TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Erstellung und Darstellung von Gaussian Splats

Optimierung des Arbeitsablaufs, der Fotobearbeitung und Floater Minimierung

Bachelorarbeit

Fachbereich Medienproduktion

Technische Hochschule Ostwestfalen Lippe

eingereicht bei:

Prof. Dr.-Ing. Alexander Kutter

Colin Behrens

eingereicht von:

Fabian Merschel

Matrikelnummer: 15454065

Studiengang: Bachelor of Arts Medienproduktion (B.A)

Lage, den 08.02.2025

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle sinngemäß und wörtlich übernommenen Textstellen aus fremden Quellen wurden kenntlich gemacht.

Lage, den 08.02.2025

Fabian Merschel

Abstract

Gaussian Splatting stellt eine innovative Methode zur 3D-Rekonstruktion dar, die im Vergleich zu klassischen Verfahren wie Photogrammetrie oder NeRFs effizientere Berechnungen und eine bessere Echtzeitanwendung ermöglicht. Trotz dieser Vorteile bestehen Herausforderungen bei der Automatisierung des Arbeitsablaufes, der Optimierung der Bildverarbeitung und der Reduzierung von Artefakten wie Floatern.

Das Ziel dieser Arbeit ist es, anhand verschiedener Methoden herauszufinden, wie die Qualität von Gaußschen Modellen optimiert und der Arbeitsprozess automatisiert werden kann. Dazu wird die Forschungsfrage gestellt*: Mit welchen Ansätzen lässt sich der Workflow effizienter gestalten und die Entstehung von Floatern reduzieren, um die Qualität und Präzision der finalen Modelle zu verbessern?*

Um die Forschungsfrage zu beantworten, wurden unterschiedliche Methoden experimentell untersucht und für die jeweiligen Anwendungsbereiche Programme geschrieben, welche die Arbeitsprozesse automatisieren. Es wurde analysiert, inwieweit die erstellten Programme zur Problemlösung beitragen und welchen Nutzen diese hinsichtlich der Automatisierung des Workflows, der Qualität der Modelle sowie der Reduktion von Floatern bieten.

Die analysierten Ergebnisse zeigen, dass mit dem erstellten Programm, GS_Auto_Ersteller, der gesamte Arbeitsprozess zur Erstellung eines Gaußschen Modells automatisiert werden konnte. Zudem trägt eine qualitativ hochwertige Bildfreistellung der Person oder des Objektes zur Qualität des 3D Modells bei und sorgt für effizientere Trainingszeiten. Außerdem hat die Bildbearbeitung einen Einfluss auf die Floater Minimierung, indem dadurch eine optimierte Sparse Point Cloud erstellt wird und eventuell anfallende Floater unsichtbar gemacht werden. Zusätzlich verhindert eine bereinigte Sparse Point Cloud in Kombination mit den freigestellten Bildern die Entstehung von Floatern.

Weiterführende Forschung könnte sich mit der Verbesserung der Bildverarbeitung durch Deep-Learning-Modelle befassen oder eine Methode entwickeln und diese in das Programm GS_Auto_Ersteller implementieren, welche effektiv die Sparse Point Cloud bereinigt, nachdem diese erstellt wurde.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Pseudocodeverzeichnis

Abkürzungsverzeichnis

API……..Application Programming Interface
AR……Augmented Reality
ASCII……. American Standard Code for Information Interchange
CGI……...Computer-Generated Imagery
CLI……...Command Line Interface
CMD……...……Command Prompt
CNN……Convolutional Neural Network
CSV……Comma-Separated Values
GPU……Graphics Processing Unit
HSV……..Hue, Saturation, Value
MCMC……Markov Chain Monte Carlo
Mask-R-CNN……Mask Region-based Convolutional Neural Network
MLP……..……Multilayer Perceptron
MVS……...Multi-View Stereo
NeRFs……Neural Radiance Fields
PLY……..……Polygon File Format
RGB……..Red, Green, Blue
SfM……...Structure from Motion
SSIM……...Structural Similarity Index Measure
VR……Virtual Reality
XML……..Extensible Markup Language

Glossar

ASCII	Der ASCII-Code ist ein international genormter Binärcode zur Darstellung und Übertragung von Daten, bei dem ursprünglich jedes Zeichen durch eine 8-stellige Binärzahl repräsentiert wurde [1].
Convolutional Networks	Ein Convolutional Neural Network (CNN) ist eine Klasse von künstlichen neuronalen Netzwerken, die speziell für die Verarbeitung von Daten mit einer Gitterstruktur, wie Bildern, entwickelt wurde[2].
Floater	Unerwünschte Gauss-Punkte, die außerhalb der eigentlichen Oberfläche eines 3D-Objekts schweben, oft durch ungenaue Rekonstruktionen oder fehlerhafte Optimierungen entstehen und zu visuellen Artefakten führen können.
Kovarianzmatrix	Die Kovarianzmatrix ist eine symmetrische Matrix, die die Streuung und gegenseitige Abhängigkeit der Daten darstellt

	und zur Definition der Form und Orientierung von 3D-Gaussians verwendet wird [3].
Morphologische Operation	Morphologische Operationen verändern den Inhalt und Struktur von Bildern. Sie lassen Bildstrukturen schrumpfen oder wachsen [4].
Multilayer Perceptron	Ein Multilayer Perceptron (MLP) ist ein künstliches neuronales Netzwerk mit mehreren Schichten, das zur Lösung von Klassifikations- und Regressionsaufgaben eingesetzt wird [5].
Radiance Fields	Ein Radiance Field ist eine Darstellung der Lichtverteilung in einem 3D Raum, welche erfasst, wie Licht mit Oberflächen und Materialien in der Umgebung interagiert [6].
Ray Marching	Ray Marching ist eine Technik, die bei volumetrischen Darstellungen verwendet wird, um Bilder durch Verfolgung des Lichtweges durch ein Volumen zu rendern [7].
Splatting	Splatting ist ein Verfahren, indem 3D Gaussians (Ellipsoids) in 2D Bilder (Ellipsen) für das rendern projiziert werden [8].
SSIM-Wert	Der SSIM-Wert ist ein Maß für die strukturelle Ähnlichkeit zwischen zwei Bildern und wird verwendet, um die Bildqualität zu bewerten [9].
Tiles	Tiles sind nicht überlappende Bildausschnitte (Patches), in die ein Bild unterteilt wird, um die Berechnungseffizienz zu steigern [10].
Voxelgrid	Ein Voxelgrid ist eine regelmäßige, dreidimensionale Gitterstruktur, in der Objekte durch diskrete Volumenelemente, sogenannte Voxels, repräsentiert werden [11].

I. Einleitung

Die Einleitung gibt eine kurze Übersicht über die Problemstellung und Zielsetzung dieser Arbeit. Es wird die Forschungsfrage vorgestellt, die Zielgruppe definiert sowie die praktische Relevanz der Arbeit dargestellt. Im Anschluss wird ein erster Überblick auf die Unterteilung dieser Arbeit gegeben.

1.1 Problemstellung

Die Erzeugung und Darstellung von komplexen 3D-Szenen und Objekten hat sich in den letzten Jahren erheblich weiterentwickelt. Technologien wie Photogrammetrie und Radiance Fields gelten in diesem Bereich als wegweisend. Mit ihrer Unterstützung können 3D-Szenen und Modelle für diverse Anwendungsbereiche wie Computergrafik, Animation oder Virtual Reality gerendert und visualisiert werden. Diese Methoden erzielen zwar beeindruckende Ergebnisse in der fotorealistischen Darstellung, sind jedoch äußerst rechenintensiv, ungeeignet für dynamische Szenen und stoßen bei der Echtzeit-Darstellung an ihre Grenzen. Im Gegensatz dazu bietet Gaussian Splatting eine vielversprechende Alternative. Diese Technik ermöglicht eine effiziente und flexible Darstellung von 3D-Szenen, die sowohl in Echtzeit als auch bei dynamischen Szenen beeindruckende Ergebnisse liefert. Statt Szenen und Objekte durch herkömmliche Herangehensweisen zu rekonstruieren, nutzt Gaussian Splatting sogenannte Gauß-Kerne, was zu neuen Möglichkeiten führt und bisherige Anwendungsbereiche erweitert. Dennoch stellt die Erstellung und Optimierung des Arbeitsablaufes von Gaussian Splats nach wie vor eine Herausforderung dar, insbesondere im Hinblick auf die Automatisierung der Prozesse, der Qualität der erzeugten Modelle und die Minimierung von Artefakten, den sogenannten Floatern.

1.2 Zielsetzung und Forschungsfrage

Das Ziel dieser Bachelorarbeit ist die Entwicklung und Analyse von Methoden zur Optimierung und Automatisierung des Workflows bei der Erstellung von Gaussian Splats. Dabei werden sowohl die Fotobearbeitung als auch die Optimierung der Sparse Point Cloud berücksichtigt.

Im Zentrum der Arbeit steht die Forschungsfrage: *Mit welchen Ansätzen lässt sich der Workflow effizienter gestalten und die Entstehung von Floatern reduzieren, um die Qualität und Präzision der finalen Modelle zu verbessern?*

Schwerpunkte sind die Automatisierung der Arbeitsprozesse von RealityCapture und PostShot sowie die Untersuchung der Rolle der Bildbearbeitung und Sparse Point Cloud Optimierung bei der Minimierung von Floatern.

Ziel ist es, eine Grundlage zu schaffen, die den aktuellen Arbeitsablauf verbessert und innovative Ansätze für mögliche Erweiterungen bildet.

1.3 Zielgruppe

Die Arbeit richtet sich an Studierende der Medienproduktion, die sich mit 3D-Visualisierung und der Optimierung von Workflows beschäftigen. Besonders für diejenigen, die sich mit der 3D-Rekonstruktionstechnik Gaussian Splatting befassen, bietet sie wertvolle Einblicke.

Von Interesse für diese Zielgruppe ist außerdem, die Automatisierung von Prozessen in Hinblick auf RealityCapture und PostShot. Die Ergebnisse dieser Arbeit sollen sowohl für die praktische Anwendung als auch der theoretischen Weiterentwicklung der Technologie dienen.

1.4 Praktische Relevanz

Die in dieser Arbeit entwickelten Ansätze und Methoden zur Optimierung des Workflows und zur Verbesserung der Qualität von Gaussian Splat Modellen weisen ein großes Potenzial für praktische Anwendungen und zukünftige Forschungen für die 3D-Modellierung auf. Die Bedeutung des Themas ergibt sich zudem aus den vielseitigen Anwendungsfeldern der 3D-Rekonstruktion, die von der Unterhaltungsindustrie über den medizinischen Bereich bis hin zur Robotik reichen. Dabei stellt die Präzision der Modelle eine wesentliche Voraussetzung dar, um realistische Rekonstruktion zu gewährleisten. Diese Arbeit schließt bestehende Forschungslücken im Bereich der Bildbearbeitung und deren Auswirkungen auf die Entstehung von Floatern, indem sie erste Ansätze und Erkenntnisse bereitstellt.

1.5 Aufbau der Arbeit

Dieses Kapitel gibt einen Überblick über den Aufbau der Arbeit und erläutert die Inhalte der einzelnen Abschnitte. Die Arbeit gliedert sich in sechs Kapitel.

Kapitel 1 führt in die Thematik ein und beschreibt die Problemstellung. Des Weiteren wird die Forschungsfrage sowie die Zielsetzung der Arbeit hergeleitet.

Kapitel 2 untersucht die Grundlagen des theoretischen Rahmens, indem relevante Theorien vorgestellt und der aktuelle Forschungsstand beleuchtet wird.

Kapitel 3 beschreibt die methodische Vorgehensweise. Es wird erläutert, welche Methoden für das Erstellen der einzelnen Programme verwendet wurden. Zusätzlich wird dessen Funktion und Aufbau erklärt.

Kapitel 4 untersucht und präsentiert die Ergebnisse, inwiefern die verwendeten Methoden zielführend waren und welche qualitativen Resultate erzielt wurden.

Kapitel 5 diskutiert die Ergebnisse. Hier wird kritisch reflektiert, inwiefern die Forschungsfrage beantwortet werden konnte, welche Beschränkungen vorliegen und welche Rückschlüsse sich daraus ergeben.

Kapitel 6 fasst die zentralen Erkenntnisse der Arbeit zusammen, zieht ein Fazit und gibt einen Ausblick auf mögliche zukünftige Forschungen.

II. Theoretischer Rahmen

In diesem Kapitel wird der aktuelle Forschungsstand zum 3D Gaussian Splatting erläutert. Dabei werden verschiedene Technologien zur 3D-Rekonstruktion vorgestellt und miteinander verglichen. Zudem werden unterschiedliche Methoden zur Bildsegmentierung und Freistellung beschrieben, die im weiteren Verlauf der Arbeit experimentell untersucht werden.

2.1 Einführung in Gaussian Splats

Die 3D-Rekonstruktion von Personen ist ein wesentlicher Bestandteil moderner Technologien, die in Bereichen wie CGI und der Medienproduktion Anwendung finden. Ziel ist es, aus einer realen Person eine exakte Kopie als digitales Modell zu erstellen, das realistische Darstellungen sowohl für statische als auch dynamische Anwendungen ermöglicht. Dieser Prozess beruht auf der Erfassung und Verarbeitung von Bild- oder Sensordaten, die anschließend durch spezialisierte Algorithmen in Punktwolken oder andere digitale Repräsentationen wie Meshes umgewandelt werden.

Innerhalb des theoretischen Rahmens wird der Fokus auf drei zentrale Technologien gelegt, die in der modernen 3D-Rekonstruktion eingesetzt werden. In dem folgenden Abschnitt wird eine kurze Übersicht zu den Technologien gegeben, die im Anschluss genauer beleuchtet werden.

- **Photogrammetrie**
  Die Photogrammetrie ist eine bewährte Methode, die Bilder aus verschiedenen Perspektiven verarbeitet, um 3D Modelle zu rekonstruieren. Sie verwendet Technologien wie Structure from Motion (SfM) und Multi-View Stereo (MVS), um präzise Punktwolken oder Mesh-Modelle zu erstellen. Photogrammetrie bietet hohe Präzision, erfordert jedoch eine Vielzahl an Bildern sowie rechenintensive Nachbearbeitungen, was den Workflow verlangsamen kann.

- **NeRFs (Neural Radiance Fields)**
  NeRFs repräsentieren eine neuartige Methode der 3D-Rekonstruktion, bei der neuronale Netze genutzt werden, um Lichtstrahlen und Oberflächeneigenschaften zu modellieren. Anstatt das 3D-Objekt als Punktwolke, Mesh-Modell oder Pixel zu speichern, stellt ein NeRF die Szene als eine mathematische Funktion dar, wodurch besonders detailreiche und visuell ansprechende Ergebnisse entstehen. Dieser Prozess erfordert eine hohe Rechenleistung und ist daher weniger effizient für Echtzeitanwendungen geeignet.

- **Gaussian Splatting**
  Gaussian Splatting bietet einen innovativen Ansatz zur Darstellung von Punktwolken, indem Gauß-Kerne verwendet werden. Diese beschreiben Datenpunkte als glockenförmige Funktionen, die in ihrer Form und Ausbreitung durch Standardabweichung und Mittelwert definiert werden. Dadurch entsteht eine glatte, speichereffiziente und kontinuierliche Repräsentation [3]. Besonders bei der 3D-Rekonstruktion von Personen ermöglicht sie eine schnelle Verarbeitung und eignet sich ideal für Anwendungen mit begrenztem Budget oder Echtzeit-Anforderungen.

2.2 Forschungslücken und Grundlagen der vorliegenden Arbeit

Obwohl in den letzten Jahren zunehmend mehr wissenschaftliche Beiträge zum Thema Gaussian Splatting veröffentlicht wurden, befindet sich diese Technologie noch in einem frühen Entwicklungsstadium. Dies erschwert eine umfassende Betrachtung der theoretischen Grundlagen, da die bisherigen Forschungen sich hauptsächlich auf die technischen Grundlagen und deren Optimierungsansätze konzentriert.

Trotz der Fortschritte in der 3D-Rekonstruktion gibt es derzeit keine spezifischen wissenschaftlichen Untersuchungen, die sich mit der Frage auseinandersetzen, ob das Freistellen von Bildern die Qualität von Gaußschen Modellen verbessert oder das Auftreten von sogenannten "Floatern" minimiert. Während Arbeiten zur Optimierung von Gaussian Splatting existieren, bleibt der Einfluss von Bildbearbeitungsschritten wie dem Freistellen unberührt.

Die in dieser Arbeit durchgeführten Experimente basieren daher überwiegend auf eigenen Hypothesen und Annahmen. Ziel ist es, diese Forschungslücke zu schließen. Dies soll einerseits das bisherige Forschungsfeld zur Verbesserung der Qualität von Gaußschen Modellen erweitern und zusätzlich neue Ansätze für zukünftige Forschungen liefern.

2.3 Überblick und Vergleich mit verwandten Technologien

2.3.1 Photogrammetrie

Die Photogrammetrie ist ein Verfahren, bei dem aus Fotografien präzise dreidimensionale Modelle von Objekten oder Landschaften erstellt werden [12]. Diese Technologie basiert auf der Aufnahme von Bildern aus verschiedenen Blickwinkeln. Diese werden durch eine spezielle Software analysiert. Es werden gemeinsame Punkte (Features) durch bestimmte Algorithmen wie Structure from Motion (SfM) [13] oder Stereografie identifiziert und aus diesen Daten sowohl eine Punktwolke als auch ein Mesh basiertes 3D-Modell rekonstruiert [12].

Die Photogrammetrie wird in unterschiedlichen Bereichen eingesetzt. In der Archäologie wird diese zur Dokumentation und Rekonstruktion von archäologischen Stätten und historischen Gebäuden verwendet. In der Industrie kommt sie zur Optimierung von Bauabläufen zur Qualitätskontrolle zum Einsatz. Eines der größten Einsatzgebiete der Photogrammmetrie stellt die Unterhaltungsindustrie dar, in der fotorealistische Assets besonders von Bedeutung sind [14].

Der Prozess der Photogrammmetrie ist rechenintensiv. Es werden komplexe Algorithmen für die Rekonstruktion verwendet, etwa zur Feature-Erkennung und Triangulation. Diese Prozesse erfordern hohe Rechenleistung und sind zeitaufwendig, was sie für Echtzeitanwendungen ungeeignet macht [15].

2.3.2 Neural Radiance Fields (NeRFs)

Die Technik der Neural Radiance Fields (NeRFs) nutzt neuronale Netzwerke, um 3D-Szenen aus 2D-Bildern zu rekonstruieren. Anders als bei herkömmlichen Methoden basieren NeRFs auf einer impliziten Darstellung von 3D-Daten. Dabei werden Informationen wie Farbe und

Dichte eines Punktes in Abhängigkeit der Position und Blickrichtung auf die Szene von dem neuronalen Netzwerk gelernt und gespeichert [16].

