter sind in Abschnitt 4.1.4 beschrieben.

Weil der Fokus jeder Kamera individuell kalibriert sein muss, ist der dafür optimale wert der *Lens Position* aus Gründen der Benutzerfreundlichkeit in der IP-Liste hinterlegt, anstelle der `camera_settings.json`. Dieser jeweils spezifische Fokuswert wird bei der Initialisierung einer Aufnahme vom Capture Controller an jeden Agent übermittelt.

Nach der Konfiguration der Kameraparameter wird bei einer Session-Aufnahme der Video-Encoder mit der zuvor festgelegten Datenrate konfiguriert und gestartet. Sobald der im Controller definierte absolute Aufnahme-

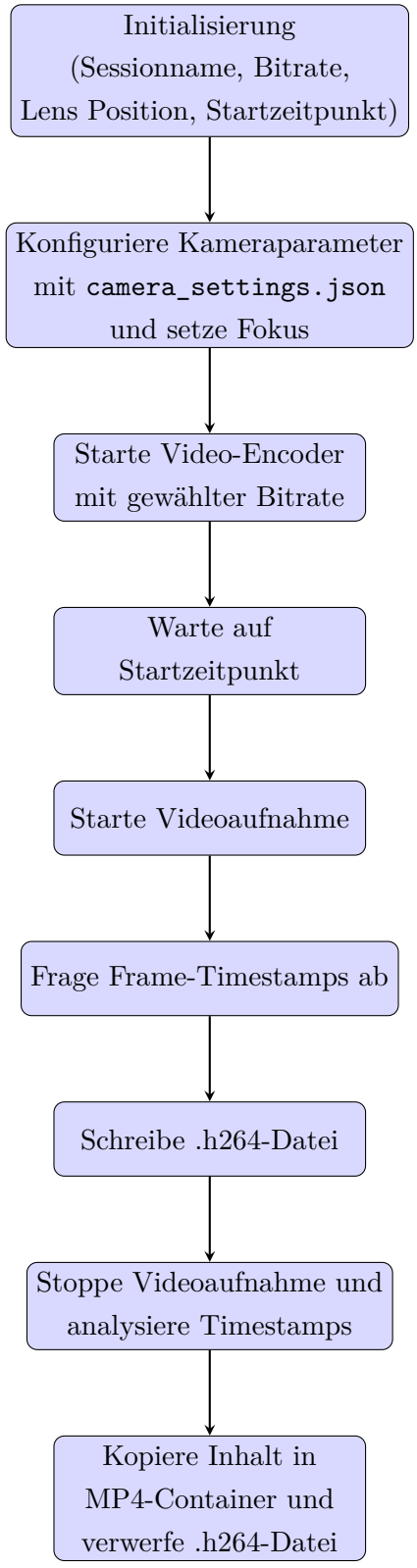


Abbildung 17: Ablauf einer Session

zeitpunkt erreicht ist, beginnt die Aufnahme. Während der Aufzeichnung analysiert der Aufnahmeprozess kontinuierlich den Kamerasensor, um die Zeitpunkte der einzelnen aufgenommenen Frames zu erfassen. Zwar beziehen sich diese Zeitpunkte auf den jeweiligen Systemstart des Agents und sind damit nur schwer referenzierbar, jedoch gewähren die Abstände dieser Zeiten zueinander unter Berücksichtigung der Framerate Aufschluss darüber, ob bei der Aufnahme eventuell Frames gedroppt wurden. Eine detaillierte Beschreibung dieses Prozesses findet sich in Abschnitt 4.1.3. Nach dem Beenden der Session erfolgt automatisch eine Analyse der erfassten Zeitstempel der Frames. In einer JSON-Datei, die denselben Namen wie die Session trägt, werden daraufhin die Aufnahmezeitpunkte aller potenziellen Frames gespeichert. Wird ein Frame nicht zum richtigen Zeitpunkt oder gar nicht aufgenommen, bekommt er anstelle eines Zeitstempels die Kennung *dropped* zugewiesen.

Bei einer Still-Aufnahme, siehe Abbildung 18, wird während der Kamerakonfiguration die Auflösung mit der vom Nutzer gewählten Auflösung gesetzt, anstatt die in der Konfigurationsdatei hinterlegte Standardauflösung zu verwenden. Nach der Konfiguration wartet das System auf das Erreichen des Auslösezeitpunkts, um das Bild aufzunehmen.

Um eine eindeutige Zuordnung sämtlicher Aufnahmen zu gewährleisten, folgt das System einer festen Dateinamensstruktur. Der gewählte Name der Session oder des Stills wird mit dem letzten Abschnitt der jeweiligen IP-Adresse kombiniert und um den entsprechenden Dateityp ergänzt. Es folgt ein Beispiel: Hat ein Agent die IP-Adresse `10.50.100.123` und lautet der Name der Aufnahme `PingPong01`, dann ergibt sich der letztendliche Dateiname:

`PingPong01_123.xxx` .

Eine aufgenommene Session erhält die Dateiendung `.h264`. Diese untypische Dateiendung wird durch den verwendeten Picamera2-Encoder verwendet, welcher Sessions mit dem H.264-Codec encodiert. Anschließend wird die aufgenommene .h264-Datei mit dem Programm FFmpeg lokal auf dem Agent in einen MP4-Container mit der Dateiendung .mp4 kopiert, ohne dabei den Videostream neu zu encodieren. Dies hat den Vorteil, dass die ursprüngliche Bildqualität erhalten bleibt und lediglich ein schneller Containerwechsel stattfindet, der wiederum keine besonderen Hardwareanforderungen stellt und somit problemlos auf den Agents nach Beenden der Session umgesetzt werden kann. Die ursprüngliche .h264-Datei wird daraufhin nach Erstellen der .mp4-Datei verworfen. Handelt es sich hingegen um eine Still-Aufnahme, erfolgt die Aufnahme im JPG-Format und der Dateityp lautet .jpg.

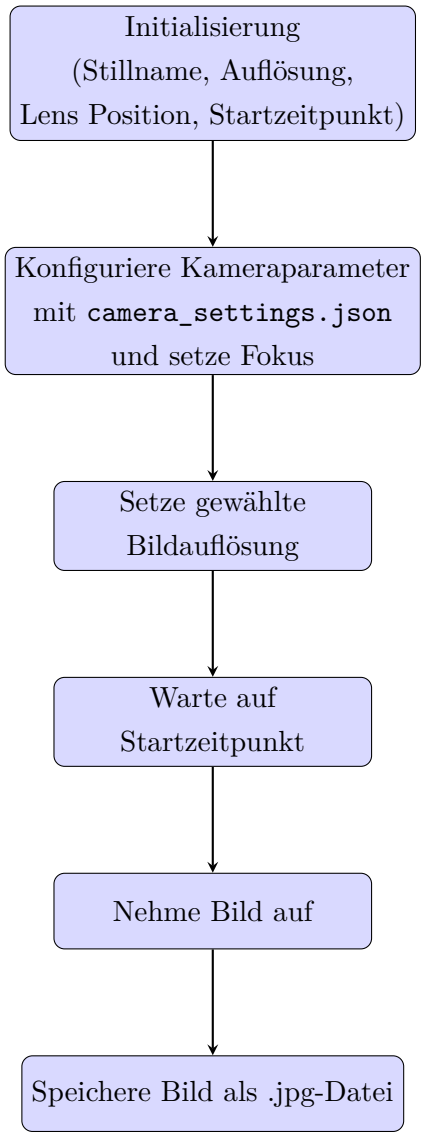


Abbildung 18: Ablauf eines Stills

Netzwerkkommunikation

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Eine zuverlässige Netzwerkkommunikation ist für gleichzeitige synchrone Aufnahmen auf allen 68 Raspberry Pis unverzichtbar. Zentrales Ziel ist, dass Sessions framegenau gestartet werden und Stills ebenfalls zeitsynchron aufgenommen werden. Es erfordert also eine äußerst latenzarme Kommunikation zwischen Capture Controller und Agents. Daneben muss gewährleistet sein, dass das System robust auf Verbindungsverluste reagiert, ohne dass wichtige Steuerbefehle aufgrund von Paketverlusten verloren gehen. Beide Punkte stehen jedoch unter dem Aspekt, dass es sich bei der Netzwerkverbindung über keine direkte physische LAN-Verbindung, sondern über eine drahtlose WLAN-Verbindung handelt. Im Folgenden werden die verschiedenen Protokollansätze (UDP, TCP und ZeroMQ) beschrieben, die während der Entwicklung getestet wurden und was im System letztendlich eingesetzt wird.

Ein erster Ansatz war das Senden von einfachen Steuerbefehlen per UDP-Broadcast. UDP zeichnet sich durch geringe Latenzzeiten und die Möglichkeit aus, bei relativ geringem Datenverkehr im Vergleich zu direkten Kommunikationswegen, Nachrichten gleichzeitig an viele Empfänger zu senden. Bei einer Anzahl von bis zu drei Raspberry Pis funktioniert dies zufriedenstellend. Mit steigender Anzahl treten jedoch zunehmend Paketverluste und Inkonsistenzen im Empfang auf, sodass nicht alle Agents verlässlich für einen lokalen Aufnahmestart getriggert werden. Standardmäßig wäre UDP das bevorzugte Protokoll für einen kontinuierlichen Datenstrom, bei dem Paketverluste gelegentlich in Kauf genommen werden können. Aber selbst wenn der vom Controller gewünschte Status kontinuierlich an die Agents gesendet werden würde, würde dies nicht nur zu einem erhöhten Datenverkehr führen, abhängig von der Dichte der Aktualisierungsrate, sondern auch das Problem fördern, dass Aufzeichnungen immer noch nicht synchron ausgelöst werden, wenn Paketverluste auftreten. UDP-Broadcast erweist sich daher bei einer größeren Anzahl an Endgeräten als unzuverlässig.

Im nächsten Ansatz wurde deshalb TCP eingesetzt, das durch eine verbindungsorientierte Übertragung und Bestätigungen deutlich zuverlässiger in puncto Paketverlusten ist. Zwar treten Paketverluste gegenüber UDP damit nicht mehr auf, dennoch kam es weiterhin zu Verzögerungen, da die Reihenfolge der Zustellung sowie Bestätigungsroutinen eine simultane Ausführung behindern. Dies führte dazu, dass bei 68 Pis nicht alle zum exakt gleichen Zeitpunkt starteten. Damit war standardmäßiges TCP alleine ebenfalls nicht geeignet, um die notwendige zeitliche Genauigkeit zu erzielen.

Bei weiterer Recherche nach alternativen Kommunikationsprotokollen ist ZeroMQ positiv aufgefallen. ZeroMQ ist eine Bibliothek, die durch Low-Latency-Eigenschaften sowie asynchrone, nicht-blockierende Kommunikation gegenüber konventioneller TCP-Kommunikation heraussticht. Sie erlaubt verschiedene Kommunikationsmuster, darunter auch die eingesetzte Router/Dealer Struktur. Sowohl der Router auf der Controller-Seite als auch der Dealer auf der Seite ermöglichen eine asynchrone, nicht-blockierende Kommunikation zwischen allen eingehenden und ausgehenden Nachrichten trotz der Verwendung des TCP-Protokolls im Hintergrund. Interne Optimierungen von ZeroMQ verbessern dabei die Leistungsfähigkeit des verwendeten TCP-Protokolls gegenüber der konventionellen Implementierung. Diese Vorteile lassen sich dabei auf eine große Anzahl von Endgeräten zu skalieren.

Im Betrieb zeigt sich, dass die Zuverlässigkeit in Bezug auf Paketverluste äußerst hoch ist und die Latenz im Durchschnitt sehr niedrig bleibt. Allerdings treten trotz der Implementierung weiterhin zufällige Latenzschwankungen bei den Empfängern auf, sodass der gemeinsame Aufnahmezeitpunkt allein durch ZeroMQ noch nicht garantiert werden kann, wenn nur ein Befehl zum Aufzeichnungsstart übermittelt wird.

Aufnahme- und Netzwerksynchronisation

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Da allein durch die Netzwerkkommunikation eine Synchronität des gesamten Systems nicht garantiert werden kann, benötigt es eine alternative Methode, um eine framegenaue Synchronität bei den Aufnahmen zu gewährleisten. Anstelle eines Triggers, welcher durch eine nicht synchrone Netzwerkkommunikation zufällig beeinflusst werden kann, und entweder zum sofortigen oder einem relativ zu Trigger verzögerten Zeitpunkt eine Aufnahme auslöst, besteht die naheliegendste Alternative darin, den Auslösezeitpunkt absolut festzulegen. Das setzt nicht mehr voraus, dass die Netzwerkkommunikation synchron erfolgen muss, sondern dass die lokalen Uhrzeiten auf den Raspberry Pis untereinander synchron sind.
Die zeitliche Synchronität der Raspberry Pis wird über einen lokalen Network Time Protocol (NTP) Server erzielt. Da Raspberry Pis keine verbaute BIOS-Batterie besitzen, müssen sie sich standardmäßig beim Startvorgang mit einem öffentlichen NTP-Server aus dem Internet synchronisieren. Damit stellen sie ihre Systemzeit bei jedem Startvorgang neu ein, weil sie nicht nachzuverfolgen können, wie lange sie ausgeschaltet waren. Da das lokale Netzwerk, mit dem die Raspberry Pis verbunden sind, so ausgelegt ist, dass kein direktes Gateway zum Internet vorliegt, wird stattdessen ein eigener lokaler NTP-Server verwendet. Dieser befindet sich auf einem zusätzlichen Raspberry Pi, dem *Distributor (dist)*.
Alle Raspberry Pis sind so konfiguriert, dass sie ihre Systemzeit sowohl beim Systemstart als auch beim Start der Statemachine und darüber hinaus beim Aufnahmestart sowie alle 10 Minuten bei ausgeführtem Remote-Skript sich mit dem Distributor synchronisieren. Da aber auch der Distributor seine Systemzeit im ausgeschalteten Zustand nicht weiterführt, wird ihm beim Starten des Capture Controllers einmalig die Systemzeit des Gerätes übermittelt, welches den Capture Controller ausführt. So wird zur Laufzeit dafür gesorgt, dass alle Geräte im System nahezu die gleiche Systemzeit haben und Aufnahmen von Sessions und Stills synchron starten.
Damit der Startvorgang nicht durch eine in der Zukunft liegende relative Zeitverschiebung bestimmt wird, werden die möglichen Startpunkte nur in 5-Sekundenschritten einer Minute vorgegeben. Zudem muss der Start mindestens 5 Sekunden in der Zukunft liegen. Beide Regeln führen dazu, dass die Startzeit mindestens 5 Sekunden und maximal 10 Sekunden in der Zukunft liegt. Innerhalb der 5 Sekunden Mindeststartverzögerung ist es dem System so möglich, alle notwendigen Aufnahmeeinstellungen zu laden und vorzunehmen. Durch das separate Starten des Encoders vor dem eigentlichen Aufnahmestart, lasst sich das gesamte System obendrein so weit vorbereiten, dass bei Erreichen der Startzeit die Frames umgehend aufgenommen und verarbeitet werden können. Dies führt zu einem framegenauen Start der Aufzeichnung von Sessions und Stills.

Framedrop-Erkennung

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Während der Testaufnahmen zur Synchronisation wurde festgestellt, dass wiederholt Bildsprünge im Videomaterial auftreten, ein Phänomen, das als *Dropped Frames* bezeichnet wird. Dropped Frames bezeichnen Lücken oder Sprünge im Bildfluss, die dadurch entstehen, dass einzelne Bilder vom Encoder nicht erfasst oder verarbeitet werden.
Da diese Dropped Frames zufällig auftreten und sich nicht eindeutig einzelnen Raspberry Pis, der Aufnahmedauer oder anderen reproduzierbaren Faktoren zuordnen lassen, liegt die Vermutung nahe, dass sie entweder auf Hardwareprobleme des Kameramoduls oder auf Interferenzen während der Datenübertragung vom Kameramodul zum Raspberry Pi zurückzuführen sind. Die Behebung beider Ursachen hätte einen erheblichen finanziellen oder zeitlichen Aufwand erfordert, der im fortgeschrittenen Projektstadium nicht mehr realisierbar war. Daher wurde entschieden, dass eine Vermeidung von Dropped Frames aktuell nicht möglich ist. Allerdings besteht die Möglichkeit, diese Frames zu erfassen und in der anschließenden Bildverarbeitung zu berücksichtigen.
Die Picamera2 Library stellt hierfür die Methode `capture_metadata` zur Verfügung, die neben weiteren Metadaten unter anderem den `sensor_timestamp` liefert. Dieser Zeitstempel gibt nicht eine absolute Zeit an, sondern die verstrichene Zeit von Systemstart bis zum ersten geschriebenen Pixel des jeweiligen Frames in Nanosekunden. `capture_metadata` wartet dabei immer auf den nächsten eingehenden Frame und ist somit *blocking*. (Raspberry Pi Foundation 2025e)
Für die Erkennung von Dropped Frames werden die eintreffenden Frames fortlaufend nummeriert und zusammen mit ihrem Zeitstempel in einer Liste gespeichert. Nach dem Empfang jedes Frames wird der aktuelle `sensor_timestamp` mit dem des vorhergehenden Frames verglichen. Dabei fließt ein Toleranzfaktor in die Berechnung ein, um möglichen *Jitter* in der Framerate zu berücksichtigen. Überschreitet die Differenz den erwarteten Zeitraum eines einzelnen Frames,

```
Initialize an empty dictionary frame_timestamps
Initialize an empty list missing_frames
Set frame_number to 0

While recording is active:
    Get the timestamp of the current frame
    Get the timestamp of the previous frame
    Compute the expected frame time (1 / fps)
    Compute the actual timestamp interval

    If interval is within an acceptable range:
        Store the current timestamp
    Else:
        Calculate the number of dropped frames
        Store missing frame indices
        Adjust frame_number accordingly
        Store the current timestamp

    Increment frame_number
```

Code 1: Pseudocode für Dropped Frame Detection

abhängig von der eingestellten Framerate, so wird dies als Hinweis auf mindestens einen Dropped Frame gewertet. Die tatsächliche Anzahl der fehlenden Frames wird ermittelt, indem die Differenz durch die erwartete Framezeit dividiert und auf das nächste Vielfache gerundet wird. Diese Anzahl wird in der Frame-Liste als Lücke vermerkt und zur aktuellen Framezahl addiert. Der Prozess wiederholt sich für jeden eintreffenden Frame und endet erst mit Abschluss der Aufnahme. Anschließend werden die Positionen der Dropped Frames in der Liste zusätzlich mit dem Label `dropped` markiert und die gesamte Liste als JSON-Datei abgespeichert.
Diese JSON-Datei wird zusammen mit der Videodatei von jeder Kamera bereitgestellt, sodass in der weiteren Verarbeitung flexibel entschieden werden kann, wie mit den Dropped Frames umgegangen wird. Abhängig von den jeweiligen Anforderungen können diese Frames entweder vollständig ignoriert oder mittels spezifischer, mehr oder weniger komplexer Rekonstruktionsverfahren wiederhergestellt werden. Seitens des Volumetric Capture Systems werden hierzu jedoch keine Vorgaben gemacht.

Automatisierung

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Da sich das Remote-Skript auf den Agents nach der Aufnahme einer Session oder eines Stills automatisch in den Standby-Status zurücksetzt, kann das Skript in der Theorie durchgehend aktiv sein und könnte der Einfachheit halber in den Autostart der Raspberry Pis integriert werden. Im Entwicklungsstadium des Systems ist dies jedoch aus mehreren Gründen unpraktikabel. Einerseits müssen die Remote-Skripte nach Änderungen im Code regelmäßig aktualisiert und neu gestartet werden, andererseits müssen Aktualisierungen stets auf allen 68 Raspberry Pis einheitlich durchgeführt werden, um die Änderungen systemweit anzuwenden.
Um den Prozess so zeiteffizient wie möglich durchzuführen, hilft ein eigens entwickelter CLI-Launcher. Dieser Launcher kann sich automatisch über einzelne SSH-Verbindungen mit allen in der IP-Liste aufgeführten Raspberry Pis verbinden. Er ermöglicht das Starten, Stoppen und Aktualisieren beliebiger Skripte sowie das Neustarten aller Raspberry Pis. Befehle, die einmalig im Launcher ausgeführt werden, werden in der Regel innerhalb von ca. drei Sekunden auf allen Raspberry Pis umgesetzt. Eine erhebliche Beschleunigung wird dabei durch Parallelisierung mithilfe eines Python-Threadpools erreicht. Die Anzahl der dort verwendeten sogenannten *Worker* wird dabei automatisch auf die doppelte Anzahl der verfügbaren CPU-Kerne des Host-Computers gesetzt, was eine signifikante Zeitersparnis beim Ausführen der Befehle ermöglicht.
Um die Nutzung des Systems weiter zu vereinfachen, wurde dieselbe Methode zum Starten und Stoppen der Remote-Skripte zusätzlich in alle Master-Skripte integriert, darunter den Capture Controller aber auch in den nachfolgend beschriebenen Camera Controller und Download Manager. Sobald eines der Master-Skripte auf dem Host-Computer startet, werden automatisch das entsprechende Remote-Skript auf allen Raspberry Pis mitgestartet. Andersherum werden beim Schließen eines Master-Skripts zunächst alle Remote-Skripte auf den Agents beendet, bevor der Master-Skript auf dem Host-Computer geschlossen wird.
Ein weiteres wichtiges Merkmal des automatisierten Starts der Remote-Skripte durch den Capture Controller besteht darin, dass die IP-Adresse des Host-Computers beim Starten als Argument an die Remote-Skripte übergeben wird. Dies dient dazu den Agents beim Start mitzuteilen,

an welche IP-Adresse sie Antwortnachrichten an den Capture Controller senden sollen, selbst wenn den Host-Computer Host-Computer zuvor eine zufällige IP-Adresse dynamisch per DHCP vom Router zugewiesen wurde.

4.1.4 Kamerasteuerung und Kalibrierung

Wie bereits zu Beginn des Abschnitts 4.1 erwähnt, besteht die Capture Software aus insgesamt drei unabhängig voneinander agierenden Programmen. Neben dem in Abschnitt 4.1.3 beschriebenen Capture Controller, welcher in der Lage ist, Sessions und Stills aufzunehmen, ist ein dedizierter *Camera Controller* dafür zuständig, eine einheitliche Kamerakalibrierung systemweit zu gewährleisten. Das Steuerungskonzept dieses Programms sowie die einstellbaren Parameter werden im folgenden Abschnitt beschrieben.

Steuerungs- und Kommunikationskonzept

Beteiligte Teammitglieder: Dennis Amuser

Das Ziel des Camera Controller ist es, Kameraeinstellungen festzulegen, welche systemweit auf allen Raspberry Pis identisch angewendet werden, und dabei eine individuelle Konfiguration so einfach wie möglich zu gestalten. Dies wird durch eine Benutzeroberfläche mit Eingabe- du Auswahlmöglichkeiten sowie Schiebereglern für verschiedene Kameraparameter inklusive einer Live-Ansicht des Kamerabildes eines Raspberry Pis ermöglicht, welche den direkten Einfluss der Parameter auf das Kamerabild zeigt. Auch hierbei wird ein Master-Remote-Konzept verfolgt. Wie zuvor beim Capture Controller werden vom Camera Controller Nachrichten mit Konfigurationsbefehlen an die Agents gesendet, wo diese lokal umgesetzt werden. Der Austausch von Nachrichten, Befehlen und der Live-Kameraansicht erfolgt hierbei per Hypertext Transfer Protocol (HTTP). Die Realisierung erfolgt durch das Web-Application-Framework Flask. Jeder Agent fungiert hierbei als Server eines dedizierten Webservers. Der Grund für die Verwendung von Flask liegt an der Einfachheit der Übertragung einer Live-Ansicht des Kamerabildes als JPEG-Bildsequenz über eine HTTP-Route. In der späteren Nutzung ist es zudem sehr einfach möglich, zwischen verschiedenen Live-Ansichten zu wechseln, da jeder Agent eine Live-Ansicht bereitstellt. Die Konfigurationsbefehle werden mittels eines HTTP POST Requests an die Agents übermittelt. Zur Kontrolle der aktuellen Einstellungen sowie zur Systemüberwachung können zudem bestimmte Werte per HTTP GET Request vom Camera Controller von den Agents abgefragt werden.

Benutzeroberfläche

Beteiligte Teammitglieder: Dennis Amuser

Die Benutzeroberfläche des Camera Controller zielt darauf ab, eine möglichst intuitive Konfiguration der Kameraparameter des Capture Systems zu ermöglichen. Dazu steht erneut eine Benutzeroberfläche zur Verfügung, welche mittels Tkinter in Python realisiert ist und in Abbildung 19 dargestellt ist. Zunächst wird eine Kamera mittels ihrer ID definiert, welche für die Live-Ansicht selektiert wird und auf die die eingestellten Kameraparameter zunächst temporär angewendet werden. Im weiteren Verlauf können verschiedene Parameter eingestellt werden.

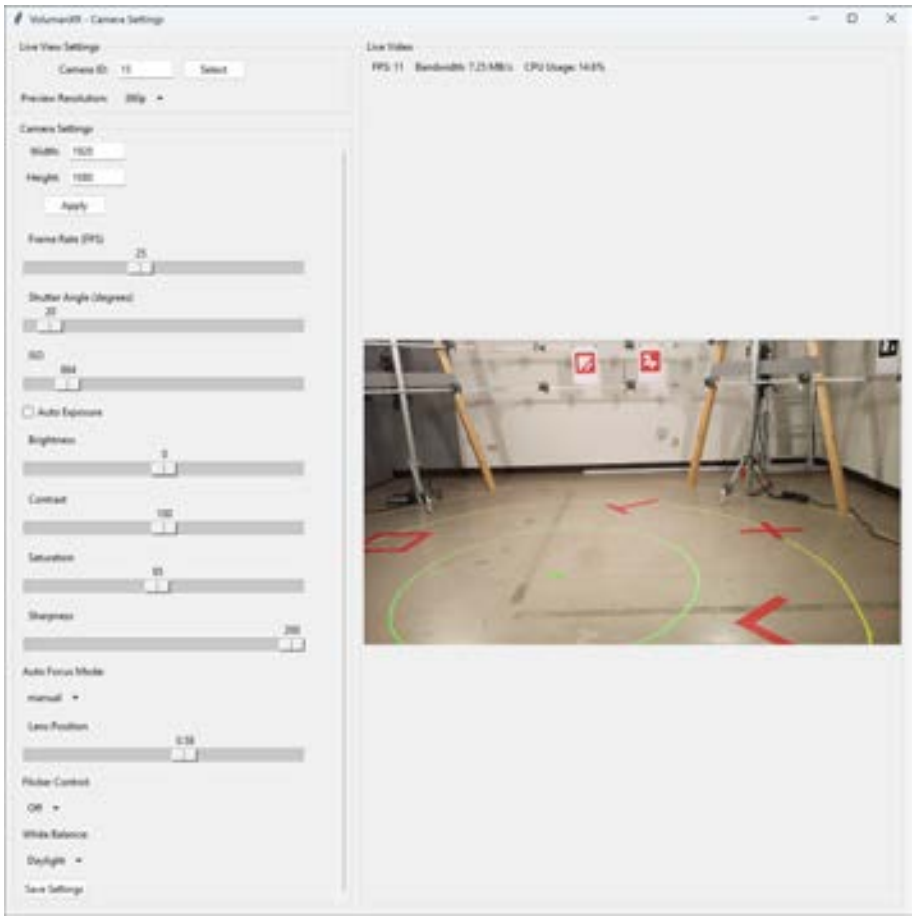


Abbildung 19: Die Benutzeroberfläche des Camera Controllers

Zu den konfigurierbaren Parametern gehören unter anderem die Auflösung, die Framerate, die Belichtungszeit und der ISO-Wert. Darüber hinaus lassen sich auch weitere bildbeeinflussende Parameter wie Helligkeit, Kontrast, Sättigung, Schärfe und Weißabgleich anpassen. Darüber hinaus lassen sich auch der Fokus der Linse sowie eine Flimmerreduktion konfigurieren. Die so konfigurierten Parameter können schließlich über einen Button auf den Agents gespeichert werden. Dieser Vorgang speichert die in der Benutzeroberfläche definierten Parameter in der `camera_settings.json`-Datei auf allen Agents. In der Folge werden die Kameraeinstellungen beim Erstellen von Aufnahmen durch den Capture Controller einheitlich über die jeweilige `camera_settings.json`-Datei angewendet. Im folgenden Abschnitt werden die einzustellenden Kameraparameter nochmals genauer erläutert.

Kameraparameter

Beteiligte Teammitglieder: Kai Altwicker, Dennis Amuser

Der Einsatz in einem Multikamera-Rig stellt besondere Anforderungen an die Belichtungssteuerung und weitere Kameraparameter. Im Folgenden werden die wichtigsten Parameter näher erläutert.

- **Belichtungszeit:** Um Bewegungsunschärfe bei bewegten Objekten zu vermeiden, muss die Belichtungszeit so gewählt werden, dass schnelle Bewegungen, wie bei menschlichen

Aktivitäten, scharf abgebildet werden. Erfahrungswerte liegen hierbei im Bereich von ca. 1/250 bis 1/500s. Um auf der sicheren Seite zu sein, wurde die Belichtungszeit auf etwa 1/500s festgelegt. In der Datei `camera_Settings.json` wird dieser Wert durch den Parameter `"shutter_angle": 20` definiert. Dies entspricht bei 25fps einer Belichtungszeit von ca. 1/450s und bei 30fps etwa 1/540s.
Ein Sonderfall besteht bei der Aufnahme von 4K-Fotos: Bei Auflösungen bis einschließlich FullHD arbeitet der Sensor in einem Modus, in dem durch Supersampling Subpixelwerte aufaddiert werden. Im 4K/UHD-Modus reduziert sich daher bei identischen Einstellungen die Belichtung um eine Blendenstufe. Um diesen Verlust auszugleichen, wird bei UHD-Still-Aufnahmen im Remoteskript automatisch die halbe, ursprünglich eingestellte Belichtungszeit verwendet.

- **Gain:** Da die Kameraobjektive über eine fixe Blende verfügen und die Wahl der Belichtungszeit aufgrund der Vermeidung von Bewegungsunschärfe stark eingeschränkt ist, verbleibt als einziger Parameter zur Steuerung der Belichtung die Sensorverstärkung. Diese Verstärkung, in den Kameraeinstellungen als `iso` definiert, beeinflusst direkt das Bildrauschen. Es muss daher ein Kompromiss gefunden werden, der ausreichende Helligkeit bei akzeptablem Rauschpegel gewährleistet. Zudem wird die automatische Belichtungssteuerung durch den Parameter `"auto_exposure": false` deaktiviert, um eine einheitliche Belichtung über alle Kameras sicherzustellen.

- **Weißabgleich:** Die Picamera2 Library unterstützt, basierend auf der Raspberry internen libcamera Bibliothek, verschiedene Weißabgleich-Modi, die in zwei grobe Kategorien unterteilt werden können:

  - *Automatikmodi:* Der Modus `Auto` berechnet anhand eines mittleren Grauwerts den optimalen Weißabgleich. Alternativ übernehmen Lichtpresets wie `Incandescent, Tungsten , Daylight` oder `Cloudy` interne Voreinstellungen, die den Weißabgleich der jeweiligen Lichtfarbe anpassen.
  - *Manuelle Modi:* Bei der Auswahl von `Manuell` oder der expliziten Angabe von Farbtemperaturen (z.B. `3200K, 4400K, 5600K`) wird der Weißabgleich manuell gesteuert. Dabei erfolgt die Einstellung über das Verhältnis von Rot- und Blauverstärkung. Im manuellen Modus können diese Werte frei gewählt werden, für die Temperaturpresets werden die zugehörigen Werte der Colour Temperature Curve (CT-Curve) (Abbildung 20) gewählt.

  Sollte mit dem Raumlicht gearbeitet werden, wird der Weißabgleich auf `Auto` gestellt, da die exakte Farbtemperatur der Deckenbeleuchtung oftmals nicht bekannt ist. Wird hingegen ausschließlich mit der internen Rig-Beleuchtung operiert, empfiehlt sich der Modus `Daylight`. Obwohl dies dem manuellen Weißabgleich von `5600K` entsprechen sollte, ergaben Tests, dass die Verwendung des Lichtpresets im Automatikmodus bessere Ergebnisse liefert. (Raspberry Pi Foundation 2025f)

- **Weitere Parameter:** Neben den oben beschriebenen Hauptparametern besteht die Möglichkeit, Helligkeit, Kontrast, Sättigung und digitale Nachschärfung einzustellen. Diese

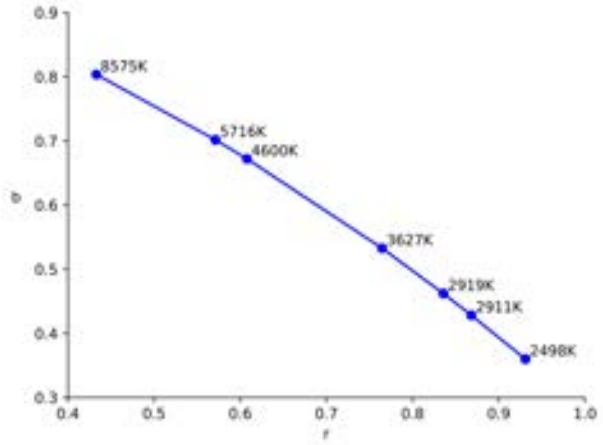


Abbildung 20: Raspberry Pi CT-Curve

wurden auf Basis subjektiver visueller Beurteilungen auf Standardwerte kalibriert. Darüber hinaus bietet das System Optionen zur automatischen Flickerreduzierung sowie verschiedene Autofokusmodi.

Diese umfassende Parametereinstellung trägt dazu bei, eine konsistente und qualitativ hochwertige Aufnahme im Kamerarig zu gewährleisten.

Individuelle Konfiguration

Beteiligte Teammitglieder: Kai Altwicker

Obwohl eine individuelle Kalibrierung jeder Kamera eine weitere Qualitätssteigerung ermöglichen würde, wurde aus Zeitgründen darauf verzichtet, da die allgemeine Kalibrierung bereits zufriedenstellende Ergebnisse liefert. Allerdings zeigen die Aufnahmen, dass insbesondere die Fokussierung von Kamera zu Kamera variiert. In Einzelfällen führen diese Abweichungen zu deutlich unscharfen Bildern. Die Raspberry Pi Foundation weist darauf hin, dass die Kameras werkseitig nur grob kalibriert sind und die angegebenen Fokuswerte lediglich als Richtwerte zu verstehen sind.
Zur Optimierung der Fokuseinstellungen wurde für jede Kamera eine Fokusreihe unter Verwendung eines Siemenssterns aufgenommen. Anschließend erfolgte die Bestimmung der optimalen Fokusposition durch visuelle Begutachtung. Die ermittelten Fokuswerte werden als zusätzlicher Parameter in der zentral verwalteten `camera_list.json` hinterlegt.
Abbildung 21 verdeutlicht eine klare Abhängigkeit der Fokusposition von der vertikalen Ausrichtung der Kamera: Je stärker die Kamera nach unten geneigt ist (rot und blau), desto niedriger ist der Fokuswert; bei einer Neigung nach oben (lila) zeigt sich der entgegengesetzte Trend. Dies legt nahe, dass das freischwebende Autofokuselement maßgeblich durch die Schwerkraft beeinflusst wird. Zudem wurde festgestellt, dass selbst bei Kameras mit identischer Ausrichtung signifikante Produktionsschwankungen auftreten, sodass eine einheitliche Kalibrierung in diesem Fall nicht zu einem akzeptablen Ergebnis führen würde.
Die nach Projektende angebrachten Deckenkameras ließen überhaupt keine zufriedenstellende Fokussierung zu, da der verwendete Fokusmotor in dieser Position zu schwach war, um das Fokuselement in eine korrekte Position zu bringen. Daher wurden diese Kameras aus der vor-

angeangenen Analyse ausgeklammert. Für eine zukünftige Nutzung müsste die Montage dieser Kameras noch einmal überarbeitet werden, dies war allerdings nicht mehr im Projektzeitraum möglich. Von einer Nutzung der vier Deckenkameras wird im aktuellen Zustand expliziet abgeraten, da nur unscharfe Bilder produziert werden können.

