



Hochschule  
**Augsburg** University of  
Applied Sciences

## Masterarbeit

Fakultät für  
Informatik

Studienrichtung  
Informatik

**Bernd Fecht**  
**3D-Visualisierung von Kariesläsionen aus  
Nahinfrarotaufnahmen**

Prüfer: Prof. Dr. Peter Rösch  
Abgabe der Arbeit am: 11.03.2019

Hochschule für angewandte  
Wissenschaften Augsburg  
University of Applied Sciences  
An der Hochschule 1  
D-86161 Augsburg  
Telefon +49 821 55 86-0  
Fax +49 821 55 86-3222  
[www.hs-augsburg.de](http://www.hs-augsburg.de)  
[info@hs-augsburg.de](mailto:info@hs-augsburg.de)

Fakultät für Informatik  
Telefon: +49 821 5586-3450  
Fax: +49 821 5586-3499



## Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbstständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

---

Ort, Datum

---

Unterschrift des/der Studierenden

## **Kurzfassung**

In der Zahnmedizin werden die Zähne oftmals mithilfe von Röntgenstrahlen untersucht, um Karies diagnostizieren zu können. Durch die ionisierende Eigenschaft der Röntgenstrahlen können dabei jedoch bleibende Gewebeschäden auftreten. Deshalb werden stets neue Diagnoseverfahren entwickelt, die ohne der ionisierenden Röntgenstrahlung auskommen. Seit ca. zehn Jahren steht ein Diagnoseverfahren zur Verfügung, das Nahinfrarot anstelle von Röntgenstrahlung verwendet. Nahinfrarot hat die Eigenschaft, tiefer in die Zahnsubstanzen eindringen zu können, als das sichtbare Licht. Somit können unter Verwendung verschiedener bildgebender Verfahren in der Zahnmedizin, detaillierte Bilder von Zähnen gemacht werden, die Defekte wie Kariesläsionen als Kontrastunterschied zu den anderen Zahnsubstanzen darstellen. In dieser Arbeit wird untersucht, ob die Nahinfrarotaufnahmen als eine geeignete Methode zur Unterstützung der zahnmedizinischen Diagnostik eingesetzt werden könnte. Dazu werden die Grundlagen, Anforderungen und Voraussetzungen für die Implementierung eines Programms untersucht, das die Kariesläsionen aus Nahinfrarotaufnahmen dreidimensional visualisieren kann. Zudem wird die prototypische Entwicklung einer solchen Software beschrieben.

## **Danksagung**

Diese Arbeit stellt den Abschluss meines Informatikstudiums dar. Das Studium war für mich sowohl inhaltlich, als auch zeitlich, aufgrund meiner Nebentätigkeit als Werkstudent, nicht immer einfach und eine große Herausforderung. Umso mehr bin ich stolz, diesen Weg beschritten zu haben. Unterstützt wurde ich dabei durch Kommilitonen, Freunde und vor allem meiner Familie. Insbesondere möchte ich meiner Partnerin Mirjam Schaich danken, die mich während dieser Zeit stets ermutigt und unterstützt hat.

Auch möchte ich meiner Schwester Verena Fecht danken, die mit ausführlichen Korrekturarbeiten zu dieser Arbeit geholfen hat.

Weiter danke ich der Firma Ghostthinker GmbH und deren Mitarbeitern, die mir während der Masterarbeit stets zur Seite standen. Insbesondere geht mein Dank hierbei an Johannes Metscher und Rebecca Gebler für ihr Verständnis und die großartige Unterstützung.

Ferner gilt mein Dank der Firma AraCom IT Services AG und deren Mitarbeitern, die mir ebenfalls stets hilfreich zur Seite standen.

Zudem möchte ich mich bei Herrn Prof. Dr. Peter Rösch für die Betreuung während dieser Arbeit und der Begleitung während meines Studiums bedanken. Mein besonderer Dank geht an Herrn Prof. Dr. Karl-Heinz Kunzelmann, der freundlicherweise die Daten, die in dieser Arbeit verwendeten wurden, generiert und bereit gestellt hat und als fachlicher Ansprechpartner stets zur Verfügung stand.

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>15</b>
1.1 Motivation . . . . .	16
1.2 Ziele der Arbeit . . . . .	17
1.3 Inhaltlicher Aufbau der Arbeit . . . . .	17
<b>2 Grundlagen</b>	<b>19</b>
2.1 Nahinfrarot-Bildgebung in der Zahnmedizin . . . . .	19
2.2 3D-Computergrafik . . . . .	26
2.3 Texture Mapping . . . . .	38
<b>3 Analyse</b>	<b>45</b>
3.1 Datenanalyse . . . . .	45
3.2 Anforderungsanalyse . . . . .	46
3.3 Fazit . . . . .	48
<b>4 Stand der Technik</b>	<b>49</b>
4.1 Kamerakalibrierung . . . . .	49
4.2 Texturierung anhand mehrerer Aufnahmen . . . . .	58
<b>5 Machbarkeitsanalyse</b>	<b>73</b>
5.1 Verfahrensanalyse . . . . .	73
5.2 Kanten und Silhouetten . . . . .	74
5.3 Mutual Information (MI) . . . . .	74
5.4 Erkenntnisse und Konsequenzen . . . . .	80
<b>6 Implementierung</b>	<b>81</b>
6.1 Verwendete Software . . . . .	81
6.2 Verwendete Softwarekomponenten . . . . .	82
6.3 Qualitätssicherung . . . . .	83
6.4 Gesamtübersicht . . . . .	84
6.5 Implementierungsdetails . . . . .	85
6.6 Zusammenfassung . . . . .	97
<b>7 Ergebnisse und Evaluierung</b>	<b>99</b>
7.1 Ergebnisse . . . . .	99
7.2 Programmalaufzeit . . . . .	99
7.3 Bekannte Probleme . . . . .	102
7.4 Anwendertest . . . . .	105