Die Struktur eines NeRFs bildet ein mehrschichtiges Perceptron (MLP). Dieses schätzt die Intensität des Lichtes und die Dichte an jedem Punkt der Szene, wodurch die Lichtstrahlen zwischen Kamera und Szene simuliert werden. Mit zusätzlichen Interaktionen durch SfM [13] und MVS [17] Algorithmen entsteht so ein fotorealistisches 3D-Modell. Fortschritte in der Optimierung und Effizienz des Trainingsprozesses haben sowohl die Geschwindigkeit als auch die Genauigkeit dieser Methode deutlich gesteigert. [18]

NeRFs finden Anwendung in zahlreichen Bereichen, darunter in der Erstellung von 3D-Szenen für Virtual (VR) und Augmented Reality (AR), der Bildbearbeitung oder der Modellierung menschlicher Avatare. Sie werden auch in der Filmindustrie genutzt, um fotorealistische Umgebungen zu erstellen sowie in der Architektur und im Metaverse, um detailreiche virtuelle Welten zu schaffen [18].

Obwohl NeRFs hervorragende Ergebnisse bei der Erstellung fotorealistischer Bilder erzielen, wird der Bedarf an schnelleren und effizienteren Rendering Methoden immer deutlicher, besonders für Anwendungen (z.B. VR und autonomes Fahren), die sehr empfindlich auf Latenz reagieren. [10]

2.3.3 Gaussian Splatting

3D-Gaussian-Splatting hat sich vor kurzem als eine innovative Technik im Bereich der Radiance Fields und Computergrafik etabliert. Dieser innovative Ansatz kennzeichnet sich durch die Nutzung von Millionen lernbaren 3D-Gauss-Funktionen aus und stellt einen bedeutenden Ansatz im Gegensatz zu traditionellen Punktwolkenansätzen dar. Anstatt die gemessenen Parameter wie Farbe und Dichte als diskreten Punkt abzuspeichern, verwendet Gaussian Splatting Wahrscheinlichkeitsverteilungen in Form von Gaus-Kernen, um eine Szene zu rekonstruieren und darzustellen [3], [10].

3D GS verspricht durch seine expliziten Szenenrepräsentationen und einen differenzierbaren Rendering-Algorithmus nicht nur die Möglichkeit, in Echtzeit zu rendern, sondern eröffnet auch bislang unerreichte Möglichkeiten zur Nachbearbeitung. Diese Innovation positioniert 3D GS als potenziellen Wendepunkt für die nächste Generation der 3D-Rekonstruktion und -Repräsentation. Durch diese Technik ergibt sich ein breites Spektrum an Anwendungen, darunter 3D-Modellierung und Animation, Roboternavigation, der Erhalt historischer Stätten, Augmented und Virtual Reality sowie autonomes Fahren [3].

Gaussian Splatting kombiniert die Vorteile der impliziten Radiance Fields, die bei der NeRF-Technologie zum Einsatz kommt [16], mit den Vorteilen der expliziten Radiance Fields, welche zum Beispiel bei der Voxelgrid Technologie verwendet wird [19]. 3D GS nutzt die Stärken beider Paradigmen, indem es lernbare 3D Gaussians als flexible und effiziente Darstellungen einsetzt [3].

Dabei stellen implizite Radiance Fields Lichtverteilungen in einer Szene dar, ohne die Geometrie der Szene ausdrücklich zu definieren. Im Bereich des Deep Learning werden oft neuronale Netzwerke benötigt, um eine kontinuierliche volumetrische Szenendarstellung zu

lernen [20], [21]. Dieses Format ermöglicht eine differenzierbare und kompakte Darstellung komplexer Szenen, allerdings oft auf Kosten einer hohen Rechenlast durch volumetrisches Ray Marching [3].

Die expliziten Radiance Fields hingegen stellen die Lichtverteilung direkt in einer diskreten räumlichen Struktur dar, wie es beispielsweise bei einem Voxelgrid oder einer Punktwolke der Fall ist [19], [22]. Diese erlauben einen direkteren und schnelleren Zugang zu Radiance Daten, jedoch zulasten eines höheren Speicherverbrauches und möglicherweise einer niedrigeren Auflösung [3].

Das Training eines 3D-Modells mittels 3D GS benötigt eine Sammlung von Bildern sowie zusätzliche Daten der Kameraausrichtungen und der Punktwolke der zu erstellenden Szene.

Diese Szene wird durch eine Vielzahl an 3D-Gaussians dargestellt, die jeweils durch Parameter wie Position, Opazität, Farbe und eine Kovarianzmatrix definiert sind. Dabei wird die Farbe ansichtsabhängig modelliert, um visuelle Veränderungen aus verschiedenen Blickwinkeln zu ermöglichen. Alle Parameter sind lernbar und werden während des Trainingsprozesses optimiert [16], [23], [24].

Die Erstellung eines 3D-Gaußschen Modells beinhaltet mehrere Schritte:

Im ersten Schritt wird eine 3D Sparse Point Cloud durch die Verwendung der Structure from Motion (SfM) Technologie erzeugt, welche aus 2D Bilddaten eine 3D-Punktwolke rekonstruiert. Anschließend wird jeder Punkt in einen Gauß umgewandelt. Dies ermöglicht eine Rasterung. Zu diesem Zeitpunkt beinhalten die umgewandelten Gaussians lediglich die Positions- und Farbinformationen, welche aus den SfM-Daten abgeleitet werden [25]. Das Training eines 3D-Gaußschen Modells beginnt mit der Initialisierung der 3D-Gaussians. Jeder Gaussian wird durch seine Position, eine Kovarianzmatrix, die seine Form und Orientierung beschreibt, eine Opazität sowie Spherical harmonics für die farbliche Darstellung definiert. Diese Parameter bilden die Grundlage für die spätere Optimierung [3].

Im nächsten Schritt werden die 3D Gaussians auf die 2D Bildfläche projiziert. Dieser Prozess wird als Splatting bezeichnet und ist ein Kernprozess, da es die Szene für das Rendering aufbereitet und differenzierbar macht. Dies ist eine notwendige Voraussetzung für das Tiling und die Optimierung. Nachdem die Gaussians gesplattet wurden, wird die Bildfläche in Kacheln unterteilt (Tiling), um die Berechnung für GPUs zu optimieren [3], [8].

Der anschließende Optimierungsprozess verfeinert wiederholend die Parameter der Gaussians (Position, Kovarianzmatrix, Opazität und Farbe). Dies geschieht durch den Vergleich der gerenderten Ergebnisse mit den Referenzbildern. Ohne die vorherigen Schritte des Splatting und Tiling könnten Gradienten Berechnungen und Rendering nicht effizient durchgeführt werden [3]. Durch die adaptive Dichtekontrolle der Gaussians werden unzureichend rekonstruierte Bereiche der Szene durch das Klonen bestehender Gaussians ergänzt, während große Gaussians in hochdetaillierten Regionen in kleinere Einheiten aufgeteilt werden. Gleichzeitig werden Gaussians mit sehr geringer Opazität entfernt, um die Speichereffizienz zu erhöhen [3].

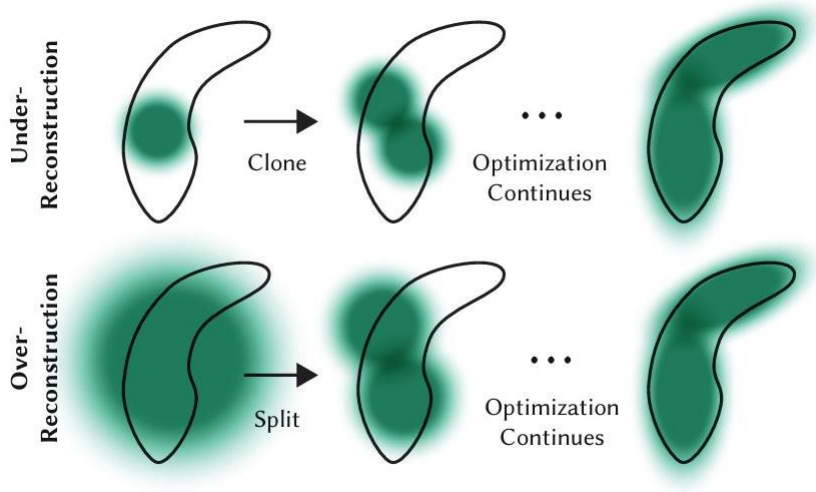


Abbildung 1 - Über- und Unterrekonstruktion von Gaussians (Clone & Split)
Quelle: Kerbl et al. [3]

Nach mehreren Iterationen entsteht so ein trainiertes Gaußschen Modell, das die Szene exakt und hochdetailliert darstellt.

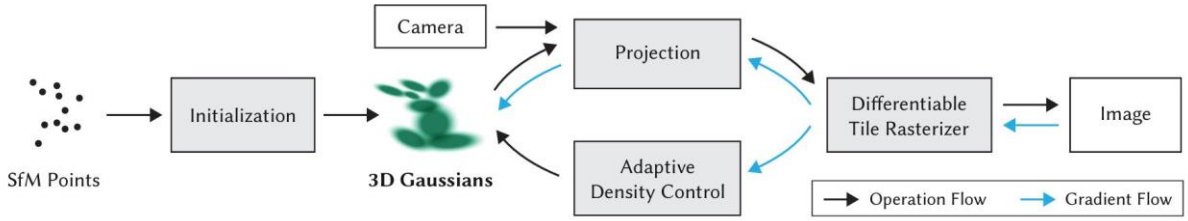


Abbildung 2 - Visuelle Darstellung der Erzeugung von Gaussian Splats
Quelle: Kerbl et al. [3]

Tabelle 1 - Gegenüberstellung der Eigenschaften von Gaussian Splatting, NeRFs und Photogrammetrie Quelle: Eigene Darstellung

Kriterium	**Gaussian Splatting**	**NeRFs**	**Photogrammetrie**
Rechenaufwand	Niedrig, ermöglicht Echtzeit-Rendering	Hoch, erfordert erhebliche Rechenressourcen	Variiert, abhängig von Bildanzahl und -qualität
Ergebnisqualität	Hohe visuelle Qualität, jedoch weniger detailliert als NeRFs	Sehr hohe Detailtreue und realistische Beleuchtung	Hohe Genauigkeit bei Texturen und Geometrie
Anwendungsbereiche	Echtzeit-Visualisierungen, VR/AR	Filmproduktion, virtuelle Welten	Architektur, Denkmalpflege, Spieleentwicklung
Datenanforderungen	Benötigt weniger Eingabedaten	Erfordert umfangreiche Datensätze	Abhängig von der Komplexität des Objekts

Im Vergleich zu herkömmlichen Methoden wie der Photogrammetrie oder NeRFs bietet Gaussian Splatting erhebliche Vorteile. Während Photogrammetrie eine hohe Präzision ermöglicht, ist der Prozess durch die Notwendigkeit zahlreicher Bilder und umfangreicher Nachbearbeitung langsam und ressourcenintensiv. NeRFs wiederum liefern beeindruckende visuelle Ergebnisse, erfordern jedoch immense Rechenleistung, was ihre Praktikabilität einschränkt. Gaussian Splatting kombiniert hingegen Effizienz und Flexibilität, da es eine speicherschonende und schnelle Verarbeitung ermöglicht. Diese Vorteile machen die Technologie besonders interessant für Echtzeit-Anwendungen und Szenarien mit begrenztem Budget.

2.4 Methoden der automatisierten Bildbearbeitung

Das Freistellen von Bildern ist eine zentrale Aufgabe in der Bildbearbeitung, die häufig in Bereichen wie der Medienproduktion, der Objekterkennung und der 3D-Modellierung Anwendung findet. Dieser Prozess ist traditionell mit hohem manuellem Aufwand verbunden. Die Automatisierung dieses Prozesses gewinnt zunehmend an Bedeutung, da sie Effizienz und Skalierbarkeit gewährleistet.

2.4.1 Differenzbildung von Bildern

Eine der wesentlichen Vorgehensweisen zur Hintergrundentfernung basiert auf der Differenzbildung von zwei Bildern. Durch die Subtraktion der beiden Bilder können Unterschiede identifiziert und Bereiche isoliert werden, die das Objekt repräsentieren. Diese Methode ist besonders effektiv in Umgebungen mit statischen Hintergründen und konstanter Beleuchtung. In der Arbeit von Bharti [26] wird ein ähnlicher Ansatz beschrieben, wie die QR-Zerlegung, die Bilder in Blöcke aufteilt und schwach bewegte Hintergrundelemente identifiziert.

2.4.2 Farbraumanalyse

Ein alternativer Ansatz basiert auf der Analyse des Farbraums des Bildes. Farbräume wie RGB oder HSV ermöglichen es, Unterschiede zwischen dem Hintergrund und dem Vordergrundobjekt auf der Basis ihrer Farbwerte zu identifizieren. Beispielsweise kann bei kontrastreichen Hintergründen (schwarz/weiß) ein Schwellenwert eingesetzt werden, um den Hintergrund zu entfernen.

Shimodaira [27] beschreibt eine Methode zur automatischen Farbbildsegmentierung, die auf dem "Seeded Region Growing and Merging"-Ansatz basiert. Diese Technik nutzt Farbinformationen, um Bilder in homogene Regionen zu unterteilen, was die Grundlage für die Trennung von Vorder- und Hintergrund bildet.

2.4.3 Photoshop-Skripting mit Aktionen

Eine weitere Methode zur Automatisierung des Freistellungsprozesses nutzt die Möglichkeit von Skripten für die Software Photoshop. Dadurch können wiederholbare Bearbeitungsschritte als „Aktionen“ aufgezeichnet und diese anschließend automatisiert auf eine große Anzahl von Bildern angewendet werden.

Adobe besitzt eine ausführliche Dokumentation, die beschreibt, wie Benutzer komplexe Aufgaben mit Aktionen automatisieren können [28]. Dieser Ansatz eignet sich besonders gut für standardisierte Workflows in der Bildbearbeitung, da er manuelle Eingriffe minimiert und sich leicht an spezifische Anforderungen anpassen lässt.

2.4.4 Deep Learning basierte Ansätze

Vortrainierte Deep-Learning-Modelle wie U-Net oder Mask R-CNN haben sich als effektive Werkzeuge für die Hintergrundentfernung etabliert. Diese Modelle sind in der Lage, komplexe Muster und Strukturen in Bildern zu erkennen und pixelgenaue Masken zu erstellen. Python-Bibliotheken wie rembg verwenden solche vortrainierten Netzwerke für die automatische Hintergrundentfernung. Ronneberger et al. [29] beschreiben die Funktionsweise des U-Net-Modells, das für Segmentierungsaufgaben entwickelt wurde. Dieses Modell nutzt Convolutional Neural Networks (CNNs), um Merkmale in Bildern zu identifizieren und Hintergrundbereiche automatisch zu trennen.

2.5 Eingesetzte Software und Tools

Für die Umsetzung der Arbeit standen folgende Ressourcen zur Verfügung:

- **Hardware-Hochschulrechner:**

  AMD Ryzen 9 5950X 16-Core Processor 3.40 GHz
  64 GB RAM
  NVIDIA GeForce RTX 4070

- **Hardware Privater Rechner:**

  AMD Ryzen 5 2600 6-Core Processor, 3400 MHz
  32 GB RAM
  NVIDIA GTX 1080

- **Scanner:**

  Full-Body Scanner Neo der Firma botspot [30]

Tabelle 2 - Verwendete Software

Programm	**Funktion**
CaptureGRID	*Einstellung der Kameraeinstellungen*
Scanner Control	*Einstellen des Shutter Lags und Bedienung des Scanners*
Photoshop	*Freistellung der gescannten Person*
Visual Studio Code	*Erstellung verschiedener Skripte mittels Python*
Blender	*Ermittlung von Koordinaten zur Punktwolken Bereinigung*
CMD-Konsole	*Basis einzelner Programme & Recherche verwendbarer Befehle*
Text-Editor	*Erstellung verschiedener Skripte*
RealityCapture	*Erstellung der Spars Point Cloud & der Cameraalignments*
PostShot	*Erstellung der Gaussian Splats*
AutoHotKey	*Erstellung eines Skriptes*
SuperSplat	*Validierung und Bearbeitung der Gaussian Splat Modelle*

III. Methodik

Im folgenden Kapitel werden die verwendeten Methoden und erstellten Programme erläutert. Dabei wird auf die Herangehensweise und verwendete Software eingegangen. Das Kapitel befasst sich mit dem bisherigen Arbeitsablauf und den dafür erstellten Programmen zur Automatisierung. Die Codes der jeweiligen Programme werden in diesem Kapitel durch mehrere Blöcke Pseudocode dargestellt. Die echten Codes können im Anhang eingesehen werden.

3.1 Beschreibung des bisherigen manuellen Arbeitsablaufs

Um den Kontext des praktischen Parts dieser Arbeit besser nachvollziehen zu können, wird in diesem Kapitel der gesamte bisherige Arbeitsablauf zur Erstellung der Gaussian Splats beschrieben. Um aus einem Datenset von Bildern ein fertiges Modell zu kreieren, sind mehrere Schritte nötig.

3.1.1 Scanprozess der Person

Der erste Schritt besteht darin, die zu scannende Person mittig im Scanner zu positionieren. Dabei helfen die am Boden angebrachten Markierungen als Orientierungshilfe. In der Regel ist die eingenommene Pose nicht entscheidend, jedoch empfiehlt es sich, je nach Verwendungszweck des Modells, eine A- oder T-Pose einzunehmen.

Die externe Steuerung des Scanners erfolgt über die Software Scanner Control, mit der sich verschiedene Kameraeinstellungen vornehmen lassen, wie beispielsweise der Shutter-Lag. Zusätzlich ermöglicht Scanner Control die Festlegung des Projektnamens und das Auslösen der Kameras. Nach der Aufnahme werden die Bilder automatisch unter dem zuvor definierten Dateipfad im entsprechenden Ordner mit dem angegebenen Projektnamen gespeichert.

3.1.2 Import in RealityCapture

Die aufgenommenen Fotos werden im nächsten Schritt in das Programm RealityCapture importiert. Dort wird der Befehl "Align Images" verwendet, um die Camera Alignments zu erstellen und eine erste Punktwolke (Sparse Point Cloud) zu generieren. Diese Punktwolke wird anschließend als PLY-Datei exportiert, während die Camera Alignment-Daten als CSV-Datei gespeichert werden. Eine unerlässliche Exporteinstellung für die Punktwolke, die für die Weiterverarbeitung in PostShot erforderlich ist, besteht darin, das ASCII-Format auf „false“ zu setzen. Falls ein zusätzliches Mesh-Modell der Person benötigt wird, kann dieses ebenfalls in RealityCapture erstellt werden. Für die Erstellung eines Gaussian Splat Modells ist dies jedoch nicht notwendig.

3.1.3 Verarbeitung in PostShot

Nach dem Export der zuvor beschriebenen Dateien aus RealityCapture ist das Datenset für die Erstellung der Gaussian Splats vollständig. Es umfasst die folgenden Elemente: einen Ordner mit den Bildern, die Datei der Sparse Point Cloud sowie die Camera Alignments-Datei. Dieses Datenset wird anschließend in das Programm PostShot importiert, entweder per Drag-and-Drop oder über die integrierte Importfunktion.

Im Programm besteht die Möglichkeit, verschiedene Parameter anzupassen, wie beispielsweise die Modellart (z. B. MCMC), die maximale Anzahl der Trainingsschritte oder die maximale Anzahl der zu erzeugenden Splats. Nachdem alle notwendigen Einstellungen vorgenommen wurden, wird der Trainingsprozess gestartet. Nach Abschluss des Trainings wird die Projektdatei gespeichert, und das erzeugte Gaussian Splat Modell wird im PLY-Format exportiert.