Abbildung 21: Verteilung Fokuswerte

4.1.5 Dateimanagement

Der *Download-Manager* stellt das dritte Teilprogramm der Capture Software dar. Die Aufgabe des Download-Managers besteht in der Verwaltung der aufgenommenen Dateien des Systems. Mittels des Download-Managers können aufgenommene Sessions und Stills zentral verwaltet werden. Wie die Verwaltung stattfindet, wird im folgenden Abschnitt beschrieben. Aus Gründen der Einfachheit werden Sessions und Stills folgend nur noch zusammengefasst als *Sessions* bezeichnet.

Dateitransfer und -management Konzept

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Der Download-Manager für die zentrale Verwaltung für Sessions füllt eine wichtige Lücke der Capture Software. Auch dieses Programm bestehlt zwei Python-Skripten, bestehend aus einem Master-Skript, welches auf dem Host-Computer, und einem Remote-Skript, welches auf den Agents ausgeführt wird. Camera und Capture Controller ermöglichen es, einheitlich kalibrierte Aufnahmen zu erzeugen. Zusammengehörige Aufnahmen werden jedoch zunächst dezentral gespeichert, denn die Agents speichern ihre jeweilige Aufnahme erst einmal lokal. Anschließend muss die entsprechende Session aber von jedem Agent heruntergeladen werden. Die Aufgabe des Download-Managers besteht darin, die erstellten Aufnahmen in einem Programm auf dem Host-Computer zentral zu verwalten. Er verbindet sich dazu mit allen Agents und lädt vom Nutzer ausgewählte Aufnahmen gesammelt herunter. Für die Verwaltung wird erneut eine Benutzeroberfläche bereitgestellt. Diese bietet eine Übersicht über zum Download verfügbare Aufnahmen

inklusive weiteren Informationen auf den Agents und kontrolliert, ob diese bereits heruntergeladen wurden. Zusätzlich lässt sich Speicherplatz lokal auf dem Host-Computer und remote auf den Agents durch Löschen der Sessions freigeben. Als weitere Ergänzung ist es möglich über den Download-Manager bereits heruntergeladen Sessions, die zunächst in Form von Videodateien vorhanden sind, in Framesequenzen zu konvertieren. Diese Konvertierung berücksichtigt die Ergebnisse der zuvor durchgeführten Framedrop-Analyse, um gewährleisten zu können, dass auch ohne besonderes Hintergrundwissen über die Handhabung der Framedrops in den Videodateien durch eine eigene Auswertung der der Videodateien gleichnamigen JSON-Dateien brauchbare Datensätze generiert werden können.

Benutzeroberfläche

Beteiligte Teammitglieder: Dennis Amuser, Kai Altwicker

Die Benutzeroberfläche des Download-Managers ist wie die anderen Teilprogramme mit Tkinter realisiert und ist in Abbildung 22 dargestellt. Im oberen Teil der Benutzeroberfläche besteht die Möglichkeit, einen individuellen Zielordner für die herunterzuladenden Aufnahmen auf dem Host-Computer zu definieren. In dem darunterliegenden Listenansicht werden alle auf den Agents zum Download verfügbaren Sessions aufgelistet. Dazu fragt der Download-Manager automatisch die im Recordings Ordner befindlichen Dateien der Agents ab. Hierbei werde Sessions hingegen nur angezeigt, solange sie auf den Agents gefunden werden. Befinden Dateien sich lediglich im Zielordner jedoch aber nicht mehr auf den Agents, werden diese nicht gelistet.
Obwohl die einzelnen Dabeinamen auf den Agents immer aus dem definierten Sessionnamen einem IP-Suffix bestehen, werden in der Listenansicht stets nur die Sessionnamen angezeigt. Zusammengehörige Sessions werden automatisch erkannt und zusammengefasst. Gleichzeitig wird über die IP-Liste kontrolliert, ob auf den Agents gefundene Sessions auch auf allen anderen Agents vorhanden sind. Ist dies bei ein oder mehreren Agents nicht der Fall, werden diese in

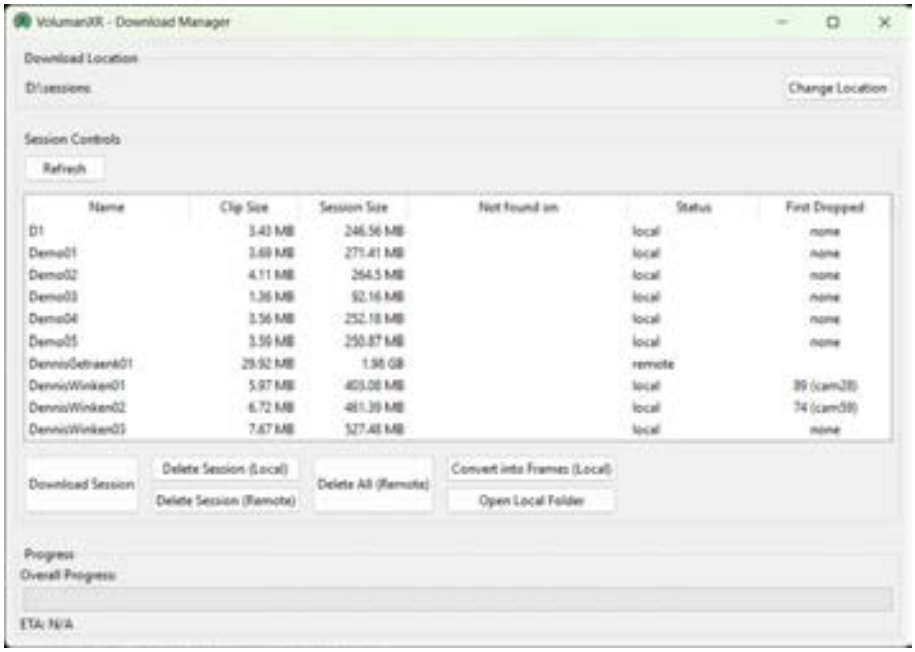


Abbildung 22: Die Benutzeroberfläche des Download Managers

der Spalte *Not found on* angezeigt. Die Spalten *Clip Size* und *Session Size* zeigen Informationen Einzel- und Gesamtgröße einer Aufzeichnung an. In der Spalte Status werden Informationen über den aktuellen Downloadstatus angezeigt. Der Status *remote* einer Session sagt aus, dass sie sich ausschließlich auf den Agents befindet und nicht lokal heruntergeladen wurde. Der Status *local* zeigt dagegen an, dass eine Session bereits heruntergeladen wurde und im Zielordner vorhanden ist. Ist sie lokal jedoch unvollständig, bekommt die Session den Status *local (incomplete)*. Die letzte Spalte der Listenansicht gibt letztlich eine kleine Einsicht und in die Framedrop-Analyse und zeigt die Dauer bis zum ersten getoppten Frame in einer Session an.
In der Listenansicht lassen sich einzelne Sessions selektieren. Über die unter der Liste befindlichen Buttons können Sessions schließlich heruntergeladen, lokal oder remote auf den Agents gelöscht, oder bereits heruntergeladenen Sessions von Videos in Frames konvertiert werden. Die Konvertierung findet dabei automatisiert als Subprozess durch FFmpeg statt.
Zu guter Letzt visualisiert eine Fortschrittsanzeige den aktuellen Downloadstatus einer Session inklusive einer noch verbleibenden Downloadzeit.
Weil die Konvertierung in Frames so implementiert ist, dass nur lokale Sessions konvertiert werden können, diese Sessions aber nur angezeigt werden, wenn sie der Download-Manager gleichzeitig auf den Agents findet, kommt es zu einem Problem, wenn Sessions nachträglich in Frames konvertiert werden sollen, ohne dass Verbindung zu den Agents besteht. Der Download-Manager würde bereits lokal vorhandene Sessions ohne eine Verbindung zu den Agents nicht auflisten. Aus diesem Grund besteht die Möglichkeit, den Download-Manager mit dem zusätzlichen Argument `offline` zu starten. Dadurch startet der Download-Manager in einem speziellen *Offline Mode* welcher dieses Problem behebt. Im Offline Mode setzt der Download-Manager keine Verbindung zu den Agents voraus. Er wird lediglich in einem reduzierten Umfang gestartet. Im reduzierten Umfang zeigt die Listenansicht alle bereits lokalen Sessions im Zielverzeichnis an und vergibt ihnen den Status *local only*. So können einzelne Sessions ausgewählt werden, um die Konvertierung in Frames unter der Berücksichtigung der Framedrop-Analyse nachträglich und ohne der Verbindungen zu den Agents durchzuführen. Im Offline Mode nicht funktionale Buttons der Benutzeroberfläche sind zusätzlich deaktiviert.

4.2 Training

Der folgende Abschnitt beschreibt die Umsetzung der Erstellung einer 3D-Szene aus den gegebenen Kamerabildern mithilfe von Spacetime Gaussians. Dabei gliedert sich der Prozess in zwei Hauptabschnitte: das Preprocessing (Abschnitt 4.2.1), in welchem die Videos vorverarbeitet werden und das Training der Szene (Abschnitt 4.2.2), in welchem aus den vorverarbeiteten Daten die finale Szene erstellt wird.

4.2.1 Preprocessing

Beteiligte Teammitglieder: David Mertens, Matthias Bullert

Parameter	Standard Wert	Beschreibung
–imageext	png	Bildformat
–videosdir		Pfad zum Ordner mit .mp4 Dateien
–startframe	0	Erster Frame des Preprocessing
–endframe	50	Letzter Frame des Preprocessing
–removebg	False	Hitnergrund Entfernung (True/False)
–stripFeatures	False	Feature Entfernung (True/False)
–select_ref_mode	file	Auswahl des Referenz Frame Modus
–showStripping	False	Anzeige der entfernten Feature punkte

Tabelle 3: Kommandozeilenparameter für das Preprocessing

Das Preprocessing umfasst die Aufbereitung der Videodateien als Grundlage für das Training. Die Eingabe der Preprocessing-Pipeline besteht aus einer variablen Anzahl an .mp4-Videodateien, die systematisch verarbeitet werden.
Der Output ist eine strukturierte Datenablage, in der für jeden einzelnen Frame alle Bilder der Kameras extrahiert und gespeichert sind. Zusätzlich werden die zugehörigen Kamerapositionen sowie die relevanten Feature-Punkte ermittelt und hinterlegt. Ein weiterer Bestandteil des Preprocessings ist die Hintergrundentfernung, die auf die extrahierten Bilder angewendet wird, um eine optimierte Datengrundlage für das Training zu schaffen.
Das Ergebnis des Preprocessings stellt den direkten Input für den Voluman Loader dar und bildet somit die Grundlage für das eigentliche Training.

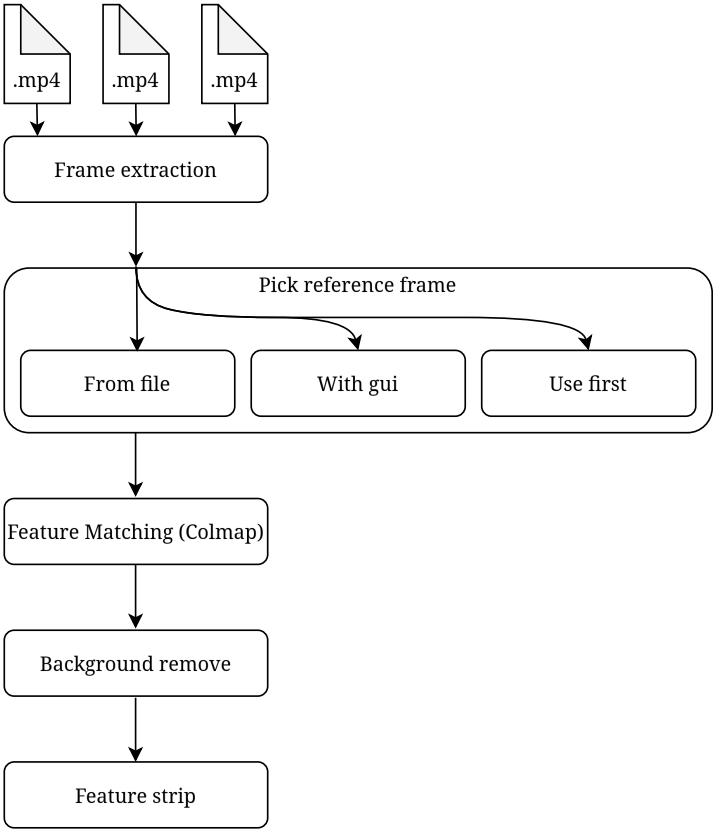


Abbildung 23: Preprocessing Pipeline

Die Datei pre_voluman ist für das Preprocessing zuständig und befindet sich im Ordner (VolumanXR/training/SpacetimeGaussians/scripts).

Referenz Frame

Beteiligte Teammitglieder: David Mertens

Auf Basis der extrahierten Frames können die Kamerapositionen sowie die Feature-Punkte der Bilder mit COLMAP berechnet werden. Die Bestimmung der Kamerapositionen wird durch die im Kamerarig angebrachten Marker zwar zuverlässiger, bleibt jedoch nicht vollkommen präzise. In der Abbildung 24 sind die Auswirkungen einer fehlerhaften COLMAP-Analyse zu erkennen. Da das Kamerarig symmetrisch aufgebaut ist und sich im verwendeten Testraum auf zwei Seiten weiße Wände befinden, ordnet COLMAP die Kamerapositionen häufig spiegelverkehrt an. Die verwendeten Marker helfen, dieses Problem zu reduzieren, können es jedoch nicht vollständig verhindern.

Abbildung 24

Um dieses Problem weiter zu minimieren, wurde das Referenzframe-System eingeführt. Anstatt sich ausschließlich auf die von COLMAP berechneten Kamerapositionen zu verlassen, wird ein einzelner Referenz-Frame bestimmt, dessen extrahierte Kameraposition als Fixpunkt dient. Die Kamerapositionen der übrigen Frames werden von diesem Referenz-Frame übernommen, wodurch inkonsistente oder spiegelverkehrte Positionierungen vermieden werden können.
Wie dieser Frame ausgewählt wird, kann von der Nutzer*in über den Kommandozeilenparameter `-select_ ref_mode` festgelegt werden.
Die möglichen Werte für den Kommandozeilenparameter sind: `first`, `interactive` und `file`.
Wird der Wert *first* gesetzt, wird der erste im Preprocessing extrahierte Frame als Referenz-Frame verwendet. Dies entspricht der ursprünglichen Methode im Spacetime Gaussian Preprocessing.
Diese Option ermöglicht eine schnelle Verarbeitung, insbesondere bei Videos, die nicht anfällig für Fehlinterpretationen sind, da keine weitere Interaktion erforderlich ist. Allerdings sollte beachtet werden, dass fehlerhafte Kamerapositionen erst während oder nach dem Training anhand unzureichender Ergebnisse und sichtbarer Artefakte erkennbar werden.
Wird der Wert `interactive` gewählt, werden für alle Frames im angegebenen Frameabschnitt die Kamerapositionen berechnet. Anschließend werden diese in einer grafischen Benutzeroberfläche dargestellt, sodass die Nutzer*in die berechneten Kamerapositionen visuell überprüfen kann.
Mit den Tasten `'a'` und `'d'` kann durch die verschiedenen Frames navigiert werden, während die 3D-Repräsentation die ermittelten Kamerapositionen darstellt. Die Maus ermöglicht eine Rotation um den Mittelpunkt der Kameras, um eine genauere Beurteilung der berechneten Positionen vorzunehmen.

Ein Frame kann durch Drücken der Taste `'g'` als Referenz-Frame markiert werden. Mit `'h'` lässt sich die extrahierte Kameraposition als Binärdatei speichern. Die GUI kann mit `'q'` verlassen werden, woraufhin die Feature-Punkte aller Frames basierend auf dem ausgewählten Referenz-Frame extrahiert werden.
Die Visualisierung der interaktiven Referenzrahmen-Auswahl erfolgt in der Datei *CameraPos-Vos*mithilfe des Python-Moduls `matplotlib`. Das Programm beginnt mit der Erstellung einer Instanz, der anschließend die aus `Colmap` extrahierten Binärdateien übergeben werden.
Im nächsten Schritt werden die Kamerapositionen und -rotationen aus den Binärdateien extrahiert. Die Positionen werden als Translationsvektoren (tx, ty, tz) und die Rotationen als Quaternionen (qw, qx, qy, qz) gespeichert. Diese Daten werden in einem Dictionary organisiert, das eine eindeutige Zuordnung der `CameraID` zu den entsprechenden Positions- und Rotationswerten ermöglicht.
Die im vorherigen Modus erstellten Binärdaten können jederzeit erneut verwendet werden. Wurde einmal die korrekte Kameraausrichtung bestimmt, lässt sie sich durch Setzen des Parameters `-select_ref_mode` auf `file` wiederverwenden. In diesem Fall werden die Kamerapositionen aller Frames durch die gespeicherte Binärdatei überschrieben.

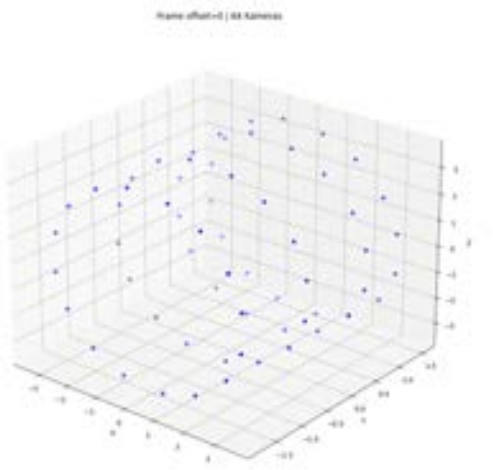


Abbildung 25: Referenzframe Auswahl

Abschließend wird die GUI erstellt, und die Kamerapositionen werden geplottet. Dies ist auch in der Abbildung 25 zu sehen.

Ursprung setzen

Beteiligte Teammitglieder: Matthias Bullert

Bei der Rekonstruktion einer Szene mittels COLMAP werden die Kamerapositionen, -orientierungen und die daraus resultierende Punktwolke in einem lokalen, nicht standardisierten Koordinatensystem bestimmt. Dies führt dazu, dass die Ausrichtung der gesamten Szene nicht mit einer realen, globalen Referenz übereinstimmt. Um eine konsistente und passend ausgerichtete Darstellung zu gewährleisten, ist eine nachträgliche Transformation erforderlich, die die Szene in eine geeignete Lage bringt und der Ursprung der Szene an der Position $(0, 0, 0)^\top$ liegt.
Die Transformation erfolgt in zwei Schritten:

1. Ausrichtung der Kameras: Eine Kamera, welche in der realen Welt parallel zum Boden ausgerichtet ist, wird als Referenz gewählt und ihre Orientierung so angepasst, dass ihre lokalen Achsen mit den Achsen des Weltkoordinatensystem übereinstimmen.

2. Translation der Szene: Die gesamte Szene wird so verschoben, dass der Mittelwert der Kamerapositionen im Ursprung des Koordinatensystems liegt. Danach wird mit einem

festgelegten Offset der Abstand zur Bodenebene justiert.

Die extrinsischen Parameter der Kameras werden durch eine Rotationsmatrix $R_i \in \mathbb{R}^{3\times3}$ und einen Translationsvektor $t_i \in \mathbb{R}^3$ beschrieben. Ziel ist es, die Orientierung der Szene durch eine geeignete Rotationsmatrix R_{align} aus zu richten.
Die gewählte Referenzkamera mit R_{ref} enthält ihre aktuelle z-Achse als Spaltenvektor $\mathbf{z}_{\text{current}}$. Die neue Ausrichtung wird so gewählt, dass diese Achse in Richtung $\mathbf{z}_{\text{target}} = (0, 0, -1)^\top$ zeigt. Der benötigte Rotationswinkel θ_z ergibt sich aus:

$$\cos\theta_z = \mathbf{z}_{\text{current}}^\top \mathbf{z}_{\text{target}}, \quad \mathbf{u}_z = \frac{\mathbf{z}_{\text{current}} \times \mathbf{z}_{\text{target}}}{\|\mathbf{z}_{\text{current}} \times \mathbf{z}_{\text{target}}\|}. \tag{7}$$

Die Rotationsmatrix R_z kann mit der Rodrigues-Formel berechnet werden:

$$R_z = I + \sin\theta_z [\mathbf{u}_z]_\times + (1 - \cos\theta_z)[\mathbf{u}_z]_\times^2, \tag{8}$$

wobei $[\mathbf{u}_z]_\times$ die Kreuzproduktmatrix von $\mathbf{u}_z$ ist.
Nachdem z korrekt ausgerichtet wurde, wird die y-Achse der Kamera so gedreht, dass sie mit $\mathbf{y}_{\text{target}} = (0, 1, 0)^\top$ übereinstimmt. Dazu wird eine weitere Rotation R_y berechnet, die auf die bereits transformierte Kamera angewendet wird:

$$R_y = I + \sin\theta_y [\mathbf{u}_y]_\times + (1 - \cos\theta_y)[\mathbf{u}_y]_\times^2. \tag{9}$$

Die endgültige Transformationsmatrix für die Szene ist dann:

$$R_{\text{align}} = R_y R_z. \tag{10}$$

Die neuen Kamerarotationen ergeben sich zu:

$$R_i' = R_{\text{align}} R_i. \tag{11}$$

Um die Szene um den Ursprung zu zentrieren, wird der Mittelwert der Kamerapositionen berechnet:

$$t_{\text{mean}} = \frac{1}{N} \sum_{i=1}^{N} t_i. \tag{12}$$

Die neue Kamerapositionen werden dann als

$$t_i' = R_{\text{align}}(t_i - t_{\text{mean}}) + (0, y_{\text{offset}}, 0)^\top \tag{13}$$

definiert.
Analog wird die Punktwolke, die durch eine Menge von 3D-Punkten $\mathbf{X}_j \in \mathbb{R}^3$ beschrieben wird, transformiert:

$$\mathbf{X}_j' = R_{\text{align}}(\mathbf{X}_j - t_{\text{mean}}) + (0, y_{\text{offset}}, 0)^\top. \tag{14}$$

Nach Anwendung dieser Transformationen sind die Kamerapositionen und die Punktwolke in einem global konsistenten Koordinatensystem verankert.

Hintergrund Entfernung auf Bildebene

Beteiligte Teammitglieder: David Mertens

Die Hintergrundentfernung erfolgt in drei aufeinander abgestimmten Schritten, die kombiniert werden, um ein optimales Ergebnis zu erzielen. Diese Schritte umfassen zunächst die Entfernung des Hintergrunds auf Bildebene, gefolgt von der Eliminierung überflüssiger Colmap-Features und abschließend der Bereinigung nicht benötigter Gaussians.
Da Colmap für eine präzise Rekonstruktion der Szene möglichst viele Bildinformationen benötigt, wird die Hintergrundentfernung erst nach Abschluss der Colmap-Rekonstruktion durchgeführt. Die Umsetzung erfolgt mithilfe des Python-Moduls 'backgroundremover', das sich im Verzeichnis 'training/SpacetimeGaussian/thirdparty' befindet. Dieses Modul nutzt ein Convolutional Neural Network (CNN), um eine Maske zu generieren, die Vorder- und Hintergrund trennt. Diese einzelnen Schritte kann man auch nochmal in der Abbildung XX erkennen.
Im Rahmen dieses Projekts wurde das Modul optimiert, indem das CNN nicht für jedes einzelne Bild separat geladen wird, sondern für ganze Bild-Batches. Diese Anpassung führte zu einer Laufzeitreduktion von etwa 29 %, was insbesondere aufgrund des hohen Zeitaufwands der Hintergrundentfernung von großer Bedeutung ist. Zudem wurde die Auflösung der generierten Masken erhöht, wodurch sich die Qualität der freigestellten Bilder deutlich verbessert. Diese Hintergrundentfernung lässt sich über den Komandozeilenparameter –removebg (bool) an oder ausschalten.

(a) Maske (b) Cutout

Abbildung 26: Verschiedene Ergebnisse bei Variationen des Datensatzes

(a) Ply Größe: 100Mb (b) Ply Größe: 3Mb

Abbildung 27: Verschiedene Ergebnisse bei Variationen des Datensatzes

Colmap Feature Entfernung

Beteiligte Teammitglieder: David Mertens

Nach der Bildsegmentierung enthalten die von Colmap extrahierten Feature-Punkte teilweise Informationen aus Bildbereichen, die in den freigestellten Bildern nicht mehr vorhanden sind. Obwohl ein Training auch mit diesen überflüssigen Feature-Punkten möglich wäre, kann deren Entfernung zu einer Reduktion der Density Control steps führen. Dies liegt daran, dass die initiale Verteilung der Gaussians näher an dem endzustand ist, da alle Gaussians für den Hintergrund garnicht erst erstellt werden.
Um dieses Ziel zu erreichen, müssen drei Binärdateien ausgelesen und gegebenenfalls bearbeitet werden. Der Prozess beginnt mit dem Einlesen der Datei Cameras.bin, die die Positionen und Orientierungen der einzelnen Kameras enthält. Basierend auf diesen Informationen lässt sich der Mittelpunkt des Kamera-Rigs berechnen.
Anschließend wird die Datei points3D.bin analysiert. Sie enthält alle von Colmap extrahierten Feature-Punkte, einschließlich ihrer 3D-Koordinaten sowie der Bildverweise, die angeben, in welchen Aufnahmen der jeweilige Punkt sichtbar ist. Zusätzlich sind für jedes Bild die entsprechenden 2D-Koordinaten hinterlegt, die die genaue Position des Punktes innerhalb des Bildes bestimmen.
Im eigentlichen Filterungsschritt werden diese Koordinaten mit der zuvor erstellten Maske aus Kapitel 4.2.1 abgeglichen. Punkte die innerhalb des als Hintergrund markierten Bereichs liegen werden und in einer separaten Liste gespeichert. Mithilfe dieser Liste werden die Dateien points3D.bin und images.bin erneut durchlaufen, um überflüssige Feature-Punkte zu entfernen.
Obwohl dieses System bereits gut funktioniert, ist es nicht fehlerfrei, da die Hintergrundentfernung aus Kapitel 4.2.1 nicht immer präzise arbeitet. Um die Robustheit zu erhöhen, erfolgt daher ein zusätzlicher Sicherheitscheck: Der Abstand jedes Feature-Punktes zum Mittelpunkt des Rigs wird überprüft. Überschreitet dieser einen festgelegten Grenzwert, wird der Punkt automatisch entfernt.
Anschließend werden alle Feature-Punkte innerhalb eines definierten Radius um den Ursprung entfernt, um eine konsistente und qualitativ hochwertige Rekonstruktion zu gewährleisten.

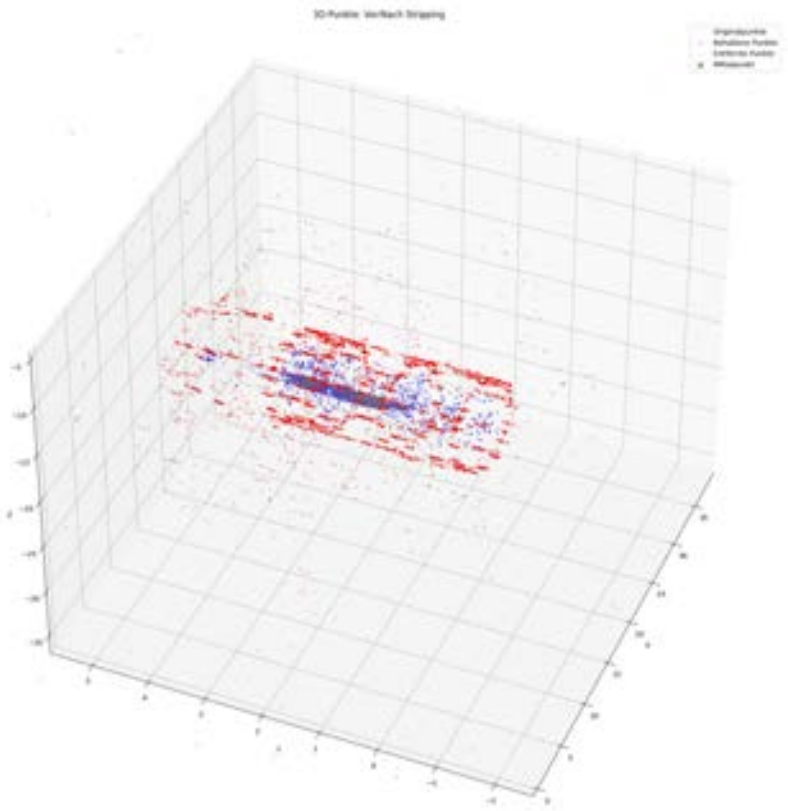


Abbildung 28: Darstellung der entfernten Colmap Features

4.2.2 Szene trainieren

Beteiligte Teammitglieder: Matthias Bullert, David Mertens

Sequenzielles Trainieren

Beteiligte Teammitglieder: David Mertens

Bevor der Trainingsprozess des Spacetime Gaussian beginnt, werden zunächst alle Bilder der Frames in den VRAM geladen. Eine größere VRAM-Kapazität ermöglicht daher das Training längerer Sequenzen. Um auch auf kostengünstiger Hardware längere `.ply`-Szenen erstellen zu können, wurde im Rahmen dieses Projekts das sequenzielle Training eingeführt. Die zentrale Idee besteht darin, einen längeren Frame-Abschnitt in kleinere Sequenzen zu unterteilen, sodass das Training auch mit begrenztem VRAM möglich bleibt.
Zur Steuerung dieses Verfahrens wurden die Kommandozeilenparameter `-slot_size` und `-overlap` implementiert. Die vom Nutzer festgelegte `slot_size` bestimmt die Länge der einzelnen Frame-Abschnitte. Beispielsweise wird eine 50 Frames lange Sequenz bei einer `slot_size` von 10 in fünf Abschnitte unterteilt und separat trainiert.
Um visuelle Artefakte an den Übergängen zwischen den Sequenzen sowie in den Start- und

Endbereichen zu minimieren, kann mit dem Parameter `-overlap` ein Bereich definiert werden, der in zwei aufeinanderfolgenden Abschnitten trainiert wird. Diese Redundanz ermöglicht eine flüssigere Wiedergabe mit weniger Artefakten.

Sparse Tensoren

Beteiligte Teammitglieder: Matthias Bullert

In der Verarbeitung der Bilddaten werden Tensoren als Speicher- und Rechenstruktur genutzt. In der Regel werden Bilder als Dense-Tensoren gespeichert, bei denen jeder Pixelwert explizit repräsentiert wird. Bei Bildern mit vielen schwarzen oder nicht genutzten Pixeln, die durch die Hinterdrundentfernung zustande kommen, können Sparse-Tensoren zu erheblichen Speicher- und Rechenvorteilen führen. Ein Sparse-Tensor reduziert den Speicherverbrauch, indem er nur Pixelwerte speichert, die nicht Null sind und für jedes gespeicherte Pixel zusätzliche Indizes speichert. Die Wahl zwischen Sparse- und Dense-Tensoren hängt demnach vom Verhältnis der Nicht-Null-Pixel zum Gesamtbild ab. Die Herausforderung besteht darin, dynamisch zu entscheiden, wann sich eine Sparse-Darstellung lohnt und wann eine Dense-Darstellung effizienter ist.
Ein RGB-Bild mit Höhe H, Breite W und drei Farbkanälen ($C = 3$) wird als Dense-Tensor mit 32-Bit-Fließkommazahlen (4 Byte pro Wert) gespeichert. Der Speicherverbrauch beträgt:

$$S_{\text{dense}} = H \times W \times C \times 4. \tag{15}$$

Ein Sparse-Tensor speichert für jedes Nicht-Null-Pixel die Position mit zwei 64-Bit-Integer-Werte (8 Byte pro Wert) für die Indizes (h, w) und deren RGb-Werte mit drei 32-Bit-Fließkommazahlen (4 Byte pro Farbkanal). Damit ergibt sich der Speicherverbrauch eines Sparse-Tensors mit N gespeicherten Nicht-Null-Pixeln:

$$S_{\text{sparse}} = N \times (2 \times 8 + 3 \times 4). \tag{16}$$

Die Sparse-Repräsentation ist effizienter als die Dense-Repräsentation, wenn gilt:

$$S_{\text{sparse}} < S_{\text{dense}}. \tag{17}$$

Einsetzen von (1) und (2) ergibt:

$$N \times (2 \times 8 + 3 \times 4) < H \times W \times 3 \times 4. \tag{18}$$

Durch Umformen nach $N/(H \times W)$ erhält man das kritische Verhältnis:

$$\frac{N}{H \times W} < \frac{3 \times 4}{2 \times 8 + 3 \times 4}. \tag{19}$$

Für ein RGB-Bild ($C = 3$) ergibt sich:

$$\frac{3 \times 4}{2 \times 8 + 3 \times 4} = \frac{12}{16 + 12} = \frac{12}{28} \approx 0.43. \tag{20}$$

Das bedeutet, dass ein Sparse-Tensor vorteilhaft ist, wenn weniger als 43% der Pixel Nicht-Null-Werte enthalten. Andernfalls ist ein Dense-Tensor speichereffizienter.
Des Weiteren wurde eine neue Methode zur Berechnung des SSIM implementiert, die zur Fehlerermittlung benötigt wird, da PyTorch standardmäßig nur eine SSIM-Berechnung für Dense-Tensoren anbietet. Diese Methode ermöglicht die Berechnung des SSIM zwischen einem Sparse- und einem Dense-Tensor, indem sie gezielt nur die nicht-null Werte des Sparse-Tensors berücksichtigt.

Gaussian Entfernung

Beteiligte Teammitglieder: Matthias Bullert

Obwohl sowohl die Hintergrundentfernung auf Bildebene als auch die Bereinigung der Colmap-Features bereits zu guten Ergebnissen führen, können während des Trainings weiterhin Hintergrundartefakte auftreten. Diese entstehen durch leere Bereiche, die in Form von schwarzen Gaussians dargestellt werden. Ein Beispiel für solche Artefakte ist in der Abb. 4.2.2 zu sehen.

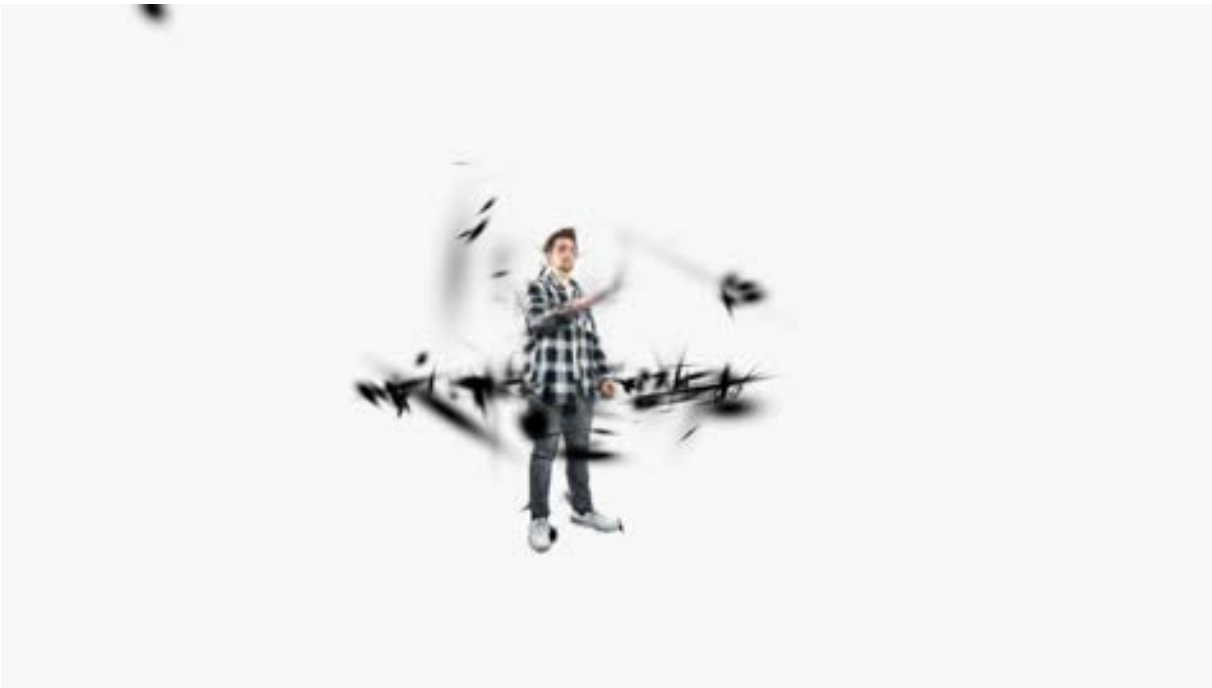


Abbildung 29: Darstellung der trainierten Szene mit überflüssigen schwarzen Gaussians

Um diese unerwünschten Gaussians automatisiert zu identifizieren und zu entfernen, wird ein Clustering-Verfahren auf die finalen Gaussians angewendet. Dabei kommt der Algorithmus *Density-Based Spatial Clustering of Applications with Noise* (DBSCAN) Ester u. a. 1996 zum Einsatz. DBSCAN klassifiziert Punkte auf Grundlage ihrer lokalen Dichteverteilung, ohne dass im Vorhinein die Anzahl der Cluster spezifiziert werden muss. Zwei Parameter steuern dabei das Verhalten des Algorithmus: der Radius eines Punktes, innerhalb dessen Nachbarn gezählt werden, sowie die minimale Anzahl an Nachbarn, die erforderlich ist, damit ein Punkt als Kernpunkt gilt. Punkte mit ausreichender Nachbarschaftsdichte werden als Kernpunkte betrachtet und mit benachbarten Punkten, die ebenfalls dicht genug liegen, zu einem Cluster zusammengefasst. Punkte, die in keinem ausreichend dichten Bereich liegen, werden als Rauschen deklariert.