3.1.4 Visualisierung und Aufbereitung mit SuperSplat

Im finalen Schritt wird das erzeugte Modell in die Browsersoftware SuperSplat importiert. Dort erfolgt die manuelle Entfernung von Floatern mithilfe verschiedener Auswahlwerkzeuge, um das Modell zu bereinigen. Anschließend kann das bereinigte Modell überprüft und begutachtet werden. Wenn das Ergebnis zufriedenstellend ist, wird die bereinigte Datei erneut als .ply-Datei exportiert.

3.2 Automatisierung durch Programmentwicklung

3.2.1 Optimierung von RealityCapture und PostShot

Beide Programme dienen dazu, aus einer Reihe von Bildern dreidimensionale Rekonstruktionen von Szenen oder Objekten zu erstellen. Die Erstellung von Gaußschen Modellen setzt unterschiedliche Schritte voraus, die bereits in Kapitel 5: Manueller Ablauf beschrieben wurden. Während RealityCapture eine auf der Structure-from-Motion-(SfM)-Technologie basierende Methode zur Erstellung von Sparse-Point-Clouds nutzt, verwendet PostShot eine Methode, um 3D-Szenen mithilfe von Gaußschen Verteilungen zu rekonstruieren. Auch für diese Arbeitsprozesse wurden unterschiedliche Ansätze getestet, um eine funktionierende, zuverlässige Automatisierung zu gewährleisten.

Teilautomatisierung RealityCapture

Mit der zu diesem Zeitpunkt gesammelten Erfahrung im Bereich der Programmentwicklung mittels Python, bestand der erste Versuch darin, einen Teil des Arbeitsprozesses von RealityCapture über einen selbstgeschriebenen Code autonom zu gestalten. Das Ziel war es, eine Grundlage zu schaffen, auf die aufgebaut werden kann. Das erstellte Python-Skript dient dazu, bereits existierende Projekte zu starten, einen vorher angepassten und exportierten Rekonstruktionsbereich zu laden, um die gewünschte Punktwolke zu importieren und im Anschluss die Point Cloud zu exportieren. Die für diesen Vorgang benötigten Bibliotheken sind:

- **OS:** Zur Überprüfung der Existenz von Dateien und Verzeichnissen.
- **Subprocess:** Um externe Programme zu starten und deren Aufgaben zu verarbeiten.
- **Pyautogui:** Zur Simulation von Mausklicks und Interaktionen mit der Benutzeroberfläche.
- **Time**: Um Verzögerungen einzufügen und Zeitlimits zu überwachen.
- **Pygetwindow**: Zum Identifizieren und Aktivieren von Programmfenstern.

Code 1 - Pseudocode Teilautomatisierung RealityCaprutre

```
START
1. Prüfe, ob alle benötigten Dateien existieren:
   a. RealityCapture.exe
   b. Projektdatei
   c. Rekonstruktionsregion
2. Wenn alle Dateien vorhanden:
   a. Starte RealityCapture mit dem Projekt und dem
Rekonstruktionsbereich.
   b. Bringe das RealityCapture-Fenster in den Vordergrund.
   c. Warte, bis das Fenster aktiv ist.
   d. Simuliere Mausklicks, um die Point Cloud zu exportieren.
   e. Zeige Nachricht: "Export abgeschlossen. "
3. Anderenfalls:
   a. Zeige Warnung: "Fehlende Dateien. "
END
```

Das Programm steuert RealityCapture nicht direkt an, sondern nutzt Funktionen, die nach gewissen Zeitspannen bestimmte Pixelregionen anklicken. Die Definition des Exportpfades sowie die Exporteinstellungen müssen bei dieser Methode weiterhin manuell vorgenommen werden.

Automatisierung des PostShot Prozesses

PostShot berechnet aus dem gesammelten Datensatz der Bilder, Point Cloud und Camera Alignments das Gaußsche Modell. Über Command-Line-Befehle ist es möglich, PostShot über die Postshot-cli.exe anzusteuern und bestimmte Befehle ausführen zu lassen. Über eine Batch-Datei lassen sich diese Befehle zusammen mit Parametern des Trainingsprozesses der Gaussians abspeichern und ausführen. Die zur Verfügung stehenden Funktionen lassen sich durch die Eingabe von ("<Installationspfad>\Jawset PostShot\bin\postshot-cli.exe" –help) in der vorinstallierten Kommandozeile des Computers ausgeben. Das entwickelte Skript ruft die postshot-cli.exe auf und lädt verschiedene Eingabedaten wie Bilder, Point Clouds und Kameraausrichtungen. Zusätzlich wird die Ausgabedatei bzw. der Export der Projekt- und Modelldatei definiert und Optionen für das zu trainierende Gaußsche Modell spezifiziert, wie Profil (MCMC) und Dateiname. Um auftretende Fehlermeldungen zu dokumentieren und auszuwerten, gibt das Skript zusätzlich eine Textdatei mit den jeweiligen Informationen aus.

Erstellung des finalen Programms GS_Auto_Ersteller

Die bisher erstellten Programme für RealityCapture und PostShot automatisieren einzelne Arbeitsschritte mit unterschiedlichen Methoden. Das finale Programm GS_Auto_Ersteller automatisiert den gesamten Arbeitsablauf und kombiniert den Workflow von RealityCapture und PostShot. Diese Methode beruht auf einem GitHub Beitrag von Azad Balabanian [31], der eine einzige Batchdatei nutzt, um den gesamten Arbeitsprozess zu optimieren. Im ersten Abschnitt des Codes werden die benötigten Pfade zu den Programmen, Ordnern oder Einstellungen definiert.

Code 2 - Pseudocode GS_Auto_Ersteller (Eigene Darstellung auf Basis von Quelle [31]) Abschnitt 1:

```
START
1. Setze die Pfade:
   a. Pfad zu RealityCapture ("RealityCapture.exe")
   b. Pfad zu Postshot ("Postshot-cli.exe")
   c. Projektwurzelverzeichnis ("RootFolder")
   d. Eingabeordner mit Bildern ("Images")
   e. Pfad zu den RealityCapture-Einstellungen ("SettingsFolder")
   f. Speicherort und Name des RealityCapture-Projekts ("Project")
```

Der zweite Abschnitt startet den RealityCapture Prozess. Dieser berechnet aus den importierten Bildern zwei Dateien, die für die Erstellung der Gaussian-Splats essenziell sind. Durch die Structure-from-Motion-(SFM)-Technologie wird aus den importierten Bildern eine Sparse-Point-Cloud erstellt, die zusammen mit den Kameraausrichtungen das Fundament für die Erstellung der Gaussian-Splats darstellt. Zusätzlich werden alle relevanten Dateien wie Projekt-, Punktwolken- und Kameraausrichtungsdatei unter den in Abschnitt 1 definierten Pfaden exportiert und gespeichert.

Code 3 - Pseudocode GS_Auto_Ersteller (Eigene Darstellung auf Basis von Quelle [31]) Abschnitt 2:

```
2. Starte RealityCapture:
   a. Erstelle eine neue Szene.
   b. Aktiviere die Option, Unterordner einbeziehen.
   c. Füge den Ordner mit den Bildern hinzu.
   d. Speichere das Projekt.
   e. Führe eine Ausrichtungsoperation durch.
   f. Wähle die größte zusammenhängende Komponente aus.
   g. Exportiere die Registrierungsdaten als "registeration.csv" unter
Verwendung der XML-Einstellungen.
   h. Exportiere die Punktwolke als "pointcloud.ply" unter Verwendung
der XML-Einstellungen.
   i. Speichere das Projekt erneut.
   j. Beende RealityCapture.
```

Um die richtigen Parameter für den Export der CSV- und PLY-Datei sicherzustellen, gibt es zusätzlich zwei XML-Dateien, die entsprechende Einstellungen beinhalten. RealityCapture greift vor dem Export der jeweiligen Datei auf diese Einstellungen zurück.

Im dritten Abschnitt des Skripts wird die Durchführung der Erstellung des Gaußschen Modells mittels der Postshot-cli gestartet. Das nun zur Verfügung stehende Datenset, bestehend aus Bildern, Punktwolke und Kameraausrichtungen, wird importiert und mit den im Code hinterlegten Parametern wie Profil, Bildskalierung, maximalen Trainingsschritten und maximaler Anzahl zu erzeugende Splats zu einem Gaußschen Modell trainiert. Nach Abschluss des Vorgangs speichert sich das Projekt automatisch im in Abschnitt 1 definierten Ordner ab.

Code 4 - Pseudocode GS_Auto_Ersteller (Eigene Darstellung auf Basis von Quelle [31]) Abschnitt 3:

```
3. Starte Postshot:
   a. Importiere die Bilder.
   b. Zeige den Trainingsfehler während des Prozesses an.
   c. Nutze das Profil "Splat MCMC".
   d. Setze die maximale Bildgröße auf 1600.
   e. Begrenze die Trainingsschritte auf 1.
   f. Begrenze die maximale Anzahl an Splats auf 1000.
   g. Speichere das Ergebnis als "output.psht" im Projektverzeichnis.
4. Zeige Nachricht: "Fertig! "
END
```

Diese Batch-Datei wurde im nächsten Schritt weiter optimiert, um auftretende Fehlermeldungen zu eliminieren und zusätzlich das Gaußsche Modell zu exportieren. Dafür konnte der Code aus den Abschnitten 1 und 2 beibehalten werden. Lediglich in Absatz 3 wurden Änderungen vorgenommen.

Code 5 - Pseudocode GS_Auto_Ersteller (Optimiert) Abschnitt 3:

```
3. Starte Postshot im "Train-Modus":
   a. Importiere die Bilder.
   b. Zeige den Trainingsfehler während des Prozesses an.
   c. Nutze das Profil "Splat MCMC".
   d. Setze die maximale Bildgröße auf 1600.
   e. Begrenze die Trainingsschritte auf 1.
   f. Begrenze die maximale Anzahl an Splats auf 1000.
   g. Speichere das Ergebnis als "output.psht" im Projektverzeichnis.
   h. Exportiere das Gaußsche Modell als "pointcloud_exported.ply".
4. Pausiere den Prozess für die Überprüfung durch den Benutzer.
5. Zeige Nachricht: "Fertig! "
END
```

Den gesamten Code von Azad Balabanian sowie den vom Autor optimierten, finden Sie im Anhang.

3.2.2 Optimierung der Kameraeinstellungen & Erstellung des Selbstauslösers

Der Scanner ist bei der Erstellung von Bildern anfällig für Fehlbelichtungen. Dies liegt daran, dass in dem Scanner unterschiedliche Kameras verbaut sind, deren Chips das Signal zum Auslösen nicht zur exakt gleichen Zeit erfassen. Stattdessen liegt eine Verzögerung im Millisekundenbereich vor, deren Dauer zudem nicht konstant ist. Diese Verzögerung führt dazu, dass manche Bilder entweder komplett schwarz oder teilweise schwarz erscheinen.

Um die Fehlbelichtungen so weit wie möglich zu reduzieren, mussten optimale Einstellungen für die Kameras und das sogenannte Shutter-Lag gefunden werden. Hierzu wurde das Trial-and-Error-Prinzip angewendet. Mithilfe der Software Scanner-Control wurden das Auslösen der Kameras und die Shutter-Lag-Einstellungen koordiniert. Die eigentlichen

Kameraeinstellungen, wie beispielsweise Belichtungszeit oder ISO-Wert, wurden über die Software CaptureGRID vorgenommen.

Zur Ermittlung der besten Einstellungen wurde zunächst eine Reihe von Leerscans (ohne Personen) durchgeführt. Dabei wurde die Anzahl der fehlbelichteten Bilder systematisch erfasst und analysiert. Durch wiederholte Anpassungen und erneutes Testen war es möglich, sich schrittweise an die optimalen Werte heranzutasten, welche die Fehlbelichtungen minimierten. Dieser iterative Prozess führte schließlich zu einer Konfiguration, die in den meisten Fällen vielversprechende Ergebnisse lieferte.

Erstellung des Selbstauslösers

Das Scannen einer Person war ohne Unterstützung einer weiteren Person nicht möglich, da die Funktion eines Selbstauslösers fehlte. Um dieses Problem zu lösen, wurde ein Selbstauslöser programmiert. Hierfür kam das Tool AutoHotKey zum Einsatz.

Die grundlegende Idee bestand darin, die x- und y-Koordinaten des "Capture"-Buttons (Auslöser) in der Software Scanner Control zu ermitteln. Hierzu wurde ein Skript erstellt, das die Koordinaten des Mauszeigers konstant anzeigt. Um präzise Werte zu erhalten, wurde Scanner Control in den Vollbildmodus versetzt und die Koordinaten des Mittelpunktes des Buttons wurden durch Positionieren des Mauszeigers über der entsprechenden Stelle notiert.

Im nächsten Schritt wurde ein weiteres Skript entwickelt, das nach einer definierten Zeitspanne die gespeicherten Koordinaten automatisch anklickt. Der Timer wurde auf 18 Sekunden gesetzt, um ausreichend Zeit zu gewährleisten, sich in den Scanner zu begeben, diesen zu verschließen und sich korrekt zu positionieren.

Die entsprechenden Skripte sind im Anhang dieser Arbeit dokumentiert.

3.2.3 Automatische Bildfreistellung und Bearbeitung

Um zu untersuchen, ob die Qualität des finalen Gaussian-Splat-Modells gesteigert und die sogenannten Floater durch das Freistellen der Person auf den einzelnen Bildern minimiert werden können, wurden zwei Ansätze für eine Automatisierung dieses Arbeitsschritts getestet.

Der erste Ansatz basiert auf der Software Photoshop, bei dem ein Skript erstellt wurde, das mit einer zuvor aufgezeichneten Aktion innerhalb von Photoshop verknüpft ist. Diese Aktion führt den Freistellungsprozess automatisch durch.

Der zweite Ansatz nutzt die Programmiersprache Python. Die entsprechenden Codes wurden in der Software Visual Studio Code entwickelt, da sie durch ihre Benutzerfreundlichkeit und die Unterstützung für verschiedene Plugins eine effiziente Arbeitsumgebung bietet. In diesem Ansatz wurden verschiedene Methoden und Algorithmen getestet, um ein automatisiertes Programm zu entwickeln, das den Freistellungsprozess automatisch ausführt.

Die Skripte und Codes der erstellten Programme sind im Anhang dieser Arbeit dokumentiert.

Ansatz mit Photoshop

Photoshop ermöglicht die Automatisierung von Arbeitsabläufen durch Skripte. Die verwendende Programmiersprache ist JavaScript. Um ein JavaScript-Skript zu erstellen, kann der Code mit einem beliebigen Texteditor verfasst und mit der Endung .jsx gespeichert werden. Anschließend wird es in Photoshop ausgeführt.

Die Entwicklung des Skripts erfolgte schrittweise. Im ersten Schritt wurde eine Basisversion erstellt, die folgende Aufgaben automatisiert:

- Auswahl eines Ordners für den Import der zu bearbeitenden Bilder sowie eines Exportordners für die Ergebnisse,
- automatisches Öffnen der Bilder aus dem Importordner,
- Speichern der Bilder im JPG-Format im Exportordner.

Diese Grundfunktionen wurden zunächst implementiert, um eine solide Basis für mögliche Erweiterungen zu schaffen.

Code 6 - Pseudocode Photoshop Basis Skript

```
START
1. Wähle Eingabe- und Ausgabeordner
2. Wenn beide Ordner ausgewählt:
   a. Hole alle unterstützten Bilddateien im Eingabeordner
   b. Für jede Datei:
      I. Öffne die Datei in Photoshop
      II. Speichere die Datei als JPG im Ausgabeordner mit hoher Qualität
      III. Schließe die Datei ohne Änderungen zu speichern
   c. Zeige Nachricht: "Fertig! "
3. Andernfalls:
   a. Zeige Warnung: "Kein Ordner ausgewählt."
END
```

Der nächste Schritt bestand darin, in Photoshop eine Aktion zu erstellen, die die manuellen Arbeitsschritte für das Freistellen aufzeichnet, speichert und wiederverwendbar macht. Für den ersten Ansatz wurde dabei ausschließlich das „Objekt-Auswahlwerkzeug“ genutzt. Zunächst wurde ein neuer Aktionssatz mit dem Namen „BA“ erstellt und darin eine neue Aktion namens „FS2“ hinzugefügt.

Nach dem Start der Aktionsaufnahme wurden die folgenden Arbeitsschritte aufgezeichnet:

1. Auswahl der gesamten Bildfläche mit dem Objekt-Auswahlwerkzeug,
2. Erstellung einer Maske basierend auf der definierten Auswahl,
3. Reduzierung aller Ebenen auf ein Smart-Objekt.

Nach Abschluss dieser Schritte wurde die Aufnahme der Aktion beendet. Anschließend wurde diese interne Automatisierung in das JavaScript-Skript integriert.

Code 7 - Pseudocode Photoshop Aktionen durchführen

```
 a. Hole alle unterstützten Bilddateien im Eingabeordner
    b. Für jede Datei:
        I. Öffne die Datei
        II. Führe die Photoshop-Aktion "FS2" aus dem Satz "BA" aus
            - Wenn ein Fehler auftritt:
                - Zeige Fehlermeldung und beende das Skript
        III. Speichere die Datei als JPG im Ausgabeordner mit hoher
Qualität
        IV. Schließe die Datei ohne Änderungen zu speichern
    c. Zeige Nachricht: "Fertig! "
```

Nach der Auswertung der Ergebnisse wurde die Aktion um eine bewährte Methode ergänzt, um dem Objekt-Auswahlwerkzeug zu helfen, die Abgrenzung zwischen Hintergrund und Person besser zu definieren. Dazu wurde der Kontrast stark erhöht, während die durch die Überbelichtung entstehende Helligkeit reduziert wurde. Die in der Aktion verwendeten Werte waren eine Kontrasterhöhung auf 100 und eine Helligkeitsreduzierung auf -110.

Ansatz per Python

Der zweite Ansatz zur Automatisierung basiert auf der Entwicklung eines eigenständigen Programms. Für die Erstellung der verschiedenen Programme wurde die Software Visual Studio Code verwendet, wobei mit der Python-Version 3.10.6 gearbeitet wurde. Jedes Programm nutzt unterschiedliche Bibliotheken, die zusätzlich installiert werden mussten. Diese werden in den jeweiligen Abschnitten genauer beschrieben. Für jedes Programm kam eine eigene Methode zur Bildbearbeitung zum Einsatz.

Programm 1:

Die Methode für das erste Programm basiert auf der Differenzbildung von zwei Bildern, die jeweils von derselben Kamera aufgenommen wurden. Dafür wurden zwei Datensets angelegt, mit und ohne Person. Diese Bilder werden Pixel für Pixel miteinander verglichen, um Unterschiede festzustellen und anhand dieser Informationen eine Maske auf das Bild mit der Person zu projizieren. Durch diese Methode lässt sich die Person mithilfe numerischer Berechnungen identifizieren und freistellen.

Dafür ist es notwendig, drei Bibliotheken zu importieren:

- **NumPy:** Bibliothek für effiziente Berechnungen mit Arrays und Matrizen.
- **matplotlib.image:** Modul zum Laden, Anzeigen und Speichern von Bildern.
- **Pillow:** Bibliothek zur Bearbeitung, Konvertierung und Speicherung von Bildern.

Mit diesen Bibliotheken stehen die notwendigen Funktionen zur Verfügung, um numerische Berechnungen durchzuführen, Bilder zu laden und zu bearbeiten. Die Bilder werden zunächst

in das Programm eingelesen und anschließend in Graustufenbilder umgewandelt. Für die Umwandlung erhalten die Farbkanäle (Rot, Grün und Blau) Gewichtungsfaktoren von 0.299, 0.587 und 0.114. Diese Werte orientieren sich an der Helligkeit, wie sie am ehesten vom menschlichen Auge wahrgenommen werden.

Im nächsten Schritt werden die zu vergleichenden Bilder in Arrays umgewandelt. Um Unterschiede zwischen den Bildern zu ermitteln, werden die Graustufenbilder pixelweise verglichen und auf einen festgelegten Schwellenwert geprüft. Liegt die Differenz über dem zuvor definierten Schwellenwert, wird der Wert des Pixels im Ergebnisbild auf 255 (weiß) gesetzt; liegt sie darunter, erhält der Pixel den Wert 0 (schwarz).

Dieser Prozess ermöglicht die Erstellung einer Maske, die am Ende des Vorgangs auf das farbige Hauptbild (das Bild mit der Person) angewendet wird. Die Ergebnisse werden anschließend in einen definierten Ordner exportiert.

```
[[[123,  64,  38], [145,  76,  50], [200, 140, 120]],  # Zeile 1
[[130,  72,  60], [150,  80,  70], [220, 180, 160]],  # Zeile 2
[[110,  50,  40], [140,  90,  80], [210, 170, 150]]]  # Zeile 3
```

Abbildung 3 - Beispiel eines Arrays

Programm 2:

Das zweite Programm basiert auf der Methode der Farbraumanalyse. Ziel ist es, farbige Bereiche in einem Bild zu isolieren. Dazu wird das Bild in den HSV-Farbraum umgewandelt, um die Verarbeitung auf Basis von Farbton (Hue), Sättigung (Saturation) und Helligkeit (Value) durchzuführen.

Die hierfür verwendeten Bibliotheken sind:

- **OpenCV (cv2):** Bibliothek für Bildverarbeitung, Objekterkennung und Computer Vision.
- **NumPy:** Bibliothek für effiziente Berechnungen mit Arrays und Matrizen.

NumPy wird genutzt, um die Bilddaten als Arrays zu verarbeiten. OpenCV wird in diesem Fall für das Einlesen und Bearbeiten der Bilddaten sowie für morphologische Operationen verwendet.

Das Programm beginnt mit dem Einlesen des Bildes und dessen Umwandlung in den HSV-Farbraum. Anschließend wird die Sättigungsebene des Bildes verwendet, um farbintensive Bereiche zu isolieren. Dazu werden Werte für die minimale und maximale Sättigung definiert, die die Bereiche kennzeichnen, die als „farbig“ gelten. Zusätzlich wird die Helligkeitsebene herangezogen, um dunkle oder sehr helle Regionen zu eliminieren.

Im nächsten Schritt wird eine Maske erstellt, um die farbigen Bereiche des Bildes hervorzuheben. Um das Ergebnis zu verbessern und kleine störende Elemente zu entfernen, werden morphologische Operationen angewandt. Hierzu wird die sogenannte „Öffnung“ verwendet, bei der kleine isolierte Pixelgruppen im Bild entfernt werden. Eine zweite

morphologische Operation („Schließen“) wird angewandt, um kleine Löcher oder Lücken in der Maske zu schließen und die erkannten farbigen Bereiche zu optimieren.

Zum Abschluss wird die Maske auf das Originalbild angewendet, sodass nur die farbigen Regionen sichtbar bleiben. Das Ergebnis wird in drei separaten Bildern dargestellt.

1. Ein Bild, das die farbigen Bereiche nach der Maskierung zeigt.
2. Eine Darstellung der Maske nach den morphologischen Operationen.
3. Ein finales Bild, bei dem kleine Löcher geschlossen wurden, um die farbigen Bereiche zu optimieren.

Alle verarbeiteten Bilder werden abschließend in einem definierten Verzeichnis gespeichert.

Programm 3:

Das dritte Programm nutzt ein vortrainiertes Modell zur automatisierten Hintergrundentfernung. Hierfür kommt die Bibliothek rembg zum Einsatz, die speziell für diese Aufgabe optimiert wurde.

Für die Umsetzung sind folgende Bibliotheken erforderlich:

- **Rembg:** Vortrainiertes KI-Modell zur automatischen Entfernung von Bildhintergründen.
- **Pillow:** Bibliothek zur Bearbeitung, Konvertierung und Speicherung von Bildern.
- **Os:** Modul für den Zugriff auf Betriebssystemfunktionen wie Dateiverwaltung und Prozesse.

Rembg wird verwendet, um die Hintergrundentfernung durchzuführen, während Pillow und os für die Bearbeitung und Organisation der Bilddateien zuständig sind.

Das vortrainierte Modell nutzt fortschrittliche Deep Learning Technologien und wurde mit umfangreichen Datensätzen trainiert. Dabei wurden verschiedene Vordergrund- und Hintergrundszenarien berücksichtigt. Dadurch ist es dem Modell möglich, relevante Objekte zuverlässig zu identifizieren und den Hintergrund präzise zu isolieren. Die Basis dafür bildet ein neuronales Netzwerk, das speziell für Segmentierungsaufgaben konzipiert wurde.

Das Programm liest die Bilder aus einem definierten Ordner ein und entfernt deren Hintergründe. Die freigestellten Objekte werden anschließend in einem definierten Ausgabeordner gespeichert. Dabei wird darauf geachtet, dass die Ergebnisse mit neuen, nachvollziehbaren Dateinamen abgelegt werden. Unterstützt werden die gängigen Bildformate (.jpg, .jpeg, .png).

Zusätzlich stellt das Programm sicher, dass der Ausgabeordner bei Bedarf automatisch erstellt wird. Während der Verarbeitung werden die Bilder schrittweise bearbeitet und gespeichert.

Am Ende des Prozesses stehen die verarbeiteten Bilder im Ausgabeordner zur Verfügung.

3.2.4 Sparse Point Cloud Bearbeitung zur Floater Minimierung

Zur Entfernung von Floatern mittels Punktwolkenoptimierung wurde ein in Visual Studio Code entwickeltes Skript erstellt. Es filtert überflüssige Punkte heraus, die außerhalb eines vordefinierten Bereichs liegen. Für die Erstellung des Skripts kamen folgende Python-Bibliotheken zum Einsatz:

- **NumPy:** Bibliothek für effiziente Berechnungen mit Arrays und Matrizen.
- **Plyfile:** Bibliothek zum Laden, Speichern und Bearbeiten von PLY-3D-Punktwolken.

Das Skript lädt die zu verarbeitende Punktwolke und extrahiert Eigenschaften wie räumliche Positionen und Farbinformationen. Anschließend werden alle Punkte überprüft, ob sie innerhalb eines vorgegebenen Bereichs liegen, der durch minimale und maximale Grenzen definiert ist. Diese Grenzen wurden mithilfe der 3D-Software Blender ermittelt.

Nur die Punkte, die diese Kriterien erfüllen, bleiben erhalten. Im Anschluss wird eine neue Punktwolke erstellt und exportiert, die die gefilterten Punkte und deren Eigenschaften enthält.

Code 8 - Pseudocode für Bildfreistellung mit rembg und U²-Net

```
START
1. Lade die Punktwolke aus der Eingabedatei:
   a. Öffne die Datei mit den Punktdaten.
   b. Extrahiere die 3D-Koordinaten (x, y, z) und die Farbinformationen
(Rot, Grün, Blau).
2. Definiere die Filtergrenzen:
   a. Setze minimale und maximale Werte für x, y und z.
3. Filtere die Punkte:
   a. Für jeden Punkt in der Punktwolke:
      i. Überprüfe, ob die Koordinaten (x, y, z) innerhalb der Grenzen
liegen.
      ii. Wenn ja, füge den Punkt der gefilterten Liste hinzu.
4. Prüfe das Ergebnis der Filterung:
   a. Wenn die gefilterte Liste leer ist:
      i. Zeige die Warnung: "Keine Punkte innerhalb des angegebenen
Bereiches gefunden."
   b. Anderenfalls:
      i. Zeige die Anzahl der gefilterten Punkte.
5. Erstelle eine neue Punktwolke:
   a. Speichere die gefilterten Punkte mit ihren Farbinformationen.
   b. Falls vorhanden, füge zusätzliche Attribute (z. B. Texturen)
hinzu.
6. Speichere die gefilterte Punktwolke:
   a. Schreibe die gefilterten Punkte in eine neue Datei.
   b. Zeige die Meldung: "Gefilterte Punktwolke wurde gespeichert."
END
```

3.3 Evaluierungskriterien

Im folgendem Kapitel Ergebnisse werden die erstellten Programme sowie die erzielten Ergebnisse analysiert und auf folgende Kriterien untersucht:

- Sind die erstellten Programme funktionstüchtig?
- Konnte durch die Automatisierung eine Zeitersparnis erzielt werden?
- Welche SSIM-Werte haben die fertigen Gaußschen Modelle?
- Wie wirkt der SSIM-Wert sich auf die tatsächliche optische Qualität aus?
- Konnten Floater mit den untersuchten Ansätzen minimiert werden?
- Hat die Fotobearbeitung zur Floater Minimierung beigetragen?
- Wie wirkt sich das Freistellen der Person auf die Trainingszeit und Qualität der finalen Gaußschen Modelle aus?

IV. Ergebnisse

In diesem Kapitel werden die entwickelten Programme, ihre Funktionalität sowie deren Einfluss auf die Qualität der Gaußschen Modelle und die Effizienz des Workflows analysiert. Es wird untersucht, inwiefern die Automatisierung der Arbeitsprozesse, die Bildbearbeitung und die Optimierung der Sparse Point Cloud zur Minimierung von Floatern sowie zur Verbesserung der finalen Modelle beigetragen haben.

4.1 Analyse der entwickelten Programme und deren Ergebnisse

4.1.1 Automatisierung von RealityCapture und PostShot

Das Programm GS_Auto_Ersteller kann den gesamten Arbeitsprozess von RealityCapture und PostShot automatisieren. Es lädt die Bilder, erstellt daraus eine Sparse Point Cloud sowie die dazugehörigen Kameraausrichtungsdateien und speichert diese mit den verfügbaren Exporteinstellungen aus den XML-Dateien zuverlässig im „image“ Ordner. Anschließend übergibt das Programm die Bilder, die PLY- und CSV-Datei an die Postshot-cli.exe und trainiert das Gaußsche Modell mit den hinterlegten Parametern. Das PostShot-Projekt sowie das Gaußsche Modell werden korrekt im Hauptordner gespeichert.

Teilautomatisierung von RealityCapture

Eine der Herausforderungen lag darin, eine Methode zu finden, um die Prozesse mittels Python zu automatisieren. Ansätze hierzu erwiesen sich jedoch als unbrauchbar, da eine mögliche Ursache fehlende APIs gewesen sein könnten und somit grundlegende Ansteuerungen einzelner Prozesse nicht möglich waren. Der Versuch, die Arbeitsschritte durch simulierte Mausklicks und angepasste Zeitverzögerungen zu automatisieren, erwies sich als ungeeignet, da zusätzliche manuelle Eingriffe nötig waren.

Zudem muss gewährleistet sein, dass die Verarbeitung der Sparse Point Cloud zu keiner Verzögerung führt, da die simulierten Mausklicks gestartet werden, sobald der dafür gesetzte Timer abläuft, auch wenn die Berechnung der Punktwolke noch nicht abgeschlossen ist. Dies führt zum Abbruch der Automatisierung, weil der Ablauf unterbrochen wurde und somit weitere Befehle nicht ausgeführt werden können.

Theoretisch wäre dieser Ansatz zur Automatisierung zwar möglich, er erfordert jedoch eine umfangreiche Erweiterung des Programms und eine präzise Feinabstimmung zwischen Verzögerungen und simulierten Mausklicks, um jeden Arbeitsschritt zuverlässig ausführen zu lassen.

Für die Übertragung auf andere Rechner ist diese Herangehensweise jedoch unpraktikabel. Alle Koordinaten, welche für die simulierten Mausklicks notwendig sind, müssten erneut ermittelt werden. Zudem muss gewährleistet sein, dass die Dauer der Berechnungen auf den Computer abgestimmt wird. Aufgrund der Fehleranfälligkeit und der nötigen Überwachung des Programms ist die Methode zur Teilautomatisierung von RealityCapture ungeeignet.

Automatisierung von PostShot

Durch die zur Verfügung stehenden Command-Line-Befehle, die über die CMD-Konsole abgefragt werden konnten, gelang es, eine Batchdatei zu erstellen. Diese ist in der Lage, voll autonom den Bilderordner, die Sparse Point Cloud und die Kameraausrichtungsdatei in die Postshot-cli.exe zu laden, mit dem Profil „MCMC“ zu trainieren und anschließend abzuspeichern. Es gelang jedoch nicht, zusätzliche Parameter wie die maximale Anzahl an Trainingsschritten oder die maximale Anzahl der zu erzeugenden Splats einzubauen. Das Gaußsche Modell wird zwar mit den in PostShot hinterlegten Standardeinstellungen trainiert, jedoch fehlt dem Benutzer mit dieser Methode die Möglichkeit, je nach Anwendungsbereich notwendige Parameter einzustellen.

Das Ziel, ein Programm zu schreiben, das den PostShot-Arbeitsprozess automatisiert, wurde mit dieser Batchdatei erfüllt. Diese Methode verbessert die Funktion, die Benutzerfreundlichkeit und den Anwendungsbereich des Programms.

Finales Programm

Der GitHub-Beitrag von Azad Balabanian [31] stellt nicht nur eine Methode zur Automatisierung von PostShot vor, sondern schafft auch einen Ansatz, um die Arbeitsprozesse von RealityCapture und PostShot zu automatisieren und zu verknüpfen. Die von ihm zur Verfügung gestellte Batchdatei ist anfangs nicht in der Lage, den Trainingsprozess des Gaußschen Modells zu starten, wohingegen die Automatisierung von RealityCapture funktioniert. Durch Erweiterungen der Batchdatei werden die Fehlermeldungen behoben. Dadurch kann der PostShot-Trainingsprozess gestartet und die Exportfunktion um eine zusätzliche PLY-Datei des Gaußschen Modells erweitert werden. Eine weitere Funktion ermöglicht es, die Konsole zu pausieren, sobald die Erstellung des Gaußschen Modells abgeschlossen ist. Dadurch können angezeigte Daten wie Trainingsdauer, SSIM-Wert oder die Anzahl der erstellten Splats gesammelt werden, die für weitere Analysen notwendig sind.

4.1.2 Anpassung der Kameraeinstellungen & Erstellung des Selbstauslösers

Bei den Anpassungen der Kameraeinstellung lag das Ziel darin, die Parameter so zu justieren, dass die Anzahl der Fehlbelichtungen auf ein Minimum reduziert werden. Die Herausforderung bestand darin, dass es keinerlei Dokumentation gab und die optimalen Werte mittels Trial-and-Error-Verfahren ermitteln werden mussten. Zudem erschwerten die inkonsistenten Ergebnisse der fehlbelichteten Bilder trotz identischer Einstellungen dieses Verfahren. Durch die Ermittlung bestimmter Parameter der Kameraeinstellungen sowie des Shutter-Lags gelang es nach mehreren Versuchen, die fehlbelichteten Bilder teilweise oder sogar vollständig zu eliminieren.

Tabelle 3 - Kameraeinstellungen für das Projekt

Kameraeinstellungen	Blendenzahl	Belichtungszeit	ISO	Brennweite	Farbdarstellung
Wert	1/14	1/60	800	25mm	sRGB

Shutter-Lag	Canon D2000	Canon EOS R7
Wert	135 ms	103 ms

Diese Ergebnisse sind nicht konstant, bieten allerdings die höchste Wahrscheinlichkeit, nur wenige oder gar keine fehlbelichteten Bilder zu erhalten.

Die Entwicklung des Selbstauslösers über das Programm AutoHotKey hat erfolgreich das Problem gelöst, Aufnahmen mit dem Scanner von sich selbst zu erstellen, ohne auf eine zweite Person angewiesen zu sein. Sie bietet zwar kein direktes Upgrade für die Software Scanner Control, erweitert aber dennoch die Anwendungsmöglichkeiten. Zudem ist das Programm einfach aufgebaut, von jedem anwendbar und lässt sich beliebig anpassen.

4.1.3 Optimierung der Fotobearbeitung

In diesem Abschnitt werden die erstellten Programme zur Bildbearbeitung anhand verschiedener Kriterien überprüft. Es wird analysiert, ob der Bearbeitungsprozess voll automatisiert werden konnte, welche Qualität die freigestellten Bilder haben und ob die Methoden zur Programmerstellung sinnvoll waren.

Ansatz mit Photoshop:

Die Methode, ein Skript zu entwickeln, welches den Freistellungsprozess über die Software Photoshop steuert, erwies sich anfangs als eine gute Lösung. Es ermöglicht die Automatisierung des gesamten Prozesses, indem ein Skript mit einer aufgezeichneten Aktion zur Freistellung verknüpft wird. Voraussetzung dafür ist, dass der Anwender eine gültige Adobe Photoshop Lizenz besitzt. Zudem muss die Software geöffnet sein, damit das Skript gestartet werden kann. Die erzielten Ergebnisse dieser Methode haben unterschiedliche Qualitäten. In einigen Kameraperspektiven sind die erstellten Masken sehr präzise, während in anderen Perspektiven teilweise die gesamte Person ausgeschnitten wird. Die Erweiterung der aufgenommenen Aktion durch gängige Methoden zur Anpassung von Kontrast und Helligkeit führt zu einer leichten Verbesserung der Ergebnisse, löst allerdings nicht das Problem.

Python-Programm 1: Erstellung einer Maske durch Differenzbildung von Bildern

Diese Methode wurde mit zwei unterschiedlichen Datensätzen getestet, um die Hypothese zu untersuchen, ob das Freistellen von Bildern mittels numerischer Vergleiche zweier Datensätze funktioniert. Die Wahl dieser Datensätze erfolgte bewusst, um die Methode zunächst unter idealen Bedingungen zu testen und anschließend unter realen Bedingungen zu validieren. Im ersten Schritt wurde eine kontrollierte Umgebung geschaffen, in der der Hintergrund zu 100 Prozent identisch gehalten wurde. Dies diente dazu, zunächst zu prüfen, ob der Ansatz grundsätzlich funktioniert, bevor er auf komplexere, realitätsnahe Szenarien angewendet wurde.

Die zu vergleichenden Bilder stammen jeweils von derselben Kamera. Es werden Pixel für Pixel verglichen, um Unterschiede auszumachen und entsprechend zu maskieren.

Bei dem ersten Test wurde eine optimale Umgebung künstlich erzeugt, indem die Person manuell ausgeschnitten und auf das zu vergleichende Referenzbild eingefügt wurde. Dieses Bild wurde im Anschluss abgespeichert und als Hauptbild deklariert. Dadurch wurde sichergestellt, dass der Hintergrund des Hauptbildes (mit Person) und des Referenzbildes (ohne Person) zu 100 % identisch ist.

Das Ergebnis dieses Testlaufes war eine saubere Maske der Person. Die Herausforderung bestand bei dieser Methode darin, dasselbe Maskenergebnis unter realen Bedingungen zu erzielen. Bei Verwendung des Originalbildes mit Person anstelle eines künstlich erstellten Hauptbildes ergibt sich ein unterschiedliches Ergebnis.