Im Rahmen dieses Projekts werden die 3D-Positionen aller optimierten Gaussians als Eingaberaum für den DBSCAN-Algorithmus verwendet. Das größte identifizierte Cluster wird dabei typischerweise der Zielperson zugeordnet, da es die höchste räumliche Kohärenz aufweist. Alle weiteren, meist kleineren Cluster sowie isolierte Einzelpunkte außerhalb dieses Hauptclusters werden als nicht zur Person gehörig betrachtet und verworfen. Das Ergebnis der Bereiningung wird in Abb. 4.2.2

Abbildung 30: Darstellung der trainierten Szene nach der Gaussianentfernung durch DBSCAN

4.3 Viewer

Beteiligte Teammitglieder: Alisa Rüge, David Martin Karg, Steffen-Sascha Stein

Im Rahmen des Projekts wurde ein Webviewer implementiert, der als Plattform zur Darstellung der erzeugten volumetrischen Inhalte dient. Die zugrunde liegende Softwarearchitektur folgt einem modularen Aufbauprinzip, das eine klare Trennung der funktionalen Teilbereiche – Rendering, Videohandling und Datenmanagement, Benutzereingabe sowie grafische Benutzeroberfläche (UI) – ermöglicht. Abbildung 31 zeigt die modulare Ordnerstruktur des Projekts und welche Komponenten jeweils dem Rendering, des Videohandlings/Datenmanagements oder der UI/Steuerung zuzuordnen sind. Diese visuelle Struktur dient als Orientierung innerhalb dieses Kapitels.

Abbildung 31: Übersicht Codestruktur des Viewers

Im Zentrum des Webviewers steht die `VolumanXRHandler.js`-Klasse, die als Main Game Loop fungiert und sämtliche Komponenten zusammenbringt. Diese Klasse besteht aus zwei zentralen Methoden: `initXR()` und `onXRFrame()`.

Die Methode `initXR()` wird einmalig zu Beginn aufgerufen und übernimmt die Initialisierung der `XR-Session`, das Laden der volumetrischen Videos inklusive ihrer Transformationsdaten, das Einrichten der Render-Pipeline (sowohl `WebGL` als auch `WebGPU`), sowie die Initialisierung der Eingabeverarbeitung durch den `InputHandler`. Sie dient somit als Setup des Gameloops.

Die Methode `onXRFrame()` stellt die den Animationsloop dar und wird von der XR-Sitzung in jedem Frame aufgerufen. Hier erfolgt die kontinuierliche Verarbeitung von Benutzereingaben, die Wiedergabe des aktuell geladenen Videos, die Sortierung der Splats, das Rendering der Szene sowie die Darstellung des User Interface (UI).

Die Datei `VolumanXRHandler.js` verbindet somit alle Kernkomponenten des Viewers: das volumetrische Videohandling (`VolumeVideo.js`), welches in der Steuerung durch Controller und Tastatur im (`InputHandler.js`) stattfindet, die UI-Komponenten (`UIRenderer.js`) sowie die verschiedenen Rendering-Backends (`WebGLRenderer.js`, `WebGPURenderer.js`, `VolumanXRRenderer.js`).

Im Folgenden werden diese einzelnen Bereiche des Viewers detailliert beschrieben.

4.3.1 Video Handling und Data Loading

Volumetric-Video Architektur Beteiligte Teammitglieder: David Martin Karg

Die Idee der Volumetric-Video-Architektur war das einfache Verwalten von mehreren volumetrischen Videos, die wiederum aus mehreren Sequenzen bestehen, die hintereinander abgespielt werden sollten. Außerdem sollte es, für zukünftiges Arbeiten, die Implementierung erleichtern, mehrere Videos gleichzeitig abzuspielen. Die Idee dabei war, dass sich zwei virtuelle Personen gegenüberstellen können. Diese wurde im Rahmen des Projektes allerdings nicht weiter verfolgt. Mit der Volumetric-Video-Architektur lassen sich nun einfach mehrere Videos laden, diese werden im Speicher gehalten und können mit der Steuerung aufgerufen werden. Im Folgenden wird auf die Grafik 32 eingegangen, die das UML-Diagramm darstellt, welches vor der Implementierung der Klasse angefertigt wurde. Das Diagramm und die schlussendliche Implementierung unterscheiden sich in Teilen, worauf an den entsprechenden Stellen eingegangen wird. Die Volumetric-Video-Klasse enthält nun folgende Parameter und Funktionen: einen dreidimensionalen Vektor für die Position des Modells, wobei die Position der Ursprung der Space-Time-Gaussians sein soll. Da nur um eine Achse rotiert werden soll, der Y-Achse, wird nur ein `float` für die Rotation um die Y-Achse gespeichert. Auch für die Skalierung wird nur ein `float` gespeichert, da die Skalierung gleichmäßig in alle Achsen erfolgt. Ein `float` soll die `playTime` speichern, die aktuell zu internalTime umbenannt wurde. Wir speichern ein Integer, das beschreibt, welche Sequenz aktuell abgespielt wird, und wir haben ein Array, das alle Sequenzen eines Volumetric-Videos halten soll. Im Diagramm ist noch zu sehen, dass das Array Daten vom Typ `ReadableStreamDefaultReader<R>[]` halten soll, allerdings werden die Sequenzen aktuell als `Uint8Array` gespeichert.
Die Volumetric-Video-Architektur hat vornehmlich zwei Funktionen. Sie soll die Sequenzen in den Speicher laden, wofür es die asynchrone Funktion `load()` gibt, und sie soll Sequenzen abspielen, was die Funktion `play()` handhabt. Ursprünglich sollte die Ladefunktion einen `boolean` zurückgeben, ob das Laden erfolgreich war, das wurde allerdings ausgenommen.
Beim Laden sollen nur die Sequenzen in das Sequenzen-Array geladen werden. Dabei wird der Pfad der jeweiligen Sequenz an die Funktion weitergegeben. Mit dem fetch-Befehl werden die Daten an dem entsprechenden Pfad zurückgegeben. Mit dem Reader sollen die Daten ausgelesen werden und in Chunks erstmal in ein Array gespeichert werden, solange der Stream nicht vollständig ausgelesen wurde. Anhand der Gesamtlänge aller Chunks wird ein `Uint8Array` erstellt, in dem nun die Daten der Chunks gespeichert werden. Dieses `Uint8Array` wird dann in den Sequences-Array gepushed. Dieser Ladeprozess wird für jede Sequenz in einem Video gemacht, jedes Video wird von einer Instanz der Volumetric-Video-Klasse repräsentiert.
Das Abspielen der Videos erfolgt mit der `play()`-Funktion. Dieser wird nur die `deltaTime` übergeben. Um die des Videos interne Zeit zu speichern, wird die `deltaTime` zu jedem `play()`-Aufruf zur `internalTime` aufaddiert. Außerdem halten wir in der Klasse einen Parameter `sequenceChangeTime`, der den internen Zeitpunkt markieren soll, an dem die nächste Sequenz geladen werden soll. Ist dieser Zeitpunkt überschritten, so wird der neue Zeitpunkt festgelegt und die nächste Sequenz wird geladen, indem die Sequenz aus dem Array in einen neuen Stream

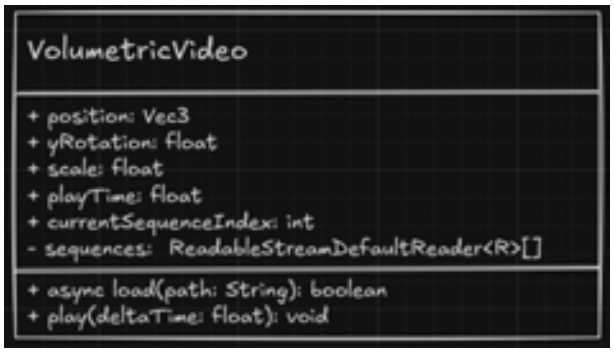


Abbildung 32: Volumetric-Video-Architektur Vorläufiges UML-Diagramm

geladen wird, dessen Reader an die Funktion `readChunks` übergeben wird, auf die im nächsten Abschnitt näher eingegangen wird.

Dateien Laden Beteiligte Teammitglieder: David Martin Karg, Steffen-Sascha Stein

Zum Laden der Dateien in den Viewer gab es zuerst die grundsätzliche Frage, wie und von woher die Dateien geladen werden sollen. In einer frühen Fassung des Viewers gab es einen Knopf auf der Desktop-Anwendung, von der ein Nutzer die einzelnen Sequenzen des Videos hatte hochladen können. Da der Viewer aber auch auf einer VR-Brille laufen soll, die nicht an einen Desktop-Rechner angeschlossen sein braucht, müssten die `.splatv`-Dateien auf der VR-Brille gespeichert werden. Außerdem hätte man innerhalb der VR-Anwendung immer zum Browser navigieren müssen, um zum Knopf zu kommen, der einem die Möglichkeit gibt, Dateien hochzuladen. Da all diese Aspekte unhandlich schienen, wurde gegen den Knopf entschieden, und stattdessen werden alle Videos als Ordner im Assets-Ordner gespeichert. Die Video-Ordner enthalten dann alle Sequenzen, die nach eindeutig nach einer vordefinierten Konvention benannt werden, sodass sie in der richtigen Reihenfolge im Sequences-Array gespeichert werden.
Das Laden der Dateien, der Videos und der Sub-Sequenzen kann in mehrere Schritte unterteilt werden. Zuallererst müssen alle verfügbaren Dateien und deren Pfade mit `require.context()` gefunden werden. Um die Sequenzen ihren entsprechenden Videos zuzuordnen, wird ein String-Array geführt mit allen Ordnern, die die Videos darstellen. Außerdem gibt es eine Liste mit den Ordner-Namen, der wir die entsprechenden Sequenz-Pfade übergeben.
In den Ordnern der entsprechenden Videos befindet sich eine Json-Datei, in der die Transformationsparameter der Videos stehen. Da beim Training die Position und Rotation der Modelle nicht immer einheitlich ist, kann man dadurch die Modelle an die gewünschte Stelle platzieren. In der Json-Datei ist auch ein Parameter, der bestimmt, ob das Video tatsächlich geladen werden soll oder nicht. Das ermöglicht das einfache Hinzufügen von neuen Videos und das deaktivieren von Videos, wenn erwünscht.
Im zweiten Schritt werden nun in der initXR-Funktion für jeden Eintrag im Video-Ordner-Array eine Volumetric-Video-Instanz erstellt und jeder Instanz werden die Sequenz-Pfade in der `load()`-Funktion übergeben.
Der dritte Schritt passiert in der `play()`-Funktion, der im Moment laufenden Volumetric-Video-Instanz. Und zwar, wird das Video das erste Mal geladen, oder, ist das Ende einer Sequenz

erreicht, dann wird aus dem Sequenz-Array in der Volumetric-Video-Instanz der entsprechende DataBuffer geladen, und ein neuer Stream daraus erstellt. Dieser Stream wird der readChunks-Funktion aus dem SplaTVLoader übergeben.
Die `readChunks`-Funktion nimmt den neu erstellten Datenstrom, einen Parameter namens `chunks` und eine Funktion, um die Chunks zu verarbeiten, den `chunkHandler`. Die `readChunks()`-Funktion soll den Datenstrom je Chunk auslesen und in einen Buffer füllen. Die `chunkHandler()`-Funktion wird zu jedem Schritt zum Füllen des Buffers aufgerufen, um zu überprüfen, um welche Art es sich bei dem aktuellen Chunk handelt. Dabei werden in drei Arten von Chunks unterschieden: `magic`, welche nur für den ersten Chunk verwendet wird, um zu überprüfen, ob es sich bei der geladenen Datei wirklich um eine .splatv-Datei Handelt; `chunks`, die Splat-Informationen enthalten, aber in mehrere Chunks aufgeteilt sind und zuletzt `splat`, welches die Splat-Informationen eines einzelnen Chunks enthalten. Diese Splat-Informationen werden pro Chunk in den Buffer geladen. Ist dieser gefüllt, wird der Buffer zuerst an den Sortworker gegeben und, sobald die splats sortiert sind, als Textur an den Renderer.

Videos Abspielen Beteiligte Teammitglieder: David Martin Karg

Für die Steuerung der Volumetrischen Videos wurde zu Anfang überlegt, welche Funktionen erwünscht sind. Die Szene soll im virtuellen Raum verschiebbar und rotierbar sein. Da es sich um eine dynamische Szene handelt, sollte es auch möglich sein, die Szene zu pausieren. Zuletzt sollte man auch noch zwischen den einzelnen Videos hin und her wechseln können. Das Video soll ganz einfach per Knopfdruck gestoppt werden. Zu Anfang war die Überlegung, dass die Play-Funktion zu jedem Frame aufgerufen wird und die `deltaTime` übergeben bekommt. Sollte auf den Stop-Knopf gedrückt werden, dann wird stattdessen die `deltaTime` auf null gesetzt, das Video schreitet als nicht voran. Da aber die `deltaTime` auch für die Evaluation abgefragt wird, wird die `deltaTime` nicht auf Null gesetzt. Stattdessen wird die Play-Funktion so lange nicht aufgerufen, bis wieder auf den Knopf gedrückt wird. Die Szene dennoch weiter rotiert oder verschoben werden.
Damit beim Halten des Knopfes das Video nicht abgehackt weiterläuft, und generell das Halten von Knöpfen keine unerwünschten Nebeneffekte hat, gibt es eine Liste namens `previousButtonState`, die den Status des Knopfes für den nächsten Frame speichert. Nur wenn der entsprechende Status des gedrückten Knopfes `false` ist, kann die Aktion erneut ausgeführt werden.
Zum Wechseln der Videos zum nächsten oder zum vorherigen Video gibt es den Parameter `currentVideoIndex`. Dieser fängt bei Null an und wird beim Wechseln zum nächsten Video inkrementiert. Damit der Wert zwischen Null und der Anzahl an Videos sich beschränkt, wird dieser wie folgt berechnet:

```
currentVideoIndex = (currentVideoIndex + 1) % voluVid.length;
```

Außerdem wird die interne Zeit des neuen Videos und auch der Sequenz-Index auf Null gesetzt.

Subsequenzen Beteiligte Teammitglieder: David Martin Karg

Da es bei dem Training Probleme mit der Länge der Videos gab, wurde beschlossen, die Videos in

Sequenzen aufzuteilen. Die Länge, beziehungsweise die Menge an Frames der Sequenzen, hängt dabei von der Rechenleistung des Computers ab, auf dem das Video trainiert wurde.
Dadurch kam es zu der Anforderung, möglichst flüssige Übergänge der einzelnen Sequenzen zu ermöglichen. Dabei geht es vor allem darum, dass die Bewegung nicht abgehackt aussieht, und dass die Sortierzeit der Gaussians nicht zu lang ist.
Da das Sortieren auf einen Worker ausgelagert wurde, wurde die Szene bereits an den Renderer übergeben, bevor alle Gaussians fertig sortiert wurden. Das führte dazu, dass zu Anfang einer jeder Sequenz, es zum Flackern kam. Auch wenn die Sortierzeit sehr kurz war, kam es zu dem Flackern, da zumindest im ersten Frame der Sequenz die Szene noch nicht fertig sortiert war. Um dieses Problem zu lösen, wurde im `chunkHandler` gewartet, bis die Szene fertig sortiert wurde, bevor sie an den Renderer übergeben wurde. Das führte zwischen den Sequenzen zu wesentlich angenehmeren Übergängen.
Da die Sequenzen im Idealfall nahtlos ineinander über gehen sollten, wurde überlegt, mit welchem Kriterium gearbeitet werden sollte, um den Wechsel zur nächsten Sequenz zu signalisieren. Wichtig ist dabei zu erwähnen, dass Space-Time-Gaussians so trainiert werden, dass jeder Splat zwei Funktionen besitzt, um die Position und Opazität zu berechnen. Die Bewegung berechnet sich aus einen Polynom, die Opazität aus einer zeitabhängigen radialen Basisfunktion. Interessant dabei für die Bewegung ist allerdings nur der Zeitabschnitt zwischen Null und Eins. Würde man die Funktion über den Wert von Eins weiterlaufen lassen, dann würde die Szene zerfallen, da die Gaussians sich in eine Richtung bewegen würden, die zwar nicht beliebig ist, aber für einen Zuschauer beliebig erscheint, und die Splats gehen auseinander.
Also muss, in der Theorie, für jede Sequenz die Zeit auf einen Wert zwischen Null und Eins beschränkt werden. Im Original-Code wurde die Zeit, wie folgt, berechnet:

```
playTime = Math.sin(time / 1000) / 2 + 1 / 2;
```

Diese Funktion würde die Sequenz nun auf einen Zeitabschnitt zwischen Null und Eins laufen lassen. Die Sinus-Funktion sorgt dafür, dass die Sequenz eine Sekunde vor und eine Sekunde zurück läuft.
Für das nacheinander Abspielen mehrerer Sequenzen, die nahtlos ineinander übergehen sollen, macht eine Sägezahn-Funktion allerdings mehr Sinn:

```
playTime = (time / 1000) - (Math.floor(time / 1000));
```

Das sorgt dafür, dass die Sequenz jedes Mal wieder bei Null startet, sollte der Wert Eins erreicht sein.
Ein Problem, das noch angesprochen werden muss, ist, dass die Sequenzen in der Regeln schon früher zerfallen als nach dem Zeitpunkt Eins, man aber für flüssige Bewegungsübergänge auf jeden Fall die Sequenz bis zum Ende laufen lassen sollte. Die Lösung dabei war, dass beim Training die jeweiligen Sequenzen mit sich überlappenden Frames trainiert wurden. Hat eine Sequenz 10 Frames, so würde für die erste Sequenz die Frames 0 bis 9 trainiert werden. Die nächste Sequenz würde für die Frames 8 bis 17 trainiert werden, also gibt es immer zwei sich überlappende Frames. Die Funktion müsste also nur noch über einen Zeitraum zwischen 0 und 0.8 laufen.

```
playTime = ((time / 1000) - (Math.floor(time/ 1000))) * 0.8;
```

Diese Methode hat die Artefakte, dass die Splats frühzeitig auseinander fallen, reduziert, allerdings hat diese Methode auch die Flexibilität beschränkt, wie viele Frames eine Sequenz enthalten darf. Würde eine Sequenz für fünfzehn Frames trainiert werden, auch mit zwei sich überlappenden Frames, würde die oben gezeigte Funktion zu abgehackten Übergängen führen. Entweder werden ab sofort nur noch Sequenzen für zehn Frames mit zwei überlappenden Frames trainiert, oder zu jedem Video gibt es eine Json mit Metadaten, um zu die Zeitfunktion zu berechnen.

4.3.2 Rendering

Beteiligte Teammitglieder: Steffen-Sascha Stein

Zur Implementierung des Renderings von Spacetime Gaussian Splatting (Zhan Li u. a. 2024), wurde auf den Open-Source Code von Kevin Kwok zurückgegriffen (Kwok 26.03.2024). Dabei handelt es sich um einen WebXR-WebGL-Viewer, der in JavaScript geschrieben wurde. Die im Abschnitt Kapitel 4.3.2 beschriebene Implementierung ist eine modifizierte Version dieser Codebasis.
Da WebGL keinen Support für Compute Shader bietet, welche z.B. für die GPU-Sortierung der Splat-Primitive benötigt werden, wurde eine alternative Umsetzung in WebGPU entwickelt, welche in Kapitel 4.3.2 erläutert wird. Zum Zeitpunkt der Erstellung dieser Arbeit gab es noch keinen WebXR Support für WebGPU im Meta Quest Browser. Daher wurde ein Workaround implementiert, in welchem die in WebGPU gerenderten Ergebnisse an einen `WebGLContext` übergeben werden, welcher mit einer `XRSession` verbunden ist. Zudem verfügt die WebGPU-Implementierung über vier verschiedene Performance-Modes, welche sich durch das Input Handling (siehe Abschnitt 4.3.3) zur Laufzeit umschalten lassen.
Der Wechsel zwischen WebGL- und WebGPU-Viewer ist jedoch nicht zur Laufzeit möglich. In `main.js` kann zwischen einer der Varianten mithilfe der `rendererMode` Variable gewählt werden.
Im Folgenden werden zunächst die Funktionsweise und der Aufbau der beiden Viewer beschrieben. Anschließend werden weiterer Optimierungsansätze erläutert, die unabhängig von der verwendeten Rendering-API genutzt werden können, jedoch hauptsächlich in der WebGPU-Implementierung Anwendung finden.

WebGL-Viewer

Zu Beginn des Projekts lag die Codebasis von Kevin Kwok als einzelne `index.html` Datei, und somit als Monolith vor. Die Struktur wurde jedoch schnell unübersichtlich, weshalb die Datei in mehrere Module aufgeteilt wurde. Dabei wurden an den Skripten `WebGLRenderer.js`, `SplaTVLoader.js` und `SortWorker.js` kaum bis keine Änderungen vorgenommen. Der `WebGLRenderer` bekommt jedoch den Zustand des derzeitigen `VolumeVideo` übergeben, damit die zugehörigen Splats transliert, rotiert und skaliert werden können. Außerdem musste die Kameramatrix vor dem Sortieren der Splats in das Koordinatensystem des `VolumeVideo` trans-

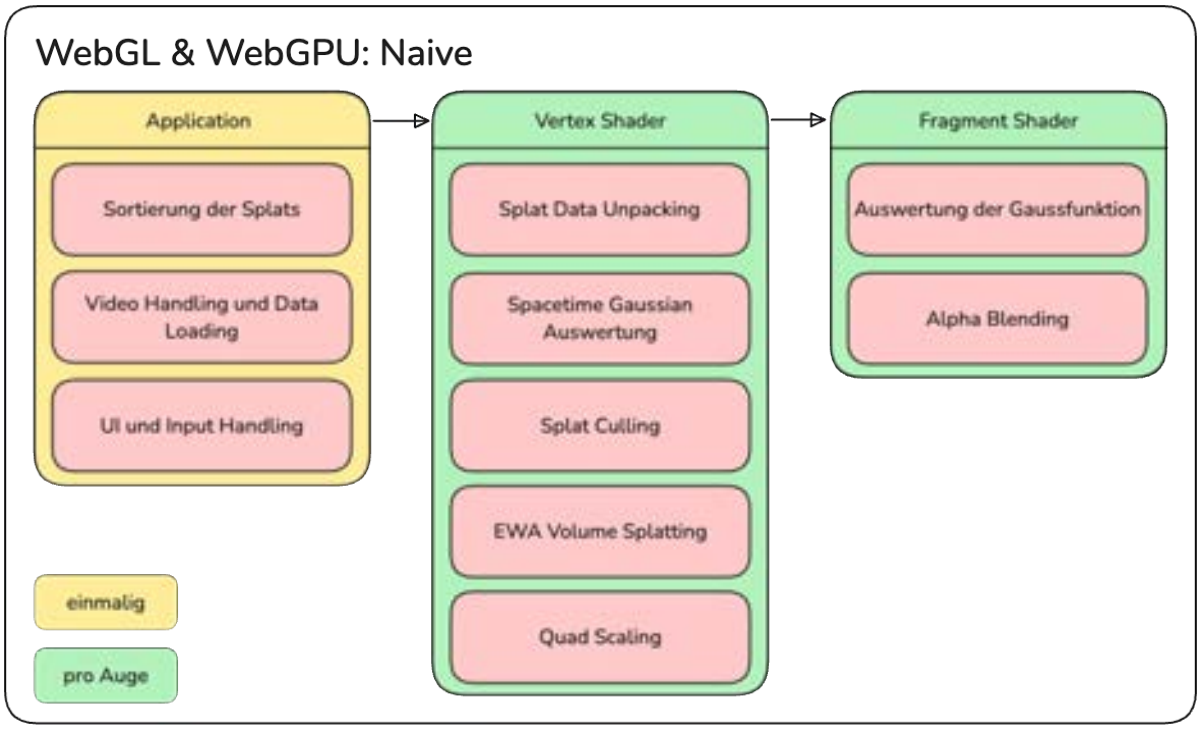


Abbildung 33: Pipeline des WebGL- und WebGPU-Viewers. Zweiterer im `Naive`-Mode.

formiert werden, damit es auch nach einer Rotation zu visuell korrekten Ergebnissen kommt. Im Folgenden wird nun der Ablauf des WebGL-Viewers beschrieben, wie er in Abbildung 33 dargestellt ist.

Die Applikation beschreibt den Central Processing Unit (CPU)-seitigen Teil des Viewers. Während eine `splatv`-Datei mithilfe von `SplaTVLoader` progressiv geladen wird, werden die Splat-Indizes mit jedem hinzukommenden Chunk mittels 16-Bit Counting Sort nach Tiefe sortiert. Dies passiert asynchron in einem `WebWorker`, um den Mainthread nicht zu blockieren. Die bisherigen geladenen Splats werden als Textur an den `WebGLRenderer` zusammen mit den sortierten Indizes übergeben. Für jeden Splat wird ein Quad als Triangle Strip gerendert, sodass nur ein einzelner Draw Call für vier Vertices und n Instanzen für das gesamte Rendering benötigt wird, wobei n der Anzahl an geladenen Splats entspricht.

Der Vertex Shader entpackt zunächst die Daten eines Splats aus der Textur. Der Zugriff erfolgt für die derzeitige Instanz über den Index, der im sortierten Buffer gespeichert ist. Zu den Daten gehören statische Attribute wie Position, Rotation, Skalierung und RGBA-Farbwert, welche dem ersten Frame des volumetrischen Videos entsprechen. Um die Splats zeitlich zu transformieren, befinden sich zudem diverse Koeffizienten unter den Daten einer Instanz. Wie in (Zhan Li u. a. 2024) beschrieben, wird der zeitliche Verlauf der Transparenz über eine Radiale Basisfunktion (RBF) interpoliert. Dies geschieht für Position und Rotation über Polynomfunktionen. Der RGB-Farbwert und die Skalierung bleiben dabei durchgehend konstant. Anschließend folgt der weitere Verlauf dem des ursprünglichen Gaussian Splatting Ansatzes (Kerbl u. a. 2023). Während die Autoren jedoch auch das Rendering der Splats in einem Computer Shader

```
in vec4 vColor;
in vec2 vPosition;

out vec4 fragColor;

void main () {
    float a = dot(vPosition, vPosition);
    if (a < 4.0) discard;
    float b = exp(-a) * vColor.a;
    fragColor = vec4(vColor.rgb, 1.0) * b;
}

```

Code 2: Fragment Shader für 3D Gaussian Splatting (Kerbl u. a. 2023).

implementiert haben, wird in dieser Umsetzung mit der Render Pipeline gearbeitet, indem jeder Splat als Quad mit vier Vertices repräsentiert wird. Dafür wird zunächst die 3D Kovarianzmatrix eines Splats nach dem Elliptical Weighted Average (EWA) Volume Splatting Verfahren (Zwicker u. a. 2001) ins Zweidimensionale projiziert. Aus der 2D Kovarianzmatrix lassen sich anschließend Haupt- und Nebenachse berechnen. Mit der Position und den Achsen eines Splats und der Größe des Viewports kann die derzeitige Vertexposition im Clip-Space berechnet werden. So werden alle vier Vertices zu einem Quad aufgespannt, in welchem die zugehörige Gaußsche Ellipse im Fragment Shader rein gezeichnet werden kann.

Der Fragment Shader in Code 2 bekommt den RGBA-Farbwert des Splats, und die Ursprungsposition des Quad-Vertex übergeben. Letztere ist interpoliert und wird genutzt, um mithilfe des Skalarproduktes aus sich selbst die Distanz des derzeitigen Pixels zum Quadzentrum zu bestimmen. Überschreitet diese Distanz die Seitenlänge des Quads, wird der Pixel verworfen. So wird aus dem Quad ein Kreis. Da der Quad entsprechend der Haupt- und Nebenachse skaliert ist, wird der Kreis zur Ellipse. Die Distanz wir außerdem benutzt, um den Abfall der Transparenz durch eine Gaußfunktion zu berechnen. So wird aus der Ellipse ein zweidimensionaler Gaussian Splat.

WebGPU-Viewer

Die WebGPU Variante unterscheidet sich bereits in ihrem grundsätzlichen Aufbau. Der `WebGPURenderer` setzt lediglich einen Rahmen, um generelles XR-Rendering zu ermöglichen. Ein übergebener `SubRenderer` übernimmt dann das tatsächliche Rendering von Inhalten. Dies soll unter anderem dafür sorgen, dass ein `SubRenderer` vom XR-Kontext abgekapselt ist und auch für Desktopanwendungen wiederverwendet werden kann.
Wie bereits erwähnt, existiert zum Zeitpunkt der Erstellung dieser Arbeit noch kein WebXR Support für WebGPU im Meta Quest Browser. Da die Nutzung von Compute Shadern jedoch die Grundlage für die überlegten Performanzoptimierungen war, wurde mithilfe des `GPU2GLHelper` die Nutzung von WebGPU und WebXR ermöglicht. Dieser verfügt über `OffscreenCanvas` Objekte für jedes Auge, in welche ein `WebGPURenderer` und der dazugehörige `SubRenderer` zu jedem Frame rein zeichnen. Die Inhalte der Canvases werden als Texturen an den

`WebGLRenderingContext` des `GPU2GLHelper` übergeben, der mit der laufenden `XRSession` verknüpft ist. Dabei gibt die Funktion `OffscreenCanvas.transferToImageBitmap()` ein `ImageBitmap` Objekt zurück, welche als Textur an WebGL ohne CPU-Readback übergeben werden kann. Anschließend wird alles über einen Fullscreen-Quad dargestellt.
Durch die erfolgreiche eigene Bereitstellung einer laufenden Umgebung für WebGPU in Kombination mit WebXR, wurde die Möglichkeit eröffnet, diverse GPU-basierter Optimierungen und Pipelines zu implementieren. Deshalb verfügt der `VolumanXRRenderer` über vier verschiedene Performance-Modes. Ursprünglich war gedacht die Modi in verschiedene `SubRenderer` aufzuteilen. Durch die große Überschneidung an Ressourcen und Funktionalitäten hätte dies jedoch zu einer Menge an Code-Duplikaten geführt. Daher existiert in der finalen Version lediglich der `VolumanXRRenderer` als `SubRenderer` in welchem sich alle Modi bündeln. Diese werden aus Kombinationen von verschiedenen `GPURenderPipeline` und `GPUComputePipeline` Objekten repräsentiert. Dieser Umstand ermöglicht es die Modi während der Laufzeit zu wechseln.

Die Modes unterscheiden sich wie folgt:

- `Naive` → Entspricht derselben Funktionsweise und demselben Aufbau wie dem des WebGL-Viewers (siehe Abbildung 33).
- `Cull` → Primitives „Culling“ der Splats welches zusammen mit dem Spacetime Gaussian Splatting in einen Computer Shader verlagert wird. Wird durch Zusammenfallen der Quad-Vertices auf einen einzelnen Punkt im Clip-Space erst im Vertex Shader realisiert.
- `IndirectCull` → Das Culling wird durch indirektes Rendering verbessert, sodass nur noch so viele Draw-Aufrufe wie nötig ausgeführt werden. Dies erfordert jedoch zusätzliches Management der sortierten Indizes.
- `IndirectCullGPUSort` → Ergänzend zur Funktionalität des `IndirectCull`-Modes wird das Sortieren der Splats von der CPU auf die GPU verlagert.

Im Folgenden werden lediglich die letzten drei Modes beschrieben, da die Funktionalität des `Naive`-Modes mit der des `WebGLRenderers` übereinstimmt. Alle Shader dieses Modes befinden sich in `naive.wgsl`. Alle anderen Modes beginnen ihre Pipeline mit dem Compute Shader `cull_compute.wgsl` und enden mit den Vertex und Fragment Shadern in `cull_render.wgsl`.

Cull Mode: Für den `WebGLRenderer` und den `VolumanXRRenderer` im `Naive`-Mode war es zunächst nicht trivial festzustellen, ob die Performanz durch den Vertex oder den Fragment Shader limitiert ist. In einer niedrigeren Auflösung zu rendern sollte zunächst vermieden werden, da neben der Echtzeit-Anforderung auch der Photorealismus für das volumetrische Video vorgesehen ist. Es wurde zudem angenommen, dass ein einmaliges Auswerten der Spacetime Gaussians zu einem geringeren Aufwand führen sollte. Der Vertex Shader im `Naive`-Mode leistet redundante Arbeit für jedes Auge und jeden Vertex – also $2 * 4 = 8$ mal.
Im Gegensatz zum `Naive`-Mode in Abbildung 33 verlagert der `Cull`-Mode in Abbildung 34 beinahe alle Aufgaben des Vertex Shaders in einen Compute Shader, welcher einmalig vor der bisherigen Render-Pipeline ausgeführt wird. Diese Verlagerung wurde in `cullShader` in

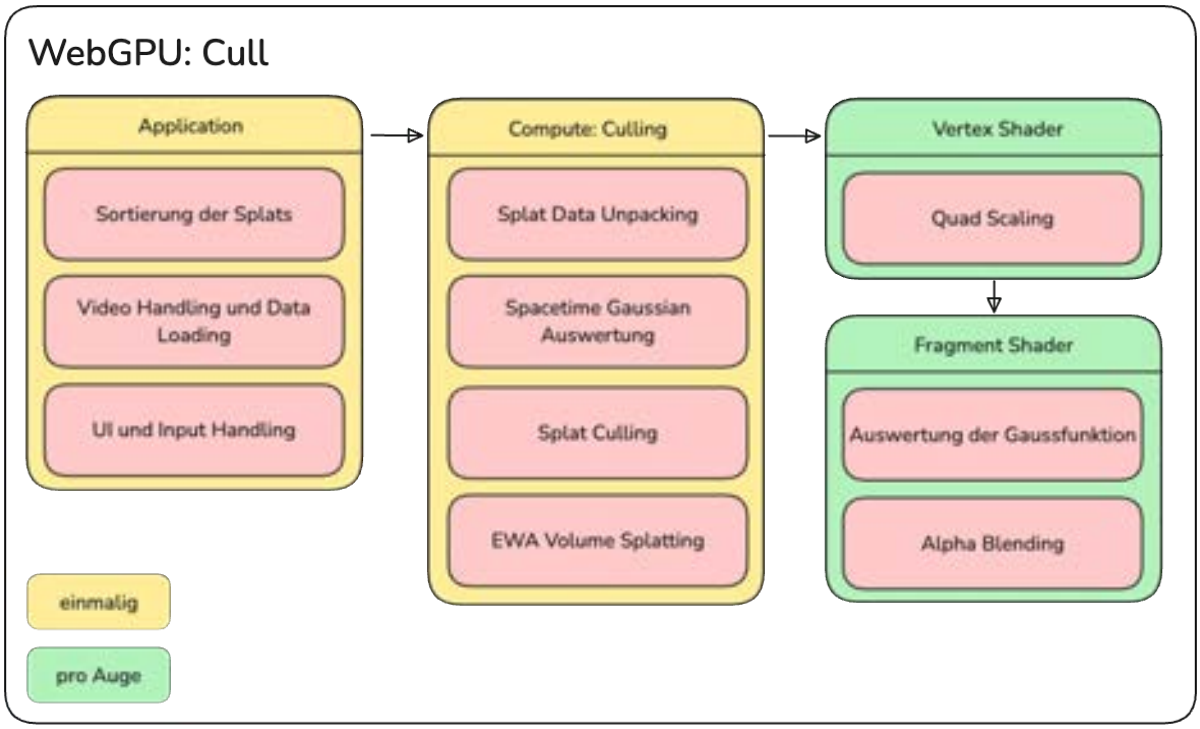


Abbildung 34: Pipeline des WebGPU-Viewers im `Cull`-Mode.