Zwei Faktoren spielen hierbei eine entscheidende Rolle. Zum einen sind die allgemeine Belichtung der Bilder sowie eventuell anfallende Fehlbelichtungen durch die Kameras, wie in Kapitel 4.1.2 beschrieben, immer minimal unterschiedlich. Zum anderen spielt das Bounce Light sowie das Blockieren des Blitzlichtes durch die zu scannende Person eine Rolle. Hierdurch verändert sich der Hintergrund des Referenzbildes so stark, dass es sehr schwierig bis unmöglich ist, passende Schwellenwerte zu definieren, um eine saubere Maske zu erstellen. Die Ergebnisse sind unsauber ausgeschnittene Bilder, die teilweise zu viel oder zu wenig von der Person entfernen.

Python-Programm 2: Erstellung einer Maske anhand der Farbraumanalyse

Diese Methode zielt darauf ab, eine Maske anhand einer Farbraumanalyse zu erstellen. Der Grundgedanke dahinter ist die signifikante Farbtrennung der zu scannenden Person und des Hintergrunds.

Während der Hintergrund (Scanner) ausschließlich aus Schwarz- und Weißtönen besteht, versucht das erstellte Programm, die HSV-Werte des Bildes zu ermitteln und farbige Bereiche, die ausschließlich von der Person hervorgebracht werden, zu isolieren.

Die Herausforderung bei dieser Methode lag darin, die minimalen und maximalen Sättigungsbereiche zu definieren, um die Auswahl der Maske auf die Person zu beschränken. Durch das Bounce Light, welches von der Person auf den Hintergrund fällt, färbt sich dieser minimal ein und beinhaltet somit nicht mehr nur noch Schwarz- und Weißtöne. Das Programm hat dadurch Schwierigkeiten, eine saubere Maske zu erstellen.

Auch die beigefügte morphologische Operation konnte die Resultate nicht verbessern. Aufgrund der mangelnden Qualität der Ergebnisse wurde dieser Ansatz nicht weiterverfolgt, und Funktionen zur Automatisierung des gesamten Arbeitsprozesses wurden nicht weiter hinzugefügt.

Python-Programm 3: Erstellung einer Maske mithilfe eines vortrainierten Modells

Dieses Programm verfolgt den Ansatz, ein mithilfe von Deep Learning trainiertes Modell zur Hintergrundentfernung zu verwenden. Es ist in der Lage, den gesamten Datensatz der Bilder am Stück zu bearbeiten und in einem separaten Ordner abzuspeichern. Das Python-Skript ermöglicht eine schnelle Bearbeitung des gesamten Datensatzes, erzielt aber unterschiedliche Qualitäten bei der Erstellung von Masken.

In der folgenden Abbildung werden repräsentative Ergebnisse des Photoshop-Skripts sowie des Python-Skripts (Programm 3) gezeigt. Aufgrund der mangelnden Qualität der Ergebnisse wurden Programm 1 und 2 für diese Visualisierung und spätere Analysen nicht berücksichtigt.

Abbildung 5 - Vergleich der Ergebnisse des Photoshop- und Python-Skripts (Prog.3)

4.1.4 Point Cloud Bearbeitung

Das erstellte Programm zur Bearbeitung der finalen Punktwolke kann überflüssige Punkte, die außerhalb des definierten Bereichs liegen, entfernen. Es speichert zudem erfolgreich eine neue, bereinigte Punktwolkendatei. Während des Exports wird jedoch ASCII wieder auf „true“ gesetzt, was die weitere Bearbeitung oder Visualisierung im Programm SuperSplat behindert. Es gelang nicht, den benötigten Befehl in das Skript zu integrieren, um das Problem zu lösen.

4.2 Bewertung der Bildqualität der Gaußschen Modelle

In diesem Abschnitt wird die Bildqualität der erstellten Gaußschen Modelle analysiert. Grundlage für die Ermittlung sind vier Datensätze, die mittels der verschiedenen Methoden produziert wurden. Die zu vergleichenden Datensätze bestehen aus:

1. Ein Bilderset ohne jegliche Bearbeitung
2. Ein Bilderset, in dem die Person manuell freigestellt wurde
3. Ein Bilderset, das mit dem Photoshop-Skript bearbeitet wurde
4. Ein Bilderset, das mit dem Python-Skript mittels eines vortrainierten Modells bearbeitet wurde

Aufgrund der mangelnden Qualität, die die Python-Programme 1 und 2 erzielten, wurden diese nicht in die Analyse einbezogen.

In dieser Analyse werden SSIM-Werte in Abhängigkeit von maximalen Trainingsschritten, maximal erzeugten Splats sowie die dafür benötigte Zeit ins Verhältnis gesetzt. Die folgende Grafik zeigt, wie die verschiedenen Datensätze in Bezug auf die genannten Kriterien abschneiden.

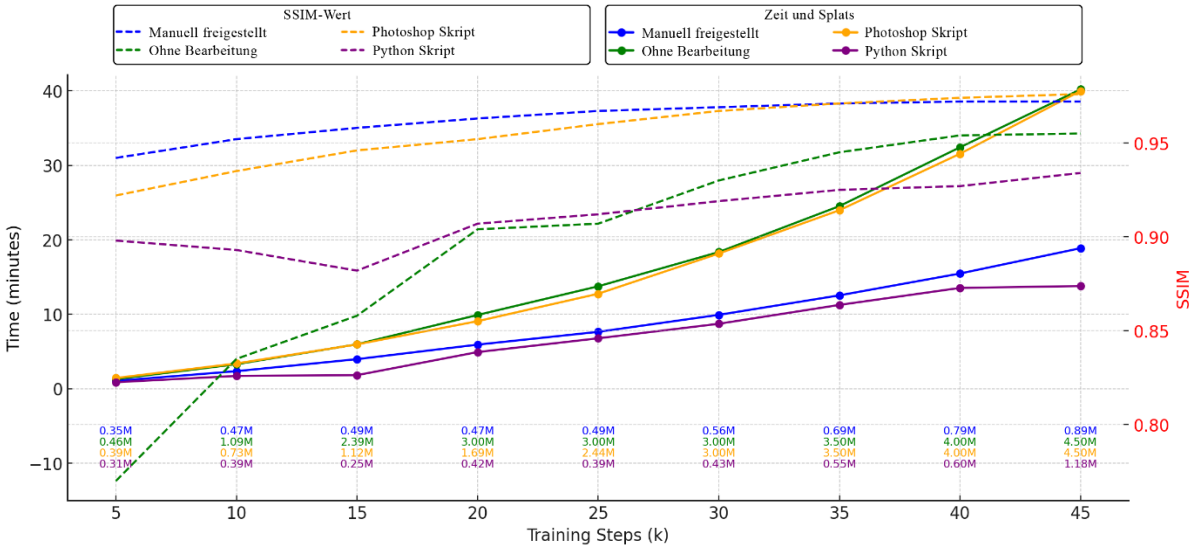


Abbildung 6 - Vergleich aller Datensätze anhand deren Trainingsschritten (5k-45k)

Die Auswertung der Grafik zeigt, dass der SSIM-Wert bei jedem Datensatz mit zunehmenden Trainingsschritten besser wird. Die Datensätze von „Manuell freigestellt“ und „Photoshop-Skript“ schneiden hierbei am besten ab, obwohl die Anzahl der erzeugten Splats bei den manuell freigestellten Bildern deutlich niedriger ist. Die Methode mit dem Python-Skript schneidet in Bezug auf den SSIM-Wert am schlechtesten ab. Bei der Qualitätsanalyse muss der SSIM-Wert allerdings ins Verhältnis zum tatsächlichen Output des Gaußschen Modells gesetzt werden. Wie in den folgenden Abbildungen zu erkennen, sagt der SSIM-Wert nichts über die tatsächliche Qualität des Modells aus. Er ermittelt lediglich, wie sehr das Modell den zum Training übergebenen Bildern ähnelt. Da die Datensätze des Photoshop- und Python-Skripts allerdings schlecht ausgeschnittene Bilder enthalten, werden aufgrund der fehlenden Bildinformationen sowohl der SSIM-Wert als auch die zum Training benötigte Zeit verfälscht.

Abbildung 7 - Teilweise bereinigtes Gaußschen-Modell / Python Datensatz 45k Steps

Wie auf der Abbildung zu sehen, ist die Qualität des Gaußschen Modells sehr schlecht. Die fehlenden Bildinformationen führen dazu, dass zum einen die Sparse Point Cloud mangelhaft erstellt wird, da das Verfahren der SfM-Technologie entscheidende Features nicht mehr ordnungsgemäß tracken kann, und zum anderen die Qualität des Modells abnimmt. Manche Bereiche, die zur Person gehören, werden stattdessen abseits an Stellen des eigentlichen Modells dargestellt. Aus diesem Grund kann der SSIM-Wert nur in die Analyse einbezogen werden, wenn die Datensätze eine entsprechende Qualität aufweisen.

Die Methode, die Bilder mit dem Photoshop-Skript freizustellen, erweist sich hingegen als zuverlässiger. Wie in Abbildung 7 zu sehen ist, weist das mit dem Photoshop-Skript erzeugte Gaußsche Modell eine bessere Qualität auf als das mit dem Python-Skript erstellte Modell, obwohl in beiden Datensätzen mangelhaft freigestellte Bilder enthalten sind.

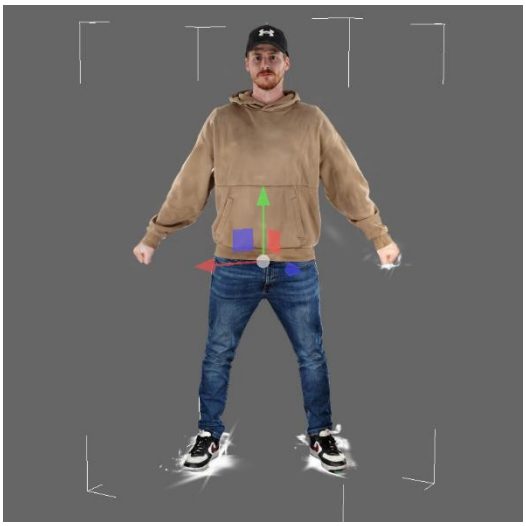


Abbildung 8 - Bereinigtes Gaußschen Modell / Photoshop Datensatz 45k Steps

Auch in diesem Datensatz treten bei der Erstellung der Sparse Point Cloud Fehler auf. Die grundlegende Qualität ist im Vergleich zur Python-Methode zwar besser, aber auch hier werden

an manchen Stellen Gaussians an der falschen Position erzeugt, wie die folgende Abbildung zeigt.

Abbildung 9 - Teilweise bereinigtes Gaußschen Modell / Photoshop Datensatz 45k Steps

Trotz Fehlern im Gaußschen Modell, erzielte der Photoshop-Datensatz den besten SSIM-Wert. In weiteren Kapiteln wird deshalb nur noch der „manuell freigestellte“ mit dem „unbearbeiteten“ Datensatz verglichen, um repräsentative Ergebnisse zu erhalten.

Wie in der ersten Grafik (Abbildung 5) zu sehen ist, liegt die Anzahl der erzeugten Splats beim unbearbeiteten Datensatz unabhängig von den Trainingsschritten weit über der des manuell freigestellten. Durch das Entfernen des Hintergrunds werden die zu trackenden Features von der SfM-Technologie auf die freigestellte Person begrenzt. Infolgedessen wird die Sparse Point Cloud, welche durch RealityCapture berechnet wird, ausschließlich von der gescannten Person erstellt. Die Punktwolke, die mit den unbearbeiteten Bildern konstruiert wird, enthält dagegen zusätzlich alle Punkte, die dem Scanner zugeordnet werden können. Da Gaussians anhand dieser Punkte trainiert werden, steigt deren Anzahl abhängig von der Anzahl der vorhandenen Punkte in der Point Cloud.

Ein zusätzlicher Test hat jedoch ergeben, dass der Hintergrund selbst bei einer auf die Person reduzierten Point Cloud mit Verwendung der unbearbeiteten Bilder den Hintergrund während des Trainings des Gaußschen Modells trotzdem erstellt wird.

Diese Problematik lässt sich durch das Freistellen der Bilder beheben.

Abbildung 10 - Vergleich Gaußschen Modelle. Links unbearbeiteter Datensatz, rechts manuell freigestellter Datensatz

Beim Vergleich der bereinigten Gaußschen Modelle, fällt auf, dass die Qualität des unbearbeiteten Datensatzes besser zu sein scheint. Je nach Perspektive ändert sich jedoch das optische Erscheinungsbild, wie in der nächsten Abbildung zu sehen ist.

Abbildung 11 - Vergleich Gaußschen Modelle. Links unbearbeiteter Datensatz, rechts manuell freigestellter Datensatz (45° Ansicht)

Aus dieser Perspektive scheinen die beiden Modelle identisch zu wirken. Bei näherer Betrachtung können jedoch weitere qualitative Unterschiede festgestellt werden. Während der Bereinigung der Modelle ist aufgefallen, dass beim rechten Gaußschen Modell im Vergleich zum linken, fast ausschließlich Floater entfernt werden mussten, die einen 100-prozentigen Alphawert haben. Diese Gaussians sind unsichtbar.

Bei der Bereinigung des linken Gaußschen Modells musste dagegen der gesamte Hintergrund (Scanner) entfernt werden, was insbesondere in Bereichen des Bodens, wo Modell und Hintergrund in Kontakt kommen, zu Problemen geführt hat. Durch die farbliche Ähnlichkeit des zu entfernenden Bodens und der Sohle der Schuhe wurden somit Gaussians, die dem Boden

zugeordnet werden konnten, teilweise jedoch auch zur Erzeugung der Schuhsohle verwendet, was bei deren Entfernung zu Qualitätseinbußen führt. Das Gaußsche Modell, das durch das manuell freigestellte Datenset rekonstruiert wurde, übertrifft in der Kategorie Detail und Klarheit das Vergleichsmodell.

Abbildung 12 - Gaußsche Modell ohne Bildbearbeitung. Detailverlust der Schuhsohle

Abbildung 13 - Gaußsche Modell manuell freigestellte Bilder. Vollständige Schuhsohle

Ein weiterer Punkt, auf den in Bezug auf die qualitativen Unterschiede eingegangen werden muss, ist die erzeugte Splat-Anzahl. Durch die Erzeugung des Hintergrunds ist die Anzahl beim Datensatz der unbearbeiteten Bilder signifikant höher. Die Annahme, dass dadurch auch mehr Splats für die Erstellung der Person verwendet werden, ist falsch. Durch die Bereinigung der nicht benötigten Gaussians nähert sich die Anzahl der übrig gebliebenen Splats stark der des Gaußschen Modells mit den freigestellten Bildern an. Die folgende Abbildung zeigt die verbliebenen sowie die gelöschten Splats der in Abbildung 9 und 10 bereinigten Modelle.

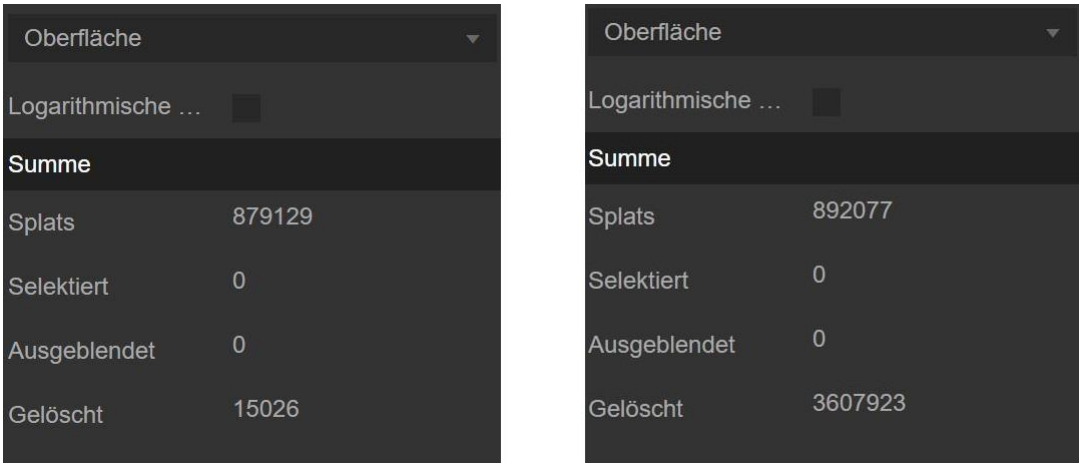


Abbildung 14 - Aktuelle sowie gelöschte Anzahl der Splats. Links manuell freigestelltes Datenset, rechts unbearbeitetes Datenset

4.3 Analyse der Zeitersparnis bei der Modellgenerierung & Bildbearbeitung

Im folgenden Abschnitt wird auf die Zeitersparnis durch die Automatisierung der Arbeitsschritte bei der Modellgenerierung und Bildbearbeitung eingegangen. Dabei werden folgende Bearbeitungswege gegenübergestellt:

1. Manuelle Freistellung der Bilder vs. automatisierte Bildbearbeitung
2. Manueller Arbeitsablauf der Gaußschen Modell Generierung vs. GS_Auto_Ersteller
3. Benötigte Zeit zur Erstellung eines Gaußschen Modells anhand verschiedener Datensätze
4. Manuelle Floater Entfernung vs. automatisierte Floater Entfernung

Bildbearbeitung: manuell vs. automatisiert

Die benötigte Zeit für die manuelle Freistellung eines kompletten Datensatzes von 124 Bildern hängt von der verwendeten Software und den Fähigkeiten des Bearbeiters ab. Das manuell freigestellte Datenset wurde mit der Software Adobe Photoshop von einem erfahrenen Bearbeiter erstellt und hat etwa ca. 4,5 Stunden in Anspruch genommen. Die Qualität der erstellten Bilder ist dafür fehlerfrei.

Ein Einflussfaktor, der beim automatisierten Verfahren zu berücksichtigen ist, ist die verwendete Hardware. Die erstellten Datensätze, die in diese Analyse einfließen, wurden mit dem Hochschulrechner bearbeitet. Die verbaute Hardware ist in Kapitel 2.5 aufgelistet. Es können nur ungefähre Zeitangaben angegeben werden, da lediglich die Bearbeitungszeit eines einzelnen Bildes gemessen und diese mit dem Faktor 124 multipliziert wurde.

Das erstellte Photoshop-Skript benötigt für ein Bild etwa 23 Sekunden, was für ein ganzes Datenset ca. 47,5 Minuten bedeutet. Während dieser Zeit kann sich der Bearbeiter um andere Aufgaben kümmern, wodurch der Arbeitsprozess effizienter gestaltet wird. Allerdings variiert die Qualität der Ergebnisse stark.

Das Python-Skript stellt ein einzelnes Bild in ca. 5 Sekunden frei. Der gesamte Datensatz benötigt somit ca. 10 Minuten. Die Qualität der Ergebnisse variiert auch hier sehr stark.

Es ist zu berücksichtigen, dass das Freistellen der Bilder ein zusätzlicher Arbeitsschritt ist. Dieser zeitliche Mehraufwand muss auf die benötigte Zeit zur Erstellung des Gaußschen Modells addiert werden. Mit den aktuell erzielten Werten verlängert sich der gesamte Arbeitsprozess im Vergleich zur manuellen Standardmethode.