`cull_compute.wgsl` umgesetzt.
Die Datenmenge, die an den Vertex Shader weitergegeben wird, sollte dabei auf das Mindeste beschränkt werden. Dafür wurde ein Quad Struct definiert, welches über den Mittelpunkt des Quads, die Haupt- und Nebenachse des Splats und dessen Farbe verfügt. Die zwei Achsen wurden dabei mittels `pack2x16float()` zu einem `vec2<u32>` zusammengefasst, sodass die Größe eines Structs sich auf 64 Byte beläuft, was der gängigen Größe einer Cache-Line entspricht. Die Zusammensetzung eines Quads ist in Tabelle 4 zu sehen. Für alle Splats eines volumetrischen Videos werden die Daten dann als Array of Structures (AoS) `quads` an den Vertex Shader übergeben. Handelt es sich bei dem Device beispielsweise um ein HMD, werden in `cullShader` die Quad-Parameter für beide Views berechnet. Das Quad-Array ist dann doppelt so groß, wobei die Elemente für das zweite Auge dann mit einem Offset beginnen, welcher der Anzahl an Splats entspricht. Die Position eines Quads im Quad-Buffer entspricht dabei dem Index des derzeitigen Splats aus dem sortiertem Storage Buffer `indexValues`. Im Vertex Shader wird zunächst mithilfe der Builtin-Variable `instance_index` der Index des Quads aus `indexValues` geholt. Über die Uniform-Variable `viewIndex` kann dann bestimmt werden, ob der Quad in der ersten oder zweiten Hälfte des Buffers liegt. Für diesen, aber auch die folgenden Modes, bleibt im Vertex Shader lediglich das Verschieben der Quad-Vertices und im Fragment Shader das Zeichnen der Splats wie in Code 2 beschrieben übrig.
Das Culling in diesem Mode ist jedoch primitiv gelöst. Ist ein Splat nicht sichtbar, wird der Alphawert von `color` des zugehörigen Quads auf null gesetzt. Die Vertices dieser Instanz werden dann außerhalb des Clip-Spaces auf dieselbe Position fallen. In diesem Fall entspricht zusätzlich der übergebene Farbvektor für den Fragment Shader dem Nullvektor.

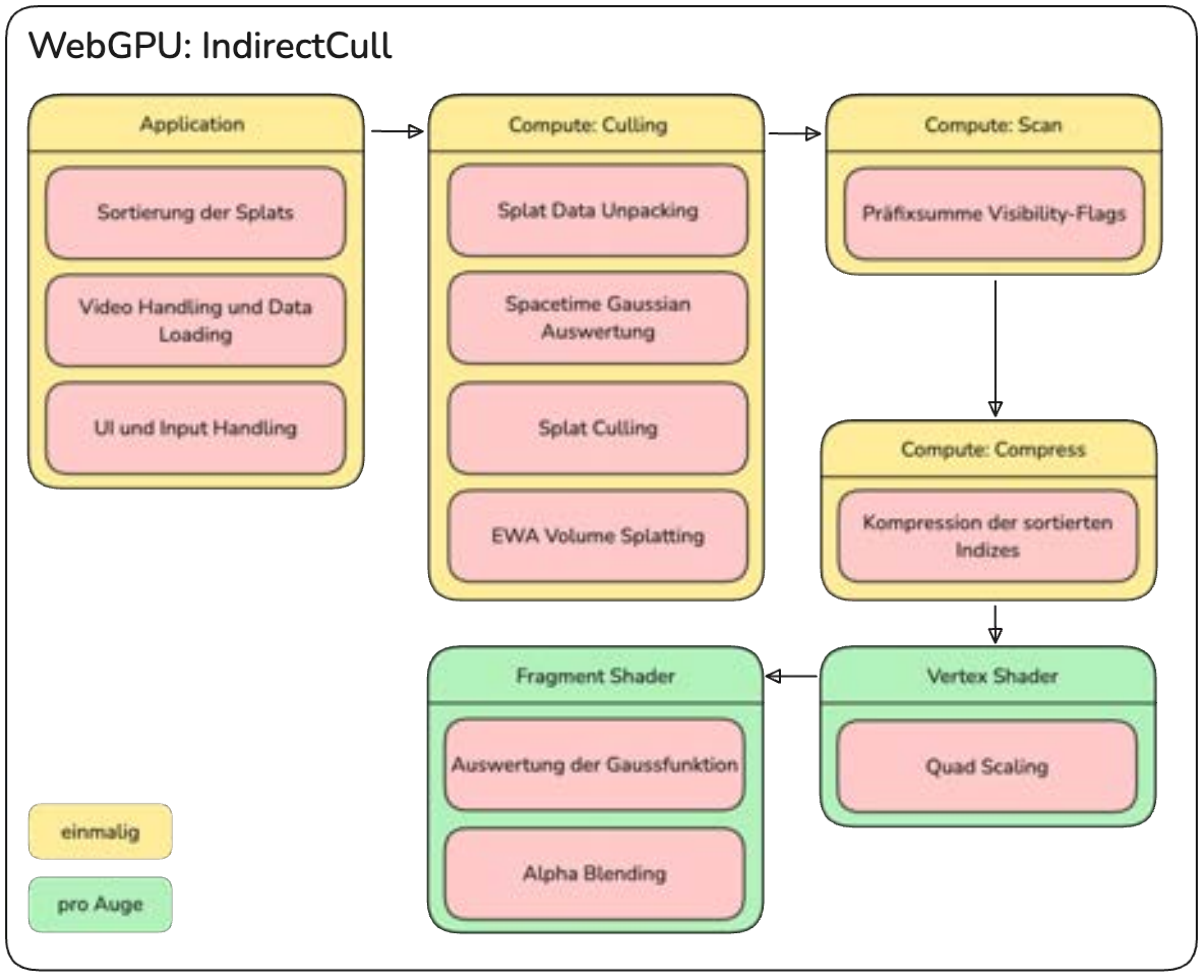


Abbildung 35: Pipeline des WebGPU-Viewers im `IndirectCull`-Mode.

IndirectCull Mode: Während im primitiven Cull-Mode das Culling erst durch das Clipping nach dem Vertex Shader umgesetzt wird, wird es in diesem Mode durch indirektes Rendering ermöglicht. Hierfür bietet WebGPU neben dem normalen `draw()`-Aufruf noch die `indirectDraw()`-Funktion. Letztere bekommt anstelle der einzelnen Parameter wie z.B. `vertexCount` und `instanceCount` lediglich einen `GPUBuffer` übergeben. Dieser kann genutzt werden, um nur die Anzahl an Instanzen rendern zu lassen, die nicht vom Culling betroffen sind. Ermöglicht wird dies durch das Hochzählen von `instanceCount` im `cullShader` mithilfe von `atomicAdd()`. Das indirekte Rendering verhindert unnötige Vertex und Fragment Shader Aufrufe und durch Ersteres auch Speicherzugriffe auf den Quad-Buffer für nicht sichtbare Instanzen. Wird ohne Weiteres im Vertex Shader anschließend auf die auf der CPU sortierten `indexValues` zugegriffen, wird es zu Artefakten kommen, wie in Abbildung 36 dargestellt. Dies geschieht, da im Vertex Shader nur

Variable	**Datentyp**	**Größe [Byte]**
`center`	`vec2<f32>`	16
`axes`	`vec2<u32>`	16
`color`	`vec4<f32>`	32
	Total	**64**

Tabelle 4: Aufbau des Quad-Structs.

a) b)

Abbildung 36: `IndirectCull`-Mode a) mit und b) ohne Präfixsumme und Kompression der sortierten Indizes. Die Abbildung zeigt den FLAMES Datensatz aus (Broxton u. a. 2020).

noch auf die ersten Instanzen in der Höhe von `instanceCount` zugegriffen wird, unter denen sich aber nicht sichtbare Splats befinden können. Dass sich ein Splat näher am Betrachter befindet als ein anderer, bedeutet nicht, dass er auch für diesen sichtbar ist.
Um diese Problematik zu umgehen, muss ein Storage Buffer an den Vertex Shader übergeben werden, welcher nur die Indizes der sichtbaren Splats enthält.
Dafür muss eine Vorverarbeitung der sortierten Indizes vor dem Rendern stattfinden, weshalb sich der Aufbau dieses Modes von dem des `Cull`-Modes unterscheidet, wie in Abbildung 35 dargestellt.
Diese Vorverarbeitung wird durch die folgenden Schritte umgesetzt:

1. In `cullShader` werden Visisbility-Flags für alle Splats gesetzt, und in `visibilityFlags` und `scanBuffer` geschrieben.

2. Der Scan-Algorithmus wird auf `scanBuffer` angewendet, um die Präfixsumme zu berechnen.

3. In `compressShader` wird für den derzeitigen Thread-Index die Sichtbarkeit über `visibilityFlags` überprüft. Ist der Splat sichtbar, kann der Schreibindex für `visibleInstances` mit dem Thread-Index aus `scanBuffer` geholt werden. Der Splat-Index wird dann aus dem sortierten `indexValues` in `visibleInstances` geschrieben.

Der Vertex Shader greift anschließend auf `visibleInstances` zu, um die Indizes der sichtbaren Splats zu holen. Im Gegensatz zu den vorherigen Modi wird dafür nicht mehr auf `indexValues` zugegriffen. Der Scan-Algorithmus wird aus der WebGPU-Radix-Sort Library (Kishimisu 2025) bezogen und dessen Implementierung orientiert sich an (Harris, Sengupta und Owens 2007).

IndirectCullGPUSort Mode: Die GPU-seitige Sortierung der Gaussian Splats wurde aufgrund zweier Annahmen implementiert. Erstens, dass das CPU-seitige Sortieren die Performanz beeinflusst, da die Sortierzeiten auf der Meta Quest 3 im zweistelligen Millisekundenbereich liegen. Zweitens, dass es zudem zu visuell unkorrekten Ergebnissen führt, da nur die statischen Positionen des ersten Frames eines volumetrischen Videos sortiert werden.
Da die Bewegung der Spacetime Gaussian Splats erst in `cullShader` auf der GPU berechnet wird, muss die Sortierung der Splats ebenfalls auf dieser und nach dem Culling stattfinden. Zur

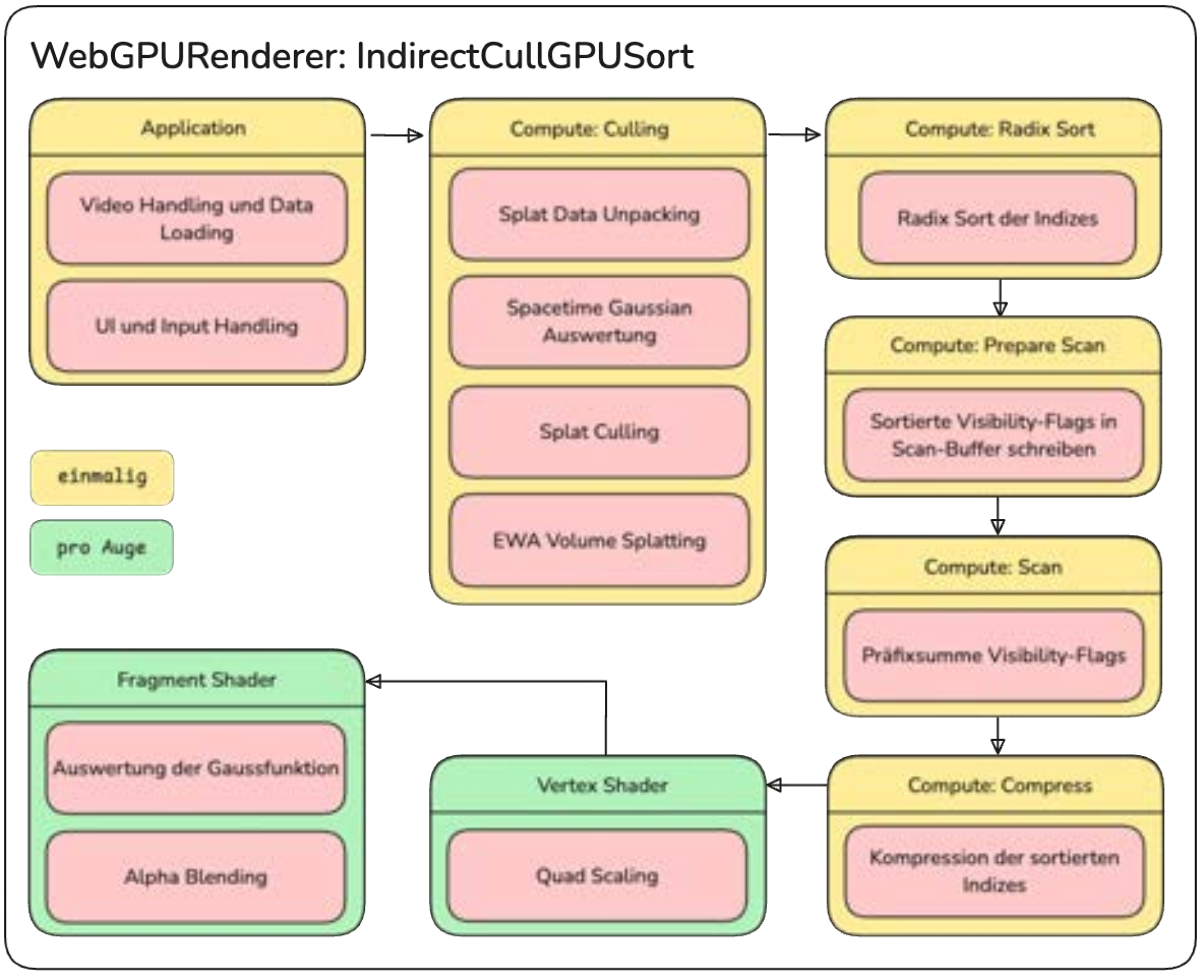


Abbildung 37: Pipeline des WebGPU-Viewers im `IndirectCullGPUSort`-Mode.

Umsetzung dieses Vorhabens wird ebenfalls die WebGPU-Radix-Sort Library (Kishimisu 2025) verwendet. Der dort implementierte Radix-Sort Algorithmus ist nach (Ha, Krüger und Silva 2009) implementiert und ermöglicht über den booleschen `check_order`-Parameter das schnellere Sortieren von bereits sortierten Arrays. Letzteres wird in der hier beschriebenen Umsetzung genutzt, zudem werden die Splats nur einmalig für das erste Auge sortiert.
In diesem Mode tritt dieselbe Problematik wie im `IndirectCull`-Mode auf, sodass auch hier nur die sortierten und sichtbaren Indizes in `visibleInstances` geschrieben werden dürfen. Zusätzlich müssen aber neben den Indizes nun auch die Visibility-Flags mit sortiert werden. Die Radix-Sort Implementierung nimmt aber nur einen Key- und zusätzlich einen optionalen Value-Buffer entgegen, wobei die Sortierung nur auf Grundlage der Keys geschieht. Existiert ein Value-Buffer, wird dieser in der Reihenfolge der Keys mit sortiert. Hierfür werden in `cullShader` die Storage Buffer `depthKeys` und `indexValues` mit den Tiefenwerten und den Indizes der Splats gefüllt und an den Radix-Sort Shader übergeben. Um zu verhindern, dass entweder etwas an der Originalimplementierung verändert werden, oder eine zusätzliche Sortierung stattfinden muss, wird die Visibility-Flag an die Stelle des Most Significant Bit (MSB) des zugehörigen Indexes gesetzt und dieser Wert in `indexValues` geschrieben. So werden die Indizes und die Visibility-Flags in einem Buffer gespeichert, und zusammen mit den Tiefenwerten sortiert.

a) b)

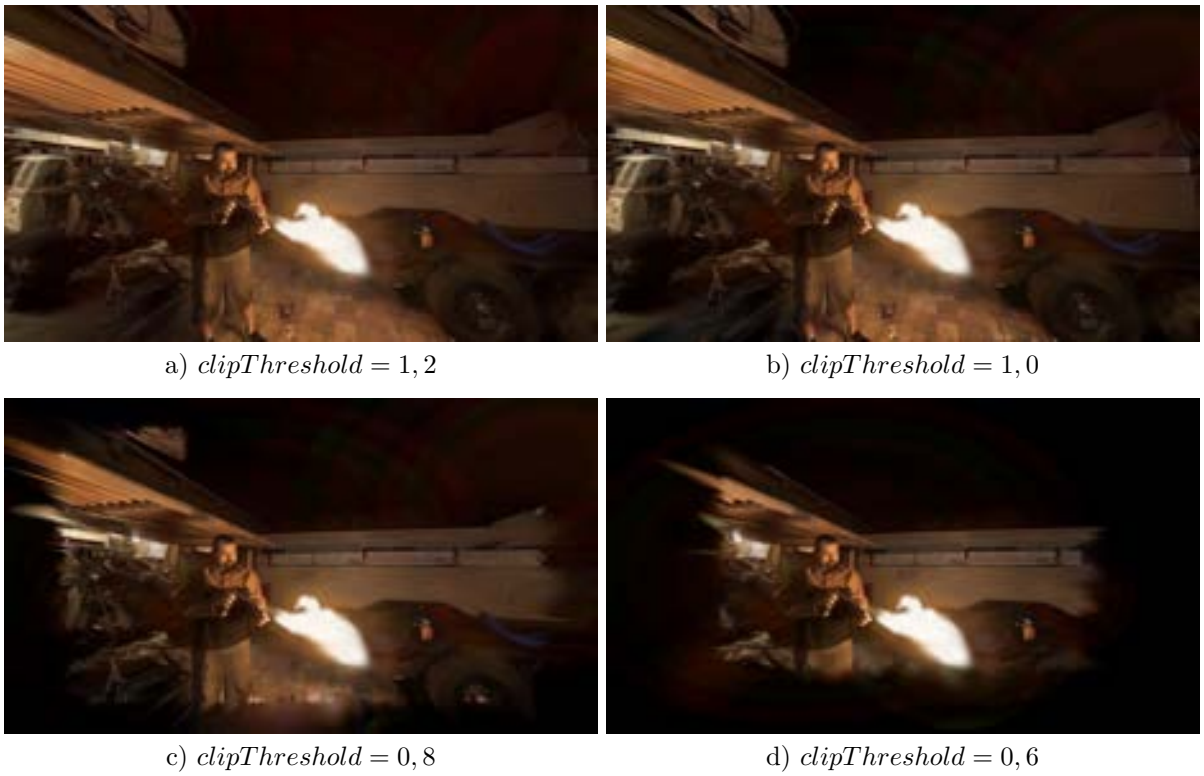
Abbildung 38: Visuelle Verbesserung durch GPU-seitige Sortierung der Splat-Inizes. a) CPU-seitige Sortierung, b) GPU-seitige Sortierung.

Zusätzlich müssen nach dem Radix-Sort und noch vor dem Berechnen der Präfixsumme die Flags in `scanBuffer` geschrieben werden. Dieser Zwischenschritt geschieht im Compute Shader `prepareScanShader`. Die nun vollständige Pipeline des `IndirectCullGPUSort`-Modes ist in Abbildung 37 dargestellt.
In Bezug auf die für diesen Mode getroffenen Annahmen gilt für die Zweite, dass die CPU-seitige Sortierung zu visuell unkorrekten Ergebnissen führt, konnte mit der Implementierung des `IndirectCullGPUSort`-Modes bestätigt werden. Die visuellen Verbesserungen sind in Abbildung 38 dargestellt. Für erste Annahme konnte jedoch keiner Verbesserung der Gesamtperformanz, sondern eine Verschlechterung festgestellt werden.

Weitere Optimierungsansätze

Zusätzlich zu dem WebGPU-Viewer und den verschiedenen Modi wurden noch einige weitere Optimierungsansätze umgesetzt, von denen sich eine Verbesserung der Performanz erhofft wurde. Da jedoch dem WebGPU-Teil des WebGPU-Viewers selbst keine WebXR-Funktionalitäten zur Verfügung stehen, kamen einige Verfahren, wie beispielsweise klassisches Foveated Rendering, nicht infrage. Diese hätten nur eine Auswirkung auf die Ausgabe des in Kapitel 4.3.2 beschriebenen `GPU2GLHelper`, welcher lediglich die von WebGPU gerenderten Ergebnisse an WebGL übergibt.

Foveated Splatting und Alpha Culling: Liegt die Position eines Splats außerhalb des Clip-Spaces, wird dieser und der dazugehörige Quad verworfen. Unter der Annahme, dass das Culling von großen Splats zu sichtbaren Artefakten führt, wurde in der Originalimplementierung von Kevin Kwok der zulässige Clip-Space um 20% vergrößert. Umgekehrt lässt sich die Beschaffenheit des Gaussian Splatting Ansatzes (Kerbl u. a. 2023) jedoch nutzen, um eine Art von Foveated Rendering zu erzeugen, welches von hier an als Foveated Splatting bezeichnet wird.
Es ist anzumerken, dass nur die Top-, Bottom-, Left und Right-Ebene für dieses Verfahren genutzt werden. Für diese muss die Position eines Splats im Bereich $[-\texttt{clipThreshold}, \texttt{clipThreshold}]$ liegen, um nicht verworfen zu werden. Die Near- und Far-Ebene sind nicht von `clipThreshold` abhängig und spannen daher, wie in WebGPU üblich, ein Intervall von $[0, 1]$ auf. Der `clipThreshold`-Parameter liegt dabei zwischen 0 und $1,2$ und kann



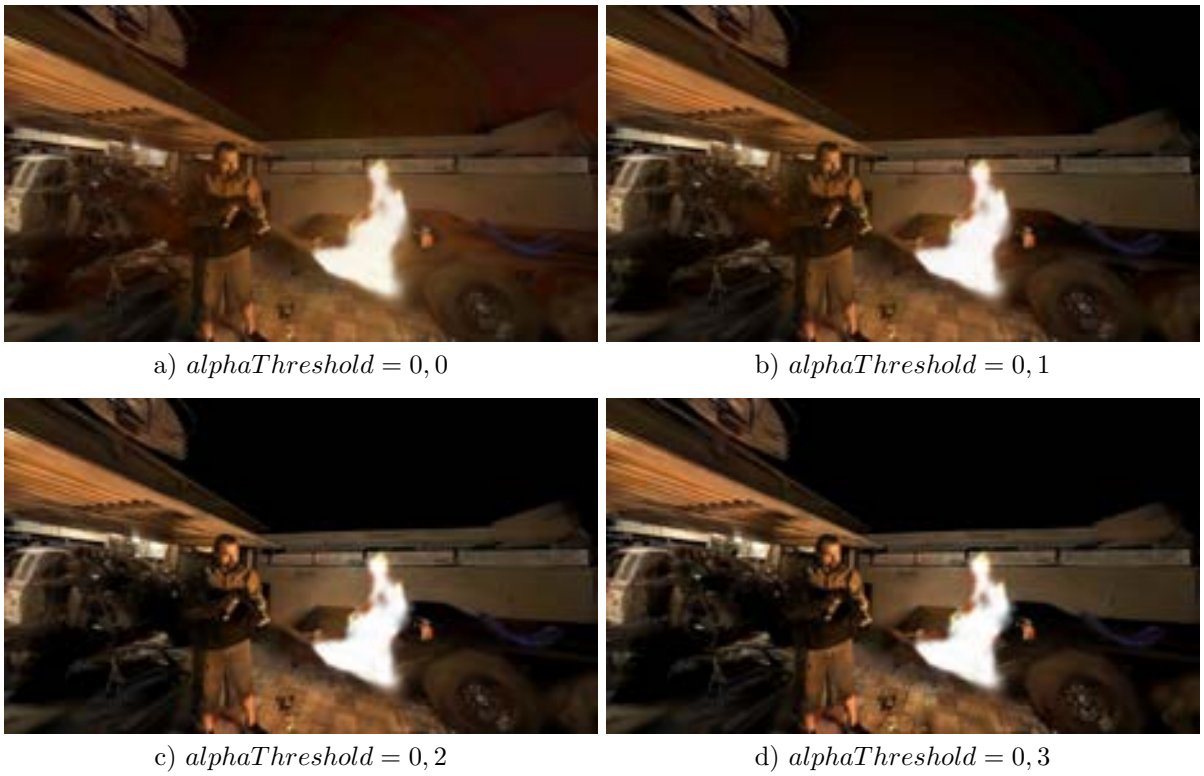
a) $clipThreshold = 1,2$ b) $clipThreshold = 1,0$

c) $clipThreshold = 0,8$ d) $clipThreshold = 0,6$

Abbildung 39: Foveated Splatting für verschiedene `clipThreshold` Parameter. Die Abbildung zeigt den FLAMES Datensatz aus (Broxton u. a. 2020).

über das Input Handling (siehe Abschnitt 4.3.3) zur Laufzeit angepasst werden. Ein visueller Eindruck dieses Ansatzes lässt sich in Abbildung 39 gewinnen.
In eigenen Tests auf der Meta Quest 3 wurde das Foveated Splatting für `clipThreshold`-Werte unter 1 teilweise kaum bis gar nicht wahrgenommen. Wird der Schwellwert jedoch zu klein gewählt, kommt es zu Doppelbildern in der Wahrnehmung, da der Bereich zwischen den Augen nicht mehr ordentlich durch beide Bilder abgedeckt wird. Eine einfache Abfrage könnte dies verhindern, indem für das linke Auge die Right-Ebene und für das rechte Auge die Left-Ebene nicht, oder nur zu einem bestimmten Grad, vom Culling betroffen sind.
Eine weitere Fragestellung ist, wie sehr die visuelle Qualität abnimmt und die Performanz steigt, wenn man Splats unter einem bestimmten Alphawert nicht rendert. Auch wenn diese Frage in dieser Arbeit nicht beantwortet werden kann, wurde ein Alpha Culling implementiert, welches Splats mit einem Alphawert unterhalb des `alphaThreshold` verwirft, welcher zwischen 0 und 1 liegt. Auch dieser lässt sich interaktiv über das Input Handling einstellen. Abbildung 40 zeigt das Alpha Culling für verschiedene `alphaThreshold`-Werte.

Dynamisches Viewport Scaling: Die Idee des dynamischen Viewport Scaling ist es, die Renderauflösung des volumetrischen Videos in Abhängigkeit von der FPS an die Renderleistung anzupassen. Gerade im Kontext von XR-Anwendungen ist eine hohe Bildrate notwendig, um ein angenehmes Benutzererlebnis zu gewährleisten und Erscheinungen wie Motion Sickness zu vermeiden. Die `XRViewport`-Klasse von WebXR stellt dafür die Methode `requestViewportScale()` zur Verfügung, die einen Skalierungsparameter übergeben bekommt. Um diesen an die derzeitige und die gewünschte Framerate anzupassen, wurde im `VolumanXRHandler` die Funktion



a) $alphaThreshold = 0,0$ b) $alphaThreshold = 0,1$

c) $alphaThreshold = 0,2$ d) $alphaThreshold = 0,3$

Abbildung 40: Alpha Culling für verschiedene `alphaThreshold` Parameter. Die Abbildung zeigt den FLAMES Datensatz aus (Broxton u. a. 2020).

`scaleFrameBuffer()` implementiert, die sowohl für den WebGL- als auch für den WebGPU-Viewer funktioniert. Für letzteren müssen jedoch zusätzlich die `OffscreenCanvas`-Objekte des `GPU2GLHelper` mit skaliert werden.

4.3.3 UI und Steuerung

Beteiligte Teammitglieder: Alisa Rüge

Für ein gutes Nutzererlebnis ist eine intuitive und benutzerfreundliche Steuerung wichtig. Damit der Nutzer sich in der Anwendung und mit der Steuerung zurecht findet, bedarf es ein UI. In diesem Kapitel wird auf diese beiden Komponenten des Viewers eingegangen.

Integration und Rendering der UI
Aus gesundheitlichen Gründen wurde das UI und die Steuerung als letzter Bestandteil zum Viewer hinzugefügt. Daher stellte es eine besondere Herausforderung dar, das UI in die bereits bestehende, sehr umfangreiche und komplexe Anwendung zu integrieren. Die Szene wurde schließlich in einen `OffScreenCanvas` gerendert, analog zum Vorgehen mit dem WebGPURenderer in Kapitel 4.3.2 Rendering.

In der Klasse `VolumanXRHandler.js` erfolgt das Rendering des `UIRenderers` innerhalb der Methode `onXRFrame`, ein Frame-Callback, der im Rahmen der WebXR-API über `xrSession.requestAnimation` registriert wird. Sie wird automatisch bei jedem Frame der XR-Sitzung aufgerufen und ist für das kontinuierliche Aktualisieren und Rendern der XR-Inhalte zuständig Group 2024a. Für jedes Auge wird zunächst der entsprechende `viewport` aus dem `XR-Frame` ermittelt. Ein `viewport` beschreibt dabei den spezifischen Ausschnitt auf dem Rendercanvas, in den das Bild für das linke oder rechte Auge gezeichnet werden soll. In stereoskopischen XR-Anwendungen erhält jede XRView einen separaten XRViewport, um die Perspektive jedes Auges korrekt darzustellen. Die Viewports werden über die getViewport()-Methode der XRWebGLLayer-Klasse bereitgestellt Group 2024b. Da die Benutzeroberfläche für beide Augen simultan dargestellt werden muss – jeweils nebeneinander für das linke und rechte Auge – wird die Gesamtbreite des `canvas` verdoppelt. Entsprechend wird die Größe so angepasst, dass beide Viewports vollständig abgebildet werden können (siehe Code 3).

```
1    for (let viewIndex = 0; viewIndex < pose.views.length; viewIndex++) {
2      const view = pose.views[viewIndex];
3      const viewport = xrSession.renderState.baseLayer.getViewport(view);
4
5      width += viewport.width;
6      if (viewIndex === 0)
7        height = viewport.height;
8    }
9    uiRenderer.renderer.setSize(width, height, false);
```

Code 3: Setzen der Größe des Canvas in VolumanXRHandler.js

Anschließend erfolgt das Rendering für jedes Auge separat mittels einer `for`-Schleife (siehe Code 4). Da Three.js in diesem Fall keine eigene `XRSession` verwaltet und diese auch nicht extern übergeben werden kann, beispielsweise aus der `VolumanXRHandler.js`-Klasse, wird nur ein Auge gerendert. Um dennoch die Benutzeroberfläche für beide Augen in XR darzustellen, muss das UI-Rendering explizit für jedes Auge durchgeführt werden. Falls das UI aktiviert ist (`showTips=true`), wird die Methode `renderView` der `UIRenderer.js`-Klasse aufgerufen. In die-

sem Schritt werden die Position und Orientierung des jeweiligen Auges übernommen und auf die Kamera angewendet. Zusätzlich wird ein `viewport`-Objekt an der Kamera gespeichert, das definiert, an welcher Stelle im `canvas` die jeweilige Ansicht gerendert werden soll. Dies ist notwendig, um sicherzustellen, dass das Bild für das linke und das rechte Auge korrekt nebeneinander auf dem `canvas` platziert wird. Der `UIRenderer` wird anschließend auf denselben `viewport` gesetzt. Abschließend erfolgt das Rendering der Szene mit der konfigurierten Kamera innerhalb des zugewiesenen `viewport` – also das Rendering der Benutzeroberfläche für genau ein Auge. Die Methode `renderView` wird innerhalb der `VolumanXRHandler.js`-Klasse im Rahmen der `for`-Schleife wie oben erwähnt zweimal aufgerufen – einmal für jedes Auge beziehungsweise jeden `viewport`, das heißt das UI wird für jedes Auge gerendert und somit korrekt in XR dargestellt.

```
    for (let viewIndex = 0; viewIndex < pose.views.length; viewIndex++) {
      const view = pose.views[viewIndex];
      const viewport = xrSession.renderState.baseLayer.getViewport(view);

      if (viewport.width === 0)
        continue;

      if(inputHandler.showTips){
        uiRenderer.renderView(view, viewport);
      }
    }
```

Code 4: Rendering der beiden Augen in VolumanXRHandler.js

Da das UI-Rendering letztlich mit erheblichem Aufwand verbunden war, werden im Abschnitt Probleme und Lösungsansätze (siehe Abschnitt 4.3.3) am Ende dieses Kapitels weitere Ansätze sowie die Gründe für ihr Scheitern näher erläutert.

UI

Bevor näher auf das UI und die Steuerung eingegangen wird, lohnt es sich die Überishct der Codestruktur in Abbildung 41 anzuschauen.

Für ein intuitives Nutzererlebnis im Bezug auf die Steuerung, ist eine geeignete Benutzeroberfläche erforderlich. Für die Umsetzung bietet sich das Framework Three.js an – eine JavaScript-basierte 3D-Bibliothek, die eine Vielzahl an Funktionen für die Darstellung und Interaktion in dreidimensionalen Szenen bereitstellt (Three.js contributors 2024a). Darüber hinaus unterstützt Three.js die WebXR-Spezifikation nativ und stellt mit dem `WebXRManager` eine integrierte Schnittstelle zur Verfügung, die eine nahtlose Integration von XR-Geräten sowie die Verwaltung der `XR-Session` ermöglicht (Three.js contributors 2024b). Die Umsetzung des UI erfolgt in der Klasse `UIRenderer.js`. Da die Vorgehensweise zum Rendering der Szene bereits im vorherigen Abschnitt ausführlich erläutert wurde, wird an dieser Stelle nicht erneut auf die Methode `renderView()` dieser Klasse eingegangen.

Zunächst wird in der Methode `_startScene()` eine Three.js-Szene initialisiert, bestehend aus einer `THREE.Scene`, einer `THREE.PerspectiveCamera` und einem `THREE.WebGLRenderer`. Diese grundlegende Struktur bildet die Basis für die Darstellung sämtlicher UI-Elemente innerhalb der XR-Umgebung. Für weiterführende Informationen zur Konfiguration und Verwendung von

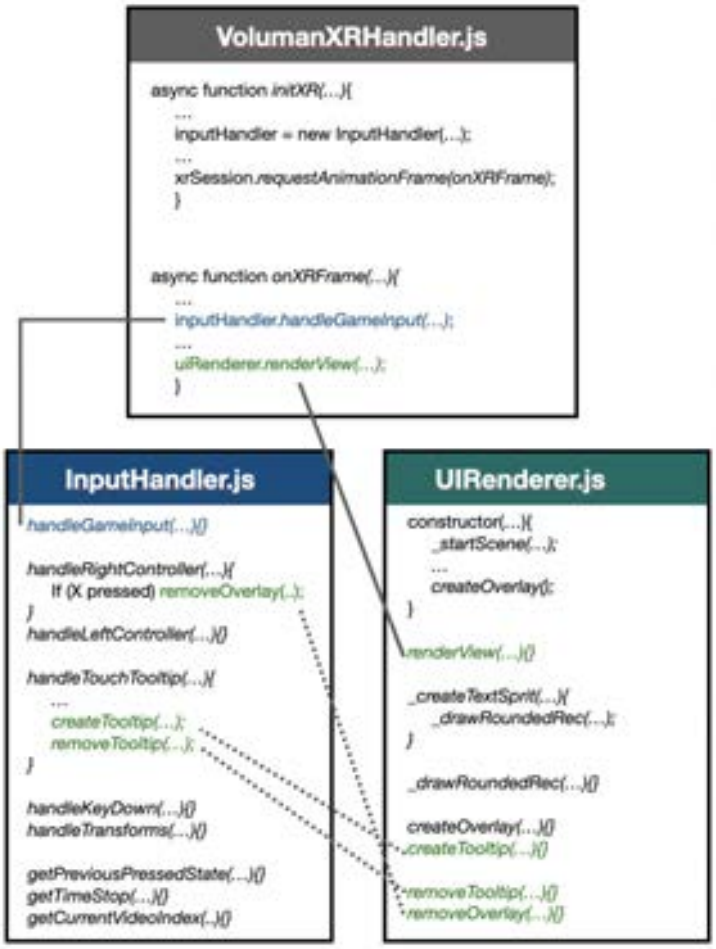


Abbildung 41: Übersicht Codestruktur UI und Steuerung

Three.js sei auf das offizielle Three.js Manual und Dokumentation verwiesen (Three.js contributors 2024c; Three.js contributors 2024d). Des Weiteren wird in der Methode `_startScene()` die Eigenschaft `this.renderer.xr.enabled` auf `true` gesetzt, wodurch die Szene für XR-Anwendungen aktiviert wird.