Arbeitsablauf von RealityCapture und PostShot: manuell vs. automatisiert (GS_Auto_Ersteller)

Bei diesem Vergleich wurde auf eine Zeiterfassung verzichtet, da die reine Zeitersparnis durch die Automatisierung der Arbeitsschritte gering ist. Es entfallen lediglich einige manuelle Eingriffe, wie das Importieren von Datensätzen, das Auswählen der Parameter oder das Exportieren der erzeugten Dateien.

Der entscheidende Faktor, der die Zeitersparnis hierbei ausmacht, liegt zwischen den Arbeitsschritten von RealityCapture und PostShot sowie in der Aufmerksamkeit des Bearbeiters während der einzelnen Arbeitsprozesse.

Die Berechnung der Sparse Point Cloud kann je nach Datensatz einige Zeit in Anspruch nehmen. Diese muss im Anschluss zusammen mit der Datei der Kameraausrichtungen gespeichert, in PostShot zusammen mit den Bildern importiert und der Trainingsprozess gestartet werden.

In der Regel wird die Wartezeit während der Berechnung der Sparse Point Cloud genutzt, um parallel andere Aufgaben auszuführen. Wird der Zeitpunkt, an dem RealityCapture die benötigten Dateien erstellt, verpasst, bleibt das System untätig, und wertvolle Zeit für die Erstellung des Gaußschen Modells verstreicht ungenutzt. Für diesen Bereich ist das Programm GS_Auto_Ersteller nützlich, da alle Arbeitsschritte automatisch nacheinander abgearbeitet werden und lediglich das Programm gestartet werden muss, um am Ende ein fertiges Gaußsches Modell zu erhalten.

Zeit für die Erstellung eines Gaußschen Modells anhand verschiedener Datensätze

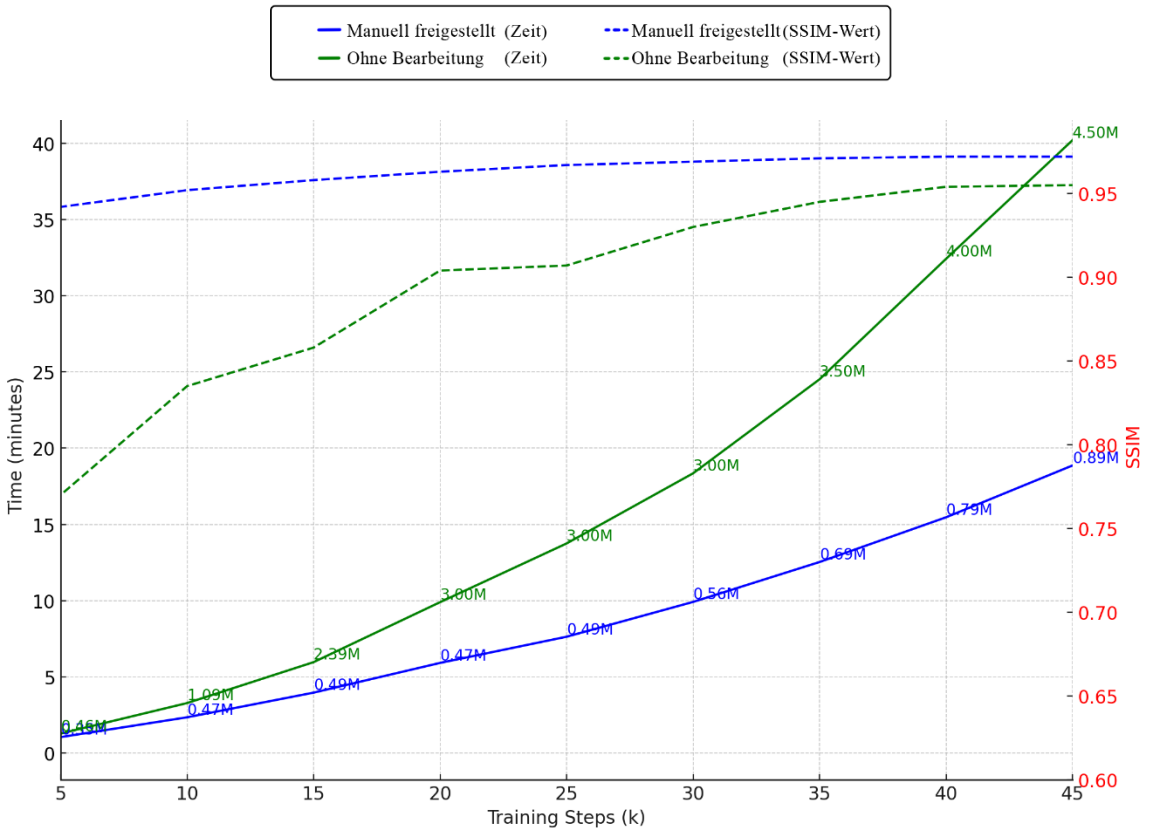


Abbildung 15 - Vergleich der benötigten Trainingszeit und SSIM-Wert (5k-45k Trainingsschritte)

Durch das Freistellen der Bilder konnte die Trainingszeit des Gaußschen Modells fast halbiert werden, während sich der SSIM-Wert bereits bei einer niedrigen Anzahl an Trainingsschritten konstant über dem des unbearbeiteten Datensatzes befindet.

Die Ergebnisse zeigen, dass je mehr sichtbare Gaussians erstellt werden, die für das Training benötigte Zeit exponentiell ansteigt. Das gilt allerdings nur für das Modell mit den unbearbeiteten Bildern.

Wie aus der Grafik zu entnehmen ist, steigt die benötigte Trainingszeit für das manuell freigestellte Datenset nahezu linear an. Diese Ergebnisse deuten darauf hin, dass die Trainingszeit durch das Freistellen der Bilder reduziert werden kann, ohne dass bei höheren Trainingsschritten ein sprunghafter Anstieg der benötigten Zeit entsteht.

4.4 Ergebnisse der Floater Minimierung

In diesem Kapitel wird die Funktionsweise des erstellten Programms zur Floater-Entfernung beurteilt. Darüber hinaus werden die gesammelten Ergebnisse erläutert, inwiefern die Fotobearbeitung zur Floater Minimierung beiträgt. Des Weiteren wird ein Lösungsweg vorgestellt, der die Entstehung von Floatern bei der Rekonstruktion von Gaußschen Modellen von Personen oder Objekten verhindern kann.

Das erstellte Programm auf Python-Basis entfernt erfolgreich einen Großteil der vorhandenen Floater. Durch den vordefinierten Bereich in Form eines Quaders erkennt das Programm die Punkte, die außerhalb dieses Bereichs liegen, und entfernt diese. In der folgenden Abbildung wird die originale Punktwolke der bereinigten gegenübergestellt.

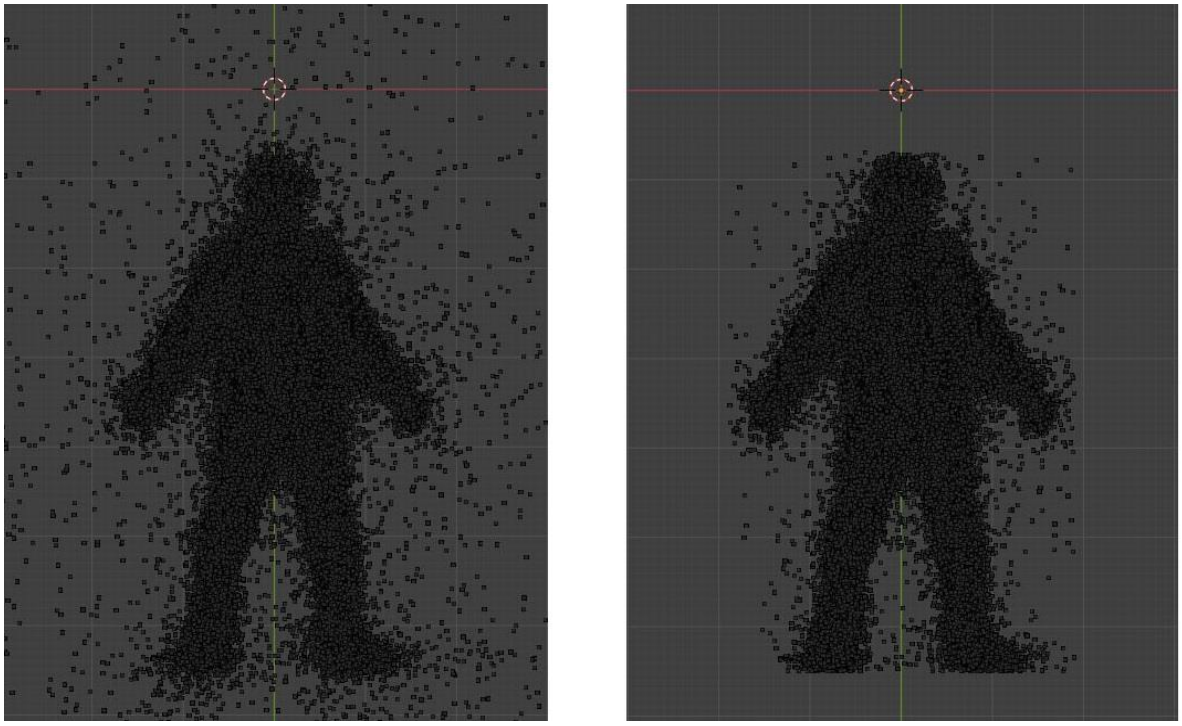


Abbildung 16 - Ursprüngliche und bereinigte Punktwolke

Der vordefinierte Bereich wurde anhand der eingescannten Person definiert. Diese ist 1.89 m groß und befindet sich in einer A-Pose. Sollten Körpermaße sowie Posen bei weiteren Scans unterschiedlich sein, ist es möglich, den Bereich des Quaders innerhalb des Python-Codes anzupassen. Das Programm ist jedoch nicht in der Lage, die Exporteinstellung ASCII auf „false“ zu setzen, was eine weitere Bearbeitung verhindert.

Die Bildbearbeitung trägt auf zwei unterschiedliche Arten zur Floater Minimierung bei. Durch das Entfernen des Hintergrunds auf allen 124 Bildern wird der Fokus bei der Erstellung der Sparse Point Cloud durch RealityCapture auf die Person gerichtet. Das ist dahingehend entscheidend, weil Gaussians anhand der zur Verfügung stehenden Punkte aus der Punktwolke erstellt werden. Je weniger überflüssige Punkte in der Szene vorhanden sind, desto weniger Floater können erstellt werden. Die Punktwolke allein ist jedoch nicht für die Erzeugung von Floatern verantwortlich. Wie in Kapitel 4.2 beschrieben, sorgen die Bildinformationen eines

unbearbeiteten Datensatzes dafür, dass der Hintergrund trotz fehlender Punkte erstellt wird. Im umgekehrten Fall geschieht es auf ähnliche Weise, jedoch mit einem entscheidenden Unterschied. Die Bildbearbeitung reduziert zwar die überflüssigen Punkte in der Punktwolke, entfernt sie aber nicht vollständig. Einzelne Punkte sind nach wie vor präsent. Das führt dazu, dass selbst mit freigestellten Bildern Floater entstehen, die jedoch durch einen maximalen Alphawert unsichtbar dargestellt werden.

Wie in der folgenden Abbildung zu sehen ist, zeigt die Punktwolke sowie die Gaußsche-Punktwolke an, dass Floater im Gaußschen Modell vorhanden sind. Wird das Hervorheben der Gaussians deaktiviert, ergibt sich ein Gaußsches Modell ohne Floater oder Artefakte.

Abbildung 17 - SuperSplat: Darstellung der Alpha Floater Punktwolkenansicht/Splatsansicht/Modell

In einem weiteren Vergleich wird deutlich, dass die Punktwolke in Verbindung mit den Hintergrundbildinformationen zur Bildung von Floatern führt. Für diesen Vergleich wurde jeweils der freigestellte Datensatz verwendet. Die von RealityCapture erstellte Sparse Point Cloud wurde beim linken Modell nicht weiterbearbeitet, die des rechten Modells wurde so weit bereinigt, dass nur noch einige Punkte um die Person vorhanden sind.

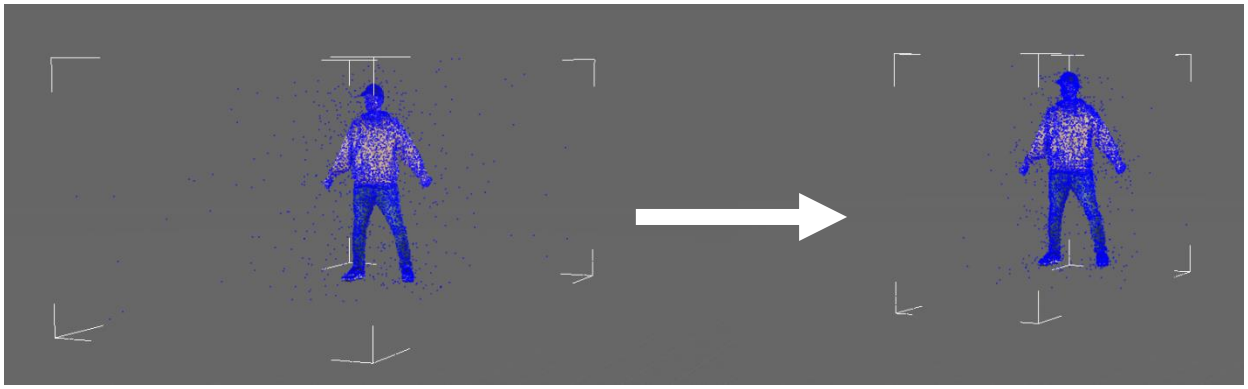


Abbildung 18 - SuperSplat: Demonstration der Floater Minimierung

Nach der Erstellung der Gaußschen Modelle und einer leichten Optimierung in SuperSplat ergeben sich folgende Ergebnisse. Um die in der obigen Abbildung gezeigten Ergebnisse zu erzielen, mussten bei der optimierten Point Cloud weniger Splats entfernt werden, wodurch gleichzeitig ein besseres Ergebnis erzielt wurde.

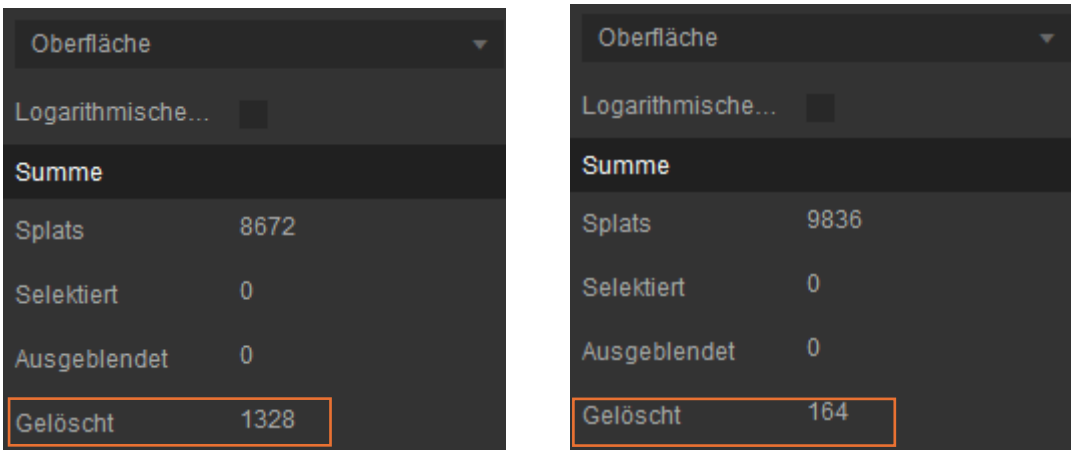


Abbildung 20 - SuperSplat: Anzahl der gelöschten Floater

Das bestätigt die Annahme, dass eine bereinigte Sparse Point Cloud in Kombination mit freigestellten Bildern die Entstehung von Floatern minimiert oder sogar verhindert.

V. Diskussion

In diesem Kapitel werden die erzielten Ergebnisse kritisch reflektiert, mögliche Limitationen der Untersuchung aufgezeigt und die Herausforderungen bei der Umsetzung der Methoden dargelegt. Neben der Analyse der Effizienz und Effektivität der angewandten Verfahren wird auch auf technische und methodische Einschränkungen eingegangen.

5.1 Kritische Reflexion der Ergebnisse

Der Arbeitsprozess von RealityCapture und PostShot konnte durch das Programm GS_Auto_Ersteller vollständig automatisiert werden. Dabei wurde die Erwartung, die jeweiligen Arbeitsschritte separat zu optimieren, übertroffen, indem beide Prozesse innerhalb eines einzigen Programms kombiniert wurden.

Der Ansatz, das Ergebnis mittels Python zu erzielen, erwies sich jedoch als ungeeignet, da fehlende APIs dazu führten, dass nicht alle notwendigen Funktionen angesteuert werden konnten. Der Versuch, dieses Problem mittels Workarounds zu umgehen, indem einzelne Pixelbereiche durch simulierte Mausklicks angesteuert werden, zeigte einige Schwächen, die eine zuverlässige Automatisierung behinderten.

Durch die vorgestellte Methode von Azad Balabanian [31] konnte eine Batchdatei erstellt werden, die zuverlässig und ohne Einschränkungen den Workflow optimiert, die manuellen Eingriffe minimiert und zugleich die Effizienz steigert.

Die Analyse, inwiefern sich das Freistellen der gescannten Bilder auf die Qualität der Gaußschen Modelle sowie der Floater Minimierung auswirkt, hat ergeben, dass visuell sichtbare Floater durch einen 100-prozentigen Alphawert eliminiert werden, jedoch nicht gänzlich aus der Datei gelöscht werden konnten.

Das liegt an der nicht perfekt optimierten Sparse Point Cloud, die dazu führt, dass Floater außerhalb des rekonstruierten Modells erzeugt werden, auch wenn diese nicht sichtbar sind. Zudem hilft das Freistellen der Bilder RealityCapture dabei, eine erste bereinigte Sparse Point Cloud zu erstellen, da der Fokus auf die gescannte Person oder das Objekt gelegt wird.

Darüber hinaus wird der Detailgrad sowie die Sauberkeit der Gaußschen Modelle durch die Bildbearbeitung verbessert, da zum Beispiel keine Gaussians entfernt werden müssen, die gleichzeitig der Umgebung und dem Objekt zugeordnet werden. Ein weiterer positiver Effekt zeigt sich in der Trainingszeit. Die Datensätze, bei denen der Hintergrund entfernt wurde, benötigten teilweise deutlich weniger Zeit als ein unbearbeitetes Datensatz. Allerdings ist der zeitliche Aufwand, alle Bilder manuell freizustellen, so hoch, dass es sich für die Erstellung eines einzelnen Objektes nicht lohnen würde.

Die Methoden, verschiedene Programme mit der Programmiersprache Python zu entwickeln, die den Freistellungsprozess automatisiert durchführen, erwiesen sich als nicht geeignet. Speziell die Herangehensweisen, Bilder per Differenzbildung und Farbraumanalyse freizustellen, lieferten mangelhafte Ergebnisse.

In Anbetracht dessen, wie lange sich Forscher und Entwickler bereits mit dieser Aufgabe auseinandersetzen, ein Tool zu programmieren, das effektiv Objekte auf Bildern erkennt und

entsprechend maskiert, stellt sich die Frage, inwieweit es sinnvoll war, diese Aufgabe ohne fundiertes Wissen im Programmieren anzugehen.

Selbst Photoshop, eine Software, die seit Jahren weiterentwickelt wird, kann trotz KI-gestützter Werkzeuge wie dem Objekt-Auswahlwerkzeug noch keine durchgängig konsistenten Ergebnisse liefern. In diesem Kontext hätte ein auf Deep Learning basierendes Modell, das speziell auf die Umgebung des Scanners trainiert wurde, mehr zum Erfolg beigetragen.

In Bezug auf die Floater Minimierung konnte ein Programm geschrieben werden, das überflüssige Punkte außerhalb eines vordefinierten Bereichs entfernt. Diese Methode ermöglicht jedoch keine saubere Bearbeitung des Gaußschen Modells. Der vordefinierte Bereich besitzt die Form eines Quaders. Dementsprechend bleiben Floater, die nahe an dem Objekt positioniert sind, weiterhin vorhanden.

Dieses Programm ist aufgrund fehlender Funktionen, wie der notwendigen Exporteinstellung, ASCII auf „false“ zu setzen, nicht für den praktischen Gebrauch geeignet. Eine weitere Validierung in SuperSplat konnte daher nicht vorgenommen werden, um eventuell zusätzliche Probleme des Programms ausfindig zu machen.

Trotz des unbrauchbaren Python-Programms zur Floater-Minimierung konnte durch weitere Tests herausgefunden werden, dass die Entstehung von Floatern das Ergebnis eines Zusammenspiels zwischen der Sparse Point Cloud und den zur Verfügung gestellten Bildinformationen ist. Das bedeutet, dass Floater auch dann entstehen, wenn die Sparse Point Cloud optimiert wurde, die Hintergrundinformationen aber weiterhin in den Bildern vorhanden sind.

Zugleich entstehen Floater, wenn die Sparse Point Cloud unbereinigt bleibt, die Bilder im Umkehrschluss aber freigestellt wurden. Der einzige Unterschied zwischen den beiden Varianten besteht darin, dass die Floater in der zweiten Methode zwar weiterhin vorhanden sind, jedoch unsichtbar bleiben und somit das rekonstruierte Modell optisch nicht beeinflussen. Wird allerdings die bereinigte Point Cloud mit den freigestellten Bildern kombiniert, entsteht ein Gaußsches Modell ohne Floater oder Artefakt.

Angesichts dieser Ergebnisse besteht ein Optimierungsansatz darin, den Fokus nicht auf die nachträgliche Bearbeitung des finalen Gaußschen Modells zur Entfernung von Floatern zu legen, sondern stattdessen eine Lösung zu entwickeln, die bereits die Sparse Point Cloud optimiert, bevor sie an PostShot zur Erstellung des Gaußschen Modells übergeben wird.

Die Wahl von Python als Programmiersprache für die Umsetzung der verschiedenen Methoden erwies sich sowohl als vorteilhaft als auch herausfordernd. Einerseits zeigte sich, dass für jede untersuchte Methode ein funktionstüchtiges Programm entwickelt werden konnte, was grundsätzlich für die Vielseitigkeit und Anwendbarkeit von Python spricht, andererseits stellten bestimmte programmiertechnische Herausforderungen eine Hürde dar, die durch iterative Anpassungen und Tests gelöst werden mussten.

Zusätzlich gab es das Problem der Systemabhängigkeit. Trotz identischer Installationsabläufe und gleicher Python-Versionen funktionierten einige Programme auf anderen Rechnern nicht

einwandfrei. Oftmals fehlten einzelne Abhängigkeiten oder Systemeinstellungen waren nicht kompatibel, was den Entwicklungsprozess weiter erschwerte.

Angesichts der begrenzten Zeit, die für die Untersuchung verschiedener Methoden zur Verfügung stand, musste der Fokus stark auf die Inbetriebnahme und Fehlerbehebung der Programme gelegt werden, anstatt sich intensiv mit den theoretischen Funktionsweisen einzelner Methoden auseinanderzusetzen. Dies führte dazu, dass einige vielversprechende Ansätze nicht weiterverfolgt werden konnten, weil der Aufwand zur Fehlerbehebung zu groß war.

Trotz dieser Herausforderungen hat sich Python als geeignete Programmiersprache für die Umsetzung der Methoden erwiesen, sofern ausreichend Erfahrung im Umgang mit Bibliotheken, Fehleranalyse und Debugging vorhanden ist. Andernfalls kann der Entwicklungsprozess unnötig langwierig und ineffizient werden, was in zeitkritischen Projekten problematisch ist.

5.2 Limitationen

Im Rahmen dieser Arbeit wurden verschiedene Methoden zur Optimierung der Erstellung und Darstellung von Gaussian Splats untersucht. Dennoch gab es einige Einschränkungen, die sowohl den Anwendungsbereich als auch die Tiefe der Untersuchung begrenzten.

5.2.1 Eingeschränkter Anwendungsbereich der Methoden

Die entwickelten Methoden zur Verbesserung der Qualität und zur Floater Minimierung wurden speziell für die Rekonstruktion von Personen und Objekten entwickelt. Die Anwendung dieser Methoden auf Szenenrekonstruktionen ist in diesem Kontext nicht sinnvoll, da in solchen Fällen möglichst viele Bildinformationen erhalten bleiben müssen. Eine Ausnahme bildet das erstellte Programm GS_Auto_Ersteller, das unabhängig vom rekonstruierenden Modell genutzt werden kann, solange RealityCapture und PostShot verwendet wird.

5.2.2 Keine Untersuchung von Deep-Learning-Methoden

Ein weiterer vielversprechender Ansatz zur automatisierten Bildfreistellung ist die Untersuchung und Entwicklung eines mittels Deep Learning trainierten Modells. Zwar haben erste Tests mit einem vortrainierten Modell gezeigt, dass die Methode grundsätzlich funktionieren kann, jedoch hätte die Entwicklung eines eigenen, speziell auf die Umgebung des Scanners trainierten Netzwerks den Rahmen dieser Bachelorarbeit gesprengt.

VI. Fazit und Ausblick

In diesem Kapitel werden die zentralen Ergebnisse der Arbeit zusammengefasst, die Forschungsfrage beantwortet und Empfehlungen für zukünftige Forschungen und Weiterentwicklungen gegeben. Dabei wird insbesondere auf die Automatisierung des Workflows, die Optimierung der Bildbearbeitung und die Reduzierung von Floatern eingegangen.

6.1 Zusammenfassung der Ergebnisse

Die zentrale Aufgabe, den Arbeitsprozess von RealityCapture und PostShot zu automatisieren, wurde nicht nur erfolgreich erfüllt, sondern so weit erweitert, dass nach dem Scannen der zu digitalisierenden Person keine weiteren Arbeitsschritte nötig sind, um ein Gaussian Splat Modell zu erstellen.

Es wurden mehrere Programme mit unterschiedlichen Methoden zur Bildbearbeitung erstellt und getestet, die jedoch keine verlässlichen Ergebnisse erzielten. Die Bildbearbeitung trägt allerdings dazu bei, die Erstellung der Sparse Point Cloud auf die wesentlichen Punkte zu beschränken und somit unerwünschte Bereiche wie den Hintergrund (Scanner) auszublenden.

Die Qualität der Gaußschen Modelle kann in puncto Sauberkeit und Detailgrad durch das Freistellen der Bilder verbessert werden und erfordert keine präzise Nachbearbeitung, welche die Qualität des Modells beeinflussen kann. Der Einfluss der Bildbearbeitung auf die Floater Minimierung kann in zwei Kategorien unterteilt werden.

Zum einen löscht diese nicht aktiv die Floater, sondern hilft RealityCapture dabei, eine erste bereinigte Punktwolke zu erstellen, wodurch die Erzeugung von Floatern minimiert wird. Zum anderen werden erstellte Gaussians, die nicht zur Person gehören und daher auch keine Farbinformationen besitzen, einem 100-prozentigen Alphawert zugeordnet. Das minimiert die Floater an sich nicht, macht sie aber unsichtbar, was zu einer Minimierung der sichtbaren Artefakte führt. Darüber hinaus wird der Trainingsprozess zur Erstellung des Gaußschen Modells beschleunigt. Der einzige Nachteil besteht in der langen Bearbeitungszeit, die benötigt wird, um einen gesamten Datensatz manuell freizustellen.

Es gelang, ein Programm zu schreiben, das die finale Punktwolke des Gaußschen Modells von den meisten Floatern befreit. Allerdings verhindert die fehlende Option, ASCII auf „false“ zu stellen, die Weiterbearbeitung des Modells in SuperSplat.

Um Floater aktiv an dessen Erstellung zu hindern, sind zwei Schritte notwendig. Die Sparse Point Cloud muss so optimiert werden, dass die Punkte, die nicht zu dem gescannten Objekt gehören, entfernt werden. Ohne die Punkte ist PostShot nicht in der Lage, an diesen Stellen Gaussians zu platzieren (Floater), sofern die Bildinformationen, die den Hintergrund definieren, entfernt werden. Dadurch entsteht ein floaterfreies, sauberes und detailreiches Gaußsche Modell.

6.2 Beantwortung der Forschungsfrage

Durch die Automatisierung der Arbeitsprozesse von RealityCapture und PostShot mittels einer Batch-Datei, die Funktionen und Befehle aus den jeweiligen Command-Line-

Dokumentationen beinhaltet, konnte der Workflow verbessert und die Effizienz gesteigert werden. Mithilfe von Bildbearbeitung im Hinblick auf das Entfernen des Hintergrunds lässt sich die Qualität der Gaußschen Modelle steigern und der Trainingsprozess beschleunigen. Des Weiteren trägt die Bildbearbeitung dazu bei, Floater durch einen hohen Alphawert unsichtbar zu machen, und hilft bei der Erstellung einer grob bereinigten Sparse Point Cloud. Die Optimierung dieser Point Cloud im Zusammenspiel mit dem Freistellen der Bilder verhindert die Entstehung von Floatern. Damit konnte die Forschungsfrage „*Mit welchen Ansätzen lässt sich der Workflow effizienter gestalten und die Entstehung von Floatern reduzieren, um die Qualität und Präzision der finalen Modelle zu verbessern?“* beantwortet werden.

6.3 Empfehlungen für zukünftige Forschung und Weiterentwicklung

Diese Arbeit liefert ein solides Fundament zur Workflow-Optimierung sowie zur Bedeutung der Bildbearbeitung für die Erstellung von Gaussian Splats. Auf der Grundlage der hier erzielten Ergebnisse empfiehlt es sich, in zukünftigen Forschungen den Fokus auf die automatisierte Bildbearbeitung zu legen, die qualitativ an die manuelle Freistellung heranreicht. Aufgrund der Schwierigkeit, ein eigenständiges Programm für dieses Thema zu entwickeln, sollte stattdessen ein Deep-Learning-Modell erstellt werden, das speziell auf den verwendeten Scanner trainiert ist, um die Aufgabe zuverlässig zu bewältigen.

Des Weiteren empfiehlt es sich, ein funktionstüchtiges Programm zu entwickeln, das die Sparse Point Cloud so genau wie möglich von überflüssigen Punkten bereinigt. In Kombination mit dem auf Deep Learning basierenden Modell zur Bildbearbeitung und dem Programm GS_Auto_Ersteller könnte so ein automatisierter Workflow entstehen, der die für das Freistellen benötigte Zeit auf ein Minimum reduziert, die Qualität des Gaußschen Modells verbessert, Floater minimiert oder sogar verhindert, die Trainingszeit des Gaußschen Modells verkürzt und somit keine manuellen Eingriffe mehr erforderlich macht.

Literatur

[1] Prof. Dr. Richard Lackes, "ASCII(-Code) Definition," *Springer Fachmedien Wiesbaden GmbH*, 19. Februar 2018. Zugriff am: 1. Februar 2025. [Online.] Verfügbar: https://wirtschaftslexikon.gabler.de/definition/ascii-code-28843/version-252467

[2] R. Yamashita, M. Nishio, R. K. G. Do und K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, Early Access. doi: 10.1007/s13244-018-0639-9.

[3] B. Kerbl, G. Kopanas, T. Leimkuehler und G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," *ACM Trans. Graph.*, Jg. 42, Nr. 4, S. 1–14, 2023, doi: 10.1145/3592433.

[4] Tobias Klinke, "Bildverarbeitung in R: Ausarbeitung zum Vortrag," 2016. [Online]. Verfügbar unter: https://wr.informatik.uni-hamburg.de/_media/teaching/sommersemester_2016/pir-16-tobias_klinke-report.pdf

[5] H. Ramchoun, M. Amine, J. Idrissi, Y. Ghanou und M. Ettaouil, "Multilayer Perceptron: Architecture Optimization and Training," *IJIMAI*, Jg. 4, Nr. 1, S. 26, 2016. doi: 10.9781/ijimai.2016.415. [Online]. Verfügbar unter: https://www.ijimai.org/journal/sites/default/files/files/2016/02/ijimai20164_1_5_pdf_30533.pdf

[6] Y. Xie *et al.,* "Neural Fields in Visual Computing and Beyond," *Computer Graphics Forum*, Jg. 41, Nr. 2, S. 641–676, 2022, doi: 10.1111/cgf.14505.

[7] V. Sitzmann, J. Thies, F. Heide, M. Nießner, G. Wetzstein and M. Zollhofer, "DeepVoxels: Learning Persistent 3D Feature Embeddings," S. 2432–2441.

[8] M. Zwicker, H. Pfister, J. van Baar und M. Gross, "EWA volume splatting," in *Proceedings Visualization, 2001. VIS '01*, null, doi: 10.1109/visual.2001.964490.

[9] G. P. Renieblas, A. T. Nogués, A. M. González, N. Gómez-Leon und E. G. Del Castillo, "Structural similarity index family for image quality assessment in radiological images," *Journal of medical imaging (Bellingham, Wash.)*, Early Access. doi: 10.1117/1.JMI.4.3.035501.

[10] G. Chen und W. Wang, "A Survey on 3D Gaussian Splatting," Jan. 2024. [Online]. Verfügbar unter: http://arxiv.org/pdf/2401.03890

[11] M. Aleksandrov, S. Zlatanova und D. J. Heslop, "Voxelisation Algorithms and Data Structures: A Review," *Sensors (Basel, Switzerland)*, Early Access. doi: 10.3390/s21248241.

[12] M. Jarofka, S. Schweig, N. Maas und D. Schramm, "Toolchain Development for Automated Scene Reconstruction using Artificial Neural Network Object Detection and Photogrammetry for the Application in Driving Simulators," S. 25–34. [Online]. Verfügbar unter: https://pdfs.semanticscholar.org/294c/d0bac39c6489d55cfac57d2fcc464d1ab4ac.pdf

[13] N. Snavely, S. M. Seitz und R. Szeliski, "Modeling the World from Internet Photo Collections," *Int J Comput Vis*, Jg. 80, Nr. 2, S. 189–210, 2008, doi: 10.1007/s11263-007-0107-3.

[14] Formlabs. "Photogrammetrie: Schritt-für-Schritt-Leitfaden und Softwarevergleich." Zugriff am: 28. Januar 2025.

[15] H.-G. Maas, *Mehrbildtechniken in der digitalen Photogrammetrie* (Zugl.: Zürich, Eidgenöss. Techn. Hochschule, Habil.-schr., 1997) (Mitteilungen / Institut für Geodäsie und Photogrammetrie an der Eidgenössischen Technischen Hochschule Zürich 62). Zürich: Eidgenössische Techn. Hochschule, 1997. [Online]. Verfügbar unter: https://ethz.ch/content/dam/ethz/special-interest/baug/igp/igp-dam/documents/PhD_Theses/62.pdf

[16] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi und R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," Mrz. 2020. [Online]. Verfügbar unter: http://arxiv.org/pdf/2003.08934

[17] M. Goesele, N. Snavely, B. Curless, H. Hoppe und S. M. Seitz, "Multi-View Stereo for Community Photo Collections," S. 1–8.

[18] K. Gao, Y. Gao, H. He, D. Lu, L. Xu und J. Li, "NeRF: Neural Radiance Field in 3D Vision, A Comprehensive Review," Okt. 2022. [Online]. Verfügbar unter: https://arxiv.org/pdf/2210.00379

[19] C. Sun, M. Sun und H.-T. Chen, "Direct Voxel Grid Optimization: Super-fast Convergence for Radiance Fields Reconstruction," Nov. 2021. [Online]. Verfügbar unter: http://arxiv.org/pdf/2111.11215

[20] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin und A. Geiger, "Occupancy Networks: Learning 3D Reconstruction in Function Space," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, doi: 10.1109/cvpr.2019.00459.

[21] J. J. Park, P. Florence, J. Straub, R. Newcombe und S. Lovegrove, "DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, S. 165–174, doi: 10.1109/cvpr.2019.00025.

[22] S. Fridovich-Keil, G. Meanti, F. R. Warburg, B. Recht und A. Kanazawa, "K-Planes: Explicit Radiance Fields in Space, Time, and Appearance," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, S. 12479–12488, doi: 10.1109/cvpr52729.2023.01201.

[23] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan und P. Hedman, "Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields," Nov. 2021. [Online]. Verfügbar unter: http://arxiv.org/pdf/2111.12077

[24] T. Müller, A. Evans, C. Schied und A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, Jg. 41, Nr. 4, S. 1–15, 2022, doi: 10.1145/3528223.3530127.

[25] "Ein Leitfaden für 3D-Gauß-Splatting und wie man es erstellt." Zugriff am: 28. Januar 2025. [Online.] Verfügbar: https://landing.pixcap.com/de/blog/gaussian-splatting

[26] T. T. Bharti, "Background Subtraction Techniques-Review," *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, Volume-2 Issue-3, 2013. [Online]. Verfügbar unter: https://www.ijitee.org/wp-content/uploads/papers/v2i3/C0454022313.pdf

[27] H. Shimodaira, *Automatic Color Image Segmentation Using a Square Elemental Region-Based Seeded Region Growing and Merging Method*, 2017.