Es wird die Methode `createOverlay()` aufgerufen, welche zu Beginn der Anwendung ein UI-Element mit einer kurzen Einführung darstellt (siehe Abbildung 42). Auf eine detaillierte Erläuterung dieser Methode wird an dieser Stelle verzichtet, da ihre Logik derjenigen der Tooltips entspricht, welche im weiteren Verlauf näher beschrieben wird.

Um nun einen Tooltip zu erzeugen, wird die Methode `createTooltip()` aufgerufen, welche wiederum das Ergebnis der Methode `_createTextSprite()` zurückgibt. Dort bildet sich die grafische Grundlage für sämtliche UI-Elemente der Anwendung. In dieser Methode

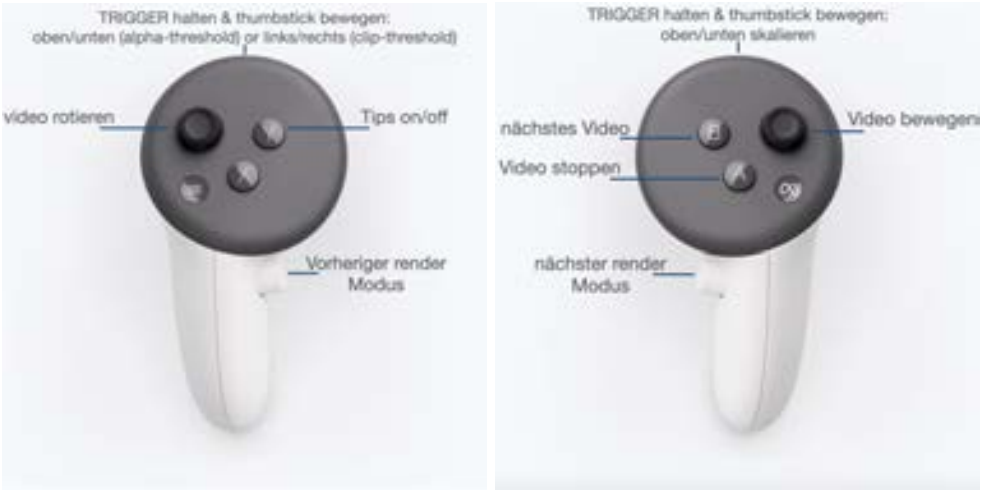
Abbildung 42: UI-Element mit Einführungstext

wird zunächst ein HTML-`canvas` erzeugt, auf dem ein Hintergrund mit abgerundeten Ecken gezeichnet und der gewünschte Text zentriert platziert wird. Die Methode `_drawRoundedRect()` ist dabei für das Zeichnen der abgerundeten Ecken verantwortlich und trägt wesentlich zur modernen und benutzerfreundlichen Gestaltung der UI-Elemente bei. Sie wird an dieser Stelle aufgerufen. Die Größe des `canvas` wird dabei in der `_createTextSprite()`-Methode dynamisch anhand der Textlänge, der Schriftgröße, Zeilenhöhe und des konfigurierten Paddings berechnet. Um eine gute Lesbarkeit zu gewährleisten, wird das `canvas` zusätzlich für das `devicePixelRatio` skaliert. Das fertige Element wird daraufhin in eine `THREE.CanvasTexture` umgewandelt und einem `THREE.Sprite` als Textur zugewiesen, das anschließend in die 3D-Szene eingefügt wird. Dabei richtet sich das THREE.Sprite automatisch zur Kamera aus und wird über einen Skalierungsfaktor an die gewünschte Größe angepasst. Durch dieses Verfahren können beliebige Texte dynamisch als Teil des 3D-UI dargestellt und skaliert werden und eignen sich somit ideal für die Anwendung in diesem Projekt. Zudem wird eine einfache, schnelle und modulare Erstellung von Tooltips ermöglicht, die sich unkompliziert in die Szene integrieren lassen. Da der Viewer in einer XR-Umgebung eingesetzt wird, ist der Hintergrund nicht festgesetzt – Farbe, Helligkeit und Inhalt der Szene können je nach Umgebung und Bewegung der Brille stark variieren. Deshalb folgt die visuelle Gestaltung der UI-Elemente einem klaren, gut lesbaren Design: Die Texte erscheinen in weißer Schrift auf einem leicht durchscheinenden, schwarzen Hintergrund, wodurch ein hoher Kontrast sowie gute Lesbarkeit gewährleistet sind – unabhängig vom dargestellten Szeneninhalt im Hintergrund (siehe Abbildung 44).
Darüber hinaus stehen zwei Methoden zur Verfügung, um UI-Elemente wieder aus der Szene zu entfernen: `removeTooltip()` und `removeOverlay()`. In beiden Fällen wird das entsprechende Sprite aus der Szene entfernt, ebenso das zugehörige Material sowie die Textur.
Wie ein Tooltip im Detail aussieht, zeigt Abbildung 44. Mit der Klasse `UIRenderer.js` ist nun die Grundlage geschaffen, um die einzelnen UI-Elemente visuell darstellen zu können. Im nächsten Schritt müssen die einzelnen Funktionen der Buttons implementiert und die Tooltips verwaltet werden. Dafür kommt die Klasse `InputHandler.js` zum Einsatz, die im folgenden Abschnitt näher erläutert wird.

Steuerung

Die Steuerung des Viewers erfolgt über die Klasse `InputHandler.js`. Sie verarbeitet Eingaben von den Quest-Controllern sowie der Tastatur am Desktop, um das volumetrische Video entsprechend zu steuern. Zusätzlich ist sie für die Erstellung und Platzierung der Tooltips verantwortlich. Der `InputHandler` wird in der Datei `VolumanXRHandler.js` erstellt. Seine zentrale Methode `handleGamepadInput` wird innerhalb der Funktion `onXRFrame` aufgerufen – und damit bei jedem Frame ausgeführt (siehe Abbildung 41).
Zunächst wird die aktive `XR-Session` abgefragt. Anschließend erfolgt eine Iteration über alle aktuell verfügbaren Eingabegeräte (`inputSources`). Für jedes Eingabegerät wird überprüft, mit welcher Hand es verbunden ist, woraufhin entweder die Methode `handleRightController()` oder `handleLeftController()` aufgerufen wird. Um die Zustände der Buttons - ob gedrückt oder berührt - zuverlässig prüfen zu können, werden die aktuellen Zustände der einzelnen Buttons pro Controller gespeichert. Diese Informationen werden im nächsten Frame für einen Vergleich benötigt, um Zustandsänderungen zuverlässig erkennen zu können.

Falls `showTips` aktiviert ist, das heißt das UI sichtbar ist, wird für jeden Button mit zugeordneter Funktion die Methode `handleTouchTooltip()` aufgerufen. Im Anschluss erfolgt eine Zustandsüberprüfung der jeweiligen Buttons, indem der aktuelle Status mit dem des vorangegangenen Frames verglichen wird. Änder sich der Zustand, werden daraufhin die jeweiligen Funktionen ausgeführt. Auf eine detaillierte Beschreibung dieser Funktionen wird in diesem Abschnitt verzichtet, für nähere Informationen siehe Abschnitt 4.3.1. Die jeweilige Funktionszuordnung der Buttons ist schematisch in Abbildung 43 dargestellt. Die beschriebene Logik zur Verarbeitung von Buttonzuständen und UI-Interaktionen wird sowohl in `handleRightController()` als auch in `handleLeftController()` identisch umgesetzt.



Linker Controller Rechter Controller

Abbildung 43: Buttons mit den jeweiligen Funktionen (Bildquelle Controller: Meta Platforms, Inc. 2025)

Damit das UI und somit die einzelnen Tooltips korrekt dargestellt werden, übernimmt die Methode `handleTouchTooltip()` die zentrale Steuerung der Tooltip-Logik. Sie sorgt dafür, dass ein Tooltip beim Berühren eines Buttons erzeugt und dessen Position kontinuierlich aktualisiert wird. Wird der Button nicht mehr berührt, so wird der Tooltip wieder von der Szene entfernt. Die Position des Controller wird mithilfe von `xrFrame.getPose()` abgerufen. Sobald ein Button erstmalig berührt wird, erfolgt die Erzeugung eines neuen Tooltips durch den Aufruf der Methode `createTooltip()`. Diese Methode ist in der Klasse `UIRenderer.js` implementiert, da dort die Three.js-Szene verwaltet wird. Diese Klasse wurde bereits ausführlich im Abschnitt UI 4.3.3 beschrieben. Bleibt der Button weiterhin berührt, so wird der entsprechende Tooltip permanent angezeigt und dessen Position aktualisiert, damit er stets am jeweiligen Controller angezeigt wird. Um eine Überlappung mehrerer Tooltips zu vermeiden und somit die Lesbarkeit sowie das Nutzererlebnis zu verbessern, wird ein Offset hinzugefügt, welcher die einzelnen Tooltips den jeweiligen Buttons am Controller etwas genauer zuordnet, sodass diese nebeneinander und nicht aufeinander angezeigt werden (siehe Abbildung 44).
Das funktioniert jedoch leider nicht einwandfrei. Dieses Problem und ein möglicher Lösungsansatz werden im Absatz Probleme und Lösungsansätze am Ende dieses Kapitels näher erläutert. Sobald der Button nicht mehr berührt wird, wird der zugehörige Tooltip aus der Szene entfernt. Auch dieser Vorgang erfolgt innerhalb der `UIRenderer.js`-Klasse durch die `removeTooltip()` Methode.

Abbildung 44: Tooltips am rechten Controller

Darüber hinaus befindet sich die Methode `handleKeyDown()` für eine Steuerung über Tastatureingaben am Desktop in der `InputHandler.js`-Klasse. Somit kann über die Tasten W, A, S und D die Position des volumetrischen Videos im Raum verändert werden. Die Pfeiltasten ermöglichen zusätzlich eine Skalierung sowie Rotation des Videos. Mit der Taste X lässt sich die Wiedergabe stoppen, während über die Tasten N und M zum nächsten beziehungsweise vorherigen Video gewechselt werden kann. Diese Funktionalitäten erweisen sich insbesondere im Entwicklungsprozess am Desktop als äußerst hilfreich. Zudem stellt die Methode `handleTransforms` sicher, dass das volumetrische Video nur innerhalb bestimmter Grenzwerte skaliert wird, wodurch eine Über- oder Unterskalierung verhindert wird. Darüber hinaus enthält die `InputHandler.js`-Klasse zwei Getter-Methoden: `getTimeStop()` und `getCurrentVideoIndex()`. Ersteres gibt zurück, ob die Zeit beziehungsweise das Video im Moment gestoppt ist, Letzteres gibt den Index des aktuell geladenen Videos zurück.

Im Hinblick auf zukünftige Arbeiten an diesem Projekt ist es wichtig, sämtliche Funktionalitäten so dynamisch wie möglich zu gestalten. Insbesondere die Belegung der Buttons und deren Tooltips sollte flexibel und ohne tiefgreifende Codeänderungen dynamisch und modular erfolgen. Dies stellte ein zentrales Ziel der Entwicklung dar und konnte erfolgreich umgesetzt werden, wodurch spätere Anpassungen oder Erweiterungen ohne größeren Entwicklungsaufwand möglich sind.

Probleme und Lösunugsansätze

In diesem Abschnitt werden die während des Projektverlaufs aufgetretenen Herausforderungen systematisch analysiert. Dabei wird auf die jeweiligen Lösungsansätze eingegangen sowie deren Umsetzung und die Gründe für ihr Scheitern erläutert. Darüber hinaus erfolgt eine Bewertung des Ergebnisses des UI und der Steuerung, gefolgt von fundierten Vorschlägen zur Optimierung und Weiterentwicklung.

Die größte Herausforderung bestand darin, dass die beiden Komponenten – UI und Steuerung – aus gesundheitlichen Gründen erst in einer späten Phase des Projekts entwickelt und integriert werden konnten. Während sich die nachträgliche Implementierung der Steuerung als weitgehend unproblematisch erwies, führte die späte Einbindung des UI zu erheblichen Schwierigkeiten. Zum Zeitpunkt der Integration war das Projekt bereits zu einer umfangreichen und komplexen Anwendung herangewachsen, wodurch insbesondere das Rendering des UI mit zahlreichen tech-

nischen Problemen behaftet war.

Das zentrale Problem ergab sich aus der Entscheidung, Three.js für die 3D-Visualisierung zu verwenden. Die Wahl fiel auf dieses Framework, da es eine signifikante Zeitersparnis verspricht: im Entwicklungsprozess erspart man sich viel Low-Level-Programmierung und es sind grundlegende 3D-Primitiven sowie eine strukturierte Szenenverwaltung bereits vorhanden. Im Zusammenspiel mit WebXR stellte sich jedoch heraus, dass Three.js eine eigene `XRSession` benötigt. Dies liegt daran, dass Three.js intern einen eigenen `WebGLRenderer` verwendet, der automatisch einens `baseLayer` anlegt und mit einer neu erzeugten `XRSession` verknüpft. Diese kann nicht vom Main Game Loop, der `VolumanXRHandler.js`-Klasse, an die `UIRenderer.js`-Klasse übergeben werden. Dies ist wichtig, da WebXR ausschließlich auf dem mit der aktiven `XRSession` verknüpften `canvas` auf der Meta Quest rendert. Der ursprünglich verfolgte Ansatz, ein separates `canvas`-Element ausschließlich für die Three.js-Szene zu verwenden, erwies sich daher als nicht praktikabel. Zwar funktionierte dieser Ansatz im Chrome-Web-Emulator auf dem Desktop, jedoch nicht auf der Meta Quest.

Ein weiterer Lösungsansatz für das Rendering des UI besteht in der Verwendung von `XRQuadLayer` aus der WebXR Layers API. Diese ermöglichen die Darstellung zweidimensionaler Flächen im 3D-Raum und sind besonders für UI-Elemente geeignet. Ein `XRQuadLayer` ist eine flache, rechteckige Fläche ohne Tiefe, die im Raum positioniert und ausschließlich von vorne sichtbar ist (MDN Web Docs 2024).

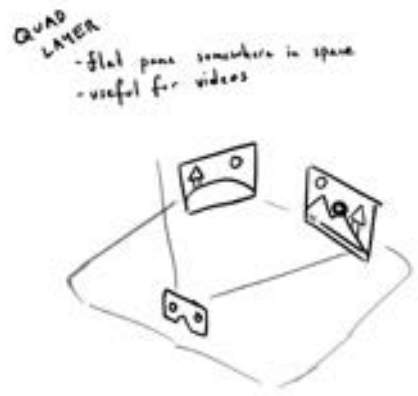


Abbildung 45: XRQuadLayer grafisch dargestellt, Bildquelle: Immersive Web Working Group 2024a

Dadurch ist das Rendering des volumetrischen Videos und des UI getrennt. Zudem bietet dieser Ansatz mehrere technische Vorteile: Die Lesbarkeit verbessert sich – insbesondere von Text – durch die Möglichkeit, UI-Elemente in hoher Auflösung darzustellen. Da die Layer nicht in jedem Frame neu berechnet werden müssen, ist eine höherer Performance und flüssigeres Rendering möglich. Darüber hinaus ermöglicht die späte Aktualisierung der Pose eine geringere Latenz, und die Unterstützung zusätzlicher Farbformate sorgt für eine verbesserte visuelle Qualität (Immersive Web Working Group 2024a).

Dieser Ansatz hätte somit eine ideale Lösung für das beschriebene Problem dargestellt. In der Theorie ermöglicht die WebXR Layers API die Übergabe eines Arrays von Layern im Rahmen der `XRSession.updateRenderState()`-Funktion, wodurch sowohl das UI als `XRQuadLayer` als auch das volumetrische Video in einem anderen Layer gleichzeitig dargestellt werden könnten. In der Praxis zeigte sich jedoch, dass der Framebuffer während des Renderings wiederholt überschrieben wurde. Dies führte dazu, dass stets nur entweder das UI oder das volumetrische Video, je nach Reihenfolge der Layer, sichtbar war – jedoch nie beides gleichzeitig. Aufgrund der fortgeschrittenen Projektphase und der limitierten verbleibenden

Entwicklungszeit konnte dieser Ansatz nicht mehr weiter erforscht und vollständig umgesetzt werden.

Ein weiterer vielversprechender Lösungsansatz war der Einsatz der WebXR DOM Overlay API (Immersive Web Working Group 2024b)), die es ermöglicht, klassische HTML-basierte Benutzeroberflächen direkt als Overlay im XR-Modus darzustellen. Die Idee bestand darin, die Three.js-Szene in einen separaten `canvas` zu rendern und diesen anschließend als DOM-Overlay einzubinden. Im Gegensatz zu renderbasierten Verfahren wie dem des `XRQuadLayers`, wird ein DOM-Overlay nicht in den Framebuffer geladen, sondern vom System als eigenständige Ebene über die XR-Szene gelegt. Auf diese Weise lassen sich Konflikte bei der Framebuffer-Nutzung vermeiden, wie sie beim gleichzeitigen Rendern von UI und volumetrischem Video in seperaten Layern auftraten. Obwohl dieser Ansatz in der Theorie vielversprechend war, konnte er im Rahmen des Projekts nicht erfolgreich umgesetzt werden. Das UI wurde nicht angezeigt. Die genaue Ursache für dieses Verhalten ließ sich nicht eindeutig feststellen. Auffällig ist jedoch, dass das Beispiel von Immersive Web Working Group 2024b keine Meta Quest als Anzeigegerät verwendet. Da – wie bereits zuvor beschrieben – die Meta Quest ausschließlich den `canvas` der aktiven `XRSession` rendert, ist es naheliegend, dass HTML-basierte DOM-Overlays außerhalb dieses Kontexts systemseitig nicht angezeigt werden. Dies könnte erklären, warum das Overlay trotz technisch korrekter Einbindung nicht sichtbar war.

Als Notfalllösung wurde in Erwägung gezogen, das UI separat vom Rendering des volumetrischen Videos aufzurufen – also mit einer eigenen `XRSession` und einem eigenen `canvas`. Dies hätte jedoch nicht dynamisch zur Laufzeit erfolgen können, sondern lediglich über einen expliziten Wechsel, beispielsweise durch einen HTML-Button oder einen dedizierten UI-Rendermodus innerhalb der `main.js`-Klasse – analog zu den bereits vorhandenen verschiedenen Rendermodi in Kapitel 4.3.2 Rendering. Diese Lösung ist jedoch aus gestalterischer wie auch aus benutzerzentrierter Sicht äußerst unvorteilhaft. Eine Benutzeroberfläche sollte integraler Bestandteil einer jeglichen Anwendung sein – insbesondere, wenn wie in diesem Projekt, eine Vielzahl von Funktionen vorhanden sind. Bei einem separaten Aufruf besteht die Gefahr, dass Nutzer den Überblick verlieren und sich nicht mehr erinnern, welcher Button welcher Funktion zugeordnet ist. Das manuelle Aktivieren der Benutzeroberfläche wäre in diesem Fall nicht nur umständlich, sondern würde auch die Bedienbarkeit und Benutzerfreundlichkeit der Anwendung erheblich beeinträchtigen.

Glücklicherweise konnte letztlich dennoch eine funktionierende Lösung gefunden werden: Durch das gezielte Rendering der Three.js-Szene in einen `OffscreenCanvas` mit Hilfe des `GPU2GLHelper` (siehe Kapitel 4.3.2 Rendering) ließ sich das UI korrekt und stabil in die Anwendung integrieren – auch wenn dies mit leichten Performanceeinbußen verbunden war.

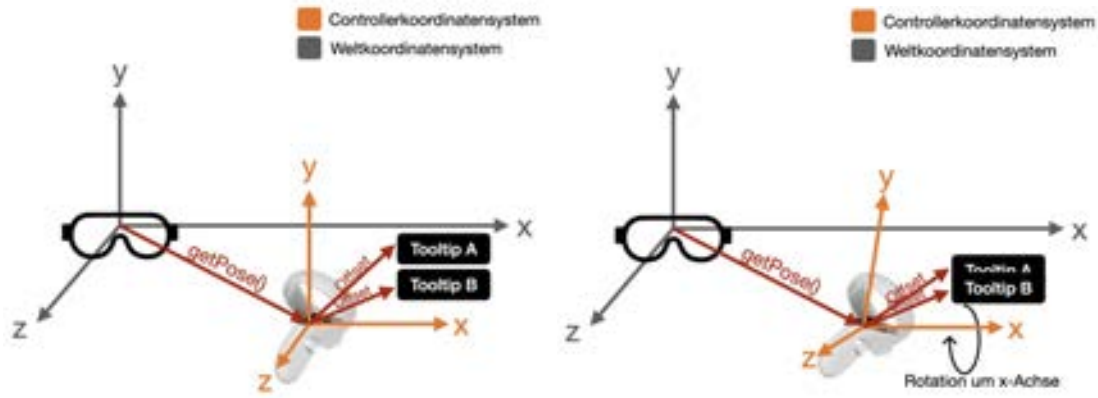
Da für das Rendering der UI viele verschiedene Lösungsansätze getestet wurden, blieb für kleinere Verbesserungen kaum noch Zeit. In der aktuellen Version des Viewers bekommt jedes Tooltip die Position des Controllers via `XRFrame.getPose()` übergeben. Darauf wurde ein fester Offset

in Form eines `THREE.Vector3` addiert:

```
// updates the position of the tooltips while button is touched
if (touched && this.currentTooltips[key]) {
const pose = xrFrame.getPose(grip, ref);
if (pose) {
const { x, y, z } = pose.transform.position;
this.currentTooltips[key].position.set(x, y, z).add(offset);
}
}
```

Code 5: Setzen der Tooltip-Position in VolumanXRHandler.js

Allerdings gibt es hierbei ein Problem: Wird der Controller nicht senkrecht vor dem Körper gehalten, sondern beispielsweise nach vorne geneigt, erscheinen die Tooltips nicht mehr an korrekter Stelle in der Nähe der Buttons. Stattdessen sind sie etwas verschoben oder überlappen sich zum Teil (siehe Abbildung 46). Leider wurde dieses Problem zu spät bemerkt.
Die Ursache liegt darin, dass der verwendete Offset direkt zur Position des Controllers im Weltkoordinatensystem addiert wird. Das Headset ist hier der Ursprung und die Position des Controllers wird als Vektor relativ zu diesem Ursprung angegeben. Die Methode `XRFrame.getPose()` liefert jedoch nicht nur die Position, sondern auch die Rotation des Controllers in Form eines Quaternions. `XRFrame.getPose()` erwartet zwei Argumente: ein `XRSpace`-Objekt, in diesem Fall `inputSource.gripSpace`, und einen `XRReferenceSpace`. Die Funktion gibt ein `XRPose`-Objekt zurück, das die Position und Orientierung des ersten Objekts relativ zum angegebenen Referenzraum beschreibt (MDN contributors 2024).



Situation vor Rotation Situation nach Rotation

Abbildung 46: Rotationsproblem grafisch dargestellt (Controller Bildquelle: 180by2 2025)

Um den Offset korrekt anzuwenden, müsste dieser relativ zur aktuellen Ausrichtung des Controllers erfolgen – also im lokalen Koordinatensystem des Controllers. Hierfür ist ein Koordinatenwechsel erforderlich: Der statische Offset-Vektor muss mithilfe der Rotation des Controllers in das Weltkoordinatensystem überführt werden. Dies entspricht mathematisch der Anwendung einer Koordinatenwechselmatrix, die die Transformation vom Controller-Koordinatensystem ins Weltkoordinatensystem beschreibt. Nur durch diese Transformation kann der Offset korrekt in Bezug zur tatsächlichen Haltung und Orientierung des Controllers angwendet werden, sodass die Tooltips zuverlässig an den richtigen Stellen angezeigt werden – unabhängig davon, wie der Controller im Raum gehalten oder geneigt ist.

Leider reichte die verbleibende Zeit nicht aus, um dieses Problem vollständig zu beheben. Dennoch ist die Einschränkung im praktischen Einsatz nicht zu drastisch, da in der Regel nicht alle Buttons gleichzeitig gedrückt werden und die Tooltips weiterhin am Controller selbst erscheinen und gut lesbar sind.

Die Umsetzung des UI und der Steuerung stellte eine Herausforderung im Projekt dar, insbesondere aufgrund der späten Integration in eine bereits komplexe und große Anwendung. Trotz technischer Hürden – etwa durch Einschränkungen der WebXR- und Three.js-Integration – konnte eine funktionierende, flexible und modular aufgebaute UI und Steuerung erfolgreich implementiert werden, welches zudem dynamisch und intuitiv ist. Somit wird eine stabile Grundlage für zukünftige Erweiterungen geboten. Es könnte beispielsweise in einem nächsten Schritt eine fest verankerte UI ergänzt werden – etwa zur Anzeige von Systeminformationen wie FPS oder der Name des aktuellen Videos. Auch das bekannte Problem der Tooltip-Überlappung bei Rotation der Controller sollte vor einer weiteren Optimierung behoben werden. Da hierfür lediglich eine Transformation des Offsets ins lokale Koordinatensystem des Controllers erforderlich ist, lässt sich diese Verbesserung mit überschaubarem Aufwand umsetzen. Insgesamt stellt die entwickelte Lösung ein solides Fundament für eine weiterentwickelbare und benutzerfreundliche XR-Anwendung dar.

4.4 HAC

Beteiligte Teammitglieder: Marvin Winkler

Die Umsetzung konzentrierte sich auf die Optimierung der Animatable Gaussians-Implementierung. Die ursprüngliche Version erreichte zwar beeindruckende visuelle Qualität, litt jedoch unter übermäßig langen Trainingszeiten, die selbst auf High-End-Hardware nicht praktikabel waren. Ein vollständiges Training hätte auf der verfügbaren RTX 3060 Grafikkarte etwa elf Tage gedauert. Die vorgenommenen Optimierungen zielten darauf ab, die Trainingszeit drastisch zu reduzieren, ohne dabei signifikante Qualitätseinbußen zu verursachen. Im Folgenden werden die wichtigsten Optimierungsbereiche, ihre Begründung und die resultierenden Vorteile im Detail erläutert.

4.4.1 Mixed Precision Training als zentrale Optimierungsstrategie

Der Kernansatz zur Verbesserung der Berechnungsgeschwindigkeit war die Implementierung von *Mixed Precision Training*. Diese Technik nutzt die Tatsache, dass moderne GPUs, insbesondere die neueren NVIDIA-Architekturen, bei Halbbyte-Präzision (FP16) deutlich schnellere Berechnungen durchführen können als bei voller Präzision (FP32).

Warum Mixed Precision Training? Mixed Precision Training bietet zwei entscheidende Vorteile: Erstens können GPUs bei FP16-Operationen einen deutlich höheren Durchsatz erreichen – auf neueren Architekturen oft den doppelten bis vierfachen Wert. Zweitens reduziert die Verwendung von FP16 den Speicherbedarf pro Tensor um 50%, was bei speicherintensiven Modellen, wie auch bei diesem Ansatz, besonders wichtig ist.

Implementierte Änderungen und ihre Vorteile

- **Integration von PyTorch AMP-Infrastruktur:** Durch die Einbindung von `GradScaler` und `autocast` kann das Netzwerk automatisch zwischen FP16 und FP32 wechseln – je nachdem, was für die jeweilige Operation numerisch stabiler ist. Dies ermöglicht eine Geschwindigkeitssteigerung ohne Präzisionsverluste.

- **Explizite Typkonvertierungen an kritischen Schnittstellen:** Um numerische Stabilität zu gewährleisten, wurden an kritischen Punkten im Code explizite Typkonvertierungen hinzugefügt:

  ```
  # Wichtige Änderung in UpFirDn2d
  kernel = kernel.to(input.dtype)

  # In FusedLeakyReLUFunction für Tensortyp-Konsistenz
  if not isinstance(scale, torch.Tensor):
      scale = torch.tensor(scale, dtype=input.dtype, device=input.device)
  ```

  Diese scheinbar kleinen Änderungen sind entscheidend, da Tensoroperationen mit gemischten Datentypen zu schwer zu diagnostizierenden Fehlern oder falschen Berechnungsergeb-

nissen führen können. Durch explizite Konvertierungen wurde die Stabilität des Trainings sichergestellt.

- **Gradientenskalierung für numerische Stabilität:** Die Implementierung von `scaler.scale(total_loss).backward()` und `scaler.step(self.optm)` verhindert das "Gradient UnderflowProblem, das bei FP16-Training auftreten kann. Dies ermöglicht stabile Gradientenaktualisierungen auch bei sehr kleinen Gradientenwerten.

Erzielte Verbesserungen Die Mixed Precision-Optimierungen führten zu einer etwa 40-45% schnelleren Ausführung pro Iteration und reduzierten den GPU-Speicherbedarf um ca. 35%.

4.4.2 Netzwerkarchitektur-Optimierungen

Die *DualStyleUNet*-Architektur stellt einen der rechenintensivsten Teile des Modells dar und wurde gezielt optimiert, um Berechnungseffizienz und Speicherbedarf zu verbessern, ohne die Ausdruckskraft des Modells signifikant zu beeinträchtigen.

Warum Netzwerkarchitektur optimieren? Die ursprüngliche Implementierung verwendete übermäßig breite Netzwerke mit einer großen Anzahl von Kanälen, was zu einer hohen Parameteranzahl und entsprechend hohem Speicherbedarf führte. Diese Überparametrisierung war für die Aufgabe nicht notwendig und führte zu längeren Berechnungszeiten ohne entsprechenden Qualitätsgewinn.

Durchgeführte Änderungen und ihre Vorteile

- **Reduktion der Kanaldimensionen:** Die Kanalanzahl wurde in allen Netzwerkebenen signifikant reduziert, jedoch in unterschiedlichem Ausmaß je nach Auflösungsstufe:

```
Geändert von:
{4: 512, 8: 512, 16: 512, 32: 512,
64: 256 * channel_multiplier(2), 128: 128 * channel_multiplier(2), ...}
Zu:
{4: 384, 8: 384, 16: 384, 32: 256,
64: 128, 128: 128, ...}
```

  Diese gestaffelte Reduktion bedeutet: In den tieferen Schichten (niedriger Auflösung) wurde die Kanalzahl nur um etwa 25% reduziert (512 → 384), während in den mittleren Auflösungen eine Reduktion um bis zu 75% erfolgte (z.B. bei 64×64 von 512 (256×2) auf 128). Zusätzlich wurde der `channel_multiplier`-Parameter, der vorher die Kanalanzahl in mittleren und höheren Auflösungen verdoppelte, komplett entfernt.

  Die Berechnungskomplexität in Faltungsschichten skaliert quadratisch mit der Anzahl der Kanäle. Bei einer Reduktion von Eingangs- und Ausgangskanälen um 25% in tiefen Schichten ergibt sich eine Parametereinsparung von etwa 44%, während eine Reduktion um 75% in mittleren Auflösungen zu einer Parametereinsparung von etwa 94% führt. Diese drastischen Reduzierungen in mehreren Schichten und über drei DualStyleUNet-Instanzen

(`color_net`, `position_net`, `other_net`) hinweg erklären die Gesamtreduktion der Parameteranzahl des AvatarNet um etwa 70%.

Erzielte Verbesserungen Die Netzwerkarchitektur-Optimierungen führen zu einer Reduzierung der Parameteranzahl um insgesamt ca. 70%, einer Beschleunigung der Vorwärts- und Rückwärtsdurchläufe um etwa 30% und einer Verringerung des Speicherbedarfs um ca. 70%, wobei die visuelle Qualität der Ergebnisse nahezu unverändert bleibt. Da die DualStyleUNet-Komponenten den Großteil der Parameter des gesamten AvatarNet-Systems ausmachen, hat diese Optimierung den größten Einfluss auf den Speicherbedarf.

4.4.3 Optimierte Trainingsparameter für beschleunigte Konvergenz

Ein weiterer kritischer Optimierungsbereich betraf die Trainingsparameter. Die ursprünglichen Parameter waren sehr konservativ gewählt, was zu einer unnötig langsamen Konvergenz führte. Durch gezielte Anpassungen konnte die Trainingszeit drastisch reduziert werden.

Warum Trainingsparameter optimieren? Die ursprüngliche Implementierung verwendete extrem niedrige Lernraten und eine sehr hohe Anzahl von Iterationen (800.000), was zu unnötig langen Trainingszeiten führte. Eine Analyse des Trainingsverlaufs zeigte, dass das Modell bereits viel früher konvergierte, aber mit den konservativen Parametern zu langsam lernte.

Durchgeführte Änderungen und ihre Begründung

- **Beschleunigter Trainingsablauf:**
  - Die Gesamtiterationen wurden von 800.000 auf 50.000 reduziert (94% weniger)
  - Die Position-Learning-Rate-Schritte wurden von 30.000 auf 15.000 halbiert
  - Der Learning-Rate-Delay-Multiplikator wurde von 0,01 auf 0,02 erhöht

  Diese drastische Reduzierung war möglich, da Experimente zeigten, dass mit optimierten Lernraten die Konvergenz viel früher erreicht wurde. Der erhöhte Delay-Multiplikator ermöglichte ein schnelleres Hochfahren der effektiven Lernrate zu Beginn des Trainings. Auch bei diesen Einstellungen führt eine Iterationszahl > 50.000 noch zu visuell besseren Ergebnissen, jedoch sind die Qualitätszugewinne ab diesem Punkt sehr gering.

- **Höhere Lernraten:** Die Lernraten wurden deutlich erhöht:
  - Positions-Lernrate: +56% (0,00016 → 0,00025)
  - Feature-Lernrate: +60% (0,0025 → 0,004)
  - Opazitäts-Lernrate: +60% (0,05 → 0,08)
  - Skalierungs-Lernrate: +60% (0,005 → 0,008)
  - Rotations-Lernrate: +200% (0,001 → 0,003)

Die höheren Lernraten waren entscheidend, um trotz der reduzierten Iterationsanzahl eine gute Konvergenz zu erreichen. Die neuen Raten wurden durch systematische Experimente ermittelt, wobei auf ein ausgewogenes Verhältnis zwischen Konvergenzgeschwindigkeit und Stabilität geachtet wurde.

- **Optimiertes Punktdichte-Management:** Die Densifikationsstrategie wurde grundlegend überarbeitet:
  - Densifikationsfrequenz: Verfünffacht (alle 100 → alle 20 Iterationen)
  - Gradientenschwelle: Um 75% gesenkt (0,0002 → 0,00005)
  - Opazitäts-Reset-Intervall: Um 33% reduziert (3000 → 2000)
  - Densifikationsstart: Verzögert (500 → 2000 Iterationen)

  Diese Änderungen waren notwendig, um die Punktverteilung schneller zu optimieren. Die höhere Frequenz und niedrigere Schwelle sorgen dafür, dass Punkte effizienter an relevante Stellen platziert werden. Der verzögerte Start ermöglicht dem Modell, zunächst die grundlegende Struktur zu lernen, bevor Details hinzugefügt werden.

- **Anpassung der Verlustgewichtungen:** Die Gewichtungen einzelner Verlustkomponenten wurden optimiert:
  - Erhöhung des Offset-Loss-Gewichts von 0,005 auf 0,1
  - Erhöhung des *LPIPS*-Gewichts von 0,1 auf 0,2 nach der Offset-Loss-Gewichtsänderung

  Die Erhöhung des Offset-Loss-Gewichts war notwendig, da die ursprüngliche Gewichtung zu gering war, um eine Konvergenz gegen Null zu erreichen. Die höhere Gewichtung verbessert insbesondere die strukturelle Konsistenz, was besonders in den Fingern sichtbar ist. Nach dieser Änderung wurde auch das LPIPS-Gewicht erhöht, um eine bessere Gesamtkonvergenz zu erzielen.

Resultierende Vorteile Die optimierten Trainingsparameter führen zu einer dramatischen Reduzierung der Trainingszeit um etwa 94%, ohne signifikante Qualitätseinbußen. Gleichzeitig wird die GPU-Auslastung besser verteilt und die Konvergenz beschleunigt. Bemerkenswert ist, dass durch das aggressivere Punktdichte-Management und die optimierten Verlustgewichtungen die Detailgenauigkeit trotz der reduzierten Trainingszeit nahezu erhalten bleibt.

4.4.4 Versuchte, aber nicht implementierte Optimierungen

Während des Optimierungsprozesses wurden verschiedene Ansätze getestet, die jedoch nicht in die finale Implementierung einflossen, weil sie entweder die visuelle Qualität zu stark beeinträchtigten oder keine signifikanten Leistungsvorteile boten.