[28] "Create actions in Adobe Photoshop." Zugriff am: 28. Januar 2025. [Online.] Verfügbar: https://helpx.adobe.com/photoshop/using/creating-actions.html
[29] O. Ronneberger, P. Fischer und T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," Mai. 2015. [Online]. Verfügbar unter: http://arxiv.org/pdf/1505.04597
[30] Firma botspot. "Full-Body Scanner Neo." [Online.] Verfügbar: https://botspot.de/products/botscan-neo
[31] Azad Balabanian. "RealityCapture-to-PostShot." Zugriff am: 28. Januar 2025. [Online.] Verfügbar: https://github.com/azadbal/RealityCapture-to-PostShot

VII. Anhang

Selbstauslöser

Codeabschnitt AutoHotKey Skript 1 - Koordinaten der Mausposition

```
SetTimer (ShowMousePosition, 100) ;
return

ShowMousePosition() {
      MouseGetPos &xpos, &ypos ;
      ToolTip "X: " xpos " Y: " ypos ;
}

^Esc::ExitApp() ;
```

Codeabschnitt AutoHotKey Skript 2 - Selbstauslöser (Scanner Control)

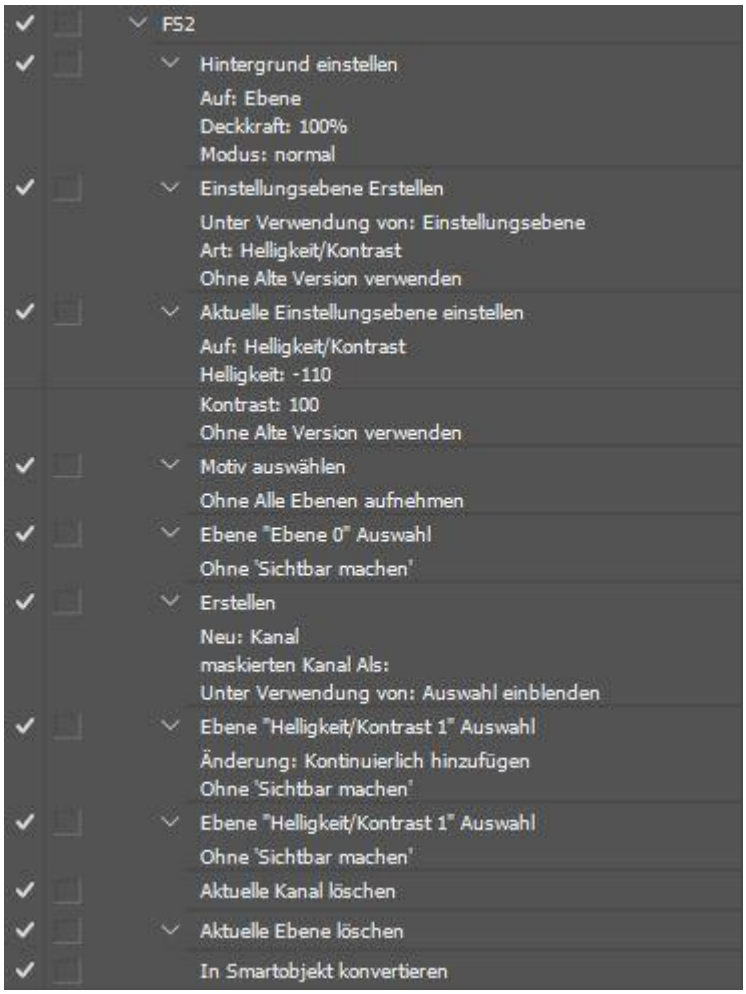
```
Sleep (18000) ;
WinActivate ("scanner-controll") ;
Click(525, 123);


Sleep (18000) ;
WinActivate ("scanner-controll") ;
Click(525, 123);
```

Codeabschnitt – Photoshop-Skript

```
#target photoshop
// Ordnerpfade anpassen
var inputFolder = Folder.selectDialog("Wähle den Ordner mit den Bildern");
var outputFolder = Folder.selectDialog("Wähle den Zielordner für die
exportierten JPGs");
if (inputFolder && outputFolder) {
    var files =
inputFolder.getFiles(/\.(jpg|jpeg|png|tif|tiff|gif|bmp|psd)$/i);
    for (var i = 0; i < files.length; i++) {
        var doc = open(files[i]);
        // Aktion ausführen
        try {
            app.doAction("FS2", "BA"); // Aktionssatz "BA" & Aktion „FS2“,
vordefiniert in Photoshop
        } catch (e) {
            alert("Fehler beim Ausführen der Aktion für " + files[i].name +
": " + e.message);
            throw new Error("Das Skript wird aufgrund eines Fehlers
beendet."); // Skript beenden
        }
        // Bild als JPG speichern
        var jpgOptions = new JPEGSaveOptions();
        jpgOptions.quality = 12; // Höchste Qualität
        var outputFile = new File(outputFolder + "/" +
doc.name.replace(/\.[^\.]+$/, '.jpg'));
        doc.saveAs(outputFile, jpgOptions, true, Extension.LOWERCASE);
        doc.close(SaveOptions.DONOTSAVECHANGES);
    }
    alert("Fertig! Alle Bilder wurden verarbeitet.");
} else {
    alert("Es wurde kein Ordner ausgewählt.");
}
```

Aktionsauflistung (Photoshop)



Python-Skripts für die Bildbearbeitung

Codeabschnitt Programm 1 – Differenzbildung von Bildern

```
import numpy as np
from matplotlib.image import imread
from PIL import Image

def greyscaler_image(image):
    return np.dot(image[...,:3], [0.299, 0.587, 0.114])

def image_to_array(image_path):
    image = imread(image_path)
    greyscale_image = greyscaler_image(image)
    image = np.array(image)
    greyscale_image = np.array(greyscale_image)

    return image, greyscale_image

def array_to_image(image_array):
    return Image.fromarray(np.uint8(image_array))
```

```
image_path = r'C:\Users\Fabi\Desktop\Python\Fabilibary\image3.jpg'
image_path_reference =
r'C:\Users\Fabi\Desktop\Python\Fabilibary\image4.jpg'

image, greyscale_image = image_to_array(image_path)
image_reference, greyscale_image_reference =
image_to_array(image_path_reference)

# Subtrahiere die beiden Bilder voneinander, mache ein neues Bild nur mit
den Unterschieden
# Herausfinden, welche Pixel in greyscale_image und
greyscale_image_reference unterschiedlich sind
def find_differences(greyscale_image, greyscale_image_reference,
threshold):
    diff_matrix = np.zeros(greyscale_image.shape)
    for i in range(0, len(greyscale_image)):
        for j in range(0, len(greyscale_image[0])):
            if greyscale_image[i][j] != greyscale_image_reference[i][j]:
                if (greyscale_image[i][j] -
greyscale_image_reference[i][j]) > threshold:
                    diff_matrix[i][j] = 255
                elif (greyscale_image[i][j] -
greyscale_image_reference[i][j]) < threshold*(-1):
                    diff_matrix[i][j] = 255
                else:
                    diff_matrix[i][j] = 0
    return diff_matrix

# apply a mask to the image, which only shows the differences
def apply_mask(image, mask):
    masked_image = np.zeros(image.shape)
    for i in range(0, len(image)):
        for j in range(0, len(image[0])):
            if mask[i][j] == 255:
                masked_image[i][j] = image[i][j]
    return masked_image

differences_image = find_differences(greyscale_image,
greyscale_image_reference, threshold=1)
masked_image = apply_mask(image_reference, differences_image)

# Masked Image as Image
image_masked_image = array_to_image(masked_image)
image_differences_image = array_to_image(differences_image)
image_masked_image.show()
```

Codeabschnitt Programm 2 – Farbraumanalyse

```
import cv2
import numpy as np

# Read image
img = cv2.imread('image2.jpg')
hh, ww = img.shape[:2]
```

```
# Convert the image to HSV color space
hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# Threshold on Saturation channel to isolate colorful areas
# Lower and upper bounds for saturation to keep only colorful regions
lower_saturation = 50  # Minimum saturation for a color to be considered
"colorful"
upper_saturation = 245  # Maximum saturation (we can keep all high
saturation colors)

# Threshold on Value (Brightness) channel to remove dark (black/white)
regions
lower_value = 40  # Lower bound for brightness to remove black/white areas
upper_value = 255  # Keep everything brighter than this value

# Create a mask where the saturation and value are above the thresholds
saturation_channel = hsv_img[:, :, 1]  # Extract the Saturation channel
value_channel = hsv_img[:, :, 2]  # Extract the Value (Brightness) channel

# Colorful mask based on both saturation and value
color_mask = cv2.inRange(saturation_channel, lower_saturation,
upper_saturation)
color_mask = cv2.bitwise_and(color_mask, cv2.inRange(value_channel,
lower_value, upper_value))

# Optional: Apply morphological operations to remove small noise (Floater)
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (1, 1))  # Slightly
larger kernel to remove small objects
morph_result = cv2.morphologyEx(color_mask, cv2.MORPH_OPEN, kernel)  # Open
operation to remove small noise

# Apply the color mask to the image
result = cv2.bitwise_and(img, img, mask=morph_result)

# Optional: Apply further morphological operations to close small holes in
the mask
kernel_close = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15, 15))  #
Larger kernel for closing small gaps
result_closed = cv2.morphologyEx(result, cv2.MORPH_CLOSE, kernel_close)

# Save results
cv2.imwrite('pills_color_filtered_fine.jpg', result)
cv2.imwrite('pills_morph_filtered_fine.jpg', morph_result)
cv2.imwrite('pills_result_closed.jpg', result_closed)
```

Codeabschnitt Programm 3 – Deep Learning U^2-Net, rembg

```
import os
from rembg import remove
from PIL import Image

def remove_background(input_image_path, output_image_path):
    # Öffne das Eingabebild
```

```
    with open(input_image_path, 'rb') as input_file:
        input_image = input_file.read()

    # Hintergrund entfernen
    output_image = remove(input_image)

    # Speichere das Ergebnis
    with open(output_image_path, 'wb') as output_file:
        output_file.write(output_image)

def process_all_images_in_folder(input_folder, output_folder):
    # Überprüfe, ob der Ausgabeordner existiert, wenn nicht, erstelle ihn
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    # Iteriere über alle Dateien im Eingabeordner
    for filename in os.listdir(input_folder):
        # Überprüfe, ob die Datei ein Bild ist (z.B. .jpg, .jpeg, .png)
        if filename.lower().endswith(('.jpg', '.jpeg', '.png')):
            input_image_path = os.path.join(input_folder, filename)
            output_image_path = os.path.join(output_folder,
f"{os.path.splitext(filename)[0]}_output.png")

            # Entferne den Hintergrund und speichere das Bild
            remove_background(input_image_path, output_image_path)
            print(f"Bild {filename} erfolgreich verarbeitet und
gespeichert.")

# Beispiel: Pfade für den Eingabe- und Ausgabeordner
input_folder = r"D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\Freisteller\Import\Set1"
output_folder = r"D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\Freisteller\Export"

# Alle Bilder im Ordner verarbeiten
process_all_images_in_folder(input_folder, output_folder)
```

Automatisierung von RealityCapture und PostShot

Codeabschnitt - RealityCapture Teilautomatisierung

```
import os
import subprocess
import pyautogui
import time
import pygetwindow as gw

# Funktion, die RealityCapture startet und die Rekonstruktionsregion setzt
def run_realitycapture():
    rc_path = r"M:\RealityCapture_1.5\AppProxy.exe"
    project_path = r"M:\Bachelor\Tutorial2.rcproj"
    region_path =
r"M:\Bachelor\Export_Reconstruction_Region\RC_Tutorial2.rcbox"
```

```
    if not os.path.exists(rc_path) or not os.path.exists(project_path) or
not os.path.exists(region_path):
        print("Fehler: Eine oder mehrere Dateien wurden nicht gefunden.")
        return

    command = [rc_path, "-load", project_path, "-selectMaximalComponent",
"-setReconstructionRegion", region_path]
    try:
        subprocess.run(command, check=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)
        bring_window_to_front("RealityCapture")
    except subprocess.CalledProcessError as e:
        print(f"Fehler: {e.stderr}")

# Bringt das RealityCapture-Fenster in den Vordergrund
def bring_window_to_front(window_title):
    try:
        window = gw.getWindowsWithTitle(window_title)[0]
        window.activate()
        wait_for_window_active(window)
    except IndexError:
        print(f"Fenster '{window_title}' nicht gefunden.")
    except Exception as e:
        print(f"Fehler: {e}")

# Wartet, bis das Fenster aktiv ist
def wait_for_window_active(window, timeout=30):
    start_time = time.time()
    while time.time() - start_time < timeout:
        if window.isActive:
            break
        time.sleep(0.1)
    else:
        print("Timeout: Fenster wurde nicht aktiv.")

# Führt den Export der Point Cloud in RealityCapture durch
def export_point_cloud():
    print("Warte auf RealityCapture...")
    while not gw.getWindowsWithTitle("RealityCapture"):
        time.sleep(2)

    try:
        time.sleep(5)
        # Simuliere Klicks, um das Menü zu öffnen und die Point Cloud zu
exportieren
        pyautogui.click(x=187, y=47)  # Klick auf die Position, um das Menü
zu öffnen
        time.sleep(1)
        pyautogui.click(x=463, y=100)  # Klick auf "Point Cloud", um den
Export zu starten
        print("Export Point Cloud wurde ausgewählt.")
    except Exception as e:
        print(f"Fehler bei der Simulation von PyAutoGUI: {e}")

# Hauptfunktion
def main():
    run_realitycapture()
    export_point_cloud()
```

```
if __name__ == "__main__":
    main()
```

Codeabschnitt - Automatisierung von PostShot

```
@echo off
"C:\Program Files\Jawset PostShot\bin\postshot-cli.exe" train -i
"D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\PostShot\Import_Daten\images" -i
"D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\PostShot\Import_Daten\Muster_freigestellt_1.csv" -i
"D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\PostShot\Import_Daten\Muster_Ready_for_Postshot.ply" -o
"D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\Postshot\Export_Test\output.psht" --profile "Splat MCMC" -
-export-splat-ply "D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\Postshot\Export_Test\splat_model.ply" >
"D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\Postshot\Export_Test\train_log.txt" 2>&1
```

Codeabschnitt – Originalcode von Azad Balabanian (GitHub)

```
::Aligns photos in RC and trains in PostShot

:: path to RealityCapture and PostShot directories
set RealityCaptureExe="C:\Program Files\Capturing
Reality\RealityCapture\RealityCapture.exe"
set PostShotEXE="C:\Program Files\Jawset Postshot\bin\postshot-cli.exe"

:: root path of project (%~dp0 means the flder in which this script is
stored)
set RootFolder=%~dp0

:: set path to the folder with your images
set Images="%RootFolder%images"

:: set the path to RealityCapture settings
set SettingsFolder="X:\FILES\RC\RC_CLI\Settings"

:: set the path, where project is going to be saved, and its name
set Project="%RootFolder%\RC.rcproj"

:: run RealityCapture
%RealityCaptureExe% -newScene ^
      -set "appIncSubdirs=true" ^
        -addFolder %Images% ^
        -save %Project% ^
        -align ^
        -selectMaximalComponent ^
```

```
        -exportRegistration %Images%\registeration.csv
"%SettingsFolder%\3DGS_reg.xml" ^
      -exportSparsePointCloud %Images%\pointcloud.ply
"%SettingsFolder%\3DGS_ply.xml" ^
      -save ^
      -quit

:: run PostShot
%PostShotEXE% --import %Images% ^
      --show-train-error ^
      --profile "Splat MCMC" ^
      --max-image-size 1600 ^
      --train-steps-limit 1 ^
      --max-num-splats 1000 ^
      --output "%RootFolder%output.psht"
```

Codeabschnitt Finales Programm – GS_Auto_Ersteller

```
@echo on
:: Aligns photos in RC and trains in PostShot

:: path to RealityCapture and PostShot directories
set RealityCaptureExe="C:\Program Files\Capturing
Reality\RealityCapture\RealityCapture.exe"
set PostShotEXE="C:\Program Files\Jawset Postshot\bin\postshot-cli.exe"

:: root path of project (%~dp0 means the folder in which this script is
stored)
set RootFolder=%~dp0

:: set path to the folder with your images
set Images="%RootFolder%images"

:: set the path to RealityCapture settings
set
SettingsFolder="C:\Users\S024_Calc2\Desktop\Automatisch_Projekt_1\Settings"

:: set the path, where project is going to be saved, and its name
set Project="%RootFolder%\RC.rcproj"

:: run RealityCapture
%RealityCaptureExe% -newScene ^
    -set "appIncSubdirs=true" ^
    -addFolder %Images% ^
    -save %Project% ^
    -align ^
    -selectMaximalComponent ^
    -exportRegistration "%Images%\registeration.csv"
"%SettingsFolder%\3DGS_reg.xml" ^
    -exportSparsePointCloud "%Images%\pointcloud.ply"
"%SettingsFolder%\3DGS_ply.xml" ^
    -save ^
    -quit

:: run PostShot and export the .ply file
```

```
%PostShotEXE% train ^
    --import %Images% ^
    --show-train-error ^
    --profile "Splat MCMC" ^
    --max-image-size 1600 ^
    --train-steps-limit 1 ^
    --max-num-splats 3000 ^
    --output "%RootFolder%output.psht" ^
    --export-splat-ply "%RootFolder%pointcloud_exported.ply"

Pause
```

Pythonprogramm zur automatischen Floater Entfernung

Codeabschnitt - Optimierung der Point Cloud/Entfernung von Floatern

```
#%%
import numpy as np
from plyfile import PlyData, PlyElement

def filter_points(ply_file, min_bounds, max_bounds, output_file):
    # Lade die .ply-Datei
    ply_data = PlyData.read(ply_file)

    # Hole die Punkte aus der .ply-Datei (meistens unter dem 'vertex'
Element)
    vertex_data = ply_data['vertex'].data

    # Extrahiere die X, Y, Z Koordinaten der Punkte sowie Farbwerte (rot,
grün, blau)
    points = np.array([(vertex_data['x'], vertex_data['y'],
vertex_data['z'],
                        vertex_data['red'], vertex_data['green'],
vertex_data['blue'])
                       for vertex_data in vertex_data])

    # Filtere die Punkte, die innerhalb des spezifizierten Bereichs liegen
    filtered_points = points[
        (points[:, 0] >= min_bounds[0]) & (points[:, 0] <= max_bounds[0]) &
        (points[:, 1] >= min_bounds[1]) & (points[:, 1] <= max_bounds[1]) &
        (points[:, 2] >= min_bounds[2]) & (points[:, 2] <= max_bounds[2])
    ]

    # Prüfe, ob noch Punkte übrig sind
    if len(filtered_points) == 0:
        print("Warnung: Es wurden keine Punkte gefunden, die innerhalb des
angegebenen Bereichs liegen.")
    else:
        print(f"Anzahl der gefilterten Punkte: {len(filtered_points)}")

    # Erstelle ein neues PlyElement mit den gefilterten Punkten (x, y, z)
und den Farben (red, green, blue)
    filtered_vertex = np.array([(x, y, z, red, green, blue)
                                for x, y, z, red, green, blue in
filtered_points],
```

```
                                dtype=[('x', 'f4'), ('y', 'f4'), ('z',
'f4'),
                                       ('red', 'u1'), ('green', 'u1'),
('blue', 'u1')])

    # Erstelle die neue PlyData und füge alle anderen Felder wieder hinzu
(z.B. Normalen, Texturen, etc.)
    other_elements = []

    for element in ply_data.elements:
        if element.name == 'vertex':
            continue
        for attribute in element.data.dtype.names:
            if attribute:

other_elements.append(PlyElement.describe(element.data[attribute],
attribute))

    # Erstelle die neue PlyData mit den gefilterten Punkten und den anderen
Attributen
    filtered_ply = PlyData([PlyElement.describe(filtered_vertex, 'vertex')]
+ other_elements, text=False)

    # Speichere die gefilterte Punktwolke in einer neuen .ply-Datei
    filtered_ply.write(output_file)
    print(f"Gefilterte Punktwolke wurde in '{output_file}' gespeichert.")

# Beispielaufruf der Funktion
min_bounds = [-10.0, -100.0, -51.25]  # Minimale Grenzen (x, y, z)
max_bounds = [10.0, -80.0, 51.25]     # Maximale Grenzen (x, y, z)
input_ply_file = r'D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\Postshot\Fertige Postshot files\Test 2 Mit
Skript\Muster2.ply'  # Deine Eingabedatei
output_ply_file = r'D:\Bachelor_Fabian_Merschel\Korrekte
Ordnerstrukturen\Postshot\Fertige Postshot files\Test 2 Mit
Skript\Muster2_Skript.ply'  # Die Ausgabe-Datei nach der Filterung
```