Style-Dimension-Reduktion Die Reduzierung der Style-Dimensionen von 512 auf 64 wurde experimentell untersucht:

```
# self.color_net = DualStyleUNet(...style_dim = 64, n_mlp = 2)
```

Obwohl dieser Ansatz theoretisch zu einer Parameterreduktion geführt hätte, war der visuelle Einfluss zu stark, während der Leistungsgewinn relativ gering ausfiel. Die Fähigkeit des Modells, feine Texturen und Farbdetails zu modellieren, wurde durch die reduzierte Style-Dimension signifikant beeinträchtigt.

Downsampling der Eingabebilder Es wurde versucht, die Eingabebilder in ihrer Größe stärker zu reduzieren. Dafür wurde auch eine alternative Implementierung der Rauscheinspeisung entwickelt, die dem Downsampling der Eingabebilder entspricht.
Dieser Ansatz sollte Verarbeitungsdauer einzelner Bilder reduzieren. Jedoch führte diese Änderung zu einer zu starken Beeinträchtigung der visuellen Qualität bei minimalem Effizienzgewinn und wurde daher nicht weiterverfolgt.

Änderungen am View-Feature-Integrationspunkt Die Änderung des View-Feature-Integrationspunkts von `i == 8` zu `i == 4` wurde untersucht, um eine frühere Integration der Viewfeatures in das StyleUNet zu ermöglichen. Diese Änderung führte jedoch nicht zu einer Verbesserung der visuellen Qualität und wurde daher nicht in die finale Implementierung übernommen.

Batch-Verarbeitung Die Implementierung einer Batch-Verarbeitung (zuvor wurde nur eine Batch-Größe von 1 unterstützt, obwohl es bereits einen Batchsize-Parameter gab) erforderte umfangreiche Codeänderungen zur Hinzufügung der Batch-Dimension. Dieser Ansatz führte jedoch zu einem Memory Leak und wurde daher wieder entfernt.

Mini-Batches für Backpropagation Die Einführung von Mini-Batches für die Backpropagation wurde ebenfalls versucht, aber aufgrund zu hohen Speicherbedarfs und fehlendem Leistungsgewinn wieder verworfen.

4.4.5 „VolumanDataset“-Implementierung

Aufgrund der Unterschiede zwischen unseren extrahierten Kameraparametern und der Formatierung des AvatarReX-Datensatzes entwickelten wir eine maßgeschneiderte *"VolumanDataset"*-Implementierung. Diese spezialisierte Lösung ermöglicht das präzise Auslesen der extrinsischen und intrinsischen Parameter aus unserer `calibration.json`-Datei, ohne diese vorher umformatieren zu müssen.

4.4.6 Kernkomponenten-Optimierungen für Stabilität und Leistung

Mehrere kritische Komponenten des Systems wurden gezielt optimiert, um die Gesamtstabilität zu verbessern und Leistungsengpässe zu beseitigen.

Warum die Kernkomponenten optimieren? Bestimmte Teile des Systems, wie die Gaussian-Rasterisierung und die Transformation zwischen kanonischem und Live-Raum, stellen kritische Leistungsengpässe und potenzielle Fehlerquellen dar. Diese Komponenten wurden gezielt optimiert, um Stabilität und Geschwindigkeit zu verbessern.

Wichtigste Optimierungen und ihre Wirkung

- **Gaussian-Rasterisierungs-Optimierungen:**
  - Einführung expliziter Tensorkonvertierungen für Matrix-Eingaben
  - Standardisierung auf `dtype=torch.float32` in der Rendering-Pipeline
  - Verbesserung der numerischen Stabilität bei Matrixinversionen

  Diese Änderungen beseitigen numerische Instabilitäten, die zuvor zu fehlerhaften Renderings oder Abstürzen führen konnten. Die Standardisierung des Typs für das Rendering ist durch die Implementierung des Mixed Precision Trainings notwendig geworden.

- **Render-Methoden-Restrukturierung:** Die vollständige Reorganisation der Render-Methode verbessert nicht nur die Codequalität und Wartbarkeit, sondern optimiert auch den Ausführungspfad für häufige Anwendungsfälle, was zu einer leichten Performance-Steigerung bei Rendering-Operationen führte.

4.4.7 Speichermanagement- und Leistungsoptimierungen

Eine besondere Herausforderung bei der Arbeit mit 3D-Gaussian-Modellen ist der hohe Speicherbedarf. Gezielte Optimierungen in diesem Bereich sind entscheidend für die Skalierbarkeit des Systems.

Warum das Speichermanagement optimieren? Der Speicherbedarf war ursprünglich einer der limitierenden Faktoren für die Modellkomplexität. Da wir die Komplexität jedoch allgemein reduzieren konnten, war dies letztendlich kein Bottleneck mehr. Durch effizienteres Speichermanagement kann das System dennoch schneller trainieren.

Kernoptimierungen und ihr Nutzen

- **Intelligente Tensor-Wiederverwendung:**
  - Implementierung von Tensor-Recycling für wiederkehrende Operationen
  - Dynamisches Sampling oder Wiederholen bei Dimensionsunterschieden
  - Explizite Speicherfreigabe mit `del items` nach der Verarbeitung
- **Leistungsüberwachung und -optimierung:** Die Integration von Timing-Infrastruktur ermöglichte die Identifikation und gezielte Optimierung von Leistungsengpässen. Das erweiterte Logging mit Timing-Statistiken half dabei, problematische Komponenten zu identifizieren und die Optimierungsentscheidungen datenbasiert zu treffen.

Erreichte Verbesserungen Die Speicher- und Leistungsoptimierungen ermöglichen eine verbesserte Trainingsgeschwindigkeit durch effizientere Speicherallokation und reduzierten Overhead.

4.4.8 Gesamtauswirkung der Optimierungen

Die Kombination aller beschriebenen Optimierungen führte zu einer transformativen Verbesserung des Systems, die es von einem theoretisch interessanten, aber praktisch kaum nutzbaren Ansatz zu einem effizienten und praktisch einsetzbaren Tool machte.

- **Trainingszeit:** Durch die Kombination aus Mixed Precision Training, Netzwerkoptimierungen und verbesserten Trainingsparametern wurde die Trainingszeit von ursprünglich etwa elf Tagen auf wenige Stunden reduziert – eine Verbesserung um mehr als 95%.
- **Speichereffizienz:** Der reduzierte Speicherbedarf ermöglichte das Training auf Standard-Hardware und verbesserte die Stabilität der Trainingsläufe.
- **Qualität der Ergebnisse:** Trotz der drastischen Effizienzsteigerungen blieb die visuelle Qualität der erzeugten Avatare nahezu unverändert.

Diese umfassenden Optimierungen demonstrieren, wie durch gezielte Anpassungen in verschiedenen Systembereichen – von der Netzwerkarchitektur über Trainingsparameter bis hin zum Speichermanagement – die praktische Nutzbarkeit eines auf Gaussian Splatting basierenden Avatarsystems entscheidend verbessert werden kann, ohne signifikante Kompromisse bei der Qualität einzugehen.

4.4.9 Preprocessing- und Nachbearbeitungs-Pipeline

Neben den Optimierungen des Trainingsmodells selbst wurde eine umfassende Pipeline für die Vor- und Nachbearbeitung der Daten entwickelt. Diese Pipeline ist essentiell für die effiziente Verarbeitung der großen Datenmengen, die bei Multi-View-Aufnahmen anfallen. Zudem werden alle notwendigen Eingabedaten generiert, die von den HAC-Gaussians benötigt werden.

Nachbearbeitung der Rendering-Ergebnisse Um die gerenderten Einzelbilder zu nutzbaren Videosequenzen zusammenzufügen, wurde ein spezielles Script entwickelt:

- **Frame-zu-Video-Konvertierung:** Ein dediziertes Script konvertiert die sequentiell gerenderten Einzelbilder automatisch in ein mp4-Video mit konfigurierbaren Parametern für Framerate und Qualität. Dies ermöglicht eine unmittelbare visuelle Überprüfung der Trainingsergebnisse.

Umfassende Preprocessing-Pipeline für Rohdaten Für die Verarbeitung der Rohdaten wurde eine mehrstufige Preprocessing-Pipeline entwickelt, die speziell auf die Herausforderungen unseres Datensatzes mit 67 Kameras zugeschnitten ist:

- **Zeitliche Datenreduktion mittels "TimelapsScript:** Anders als bei den Beispieldatensätzen, die nur 16 Kameras nutzen, erfordert unser Setup mit 67 Kameras eine Reduktion in der zeitlichen Dimension bereits vor dem Training und den meisten Preprocessingschritten. Das entwickelte Script reduziert die Videodaten auf 1/30 oder 1/15 der ursprünglichen Framerate, um eine handhabbare Datenmenge zu gewährleisten.

- **Automatisiertes Videomatting:** Ein spezialisiertes Script führt das *RobustVideoMatting* (mobilenetv3) von Lin u. a. 2021 für jedes Video durch. Wir verwenden RobustVideoMatting, da die bei den Spacetime-Gaussians verwendete Lösung bei der Aufnahmelänge der Daten für die HAC-Gaussians zu langsam ist.
- **Strukturierte Datenorganisation für Colmap:** Das `colmap_setup`-Script organisiert und formatiert die vorverarbeiteten Daten in einer für Colmap optimierten Struktur.
- **Colmap-Prozessierungskette:** Als Nächstes wird eine Abfolge von Colmap-Operationen durchlaufen:
  - Feature-Extraktion aus allen Kamerabildern
  - Exhaustive-Matcher für robuste Feature-Korrespondenzen zwischen verschiedenen Ansichten
  - *Structure-from-Motion*-Mapping zur 3D-Rekonstruktion und Kamerakalibrierung
- **Konvertierung für EasyMocap:** Nach der Colmap-Verarbeitung werden die Daten in einem zweistufigen Prozess für *EasyMocap* aufbereitet:
  - Neuorganisation der Daten entsprechend der EasyMocap-Anforderungen
  - Extraktion und Formatierung der extrinsischen Kameraparameter aus den Colmap-Ergebnissen
- **Posenextraktion mit EasyMocap:** EasyMocap wird genutzt, um für jeden Frame die präzisen Körperposen zu extrahieren, was eine essentielle Grundlage für für den nächsten Schritt darstellt.
- **Generierung von Positionskarten:** Basierend auf den extrahierten Posen werden mit dem Script von Liu u. a. 2024 Positionskarten generiert, die für das Training des Animatable-Gaussians-Modells verwendet werden.

Diese umfassende Preprocessing-Pipeline ermöglicht die Verarbeitung komplexer Multi-View-Aufnahmen mit einer hohen Kamerazahl, was über die ursprüngliche Implementation hinausgeht und eine wesentliche Erweiterung für den praktischen Einsatz darstellt. Die Kombination aus optimiertem Trainingsmodell und effizienter Datenpipeline macht das System nun auch für Datensätze mit ungewöhnlich hoher Kameradichte praktikabel.

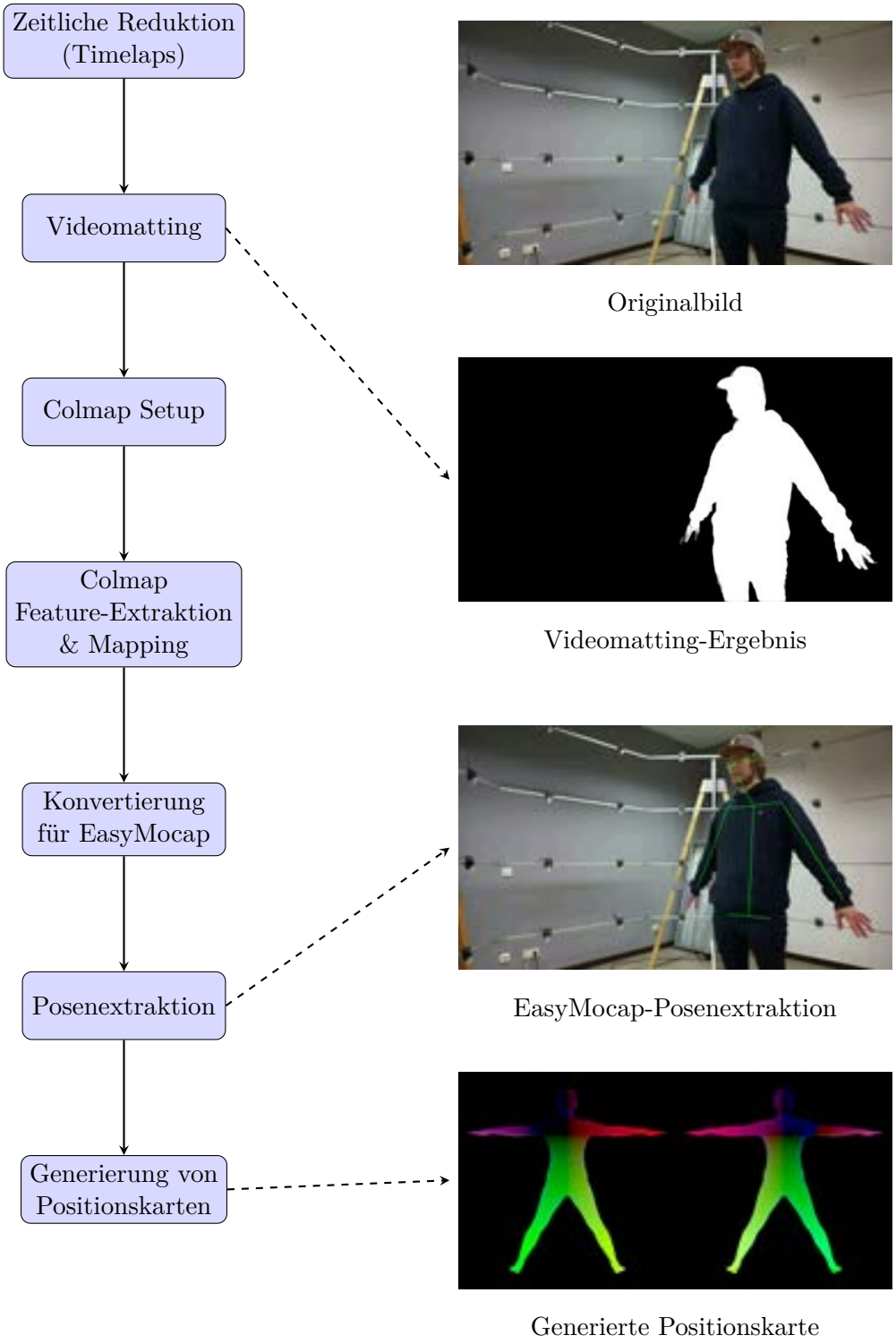


Abbildung 47: Visualisierung der vollständigen Preprocessing-Pipeline: Die Verarbeitung beginnt mit der zeitlichen Reduktion des Rohmaterials, gefolgt vom Videomatting zur Segmentierung der Person. Nach der Verarbeitung durch Colmap und Konvertierung der Daten erfolgt die Posenextraktion mit EasyMocap. Die generierten Positionskarten bilden schließlich die Grundlage für das Training der Animatable Gaussians.

5 Evaluation

5.1 Volumetric Capture System

Beteiligte Teammitglieder: Kai Altwicker

Um die Reproduktionsqualität des Volumetric Capture Systems beurteilen zu können, wurde während des gesamten Projektverlaufs, wie bereits erwähnt, das Programm Jawset Postshot in der Version 0.5.115 eingesetzt. Da das Rig Aufnahmen produzieren soll, die auch unabhängig von der in diesem Projekt entwickelten Verarbeitungspipeline nutzbar sind, war so eine Evaluierung durch ein externes Tool möglich. Dieser Abschnitt befasst sich mit der allgemeinen Bildqualität des Systems sowie mit Überlegungen zu Einstellparametern in Postshot. Eine detaillierte Evaluation der entwickelten Trainingsverfahren ist in Kapitel 5.2 aufgeführt.
Die Berechnungen erfolgten auf einem PC, der mit einer Nvidia GeForce RTX3090 und einem Intel i7-14700K ausgestattet ist. Die für die Berechnung benötigten Daten lagen auf einer 2TB NVMe, die eine Lesegeschwindigkeit von 5600MB/s bietet. Die Datensätze wurden mit einer Auflösung von 3840×2160 pro Bild aufgenommen. Für geringere Auflösungen kam die in Postshot integrierte Downsampling-Funktion zum Einsatz.
Postshot bietet zwei unterschiedliche Algorithmen zur Berechnung von Gaussian Splats an: Markow-Chain-Monte-Carlo (MCMC) und Additive Data Composition (ADC). Für die meisten Anwendungen wird MCMC als der bevorzugte Algorithmus empfohlen und ist standardmäßig aktiviert. Bei dieser Option lässt sich die maximale Anzahl an Gaussian Splats und damit direkt der benötigte VRAM begrenzen. Im Vergleich dazu arbeitet ADC ähnlich wie MCMC, erzeugt jedoch Details auf eine andere Art und Weise. Hier kann die Splat Density eingestellt werden: Ein Wert größer als 1 bewirkt, dass das Verfahren empfindlicher auf feine Details reagiert und somit mehr Splats generiert. (Jawset Visual Computing 2025) Für alle in Tabelle 5 dargestellten Tests wurde die Anzahl der Iterationsschritte auf 30.000 begrenzt, sofern nicht anders angegeben.

Test	**Algorithmus**	**Auflösung**	**Max Splat Count (MCMC)**	**Splat Density (ADC)**	**Accuracy (SSIM)**	**Anzahl Splats**	**Trainings-zeit**
1	MCMC	1600x900	3M		0,971	3M	15min
2	MCMC	3840x2160	3M		0,964	3M	40min
3	MCMC	3840x2160	25M		0,971	18,7M	100min
4	ADC	1600x900		1	0,966	432k	5min
5	ADC	1600x900		1,5	0,972	698k	10min
6	ADC	1600x900		3	0,978	1,39M	15min
7	ADC	3840x2160		1,5	0,968	1,02M	30min
8 (90kSteps)	MCMC	3840x2160	3M	0,976		3M	165min
9	MCMC	3840x2160	3M		0,964	3M	40min
10	MCMC	3840x2160	3M		0,962	3M	40min

Tabelle 5: Übersicht Postshot Benchmark

5.1.1 MCMC vs. ADC

Aufgrund ihrer verschiedenen Arbeitsweisen erzeugen MCMC und ADC auch eine unterschiedliche Anzahl an Gaussian Splats, was einen direkten Vergleich der Ergebnisse erschwert. Dennoch lassen sich anhand der Verarbeitungszeiten einige Rückschlüsse ziehen. Bei annähernd gleicher Berechnungsdauer, wie im Fall von Test 1 und 6, führen beide Methoden zu einer vergleichba-

Abbildung 48: Vergleich Ergebnisse MCMC vs ADC (Test 1, 4, 5 & 6)

ren visuellen Qualität (Abbildung 48). Zwar erzeugt ADC weniger als die Hälfte der Gaussian Splats im Vergleich zu MCMC, dennoch erscheinen die feinen Details minimal schärfer. Gleichzeitig weist die durch ADC generierte Gesamtszene jedoch deutlich mehr *Floating Artifacts* auf (Abbildung 49b & c). MCMC hingegen bildet die Gaussian Splats gleichmäßiger ab, was möglicherweise den Eindruck einer insgesamt etwas unschärferen Detailwiedergabe erzeugt.
Für schnelle Berechnungen, bei denen die visuelle Detailgenauigkeit weniger kritisch ist, können die schnelleren ADC-Verfahren genutzt werden, die bereits nach etwa fünf Minuten zufriedenstellende Ergebnisse liefern (Abbildung 49a).

(a) Detailreproduktion
(Test 1, 4, 5 & 6)

(b) Gesamtszene Test 1 (c) Gesamtszene Test 6

Abbildung 49: Vergleich Detail & Gesamtszene MCMC vs. ADC

Test 1 Test 2 Test 7

Abbildung 50: Vergleich Downsampled vs. volle Auflösung

5.1.2 UHD vs. Downsampled

Abbildung 50 zeigt, dass durch die Verwendung vollaufgelöster UHD-Bilder eine deutlich gesteigerte optische Qualität erreicht werden kann. Wird das Originalbild in gleicher Auflösung den von Postshot berechneten Gaussian Splats gegenübergestellt (Abbildung 51), wird erkennbar, dass Postshot das Originalbild, abgesehen von den feinsten Details in den Haaren, sehr gut nachbilden kann. Allerdings geht diese verbesserte Qualität mit einer deutlich gesteigerten Verarbeitungszeit einher. Eine weitergehende Detailverbesserung wird zudem letztlich durch das optische Auflösungsvermögen der verwendeten Kameras beziehungsweise Objektive begrenzt.

Realbild Gaussian Splats

Abbildung 51: Vergleich Kamera vs. Postshot (100% Ausschnitt)

5.1.3 Große Anzahl Splats vs. Viele Iterationsschritte

Für Abbildung 52 wurden zum einen die maximale Anzahl an Gaussian Splats erhöht(52b), zum anderen wurde in einem separaten Test die Erhöhung der Iterationsschritte untersucht (52c). Obwohl sich die Anzahl der Gaussian Splats auf 18,7 Mio. versechsfacht hat, konnte keine relevante Verbesserung der dargestellten Schärfe gegenüber dem Standardverfahren (52a) festgestellt

(a) Test 2 (b) Test 3 (c) Test 8

Abbildung 52: Vergleich Splat Anzahl vs. Iterationsschritte

werden. Lediglich sehr feine Haarstrukturen konnten besser dargestellt werden. Ebenso führt eine Verdreifachung der Iterationsschritte nur zu einer minimalen sichtbaren Qualitätssteigerung. Zwar erzielen beide veränderte Einstellungen einen leicht besseren SSIM-Wert, allerdings gehen diese Verbesserungen mit massiv verlängerten Verarbeitungszeiten einher. Aus diesen Ergebnissen lässt sich ableiten, dass für das vorliegende System die Einstellungen mit 3 Mio. Gaussian Splats und 30k Iterationsschritten in Kombination mit der vollen UHD-Auflösung als optimal betrachtet werden können.

5.1.4 Positionierung im Interpolationsvolumen

Um die positionsabhängige Reproduktionsfähigkeit zu testen, wurden gemäß Abbildung 53 Aufnahmen an drei unterschiedlichen Positionen durchgeführt. Position A (Test 2) entspricht der beabsichtigten Nutzung des Rigs, bei der die Person zentral in der Mitte steht und sich mit dem Großteil ihres Körpers im maximalen Interpolationsvolumen befindet, nur die Hände reichen leicht darüber hinaus. Dieser Test ist vergleichbar mit Aufnahmen, bei denen Personen entweder still in der Mitte stehen oder sich leicht innerhalb des grünen Bereichs bewegen. Position B (Test 9) repräsentiert die extremste Nutzung des Rigs, da hierbei das maximal nutzbare Interpolationsvolumen ausgeschöpft wird, wobei einzelne Elemente bereits außerhalb des Erfassungsrahmens liegen. Position C (Test 10) stellt das Worst-Case-Szenario dar, in dem sich die aufzunehmende Person vollständig außerhalb des Interpolationsvolumens befindet und nur noch von vereinzelten Kameras erfasst wird.

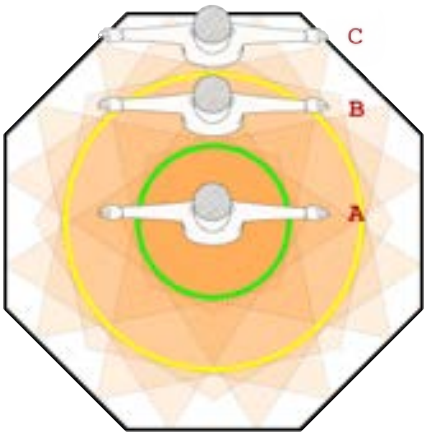


Abbildung 53: Positionierung im Rig

Abbildung 54a (Position A) zeigt, dass innerhalb des Interpolationsvolumens aus allen Perspektiven eine gute optische Güte erreicht werden kann und selbst Objekte, die sich leicht außerhalb des optimalen Volumens befinden, grundsätzlich gut reproduziert werden.
Für Position B verdeutlicht Abbildung 54b, dass aus der Front- und Rückansicht noch eine

ausreichende visuelle Qualität erzielt wird. Allerdings verschlechtert sich die Bildqualität für Objekte, die sich außerhalb des Interpolationsvolumens befinden, signifikant, sodass beispielsweise die Hände je nach Perspektive nicht mehr vollständig reproduziert werden können.
Dieser Trend setzt sich bei Position C (Abbildung 54c) fort, wobei mit Ausnahme der Frontperspektive in den übrigen Ansichten keine hinreichende Qualität erreicht wird, da zu viele Objekte aus dem Erfassungsbereich herausfallen.

(a) Ergebnisse Position A (Test 2)

(b) Ergebnisse Position B (Test 9)

(c) Ergebnisse Position C (Test 10)

Abbildung 54: Vergleich Position A, B & C

5.2 Training

Beteiligte Teammitglieder: Matthias Bullert

In diesem Abschnitt werden die Ergebnisse des Trainingsprozesses analysiert. Hierfür wird ein Datensatz bestehend aus 68 Videos verwendet, die mit den in Kapitel 4.1.2 beschriebenen Kameras aufgenommen wurden. Ein exemplarischer Ausschnitt des verwendeten Datensatzes ist in Abbildung 55 dargestellt.

Abbildung 55: Ausschnitt aus dem Testdatensatz

Wie in Kapitel 4.2.1 beschrieben, wird der Hintergrund bereits in der Bildebene entfernt. Abbildung 56 zeigt einen Ausschnitt des Datensatzes nach der Hintergrundentfernung.

Abbildung 56: Ausschnitt aus dem Testdatensatz nach der Hintergrundentfernung

Für die nachfolgenden Tests wird eine Kamera verwendet, die nicht im Trainingsprozess berücksichtigt wurde T. Li u. a. 2022. Diese Kamera wird im weiteren Verlauf als *Testkamera* bezeichnet, während alle anderen Kameras als *Trainingskameras* klassifiziert werden.

5.2.1 Trainingsparameter

Zur Optimierung der Modellleistung werden im Folgenden zentrale Trainingsparameter bestimmt. Die ermittelten Werte dienen als Grundlage für die weitere Evaluierung. Ziel ist es, eine optimale Balance zwischen Modellqualität, vertretbarer VRAM-Auslastung und akzeptabler Trainingszeit zu finden.
Die zu untersuchenden Parameter umfassen die Anzahl der Iterationen (Kapitel 5.2.1), die Anzahl der density control steps (Kapitel 5.2.1) sowie die Bildauflösung der Trainingsdaten (Kapitel 5.2.1). Durch eine gezielte Anpassung dieser Parameter soll eine effiziente Trainingskonfiguration gewährleistet werden.

Iterationen In dieser Testreihe wird die Anzahl der benötigten Iterationen zur Optimierung des Trainingsprozesses untersucht. Ziel ist es, zu analysieren, inwiefern eine höhere Anzahl an Iterationen zu einer Verbesserung der Ergebnisse führt und ob ein Plateau in der Fehlerreduktion erreicht wird.
Abbildung 57 zeigt die aus Sicht der Testkamera gerenderten Bilder nach verschiedenen Iterationsstufen. Zudem ist der Verlauf des berechneten Fehlers über die Iterationen hinweg dargestellt.

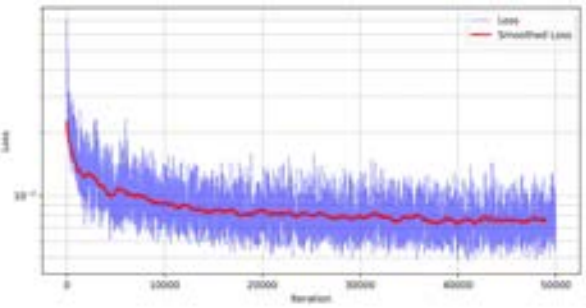
gerendertes Bild aus Sicht der Testkamera
Iterationen : 10000
Trainingsdauer : 2,5 Minuten
Durschnittlicher Fehler der letzten
100 Iterationen : 0.0092

gerendertes Bild aus Sicht der Testkamera
Iterationen : 20000
Trainingsdauer : 4,81 Minuten
Durchschnittlicher Fehler der letzten
100 Iterationen : 0.0082

gerendertes Bild aus Sicht der Testkamera
Iterationen : 30000
Trainingsdauer : 7,17 Minuten
Durschnittlicher Fehler der letzten
100 Iterationen : 0.0076

gerendertes Bild aus Sicht der Testkamera
Iterationen : 40000
Trainingsdauer : 9,58 Minuten
Durschnittlicher Fehler der letzten
100 Iterationen : 0.0076

gerendertes Bild aus Sicht der Testkamera
Iterationen : 50000
Trainingsdauer : 11,96 Minuten
Durchschnittlicher Fehler der letzten
100 Iterationen : 0.0076



Verlauf des Fehlers über 50000 Iterationen (blau)
Verlauf des gleitenden Durchschnitt des Fehlers
über 50000 Iterationen (rot)

Abbildung 57: Vergleich verschiedener Iterationen

Abbildung 57 zeigt den Fehlerverlauf über die gesamte Trainingszeit. Die blaue Kurve stellt den berechneten Fehler dar, während die rote Kurve die geglättete Version mittels eines gleitenden Durchschnitts (über die letzten 100 Werte) zeigt.

Die durchschnittlichen Fehlerwerte der letzten 100 Iterationen nehmen zwischen 10.000 und 30.000 Iterationen kontinuierlich ab. Allerdings ist ab 30000 Iterationen keine signifikante Verbesserung des Fehlers mehr erkennbar.
Es ist keine erkennbare Verbesserung des gerenderten Bildes, sowie des Fehlers nach 30000 Iterationen zu erkennen.
Basierend auf diesen Ergebnissen kann festgestellt werden, dass eine Erhöhung der Iterationen über 30000 hinaus keinen signifikanten Nutzen hinsichtlich der Fehlerreduktion bringt.

Density Control Steps

In diesem Abschnitt wird untersucht, wie sich die Anzahl der density control steps während des Trainings auf die Qualität der gerenderten Bilder, die Anzahl der Gaussians sowie die Trainingsgeschwindigkeit auswirkt. Wie in Kapitel 2.2 beschrieben, erfolgt die Anpassung der Dichte der Gaussians alle 100 Iterationen durch Teilung und Klonen. Dabei beeinflusst die Anzahl dieser Anpassungsschritte maßgeblich die Komplexität des Modells und die Effizienz des Trainings.

Zur Analyse wurden Modelle mit einer festen Anzahl von 30000 Iterationen trainiert, wobei die Anzahl der density control steps variiert wurde.
Für jede Testreihe wurden die resultierenden Bilder aus Sicht der Testkamera gerendert und die folgenden Metriken erfasst:

- **Trainingsdauer**
- **Maximale Anzahl der Gaussians** während des Trainings
- **Endgültige Anzahl der Gaussians** nach Abschluss des Trainings
- **Durchschnittlicher Fehler** der letzten 100 Iterationen

Die Ergebnisse sind in Abbildung 58 dargestellt.

gerendertes Bild aus Sicht der Testkamera
density control steps: 10
Trainingsdauer: 6,22 Minuten
Maximale Anzahl der Gaussians: 67545
Endgültige Anzahl der Gaussians: 24386
Durchschnittlicher Fehler der letzten
100 Iterationen: 0.0087

gerendertes Bild aus Sicht der Testkamera
density control steps: 20
Trainingsdauer: 7,26 Minuten
Maximale Anzahl der Gaussians: 96590
Endgültige Anzahl der Gaussians: 44865
Durchschnittlicher Fehler der letzten
100 Iterationen: 0.0081

gerendertes Bild aus Sicht der Testkamera
density control steps: 30
Trainingsdauer: 7,65 Minuten
Maximale Anzahl der Gaussians: 116863
Endgültige Anzahl der Gaussians: 56759
Durchschnittlicher Fehler der letzten
100 Iterationen: 0.0080

gerendertes Bild aus Sicht der Testkamera
density control steps: 40
Trainingsdauer: 7,99 Minuten
Maximale Anzahl der Gaussians: 135178
Endgültige Anzahl der Gaussians: 75412
Durchschnittlicher Fehler der letzten
100 Iterationen: 0.0076

gerendertes Bild aus Sicht der Testkamera
density control steps: 50
Trainingsdauer: 8,32 Minuten
Maximale Anzahl der Gaussians: 157098
Endgültige Anzahl der Gaussians: 86389
Durchschnittlicher Fehler der letzten
100 Iterationen: 0.0077

gerendertes Bild aus Sicht der Testkamera
density control steps: 60
Trainingsdauer: 8,74 Minuten
Maximale Anzahl der Gaussians: 171623
Endgültige Anzahl der Gaussians: 104274
Durchschnittlicher Fehler der letzten
100 Iterationen: 0.0074

Abbildung 58: Vergleich der Bildqualität bei unterschiedlichen density control stepsn.

Abb. 58 zeigt, dass eine geringe Anzahl von density control steps (5–30) zu einem erhöhten Fehler führt. Dies ist bedingt durch eine geringere Anzahl finaler Gaussians, wodurch, dass das Modell nicht ausreichend feine Details erfassen kann.
Mit zunehmender Anzahl der density control steps über 40 verbessert sich die Bildqualität, kaum noch. Allerdings nimmt die Trainingsgeschwindigkeit durch die höhere Anzahl an Gaussians ab. Die Ergebnisse zeigen, dass eine Anzahl von 40 density control stepsn einen optimalen Kompromiss zwischen Bildqualität und Trainingsgeschwindigkeit bietet.

Bildauflösung der Trainingsdaten

In diesem Abschnitt wird der Einfluss der Auflösung der Trainingsbilder auf die Qualität der generierten Ergebnisse sowie auf die Trainingsdauer untersucht. Ziel dieser Analyse ist es, eine

optimale Balance zwischen visueller Qualität und Trainingszeit zu identifizieren.

Das Training wurde mit vier unterschiedlichen Auflösungen der Trainingsbilder durchgeführt:

- **Niedrigste Auflösung**: 236 × 132
- **Mittlere niedrige Auflösung**: 471 × 265
- **Mittlere hohe Auflösung**: 942 × 529
- **Höchste Auflösung**: 1413 × 794

Für jede dieser Auflösungen wurde ein Modell trainiert und anschließend die Qualität der generierten Bilder aus Sicht der Testkamera analysiert. Zusätzlich wurde die benötigte Trainingsdauer dokumentiert.

Die resultierenden Bilder sind in Abbildung 60 dargestellt und die Trainingszeiten sind in Abb. 59 visualisiert.

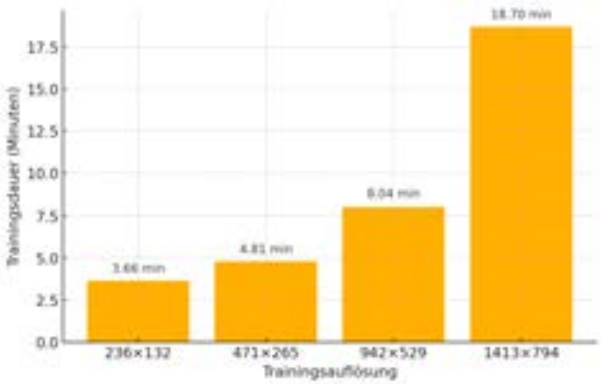


Abbildung 59: Trainingszeiten für verschiedene Auflösungen der Trainingsbilder

gerendertes Bild aus Sicht der Testkamera
Trainingsauflösung: 236 × 132
Trainingsdauer: 3.66 Minuten

gerendertes Bild aus Sicht der Testkamera
Trainingsauflösung: 471 × 265
Trainingsdauer: 4,81 Minuten

gerendertes Bild aus Sicht der Testkamera
Trainingsauflösung: 942 × 529
Trainingsdauer: 8,04 Minuten

gerendertes Bild aus Sicht der Testkamera
Trainingsauflösung: 1413 × 794
Trainingsdauer: 18,7 Minuten

Abbildung 60: Vergleich der generierten Bilder bei unterschiedlichen Auflösungen der Trainingsdaten

Die Ergebnisse zeigen, dass die beiden niedrigsten Trainingsauflösungen (236 × 132 und 471 × 265) zu deutlichen Artefakten in den generierten Bildern führen.
Die mittlere hohe Auflösung (942 × 529) liefert hingegen ein deutlich verbessertes visuelles Ergebnis, während die Trainingszeit im Vergleich zur höchsten Auflösung (1413 × 794) signifikant geringer ausfällt. Dies spricht für eine effiziente Nutzung der Rechenressourcen bei gleichzeitig hoher Bildqualität.
Die höchste getestete Auflösung (1413 × 794) liefert zwar potenziell noch detailliertere Bilder, jedoch steigt die Trainingszeit stark an, ohne einen entsprechend Gewinn an visueller Qualität zu erzielen.

5.2.2 Hintergrundentfernung

Trainingseinfluss
Um den Einfluss der Hintergrundentfernung auf das Training zu analysieren, wurden drei verschiedene Ansätze mit je 5 Frames über 30000 Iterationen getestet:

- **Ohne Hintergrundentfernung**: Das Modell wird mit den Originalbildern trainiert.
- **Mit Hintergrundentfernung (Dense-Matritzen)**: Der Hintergrund der Trainingsdaten wird entfernt und die daraus resultierenden Bilder werden mit Dense-Matritzen trainiert.
- **Mit Hintergrundentfernung (Sparse-Matritzen)**: Der Hintergrund der Trainingsdaten wird entfernt und die daraus resultierenden Bilder werden mit Sparse-Matritzen trainiert.

Die Ergebnisse dieser Tests sind in Tabelle 6 zusammengefasst. Die VRAM Auslastung wurde anhand der Windows internen *Game-Bar* ausgelesen.

	Ohne Hintergrundentfernung	**Mit Hintergrundentfernung (dense)**	**Mit Hintergrundentfernung (sparse)**
Trainingszeit	64,03 Minuten	10,57 Minuten	7,83 Minuten
VRAM-Auslastung	7.9 GB	7.9 GB	5.8 GB
Anzahl Gaussians	1575874	112389	105184

Tabelle 6: Vergleich der Trainingszeiten, der VRAM-Auslastung, der Dateigrößer der .ply Datein und der Anzahl der Gaussians mit und ohne Hintergrundentfernung

Die Ergebnisse zeigen, dass die Hintergrundentfernung einen erheblichen Einfluss auf die Trainingsdauer und die Fehlerreduktion hat. Während das Training ohne Hintergrundentfernung mehr als 1 Stunde in Anspruch nimmt, kann die Trainingszeit mit Hintergrundentfernung auf weniger als 8 Minuten reduziert werden. Auch ohne Verwendung von Sparse-Matritzen wird die Traingsdauer bereits auf 10,57 Minuten gesenkt, was allein durch die deutlich geringere Anzahl der Gaussians zu erklären ist. Ein weiterer positiver Effekt ist die geringere VRAM-Auslastung, durch die Verwendung von Sparse-Matrizen. Ein Visueller Vergleich der Ergebnisse ist in Abb. 61 zu sehen.

gerendertes Bild aus Sicht der Testkamera Mit Hintergrund

gerendertes Bild aus Sicht der Testkamera Ohne Hintergrund mit Dense-Tensoren

gerendertes Bild aus Sicht der Testkamera Ohne Hintergrund mit Sparse-Tensoren

Abbildung 61: Vergleich der generierten Bilder bei verschiedenen Trainingsansätzen

Qualität der Hintergrundentfernung

Die implementierte Hintergrundentfernung hat den Nachteil, das teilweise in Bilder Hintergrund vorhanden bleibt (siehe Abb. 63) oder Teile der Person entfernt wurden (siehe Abb. 62).

Abbildung 62: Links Bild mit Hintergrund | Rechts Bild ohne Hintergrund

Abbildung 63: Links Bild mit Hintergrund | Rechts Bild ohne Hintergrund

Dies führt zu löchern und fehlenden Teilen in der trainierten Szene. Wie in Abb. 64 an dem rechten Bein und dem Flaschenhals zu erkennen.

Abbildung 64: gerendertes Bild aus Sicht der Testkamera

5.3 Viewer

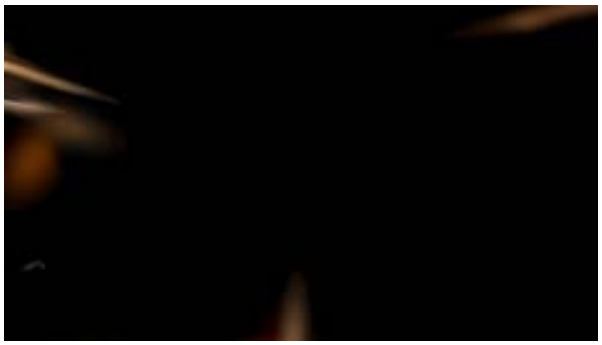
Beteiligte Teammitglieder: Steffen-Sascha Stein

Abbildung 65: Der Performance Heads-Up Display (HUD) Mode des OVR Metric Tools. Die Abbildung stammt aus (Meta 2025a).

In diesem Kapitel werden die Implementierungen der in Abschnitt 4.3.2 vorgestellten Viewer aufgrund von visuellen und zeitlichen Aspekten evaluiert. Vorab ist zu erwähnen, dass für die Meta Quest diverse Performance-Tools zur Verfügung stehen, von denen aber nur ein Bruchteil für das Profiling von WebXR-Anwendungen geeignet ist. Darunter sind beispielsweise das OVR Metrics Tool (Meta 2025a) und der Ovrgpuprofiler (Meta 2025b), welche für die Evaluation dieser Arbeit geplant waren. Ersteres verfügt über einen sogenannten Report Mode, welcher es ermöglicht, die Performance-Daten einer Session in eine CSV-Datei zu exportieren. Leider funktioniert dies aber nur für native und nicht für WebXR-Anwendungen. Zusätzlich stellt das OVR Metrics Tool einen Performance HUD Mode zur Verfügung, welcher Graphen und Metriken in Echtzeit über ein Overlay anzeigt, wie in Abbildung 65 zu sehen ist. Welche Daten angezeigt werden sollen, ist konfigurierbar, wobei der Großteil der GPU-spezifische Metriken erst angezeigt werden kann, wenn über den Ovrgpuprofiler GPU Profiling aktiviert wurde. Der Ovrgpuprofiler ist ein Command Line Interface (CLI) Tool für das Performance Monitoring von laufenden Anwendungen auf der Meta Quest. Dieser sollte genutzt werden, um mittels Render Stage Traces die Performanz der verschiedenen Ansätze präzise zu messen. Leider waren die gemessenen Ergebnisse für den in Kapitel 4.3.2 beschriebenen WebGPU-Viewer nicht interpretierbar. Diese könnte entweder daran liegen, dass der generelle Support von WebGPU gegenüber dem vom WebGL noch nicht ausgereift ist, oder dass das Zwischenspiel der beiden Application Programming Interfaces (APIs) durch den `GPU2GLHelper` das Performance-Tool überfordert. Auf dieser Grundlage wurde entschieden, dass die Performance-Analyse der GPU-Zeiten für WebGPU über Timestamp-Queries geschieht und für WebGL nicht ausgeführt wird. Es werden

a) Rotation der Y-Achse um 0°.

b) Rotation der Y-Achse um 90°.



c) Rotation der Y-Achse um 180°.

d) Rotation der Y-Achse um 270°.

Abbildung 66: Verschiedene Perspektiven des FLAMES Datensatzes (Broxton u. a. 2020) zur Evaluation. Das Modell wurde um -1 auf der Z-Achse verschoben, und unterscheidet sich für jede Perspektive nur in der Rotation um die eigene Y-Achse.

jedoch die FPS und die Framedauer für beide Viewer-Varianten gemessen, um einen groben Vergleich zu ermöglichen. Zusätzlich werden das in Kapitel 4.3.2 beschriebene Foveated Splatting und Alpha Culling in dieser Evaluation nicht erfasst, da ein Performanzgewinn nur in Relation zur Wahrnehmung des Betrachters aussagekräftig wäre. Dies würde die Ausführung einer Studie erfordern, die außerhalb des Rahmens dieser Arbeit liegt.

5.3.1 Versuchsaufbau

Die hier vorgenommene Evaluation wird auf der Meta Quest 3 mit dem Snapdragon XR2 Gen 2 und der darauf liegenden Adreno 740 GPU durchgeführt. Um WebGL- und WebGPU-Anwendungen ansatzweise miteinander vergleichen zu können, wurden die in (Jones 2025) beschriebenen Vorkehrungen getroffen. In dem Artikel wird unter anderem darauf hingewiesen, dass die Einstellungen für Antialiasing, Depth- und Stencil-Tests und Alpha-Blending in WebGL und WebGPU identisch sein sollten. Für Performanzmessungen wird die Anwendung in Virtual Reality (VR) gestartet und es wird auf Augmented Reality (AR) verzichtet, um einen Overhead durch das Passthrough-Rendering auszuschließen. Des Weiteren wird die Anwendung für vier verschiedene Perspektiven des FLAMES-Datensatzes aus (Broxton u. a. 2020) ausgewertet, welche in Abbildung 66 zu sehen sind. Während der Messungen wird das HMD in einer stabilen Haltung positioniert und nicht bewegt, um eine Reproduzierbarkeit dieses Versuchs zu gewährleisten. Alle gemessenen Werte werden jeweils über 1000 Frames erfasst und gemittelt.

Perspektive	API	Framedauer [ms]	FPS
0°	WebGL	83.29	12.01
	WebGPU	83.09	12.03
90°	WebGL	55.55	18.00
	WebGPU	59.70	16.75
180°	WebGL	42.33	23.62
	WebGPU	39.59	25.26
270°	WebGL	62.37	16.03
	WebGPU	63.25	15.81

Tabelle 7: Vergleich der Framedauer und FPS des WebGL- und WebGPU-Viewers. Zweiterer im Naive-Mode.

5.3.2 WebGL vs. WebGPU

Wie bereits erwähnt, können der WebGL- und WebGPU-Viewer in Bezug auf die Performanz nicht präzise miteinander verglichen werden. Es lassen sich jedoch die Zeiten zwischen den einzelnen Frames und die daraus resultierenden FPS messen. Diese werden für beide Viewer-Varianten in Tabelle 7 miteinander verglichen, wobei der WebGPU-Viewer im Naive-Mode läuft, da dieser mit der Implementierung des WebGL-Viewers identisch ist. Die gemessenen Werte liegen sehr nah beieinander und lassen keine signifikanten Unterschiede erkennen. Es ist jedoch anzumerken, dass der WebGPU-Viewer durch den `GPU2GLHelper` einen zusätzlichen Overhead durch das Weiterreichen der gerenderten Daten an WebGL hat. Außerdem profitiert das WebGPU-Rendering womöglich nicht von den Optimierungen, die für ein Rendering über die WebXR-APIs gewährleistet werden. Es ist also anzunehmen, dass der WebGPU-Viewer in der Zukunft eine bessere Performance aufweisen wird, wenn der Meta Quest Browser das WebXR-WebGPU-Binding supportet. Eine weitere Auffälligkeit war, dass wenn man für den WebGL-Viewer das Antialiasing aktiviert, dieser beinahe doppelt so schnell wird. Aufgrund der fehlenden Zeit und der nicht vorliegenden Erklärung für dieses Phänomen, wurde auf eine genauere Untersuchung verzichtet. Für den WebGPU-Viewer hingegen verlangsamt sich die Anwendung um wenige Millisekunden bei aktivem Antialiasing. Zudem scheint die WebGL-Variante die Auflösung der Anwendung zu beeinflussen, was intern und außerhalb der Reichweite der WebXR-API geschieht, da Letztere darauf hinweist, dass stets in voller Auflösung gerendert wird. Um dies zu veranschaulichen und gleichzeitig die visuelle Verbesserung durch den WebGPU-Viewer im `IndirectCullGPUSort`-Mode zu zeigen, wurde in Abbildung 67 eine Vergleichsaufnahme des FLAMES (Broxton u. a. 2020) und des Dennis PingPong Datensatzes – der aus dieser Arbeit stammt – dargestellt.

5.3.3 WebGPU Performance-Modes

Die Performance-Modes des WebGPU-Viewers werden in Tabelle 9 miteinander verglichen. Zudem werden in Tabelle 8 die Anzahl und der Anteil der gezeichneten und ausgeblende-

Perspektive	Drawn	Culled	Drawn [%]	Culled [%]
0°	110032	223792	32,96%	67,04%
90°	62031	271793	18,58%	81,42%
180°	40	333784	0,01%	99,99%
270°	21467	312357	64,31%	35,69%

Tabelle 8: Anzahl und Anteil gezeichneter vs. ausgeblendeter Objekte pro Perspektive

a) WebGL FLAMES. b) WebGPU FLAMES.

c) WebGL Dennis PingPong. d) WebGPU Dennis PingPong.

Abbildung 67: Visueller Unterschied zwischen dem WebGL- und WebGPU-Viewer. Zweiterer ist im `IndirectCullGPUSort`, welche mittels Radix Sort eine Sortierung der Splats auf der GPU ermöglicht. Die Bilder a) und b) zeigen den FLAMES Datensatz (Broxton u. a. 2020) und c) und d) den Dennis PingPong Datensatz, der aus dieser Arbeit stammt.

ten Splats pro Perspektive angegeben. Das ausschließlich für die Modi `IndirectCull` und `IndirectCullGPUSort` verwendete indirekte Rendering macht sich vor allem bei der Perspektive 180° in den Renderzeiten bemerkbar. Ansonsten konnten alle anderen Modi gegenüber dem Naive-Mode keine signifikante Verbesserung der Performanz aufweisen. Die Verlagerung der redundanten Arbeit des Vertex-Shaders in den Cull-Compute-Shader hat nicht den gewünschten Effekt erzielt und die Performanz verschlechtert. Dies lässt zum einen darauf schließen, dass die Anwendung nicht durch den Vertex-Shader, sondern durch den Fragment-Shader limitiert wird. Zum anderen könnte es daran liegen, dass Computer-Shader auf dem Snapdragon XR2 Gen 2 nicht dieselben Beschleunigungen erfahren wie Vertex- und Fragment-Shader, da der Chip darauf optimiert ist, Bilder für zwei Augen in Echtzeit zu rendern. Es ist auch nicht auszuschließen, dass die auf das Quad-Array ausgeführten Schreibzugriffe in `cullShader` und die Lesezugriffe in der Render-Pipeline inperformant sind.

Perspektive	Modus	Cull	Radix Sort	Prepare Scan	Scan	Compress	Render L / R	GPU Total
0°	Naive	–	–	–	–	–	28.66 / 28.73	57.40
	Cull	8.31	–	–	–	–	27.05 / 27.26	62.62
	IndirectCull	7.80	–	–	0.45	0.10	26.76 / 27.00	62.11
	IndirectCullGPUSort	8.12	18.95	0.05	0.29	0.04	26.55 / 26.75	80.75
90°	Naive	–	–	–	–	–	14.09 / 18.84	32.93
	Cull	9.47	–	–	–	–	11.65 / 16.34	37.46
	IndirectCull	12.90	–	–	0.69	0.11	11.81 / 16.66	42.16
	IndirectCullGPUSort	8.34	21.82	0.07	0.35	0.05	10.59 / 14.92	56.14
180°	Naive	–	–	–	–	–	5.97 / 5.67	11.64
	Cull	14.81	–	–	–	–	4.74 / 4.29	23.84
	IndirectCull	14.63	–	–	0.67	0.08	0.96 / 1.12	17.45
	IndirectCullGPUSort	11.74	33.22	0.11	0.58	0.07	0.89 / 0.87	47.46
270°	Naive	–	–	–	–	–	17.80 / 16.81	34.61
	Cull	9.55	–	–	–	–	15.75 / 14.30	39.60
	IndirectCull	12.57	–	–	0.70	0.10	15.13 / 13.49	41.99
	IndirectCullGPUSort	7.73	21.83	0.07	0.36	0.04	13.70 / 12.07	55.80

Tabelle 9: GPU-Zeiten der verschiedenen Performance-Modes pro Perspektive. Alle Angaben sind in Millisekunden und wurden über WebGPU Timestamp-Queries gemessen.

5.4 HAC

Beteiligte Teammitglieder: Marvin Winkler

Die durchgeführten Optimierungen wurden systematisch evaluiert, um ihren Einfluss auf Trainingsgeschwindigkeit, Renderperformanz und visuelle Qualität zu quantifizieren. Die Ergebnisse zeigen eine dramatische Verbesserung der Effizienz bei nahezu gleichbleibender visueller Qualität. Im Folgenden werden die wichtigsten Aspekte der Evaluation und die daraus gewonnenen Erkenntnisse detailliert dargestellt.

5.4.1 Trainings- und Renderperformanz

Der augenfälligste Erfolg der durchgeführten Optimierungen zeigt sich in der drastischen Reduzierung der Trainingszeit. Durch die Kombination aus Mixed Precision Training, Netzwerkoptimierungen und verbesserten Trainingsparametern konnte ein beeindruckender Beschleunigungsfaktor erzielt werden.

Trainingszeit

- **Ursprüngliche Version:** Ca. 11 Tage für 800.000 Iterationen auf einer RTX 3060
- **Optimierte Version:**
  - 8,3 Stunden für 50.000 Iterationen (Faktor 31,8×)
  - 16,6 Stunden für 100.000 Iterationen (Faktor 15,9×)

Diese extrem verkürzte Trainingszeit transformiert das System von einem rein akademischen Experiment zu einem praktisch einsetzbaren Werkzeug. Während die ursprüngliche Implementierung aufgrund der langen Trainingszeit selbst auf High-End-Hardware kaum praktikabel

war, ermöglicht die optimierte Version iteratives Arbeiten und schnelles Experimentieren auf Standard-Hardware.

Renderperformanz Neben der Trainingsgeschwindigkeit wurde auch die Renderleistung signifikant verbessert:

- **Ursprüngliche Version:** Etwas über 1 FPS
- **Optimierte Version:** Über 3 FPS (Steigerung um mehr als 200%)

Diese Verbesserung bringt das System deutlich näher an die Echtzeitfähigkeit heran und erweitert die potenziellen Anwendungsmöglichkeiten erheblich. Während 3 FPS noch nicht für flüssige Interaktionen ausreichen, stellen sie eine wesentliche Verbesserung dar und ermöglichen ein deutlich responsiveres Arbeiten mit dem System.

Speichereffizienz Die Optimierungen führten zudem zu einer substantiellen Reduktion des GPU-Speicherbedarfs:

- **Netzwerkarchitektur-Optimierungen:** Reduktion der Parameteranzahl um ca. 70%
- **Mixed Precision Training:** Weitere Reduktion des GPU-Speicherbedarfs um ca. 35%

Dieser reduzierte Speicherbedarf ist besonders bedeutsam, da er die Nutzung des Systems auf GPUs mit begrenztem Speicher ermöglicht und gleichzeitig die Stabilität des Trainings verbessert.

5.4.2 Visuelle Qualität

Ein entscheidender Aspekt bei der Evaluation von Optimierungsmaßnahmen ist die Frage, ob und inwieweit diese die Qualität der erzielten Ergebnisse beeinflussen. Detaillierte Vergleiche zwischen den mit dem ursprünglichen und dem optimierten System erzeugten Avataren zeigen, dass die visuellen Unterschiede minimal sind.

Qualitative Bewertung Die visuellen Unterschiede zwischen dem Original- und dem optimierten Ansatz sind bei subjektiver Betrachtung geringfügig und rechtfertigen in keiner Weise den erheblich höheren Berechnungsaufwand des ursprünglichen Systems. Insbesondere wurden folgende Aspekte evaluiert:

- **Detailgenauigkeit:** Die optimierte Version zeigt eine etwas verschlechterte Detailtiefe bei der Darstellung von Gesichtszügen, Kleidungsfalten und feinen Strukturen.
- **Farbtreue:** Trotz der reduzierten Netzwerkkomplexität bleibt die Farbwiedergabe nahezu identisch.
- **Animationsqualität:** Die Bewegungen wirken natürlich und fließend, ohne erkennbare Artefakte durch die vorgenommenen Optimierungen.
- **Stabilität über verschiedene Perspektiven:** Der Avatar behält seine visuelle Konsistenz auch bei wechselnden Betrachtungswinkeln bei.

Optimiert: nach 50.000 Iterationen (8,3 Stunden) | Optimiert: nach 100.000 Iterationen (16,6 Stunden) | Original: nach 800.000 Iterationen (264 Stunden)

Abbildung 68: Visueller Vergleich zwischen optimierter und originaler Implementation: Der Qualitätsunterschied ist minimal, während die Trainingszeit drastisch reduziert wurde.

5.4.3 Herausforderungen bei der Preprocessing-Pipeline

Ein wichtiger Aspekt der Evaluation ist die Analyse der entwickelten Preprocessing-Pipeline, die sich als komplexer erwies als zunächst angenommen.

Späte Erkenntnis der Preprocessing-Notwendigkeit Durch die initiale Fokussierung auf den Testdatensatz wurde erst spät im Projektverlauf deutlich, dass ein erhebliches Preprocessing für eigene Datensätze notwendig ist. Die Preprocessing-Pipeline musste daher noch recht spontan entwickelt werden, nachdem die eigentliche Optimierung des Trainingsmodells bereits abgeschlossen war.

Technische Herausforderungen Bei der Anwendung der Pipeline auf Daten aus unserem eigenen Kamera-Rig traten Probleme auf, die trotz intensiver Fehlersuche nicht vollständig gelöst werden konnten. Vermutlich hat sich bei der Implementierung ein Fehler eingeschlichen, der zu Inkompatibilitäten zwischen den generierten Daten und dem Trainingsmodell führt. Die genaue Ursache konnte innerhalb des Projektzeitraums nicht identifiziert werden.

Komplexität der Pipeline Die implementierte Pipeline umfasst zahlreiche Schritte, von der zeitlichen Datenreduktion über Videomatting bis hin zur Posenextraktion und Generierung von Positionskarten. Diese Komplexität macht die Fehlersuche schwierig und erhöht die Wahrscheinlichkeit von Inkompatibilitäten zwischen den verschiedenen Komponenten.
Trotz dieser Herausforderungen stellt die entwickelte Preprocessing-Pipeline einen wichtigen Beitrag dar, da sie die Verarbeitung von Daten aus unserem 67-Kamera-Setup ermöglicht – eine Konfiguration, die deutlich komplexer ist als die in der ursprünglichen Implementierung vorgesehenen 16 Kameras.

6 Fazit

6.1 Zusammenfassung der Ergebnisse

6.1.1 Volumetric Capture System

Das Volumetric Capture System wurde mit dem Ziel konzipiert, 1080p-Aufnahmen bei einer Bildrate von bis zu 30 Bildern pro Sekunde zu ermöglichen. Das Aufnahmerig wurde so dimensioniert, dass es die Bewegungen von Personen vollständig erfassen und eine lückenlose 360°-Abdeckung gewährleisten kann. Priorität lag auf der Realisierung einer stabilen und synchronisierten Erfassung großer Datenmengen innerhalb definierter Budgetrestriktionen.
Im Entwicklungsverlauf erwies sich die gewählte modulare Systemarchitektur sowie die dezentrale Steuerung der Kameraeinheiten als entscheidend vorteilhaft. Diese Ansätze erhöhten nicht nur die funktionale Effizienz im Aufbau und Betrieb, sondern ermöglichten darüber hinaus eine signifikante Flexibilität und Erweiterbarkeit des Systems — Eigenschaften, die zunächst nicht im primären Fokus der Planung standen, jedoch im Projektverlauf als wesentliche Zusatznutzen identifiziert wurden.
Die Bildqualität des final realisierten Systems erreicht ein Niveau, das mit deutlich kostenintensiveren professionellen Lösungen vergleichbar ist. Durch den Einsatz des Camera Module V3 konnte eine wirtschaftliche, zugleich jedoch qualitativ überzeugende Lösung implementiert werden. Insbesondere bei Nutzung der UHD-Aufzeichnungsoption zeigt sich eine erhöhte Detailwiedergabe, insbesondere in den Randbereichen sowie bei der Texturerhaltung von Haaren und Bekleidung. Die maximale Bildqualität ist jedoch durch die native Auflösung der Kameramodule physikalisch begrenzt. Für Ganzkörperaufnahmen ist die Reproduktionsqualität hervorragend geeignet; Aufnahmen im Halbnahbereich (Kopf bis Hüfte) sowie Gesichtsnahaufnahmen sind, unter Berücksichtigung gewisser Einschränkungen, ebenfalls möglich.
Das System bewährte sich im praktischen Einsatz durch hohe Robustheit und Wartungsfreundlichkeit. Die implementierte Netzwerksynchronisation, die eigens entwickelte Kontrollsoftware sowie die modulare Hardwarearchitektur gewährleisten eine intuitive Bedienbarkeit sowie eine schnelle Reaktion auf etwaige technische Störungen. Einzelne Komponenten können bei Bedarf unkompliziert ersetzt oder erweitert werden, was eine flexible Anpassung an veränderte Anforderungen ermöglicht.
Ein weiterer Vorteil des Systems liegt in seiner vielseitigen Verwendbarkeit: Das erzeugte Bildmaterial ist nicht ausschließlich für die im Projekt eingesetzte 4D-Gaussian Splatting Pipeline nutzbar, sondern eignet sich ebenso für alternative volumetrische Verfahren wie Multi-View Stereo, Photogrammetrie, Neural Rendering oder Motion-Capture-Analysen. Somit bietet das Capture-Rig eine flexible Plattform für Forschungs- und Entwicklungsaktivitäten im Bereich immersiver Medientechnologien.
Zusammenfassend stellt das entwickelte Volumetric Capture System nicht nur eine tragfähige technische Grundlage für das aktuelle Projekt dar, sondern bildet auch ein zukunftsfähiges Werkzeug für weitere studentische Forschungsprojekte an der Schnittstelle von Medientechnologie, Computergrafik und KI-gestützter Bildverarbeitung. Die Kombination aus kosteneffizienter Hardware, hoher Bildqualität und modularer Erweiterbarkeit qualifiziert das System als wichtigen Enabler für explorative und praxisnahe Forschungsansätze in diesem Themenfeld.

6.1.2 Training

Beteiligte Teammitglieder: David Mertens

Die Methode der Spacetime Gaussians stellt einen innovativen Ansatz zur Darstellung dynamischer 3D-Szenen dar, indem sie zeitlich variierende Inhalte durch die Kombination von 3D-Gaussians mit temporalen Komponenten modelliert. Im Gegensatz zu früheren Arbeiten, die Spherical Harmonics (SH) zur Modellierung richtungsabhängiger Lichtverteilungen nutzten, ersetzt diese Methode die SH durch neuronale Merkmale, die über ein flaches neuronales Netzwerk (MLP) in RGB-Farben umgewandelt werden.
Diese Entscheidung reduziert zwar die Anzahl der Parameter pro Gaussian (9 statt 48 bei SH vom Grad 3) und ermöglicht eine schnellere Verarbeitung, führt jedoch zu Einschränkungen bei der Darstellung von Richtungsabhängigkeiten. Insbesondere bei omnidirektionalen Aufnahmen, bei denen die Lichtverhältnisse stark von der Blickrichtung abhängen, kann das Fehlen expliziter Richtungsmodelle wie SH zu einer verminderten Darstellungsqualität führen.
In der Originalpublikation werden hauptsächlich Videos mit begrenztem Blickwinkel präsentiert, was darauf hindeutet, dass die Methode bei omnidirektionalen Szenarien an ihre Grenzen stößt. Die neuronalen Merkmale können komplexe Richtungsabhängigkeiten möglicherweise nicht so effektiv erfassen wie SH, was zu Artefakten oder Unschärfen in der Darstellung führen kann.
Um die Eignung von Spacetime Gaussians für vollständig omnidirektionale Anwendungen zu verbessern, könnte die Integration von Spherical Harmonics oder ähnlichen richtungsabhängigen Modellen in die neuronalen Merkmale ein vielversprechender Ansatz sein. Dies würde eine genauere Modellierung der Lichtverhältnisse aus verschiedenen Blickwinkeln ermöglichen und die Darstellungsqualität in solchen Szenarien erhöhen.

6.1.3 Viewer

Beteiligte Teammitglieder: Steffen-Sascha Stein

Der in dieser Arbeit implementierte WebGPU-Viewer für Spacetime Gaussian Splatting (Zhan Li u. a. 2024) baute auf der WebGL-Implementierung von Kevin Kwok (Kwok 26.03.2024) auf. Obwohl die verschiedenen Performance-Modes keine Verbesserung der Laufzeit hervorriefen, konnte die Verlagerung des Sortierens der Splats von der CPU auf die GPU die visuelle Qualität der Darstellung unserer Modelle deutlich verbessern. Eine besondere Erkenntnis dieser Arbeit ist jedoch, dass das CPU-seitige Sortieren der Splats einen performanten Vorteil für die Gesamtanwendung gegenüber dem GPU-seitigen Sortieren aufweist, wenn es asynchronisiert wird. Zudem konnte letztendlich der Fragment-Shader als Bottleneck der Pipeline identifiziert werden. Auch die Implementierung von Foveated Splatting, Alpha Culling und des dynamischem Viewport Scaling zeigt, dass weitere Optimierungsansätze grundsätzlich möglich sind, deren Wirksamkeit jedoch stark vom Rendering-Kontext, den Wahrnehmungseigenschaften der Nutzer und den Einschränkungen der WebXR-API abhängt. Der nicht vorhandene Support von WebXR für WebGPU stellte außerdem eine große Schwierigkeit in diesem Projekt dar und kostete letztendlich Zeit in der Umsetzung eines Workarounds, aber auch im Rendering in der Form von Overhead.

6.2 Überlegungen für weitergehende Arbeiten

6.2.1 Volumetric Capture System

Wie bereits erwähnt, lässt sich das Volumetric Capture System hervorragend für zukünftige studentische Projekte weiterentwickeln. Dabei besteht insbesondere im Bereich der Kameratechnik und -ansteuerung noch viel Potenzial: Eine automatische Kamerakalibrierung – insbesondere auch für Einzelkameras nach Austausch oder Neujustierung – könnte die Nutzbarkeit deutlich verbessern und den manuellen Aufwand verringern. Entsprechende Verfahren, etwa auf Basis von ArUco-Mustern in Kombination mit automatisierter Colmap-Integration, könnten hier evaluiert und implementiert werden.
Ein weiterer Ansatzpunkt ist die Verbesserung der elektrischen und mechanischen Stabilität der Verbindungen. Aktuell sind ungeschirmte CSI-Kabel im Einsatz, die unter bestimmten Bedingungen zu Signalstörungen und Framedrops führen können. Der gezielte Einsatz geschirmter CSI-Kabel oder alternativer Steckverbindungen könnte hier zu einer höheren Ausfallsicherheit und verbesserten Datenintegrität beitragen. Auch eine Optimierung der Deckenkameras wäre ein weiteres mögliches Projekt, da hier eine Alternative zur top-down Montage gefunden werden muss, um diese Kameras nutzen zu können.
Darüber hinaus bietet das System durch die gezielte Ansteuerbarkeit jedes einzelnen Raspberry Pis über das Netzwerk eine spannende Grundlage für experimentelle Lichtsteuerung und Relighting-Ansätze. So ließen sich etwa mit LED-Arrays ansteuerbare Lichtszenarien realisieren, bei denen für jedes Frame eine gezielte Lichtumgebung geschaffen wird – ein besonders relevanter Aspekt für weiterführende Forschung im Bereich realitätsnaher Darstellung und datengetriebener Materialrekonstruktion.
Nicht zuletzt wäre auch eine Integration mobiler Kalibrier- und Monitoring-Stationen für die Inbetriebnahme und Wartung des Systems vor Ort denkbar. Hier könnten z.B. Web-UIs zur Statusanzeige und Kalibrierung einzelner Kameraeinheiten in zukünftige studentische Arbeiten einfließen.

Zusammenfassend lässt sich sagen, dass das Volumetric Capture System eine technisch anspruchsvolle, dabei aber praxistaugliche Lösung darstellt, die sowohl in ihrer jetzigen Form überzeugt als auch Raum für weiterführende studentische Forschung und Entwicklung bietet. Die Kombination aus niedrigen Hardwarekosten, hoher Flexibilität und systematischer Erweiterbarkeit prädestiniert es als Plattform für zukünftige Projekte im Bereich volumetrischer Videoaufnahme, Lichtsimulation und Echtzeit-Visualisierung.

6.2.2 Training

Beteiligte Teammitglieder: Matthias Bullert

Die im Rahmen des Projekts entwickelte Trainingspipeline bietet eine solide Grundlage für die Erzeugung dynamischer volumetrischer Darstellungen mittels Spacetime Gaussian Splatting. Dennoch ergeben sich mehrere Ansätze für weiterführende Optimierungen und Erweiterungen.

- **Integration fortgeschrittener Hintergrundsegmentierung:** Die aktuelle Methode zur Hintergrundentfernung basiert auf bildbasierten Maskierungsansätzen. Eine Verbesserung könnte durch den Einsatz semantischer Segmentierungsnetzwerke oder durch die Fusion mehrerer Ansätze (z.B. DeepLabV3+, SAM) erzielt werden, insbesondere für Szenen mit komplexem oder dynamischem Hintergrund.
- **Skalierbarkeit und Parallelisierung:** Für größere Szenen oder höhere Auflösungen wäre eine weitere Parallelisierung des Trainingsprozesses – sowie der Preprocessing-Pipeline – mittels Multi-GPU-Ansätzen oder verteiltem Rechnen sinnvoll. Auch könnte die Nutzung von Sparse Tensor Representations weiterentwickelt werden, um Speicher- und Rechenressourcen effizienter zu nutzen.
- **Echtzeit-Feedback während des Trainings:** Eine visuelle Echtzeitkontrolle der Trainingsergebnisse – etwa durch ein Preview-Rendering der Splats – könnte dazu beitragen, suboptimale Konfigurationen frühzeitig zu erkennen und Korrekturen vorzunehmen, bevor umfangreiche Ressourcen investiert werden.

Diese Aspekte bieten vielversprechende Ansatzpunkte für weitergehende Forschungs- und Entwicklungsarbeiten.

6.2.3 Viewer

Beteiligte Teammitglieder: Steffen-Sascha Stein

Die umgesetzten Performance-Modes des WebGPU-Viewers bieten eine solide Grundlage für zukünftige Optimierungen. Beispielsweise könnte die Limitierung durch den Fragment-Shader durch eine vollständige Compute-Pipeline umgangen werden, wie sie auch im ursprünglichen Gaussian Splatting Ansatz (Kerbl u. a. 2023) beschrieben ist. Hier wird es einerseits ermöglicht das Front-to-Back Rendering für einzelne Pixel zu unterbrechen, sobald diese opak sind. Andererseits nutzt die Implementierung dieses Ansatzes Tile-Based-Rendering, welches auch intern auf allen Meta Quest Geräten genutzt wird. Eine gezielte Umsetzung könnte also mit der Hardware der Meta Quest harmonieren. Bezüglich des Foveated Splattings müsste für eine genauere Aussage über einen Performanzgewinn dieser auch in Relation zur Wahrnehmung des Betrachters evaluiert werden. Auch hier gibt es Raum für Optimierungen, denn während in dieser Umsetzung das Verwerfen der Splats in einem Rechteck resultiert, könnte in Bezug auf das menschliche Auge eine Ellipse sinnvoller sein.

6.3 HAC

Beteiligte Teammitglieder: Marvin Winkler

Die durchgeführten Optimierungen stellen einen signifikanten Fortschritt dar und transformieren das ursprünglich kaum praktikable System in ein effizientes Werkzeug für die Erzeugung fotorealistischer, animierbarer 3D-Avatare.

6.3.1 Gesamtbewertung und Ausblick

Stärken

- Drastische Reduzierung der Trainingszeit (Faktor 15,9-31,8×)
- Signifikante Verbesserung der Rendergeschwindigkeit (>200%)
- Erhebliche Reduktion des Speicherbedarfs (≈ 70%)
- Nahezu unveränderte visuelle Qualität
- Umfassende Preprocessing-Pipeline für unser komplexeres Kamerasetup

Verbesserungspotenzial

- Behebung der Inkompatibilitäten in der Preprocessing-Pipeline
- Weitere Optimierung der Rendergeschwindigkeit in Richtung Echtzeit
- Integration physikbasierter Simulationen für realistischere Bewegungsdynamik, insbesondere für Haare
- Verbesserung der Steuerung feiner Gesichtsausdrücke und Handbewegungen

Ausblick Die erzielten Optimierungen bilden eine solide Grundlage für zukünftige Weiterentwicklungen. Mit weiteren Verbesserungen, insbesondere im Bereich der Rendereffizienz und der Integration physikbasierter Simulationen, könnte das System zu einer leistungsfähigen Plattform für die Erzeugung und Animation fotorealistischer virtueller Menschen werden.
Besonders vielversprechend wäre eine Kombination der hier entwickelten Optimierungstechniken mit neueren Ansätzen wie Zielonka u. a. 2025 oder Moon, Shiratori und Saito 2024, die zusätzliche Vorteile im Bereich der expressiven Gesichts- und Handanimation bieten.
Insgesamt kann die durchgeführte Optimierung als großer Erfolg gewertet werden, da sie die praktische Anwendbarkeit des Systems erheblich verbessert und gleichzeitig die hohe visuelle Qualität beibehält. Die verbleibenden Herausforderungen, insbesondere im Bereich des Preprocessings, stellen interessante Aufgaben für zukünftige Arbeiten dar.

Literatur

Guo, Kaiwen u. a. (31. Dez. 2019). "The relightables: volumetric performance capture of humans with realistic relighting". In: *ACM Transactions on Graphics* 38.6, S. 1–19. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3355089.3356571.

Palladino, Tommy (19. Nov. 2019). *Google Researchers Develop Method for Capturing 'Relightable' Volumetric Video*. Next Reality. URL: https://next.reality.news/news/google-researchers-develop-method-for-capturing-relightable-volumetric-video-0213517/ (Zuletzt besucht: 2 Mai 2025).

Schreer, Oliver u. a. (Juni 2019). "Advanced Volumetric Capture and Processing". In: *SMPTE Motion Imaging Journal* 128.5, S. 18–24. ISSN: 1545-0279, 2160-2492. DOI: 10.5594/JMI.2019.2906835.

befores und afters (27. Mai 2022). *Inside Volucap's new volumetric tech set-up, and its new 4D people*. befores and afters. URL: https://beforesandafters.com/2022/05/28/inside-volucaps-new-volumetric-tech-set-up-and-its-new-4d-people/ (Zuletzt besucht: 3 Mai 2025).

Ehrenhöfer, Alex (2025). *Volucap Max*. Volucap. URL: https://volucap.com/volucap-max/ (Zuletzt besucht: 3 Mai 2025).

Collet, Alvaro u. a. (27. Juli 2015). "High-Quality Streamable Free-Viewpoint Video". In: *ACM Transactions on Graphics* 34.4, S. 1–13. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/2766945.

Lowpass, Janko Roettgers (28. Juli 2023). *Microsoft lays off Mixed Reality Capture Studios team*. Lowpass. URL: https://www.lowpass.cc/p/microsoft-mixed-reality-capture-studios-layoffs (Zuletzt besucht: 3 Mai 2025).

Intel (31. Aug. 2020). *Intel Studios Showcases Volumetric Production at 77th Venice...* URL: https://download.intel.com/newsroom/archive/2025/en-us-2020-08-31-intel-studios-showcases-volumetric-production-at-77th-venice-international-film-festival.pdf (Zuletzt besucht: 3 März 2025).

Farrell, Nick (2025). *Intel shuts Intel Studios*. URL: https://www.fudzilla.com/news/51949-intel-shuts-intel-studios (Zuletzt besucht: 3 Mai 2025).

Polymotion Stage (2025). Mark Roberts Motion Control. URL: https://www.mrmoco.com/polymotion-stage/ (Zuletzt besucht: 3 Mai 2025).

Polymotion Stage Case Study (2025). *Polymotion Stage Case Study: The 148th Open Championship, Royal Portrush*. Mark Roberts Motion Control. URL: https://www.mrmoco.com/case-studies/polymotion-stage-case-study-the-148th-open-championship-royal-portrush/ (Zuletzt besucht: 3 Mai 2025).

Bönsch, Andrea u. a. (2019). "Volumetric Video Capture using Unsynchronized, Low-cost Cameras:" In: *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. 14th International Conference on Computer Graphics Theory and Applications. Prague, Czech Republic: SCITEPRESS - Science und Technology Publications, S. 255–261. ISBN: 978-989-758-354-4. DOI: 10.5220/0007373202550261.

Riegler, Gernot und Vladlen Koltun (2020). “Free view synthesis”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIX 16*. Springer, S. 623–640.

Yang, Ming-Der u. a. (2013). “Image-based 3D scene reconstruction and exploration in augmented reality”. In: *Automation in Construction* 33, S. 48–60.

Zhou, Qian-Yi und Vladlen Koltun (2013). “Dense scene reconstruction with points of interest”. In: *ACM Transactions on Graphics (ToG)* 32.4, S. 1–8.

Mildenhall, Ben u. a. (2021). “Nerf: Representing scenes as neural radiance fields for view synthesis”. In: *Communications of the ACM* 65.1, S. 99–106.

Kerbl, Bernhard u. a. (2023). “3d gaussian splatting for real-time radiance field rendering.” In: *ACM Trans. Graph.* 42.4, S. 139–1.

Liu, Yang u. a. (Okt. 2024). “Animatable 3D Gaussian: Fast and High-Quality Reconstruction of Multiple Human Avatars”. In: *Proceedings of the 32nd ACM International Conference on Multimedia (MM ’24)*. DOI: 10.1145/3664647.3680674.

Moreau, Antoine u. a. (2024). “Human Gaussian Splatting: Real-time Rendering of Animatable Avatars”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, S. 788–798.

Qian, Zhengming u. a. (2024). “3DGS-Avatar: Animatable Avatars via Deformable 3D Gaussian Splatting”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, S. 5020–5030.

Zielonka, Wojciech u. a. (Feb. 2025). “Drivable 3D Gaussian Avatars”. In: *arXiv preprint arXiv:2311.08581v2*. DOI: 10.48550/arXiv.2311.08581.

Jung, HyunJun u. a. (Dez. 2023). “Deformable 3D Gaussian Splatting for Animatable Human Avatars”. In: *arXiv preprint arXiv:2312.15059v1*. DOI: 10.48550/arXiv.2312.15059.

Shao, Zhan u. a. (2024). “SplattingAvatar: Realistic Real-Time Human Avatars with Mesh-Embedded Gaussian Splatting”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, S. 1606–1616.

Li, Zhe u. a. (März 2024). “Animatable Gaussians: Learning Pose-dependent Gaussian Maps for High-fidelity Human Avatar Modeling”. In: *arXiv preprint arXiv:2311.16096v3*.

Hu, Liangxiao u. a. (März 2024). “GaussianAvatar: Towards Realistic Human Avatar Modeling from a Single Video via Animatable 3D Gaussians”. In: *arXiv preprint arXiv:2312.02134v3*.

Saito, Shunsuke u. a. (2024). “Relightable Gaussian Codec Avatars”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, S. 130–141.

Qian, Siyou u. a. (2024). “GaussianAvatars: Photorealistic Head Avatars with Rigged 3D Gaussians”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, S. 20299–20309.

Xiang, Jiaxing u. a. (2024). “FlashAvatar: High-fidelity Head Avatar with Efficient Gaussian Embedding”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, S. 1802–1812.

Moon, Gyeongsik, Takaaki Shiratori und Shunsuke Saito (Juli 2024). "Expressive Whole-Body 3D Gaussian Avatar". In: *arXiv preprint arXiv:2407.21686v1*. DOI: 10.48550/arXiv.2407.21686.

Yuan, Yuelang u. a. (2024). "GAvatar: Animatable 3D Gaussian Avatars with Implicit Mesh Learning". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2024)*, S. 896–905.

Raspberry Pi Foundation (2025a). *Raspberry Pi Zero 2 W*. https://www.raspberrypi.org/products/raspberry-pi-zero-2-w. (Zuletzt besucht: 24 Apr. 2025).

— (2025b). *Raspberry Pi 4 Model B*. https://www.raspberrypi.org/products/raspberry-pi-4-model-b/. (Zuletzt besucht: 24 Apr. 2025).

— (2025c). *Introducing: Raspberry Pi 5*. https://www.raspberrypi.com/products/raspberry-pi-5/. (Zuletzt besucht: 24 Apr. 2025).

— (2025d). *Camera*. https://www.raspberrypi.com/documentation/accessories/camera.html. (Zuletzt besucht: 24 Apr. 2025).

Kwok, Kevin (26.03.2024). *splaTV: video splat*. URL: https://github.com/antimatter15/splat.

Li, Zhan u. a. (2024). *Spacetime Gaussian Feature Splatting for Real-Time Dynamic View Synthesis*. arXiv: 2312.16812 [cs.CV]. URL: https://arxiv.org/abs/2312.16812.

Franke, Linus u. a. (2024). "TRIPS: Trilinear Point Splatting for Real-Time Radiance Field Rendering". In: *Computer Graphics Forum* 43.2. DOI: https://doi.org/10.1111/cgf.15012.

Rückert, Darius, Linus Franke und Marc Stamminger (Juli 2022). "ADOP: approximate differentiable one-pixel point rendering". In: *ACM Trans. Graph.* 41.4. ISSN: 0730-0301. DOI: 10.1145/3528223.3530122. URL: https://doi.org/10.1145/3528223.3530122.

Ponn, Josef und Udo Lindemann (2011). *Konzeptentwicklung und Gestaltung technischer Produkte – Systematisch von Anforderungen zu Konzepten und Gestaltlösungen*. Springer Vieweg. URL: https://link.springer.com/book/10.1007/978-3-642-20580-4 (Zuletzt besucht: 24 Apr. 2025).

Raspberry Pi Foundation (2024). *Power supply*. https://github.com/raspberrypi/documentation/blob/develop/documentation/asciidoc/computers/raspberry-pi/power-supplies.adoc. (Zuletzt besucht: 24 Apr. 2025).

Schönberger, Johannes Lutz und Jan-Michael Frahm (2025). *Structure-from-Motion Revisited*. https://colmap.github.io/. (Zuletzt besucht: 24 Apr. 2025).

Raspberry Pi Foundation (2025e). "Capture Metadata". In: *The Picamera2 Library*. Raspberry Pi Foundation. Kap. 6.2. URL: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf (Zuletzt besucht: 24 Apr. 2025).

— (2025f). "Available camera controls". In: *The Picamera2 Library*. Raspberry Pi Foundation, S. 86–87. URL: https://datasheets.raspberrypi.com/camera/picamera2-manual.pdf (Zuletzt besucht: 24 Apr. 2025).

Ester, Martin u. a. (1996). "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *kdd*. Bd. 96. 34, S. 226–231.

Zwicker, M. u. a. (2001). “EWA volume splatting”. In: *Proceedings Visualization, 2001. VIS ’01.* S. 29–538. DOI: 10.1109/VISUAL.2001.964490.

Kishimisu (2025). *WebGPU-Radix-Sort*. https://github.com/kishimisu/WebGPU-Radix-Sort. (Zuletzt besucht: 18 Mai 2025).

Harris, Mark, Shubhabrata Sengupta und John D Owens (2007). “Parallel prefix sum (scan) with CUDA”. In: *GPU gems* 3.39, S. 851–876.

Broxton, Michael u. a. (2020). “Immersive Light Field Video with a Layered Mesh Representation”. In: 39.4, 86:1–86:15.

Ha, Linh, Jens Krüger und Cláudio T. Silva (2009). “Fast Four-Way Parallel Radix Sorting on GPUs”. In: *Computer Graphics Forum* 28.8, S. 2368–2378. DOI: https://doi.org/10.1111/j.1467-8659.2009.01542.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01542.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01542.x.

Group, Immersive Web Working (2024a). *WebXR Device API*. W3C Editor’s Draft. URL: https://immersive-web.github.io/webxr/ (Zuletzt besucht: 23 Mai 2025).

— (2024b). *WebXR Layers API — XRWebGLLayer.getViewport()*. W3C Editor’s Draft. URL: https://immersive-web.github.io/webxr/#dom-xrwebgllayer-getviewport (Zuletzt besucht: 23 Mai 2025).

Three.js contributors (2024a). *Three.js – JavaScript 3D Library*. URL: https://threejs.org.

— (2024b). *Three.js Documentation: WebXRManager*. URL: https://threejs.org/docs/#api/en/renderers/webxr/WebXRManager.

— (2024c). *Three.js Documentation*. URL: https://threejs.org/docs/.

— (2024d). *Three.js Manual*. URL: https://threejs.org/manual/.

Meta Platforms, Inc. (2025). *Meta Quest Touch Plus Controller*. URL: https://www.meta.com/de/quest/accessories/quest-touch-plus-controller/.

MDN Web Docs (2024). *XRQuadLayer - Web APIs | MDN*. https://developer.mozilla.org/en-US/docs/Web/API/XRQuadLayer. https://developer.mozilla.org/en-US/docs/Web/API/XRQuadLayer. (Zuletzt besucht: 22 Mai 2025).

Immersive Web Working Group (2024a). *WebXR Layers API - Specification Draft*. https://immersive-web.github.io/layers/. https://immersive-web.github.io/layers/. (Zuletzt besucht: 22 Mai 2025).

— (2024b). *WebXR DOM Overlay API - Explainer*. https://github.com/immersive-web/dom-overlays/blob/main/explainer.md. (Zuletzt besucht: 22 Mai 2025).

MDN contributors (2024). *XRFrame.getPose() - Web APIs | MDN*. https://developer.mozilla.org/en-US/docs/Web/API/XRFrame/getPose. (Zuletzt besucht: 22 Mai 2025).

180by2 (2025). *Oculus Quest 2 Touch Controller (Right)*. https://180by2.co.za/cdn/shop/products/oculus-quest-2-touch-controller-right.png?v=1622543703. (Zuletzt besucht: 22 Mai 2025).

Lin, Shanchuan u. a. (2021). *Robust High-Resolution Video Matting with Temporal Guidance*. arXiv: 2108.11515 [cs.CV]. URL: https://arxiv.org/abs/2108.11515.

Jawset Visual Computing (2025). *Postshot User Guide: Training Configuration*. https://www.jawset.com/docs/d/Postshot%2BUser%2BGuide/Interface/Training%2BConfiguration. (Zuletzt besucht: 24 Apr. 2025).

Li, Tianye u. a. (2022). "Neural 3d video synthesis from multi-view video". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, S. 5521–5531.

Meta (2025a). *Monitor Performance with OVR Metrics Tool*. https://developers.meta.com/horizon/documentation/unity/ts-ovrmetricstool/. (Zuletzt besucht: 18 Mai 2025).

— (2025b). *Use ovrgpuprofiler for GPU Profiling*. https://developers.meta.com/horizon/documentation/unity/ts-ovrgpuprofiler/. (Zuletzt besucht: 18 Mai 2025).

Jones, Brandon (2025). *WebGPU/WebGL Performance Comparison Best Practices*. https://toji.dev/webgpu-best-practices/webgl-performance-comparison. (Zuletzt besucht: 18 Mai 2025).

Zhang, Felix (13.06.2023). *Immersive Web Emulator 2.0*. URL: https://github.com/antimatter15/splat.

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

ADC Additive Data Composition

AoS Array of Structures

API Application Programming Interface

AR Augmented Reality

CAD Computer Aided Design

CLI Command Line Interface

CNN Convolutional Neural Network

CPU Central Processing Unit

CT-Curve Colour Temperature Curve

dhcpcd Dynamic Host Configuration Protocol Client Daemon

EWA Elliptical Weighted Average

FPS Frames Per Second

GPU Graphics Processing Unit

HAC Humanoide Animierte Charaktere

HMD Head-Mounted Display

HTTP Hypertext Transfer Protocol

HUD Heads-Up Display

LBS Linear Blend Skinning

MCMC Markow-Chain-Monte-Carlo

MSB Most Significant Bit

NeRF Neural Radiance Fields

NTP Network Time Protocol

PLA Polylactide

RBF Radiale Basisfunktion

SCP Secure Copy Protocol

SSH Secure Shell

UI	User Interface
USB-PD	USB Power Delivery
VR	Virtual Reality

A Anhang

A.1 Capture Software User Guide

A.1.1 Programm und Skriptübersicht

Während der Entwicklung der Capture Software wurden mehrere Skripte und Programme entwickelt, welche zum Erstellen von Aufnahmen mit dem Capture Rig, zu Kalibrierungszwecken, zum Datentransfer oder zur allgemeinen Entwicklung dieser Software verwendet werden können. Darunter befinden sich die folgenden Programme:

Skript	**Kurzbeschreibung**
`master_capture_controller.py`	Erstellt Sessions (Videos) und Stills (Standbilder) synchronisiert über alle Kameras
`master_camera_controller.py`	Kalibriert das globale Kamerasystem und bietet eine Live-Ansicht der Kameraperspektiven
`master_download_manager.py`	Verwaltet aufgenommene Dateien auf den Raspberry Pis, kann diese gesammelt herunterladen und löschen und Videodateien in Frames konvertieren
`launcher.py`	Bietet Optionen zum Starten, Stoppen, Hochladen, Aktualisieren und Löschen von Dateien auf allen Raspberry Pis gleichzeitig und kann alle Raspberry Pis neu starten
`search_devices.py`	Sucht nach allen Geräten innerhalb des lokalen Netzwerks
`master_set_cam_number.py`	Konfiguriert Hostname und IP-Adresse eines Raspberry Pis
`set_up_chrony_via_ssh.py`	Konfiguriert mehrere Raspberry Pis mit einer Abfolge an SSH-Befehlen, um neue Konfigurationen systemweit in kurzer Zeit einzustellen

Tabelle 10: Übersicht der entwickelten Skripte und Programme

Die folgenden Skripte stellen zusätzlich eine ausführbare Datei (Executable) bereit:

- `master_capture_controller.py`
- `master_camera_controller.py`
- `master_download_manager.py`

A.1.2 Systemvoraussetzungen

Alle Programme wurden in Python entwickelt und erfolgreich mit den Versionen 3.10, 3.11 und 3.12 getestet. Vor der Ausführung der Skripte ist sicherzustellen, dass alle in der `requirements.txt` aufgeführten Python-Bibliotheken installiert sind. Es wird empfohlen, hierfür ein virtuelles Python-Environment zu verwenden, entweder über die Standard-`venv`-Funktionalität von Python oder über `conda`.
Dank der Plattformunabhängigkeit von Python sind alle Skripte und Programme sowohl unter Windows als auch auf Unix-basierten Systemen lauffähig. Die Kompatibilität wurde getestet unter Windows 11 sowie macOS Sequoia.

Für die Umwandlung aufgezeichneter Videodateien in Frame-Sequenzen mithilfe des `download_manager.py` ist zusätzlich das Tool `FFmpeg` erforderlich. Dieses muss systemweit verfügbar sein, was bedeutet, dass der Pfad zur ausführbaren Datei von `FFmpeg` in die Umgebungsvariablen des jeweiligen Betriebssystems aufgenommen werden muss.

A.1.3 Funktionsweise des `launcher.py`-Skritps

Das Skript `launcher.py` befindet sich im `utils`-Ordner und dient dazu, verschiedene Skripte systemweit gleichzeitig auf allen Raspberry Pis auszuführen, zu verwalten oder die Geräte neu zu starten. Es handelt sich um ein rein textbasiertes CLI-Tool (Command Line Interface) ohne grafische Benutzeroberfläche.
Folgende Operationen werden unterstützt:

- **Starten eines Skripts:**
  `launcher.py start <script_name.py>`
  Startet ein Skript mit dem Namen `<script_name.py>` auf allen Raspberry Pis der konfigurierten IP-Liste.
- **Stoppen eines Skripts:**
  `launcher.py stop <script_name.py>`
  Beendet ein laufendes Skript mit dem Namen `<script_name.py>` auf allen Raspberry Pis.
- **Löschen eines Skripts:**
  `launcher.py delete <script_name.py>`
  Entfernt das angegebene Skript aus dem Home-Verzeichnis (`/home/voluman/`) aller Raspberry Pis.
  Wildcards (z. B. `'test_*.py'`) können verwendet werden, um mehrere Dateien zu löschen.
  Wichtig: Die Wildcards müssen in einfache Anführungszeichen gesetzt werden, um eine Interpretation durch die lokale Shell zu vermeiden.
- **Hochladen eines Skripts:**
  `launcher.py upload [<local_script_path>]`
  Überträgt ein lokales Skript auf alle Raspberry Pis. Wird kein Pfad angegeben, öffnet sich (sofern `tkinter` installiert ist) ein Dateiauswahlfenster.
- **Neustarten aller Geräte:**
  `launcher.py reboot`
  Führt auf jedem Raspberry Pi einen Neustart (`sudo reboot`) durch.

Das Skript nutzt `ThreadPoolExecutor`, um alle Operationen parallel auszuführen und so eine effiziente Verwaltung mehrerer Geräte zu ermöglichen.

A.1.4 Netzwerkerkennung mit `search_devices.py`

Das Skript `search_devices.py` dient zur Erkennung aller erreichbaren Geräte in einem definierten Subnetz, die auf ICMP-Ping-Anfragen reagieren. Es handelt sich um ein reines CLI-Tool

(Command Line Interface) ohne grafische Benutzeroberfläche und kann direkt über das Terminal ausgeführt werden.

Funktionsweise
Beim Start des Skripts wird automatisch das Subnetz `10.50.100.0/24` (IP-Adressen von `10.50.100.1` bis `10.50.100.254`) überprüft. Für jede dieser IP-Adressen wird ein Ping-Befehl abgesetzt:

- Unter **Windows**: `ping -n 1 -w 1 <IP>`
- Unter **Linux/macOS**: `ping -c 1 -W 1 <IP>`

Wenn ein Gerät auf den Ping reagiert, wird zusätzlich versucht, den Hostnamen über einen Reverse-DNS-Lookup zu ermitteln. Falls dies fehlschlägt, wird als Hostname `Unknown` angezeigt.

Parallelisierung
Zur Beschleunigung des Scanvorgangs wird ein `ThreadPoolExecutor` eingesetzt, der bis zu 10 IP-Adressen gleichzeitig prüft.

Ausgabe
Am Ende gibt das Skript eine Liste aller Geräte aus, die auf die Ping-Anfragen geantwortet haben. Für jedes erkannte Gerät werden die IP-Adresse sowie – sofern ermittelbar – der zugehörige Hostname ausgegeben:

```
Devices responding to ping:
IP: 10.50.100.12 - Hostname: raspberrypi
IP: 10.50.100.42 - Hostname: Unknown
...
```

Dieses Skript eignet sich besonders für eine schnelle und einfache Überprüfung, welche Geräte im lokalen Netzwerk aktiv und erreichbar sind.

A.1.5 Konfiguration der NTP-Kommunikation mit `set_up_chrony_via_ssh.py`

Das Skript `set_up_chrony_via_ssh.py` dient der automatisierten Konfiguration des zur Abfrage des NTP-Servers mittels `Chrony` auf mehreren Raspberry Pis innerhalb eines definierten IP-Bereichs. Es handelt sich um ein reines CLI-Tool (Command Line Interface), das per SSH eine Abfolge von Systembefehlen ausführt.

Zweck und Anwendungsfall
Das Skript aktualisiert die Paketlisten, installiert `chrony`, ersetzt die Konfigurationsdatei `/etc/chrony/chrony.conf` mit einer vordefinierten Version, startet den Dienst neu, aktiviert den automatischen Start und führt schließlich einen Reboot des Raspberry Pis durch. Dies war während der Entwicklung notwenidig, um die Raspberry Pis nachträglich einheitlich so zu konfigurieren, damit sie den Distrobutor als NTP-Server abfragen.

Dank seiner flexiblen Struktur eignet sich das Skript hervorragend als Vorlage für andere systemweite Konfigurationen über SSH – z. B. zur Verteilung von Zertifikaten, Systemupdates oder Softwareinstallationen.

Ablauf

1. Aufbau einer SSH-Verbindung zu jedem Raspberry Pi (Anmeldedaten aus `lib/config.py`)
2. Ausführung folgender Befehle:
   - `sudo apt update -y`
   - `sudo apt install chrony -y`
   - Upload einer neuen `chrony.conf` nach `/tmp/chrony.conf`
   - Verschieben nach `/etc/chrony/chrony.conf`
   - Neustarten und Aktivieren des Chrony-Dienstes
   - Reboot des Raspberry Pi

Aufruf und Argumente
Das Skript wird im Terminal mit zwei Argumenten ausgeführt:

```
set_up_chrony_via_ssh.py <start_cam> <end_cam>
```

Dabei stehen `start_cam` und `end_cam` für durchnummerierte Raspberry Pis (z. B. 1 bis 20). Die zugehörigen IP-Adressen werden automatisch anhand des Musters `10.50.100.<100 + cam_nr>` generiert.
Beispiel:

```
set_up_chrony_via_ssh.py 1 3
```

Konfiguriert die Geräte `cam01`, `cam02` und `cam03`, entsprechend den IPs `10.50.100.101` bis `10.50.100.103`.

Hinweis
Das Skript kann einfach angepasst werden, um alternative Befehle auszuführen. Es bietet somit eine flexible Grundlage zur Automatisierung administrativer Aufgaben auf mehreren Geräten innerhalb eines Netzwerks.

A.1.6 Zentrale Aufnahmesteuerung mit `master_camera_controller.py`

Zur Durchführung von Aufnahmen mit dem Capture Rig von VolumanXR wird das Programm `master_camera_controller.py` ohne zusätzliche Kommandozeilenargumente gestartet. Beim Start initiiert das Skript automatisch die zugehörigen Remote-Skripte auf allen Raspberry Pis der IP-Liste – ein separater Start über das `launcher.py`-Skript ist nicht erforderlich.

Benutzeroberfläche
Das Programm bietet eine vollwertige grafische Benutzeroberfläche (GUI), über die sowohl Videoaufnahmen (Sessions) als auch Standbilder (Stills) synchron über alle Kameras erstellt werden können.

- **Sessions (Videos):** Der Benutzer muss einen beliebigen Sessionnamen sowie eine gewünschte Zieldatenrate angeben.
- **Stills (Einzelbilder):** Es ist ein Name sowie eine Zielauflösung für die einzelnen Bilder festzulegen.

Die jeweiligen Aufnahmen können über entsprechende Schaltflächen in der Benutzeroberfläche gestartet werden. Nach kurzer Initialisierungszeit beginnt die Aufnahme.

Systemüberwachung
Der gesamte Ablauf sowie der aktuelle Status jedes einzelnen Raspberry Pis können in den Bereichen **Session Info** und **Camera Status** in Echtzeit überwacht werden. So ist jederzeit ersichtlich, welche Geräte aktiv sind und ob mögliche Probleme vorliegen.

A.1.7 Kamerakalibrierung mit `master_camera_controller.py`

Zur einheitlichen Kalibrierung aller Kameras im Kamera-Rig wird ebenfalls das Programm `master_camera_controller.py` verwendet. Es wird ohne zusätzliche Kommandozeilenargumente gestartet. Beim Start initiiert das Programm automatisch die entsprechenden Remote-Skripte auf allen Raspberry Pis der konfigurierten IP-Liste – ein manuelles Starten über das `launcher.py`-Skript ist nicht notwendig.

Benutzeroberfläche
Die grafische Benutzeroberfläche ermöglicht die Konfiguration aller relevanten Kameraparameter über intuitive Eingabefelder, Auswahlmenüs und Schieberegler. Als visuelle Referenz dient eine Live-Ansicht einer frei wählbaren Kamera des Rigs, die direkt in der Oberfläche angezeigt wird.

Synchronisierung und Speicherung
Über den Button `SSave Settings"` werden sämtliche Kameraeinstellungen zentral gespeichert und automatisch auf allen Raspberry Pis synchronisiert. Die Einstellungen werden in der Datei `camera_settings.json` abgelegt. Diese Datei wird anschließend vom Capture Controller bei der Durchführung von Aufnahmen verwendet, um sicherzustellen, dass alle Kameras mit identischer Konfiguration arbeiten.

A.1.8 Datentransfer und Management mit `master_download_manager.py`

Zur Verwaltung und zum Herunterladen der aufgenommenen Daten wird das Programm `master_download_manager.py` verwendet. Beim Start werden automatisch die zugehörigen Remote-Skripte parallel auf allen Raspberry Pis der IP-Liste ausgeführt – ein separater Start der Remote-Skripte über `launcher.py` ist nicht erforderlich.

Benutzeroberfläche
Das Programm stellt eine grafische Benutzeroberfläche zur Verfügung, über die die Aufnahmen zentral verwaltet werden können. Im oberen Bereich der Oberfläche lässt sich ein Zielverzeichnis definieren. Dieses Verzeichnis wird beim Herunterladen und zur Statusprüfung verwendet, um bereits übertragene Dateien zu erkennen.

Funktionen
In einer übersichtlichen Liste werden alle verfügbaren Aufnahmen auf den Raspberry Pis dargestellt, die sich im Ordner `Recordings` befinden. Zu jeder Aufnahme werden Zusatzinformationen angezeigt.
Die folgenden Aktionen können für eine oder mehrere ausgewählte Aufnahmen ausgeführt werden:

- **Download:** Überträgt die ausgewählten Dateien in das definierte Zielverzeichnis.
- **Löschen (lokal oder remote):** Entfernt Aufnahmen entweder vom lokalen Rechner oder von den Raspberry Pis.
- **Konvertieren in Einzelbilder:** Für Sessions kann eine Umwandlung der Videodateien in Einzelbilder (Frames) mit Hilfe von `FFmpeg` erfolgen.

Dieses Tool bietet somit eine zentrale Lösung für den effizienten Datentransfer, das Aufräumen und die Weiterverarbeitung der aufgenommenen Daten.

Offline-Modus
Standardmäßig startet das Skript `master_download_manager.py` beim Programmstart automatisch die Remote-Skripte auf den Agents und fragt diese nach den aktuell vorhandenen Dateien im Ordner `Recordings` ab. Die gefundenen Dateien werden anschließend in der Liste der Benutzeroberfläche angezeigt und mit dem Download-Verzeichnis auf dem Host-Computer abgeglichen. Aufnahmen, die sich ausschließlich lokal befinden, aber auf den Agents bereits gelöscht wurden, werden dabei **nicht** in der Liste angezeigt. Dies sorgt zwar für Übersichtlichkeit, hat jedoch den Nachteil, dass solche Sessions nicht mehr in Frames konvertiert werden können – obwohl dieser Vorgang vollständig lokal auf dem Host-Computer abläuft.
Aus diesem Grund kann das Skript alternativ mit dem zusätzlichen Argument `offline` gestartet werden:

```
python master_download_manager.py offline
```

Der Download Manager startet dann in einem speziellen *Offline-Modus*. In diesem Modus werden ausschließlich die lokal im Download-Verzeichnis vorhandenen Dateien angezeigt, ohne dass die Remote-Skripte auf den Agents gestartet oder abgefragt werden. Das Konvertieren lokal gespeicherter Sessions in Einzelbilder ist dabei aber weiterhin vollständig möglich.

A.1.9 Inbetriebnahme eines neuen Raspberry Pi

Die Inbetriebnahme eines neuen Raspberry Pi im System erfolgt in mehreren Schritten:

1. **Raspberry Pi Image flashen**
   Die am Ende des Projekts bereitgestellten Dateien enthalten ein vorkonfiguriertes Raspberry-Pi-Image, das als Ausgangspunkt für die Konfiguration neuer Geräte verwendet werden kann. Mit dem `Win32DiskImager` lässt sich dieses Image als Custom Image auf eine microSD-Karte flashen. Das Image basiert auf der Lite-Version des Raspberry Pi OS (64-Bit), die keine grafische Desktopumgebung, sondern ausschließlich eine Konsole zur lokalen oder SSH-basierten Nutzung bereitstellt.

   Nach dem Flashen wird dem Gerät automatisch die IP-Adresse `10.50.100.200` zugewiesen. Benutzername und Passwort sind vorkonfiguriert auf `voluman` bzw. `xr`. Im Home-Verzeichnis des Benutzers (`/home/voluman`) befinden sich alle aktuellen Skripte zum Zeitpunkt des Projektabschlusses.

   Das System ist außerdem so eingerichtet, dass es den Distributor unter der IP-Adresse `10.50.100.5` als NTP-Server zur Zeitsynchronisation kontaktiert.

2. **Konfiguration von IP-Adresse und Hostnamen**
   Um einen neuen Raspberry Pi in das bestehende System zu integrieren, müssen seine IP-Adresse und sein Hostname angepasst werden. Dies kann automatisiert über das Skript `master_set_cam_number.py` im Ordner `utils` erfolgen. Das Skript wird auf dem Host-Computer ausgeführt.

   Nach dem Start fragt das Skript nach der gewünschten Gerätenummer. Anschließend kontaktiert es den Raspberry Pi unter der Standard-IP-Adresse `10.50.100.200` und konfiguriert automatisch die neue IP-Adresse sowie den neuen Hostnamen. Danach wird der Raspberry Pi automatisch neu gestartet und startet mit den neuen Einstellungen.

3. **Hinzufügen des Raspberry Pi zur IP-Liste**
   Damit eine Kommunikation zwischen den Master-Skripten und dem neu hinzugefügten Raspberry Pi möglich ist, muss dieser in der Datei `camera_list.json` mit seinem Hostnamen und seiner IP-Adresse eingetragen werden.

   Zusätzlich muss jedem Raspberry Pi ein individueller Wert für die `Lens Position` zugewiesen werden, um ein fokussiertes Bild sicherzustellen. Dieser Wert kann mithilfe des Camera Controllers über einen Slider bestimmt werden. Die über den Camera Controller gespeicherte Konfiguration wird jedoch bei Aufnahmen nicht verwendet; stattdessen greifen die Master-Skripte auf die Werte aus der IP-Liste zurück.

   Es wird empfohlen, verschiedene Fokuswerte zu testen und anhand von aufgenommenen UHD-Bildern subjektiv zu beurteilen. Alternativ kann ein eigenes Skript erstellt werden, das automatisch eine Reihe von Bildern mit unterschiedlichen Fokuswerten aufnimmt.

4. **Aktualisierung der Remote-Skripte (optional)**
   Nachdem der neue Raspberry Pi erfolgreich integriert wurde, kann es sinnvoll sein, die Remote-Skripte auf dem Gerät auf den neuesten Stand zu bringen. Dies ist insbesondere

dann empfehlenswert, wenn Weiterentwicklungen oder Korrekturen an den Skripten nach Erstellung des Images erfolgt sind.

Die Aktualisierung erfolgt über das `launcher.py`-Skript auf dem Host-Computer, mit dem aktuelle Versionen der Skripte über den Befehl `launcher.py upload` auf alle Raspberry Pis verteilt werden können. Nach dem Upload sollte geprüft werden, ob alle Skripte ordnungsgemäß aktualisiert wurden.

A.2 Setup Viewer

Requirements Um den Viewer zu nutzen, braucht es einen Computer, auf dem Google Chrome läuft, und eine VR-Brille. Beide Geräte müssen mit demselben Netzwerk verbunden sein.

Installation Um den Viewer aufzusetzen muss Die Repository des Projektes zuerst geklont werden. Da es sich um eine Private Repository handelt braucht es eine Einladung vom Administrator des Repositorys. Wurde die Repository geklont, navigiert man zum Ordner mit dem Namen "web-viewer". In diesem gibt es eine README-Datei, die alle wichtigen Schritte erklärt zum Aufsetzen, die aus Gründen der Vollständigkeit hier nochmal genannt werden. In der Kommandozeile muss ebenso zum Ordner "web-viewer"navigiert werden, zum Beispiel mit dem Befehl:

```
cd C:\Users\User\path\to\web-viewer
```

In der Kommandozeile im Web-Viewer-Ordner nun, müssen die Abhängigkeiten installiert werden mit dem Befehl:

```
npm install
```

Sind die Abhängigkeiten installiert, gibt es nun zwei Möglichkeiten fortzufahren. Entweder man lässt einen Development-Server laufen mit:

```
npm run dev
```

oder man baut eine fertige Applikation für den Betrieb:

```
npm run build
```

Lässt man einen Development-Server laufen mit dem `dev`-Befehl, öffnet sich automatisch ein Chrome-Browser. Der `build`-Befehl erstellt einen "distOrdner mit allen Dateien für die Produktion.
Um die Applikation auf einer VR-Brille laufen zu lassen, kann man für Chrome eine Browser Erweiterung installieren, die die Webseite an ein angeschlossenes Head-Mount-Display sendet. Das Plugin heißt Ïmmersive-Web-Emulatoründ ist auf Github zu finden Zhang 13.06.2023. Im Browser kann man auf der oberen rechten Seite dann das Add-On anklicken und ein kleines Fenster öffnet sich mit drei Optionen. Beim klicken des Knopfes SSend Page to Device"öffnet sich ein neuer Tab zum anmelden eines Meta-Accounts. Nach der Anmeldung sind alle mit dem Account verbundenen Geräte gelistet und man die Seite wird dem ausgewähltem Gerät gesendet.