<b>8 Zusammenfassung und Ausblick</b>	<b>107</b>
8.1 Zusammenfassung . . . . .	107
8.2 Ausblick . . . . .	108
<b>Literaturverzeichnis</b>	<b>113</b>
.1 Tabellen . . . . .	118

# Abbildungsverzeichnis

2.1	Elektromagnetisches Spektrum. . . . .	20
2.2	Aufbau und Anatomie eines Zahns. . . . .	21
2.3	Optische Dämpfung von Kariesläsionen. . . . .	22
2.4	Herkömmliche NIR Bildgebungsverfahren. . . . .	23
2.5	Approximale Kariesläsion eines Zahnes dargestellt mit allen herkömmlichen Nahinfrarot (NIR)-Modalitäten der bildgebenden Verfahren. . . . .	24
2.6	In vivo Aufnahmen der selben approximalen Kariesläsion als NIR- und Röntgenaufnahme. . . . .	25
2.7	In vitro Aufnahmen der okklusalen Zahnoberfläche mit NIR und sichtbarem Licht. . . . .	26
2.8	Polygonnetz eines 3D-Modells in Form eines Zahns. . . . .	28
2.9	Indices für ein Quadrat. . . . .	29
2.10	Beleuchtung von Flächen abhängig von der Richtung der eintreffenden Lichtstrahlen. [GC18a] . . . . .	30
2.11	Berechnung des Normalen-Vektors. [GC18a] . . . . .	30
2.12	Repräsentation der Texturkoordinaten einer Textur. [GC18a] . . . . .	31
2.13	Die verschiedenen Koordinatensysteme der 3D-Computergrafik. . . . .	31
2.14	Funktionsweise einer Videokamera. . . . .	40
2.15	Schematisches Prinzip des Kameramodells. [Sch05] . . . . .	40
2.16	Extrinsische Transformation der Weltkoordinaten in Kamerakoordinaten. [Sch05]	41
2.17	Verschiebung des Kamerahauptpunktes in den Ursprung der diskreten Bildmatrix.	42
2.18	Die Arten der Verzeichnung. [Vol19] . . . . .	44
3.1	Bereitgestelltes 3D-Zahnmodell. . . . .	46
3.2	Auszug aus dem NIR-Datensatz. . . . .	47
4.1	Bestimmung korrespondierender Punkte anhand eines Kalibrierungskörpers. [Sch05]	51
4.2	Erste Iteration zur Generierung der 2D-Aufnahmen für den Vergleich mit der Texturaufnahme. [UPP11]. . . . .	52
4.3	Iterative Annäherung an die Kameraperspektive. [UPP11] . . . . .	53
4.4	Korrespondierende Kanten. [MK99] . . . . .	55
4.5	Das gegebene 3D-Modell des Zahnes projiziert auf eine NIR-Textur. . . . .	59
4.6	Illustriertes Z-Buffer Verfahren für zwei Dreiecke. [19j] . . . . .	61
4.7	Sichtbarkeitsanalyse des Maler Algorithmus. [Han09b] . . . . .	63
4.8	Überlappungsbereiche zweier Texturen. . . . .	64
4.9	Illustration der versetzte Kanten. . . . .	68
4.10	Versatz der Textur an den Kanten von Dreiecken, die von unterschiedlichen Texturen eingefärbt wurden. [NB95] . . . . .	69
4.11	Prinzip des Blending. [LHS00] . . . . .	69

4.12	Mitteln der Textur für zwei angrenzende Dreiecke mit unterschiedlichen Texturen, sodass ein weicher Übergang entsteht. [NB95] . . . . .	70
4.13	Multi-band blending in 3D. [Bau03] . . . . .	71
5.1	Darstellung verschiedener Zähne. [19g] . . . . .	74
5.2	Manuelle Bestimmung der Kameraposition für eine NIR-Aufnahme mit Blender. . . . .	75
5.3	Gemeinsames Histogramm von 8-Bit-Grauwertbildern. . . . .	77
6.1	Gesamtablauf des Programms. . . . .	85
6.2	UML-Klassendiagramm, der für die die Texturierung verantwortlichen Klassen. . . . .	86
6.3	Ablauf der Rasterisierung eines Dreiecks in der Bildebene. [Scr15] . . . . .	91
6.4	Baryzentrische Koordinaten können als Sub-Dreiecke CAP (für u), ABP (für v) und BCP (für w) interpretiert werden. [Scr14] . . . . .	92
6.5	Beschleunigung des Rasterisierungs-Verfahrens durch die Bestimmung einer Bounding-Box. [Scr15] . . . . .	93
6.6	Ergebnis des Texture Mapping für die sichtbaren Vertices zweier Texturen. . . . .	94
7.1	Ergebnis des Texture Mapping für fünf Texturen, die auf ein 3D-Modell mit 20.468 Dreiecken appliziert wurden, dargestellt in Blender. . . . .	100
7.2	Gegenüberstellung der manuell gemappten Textur mit dem Ergebnis . . . . .	100
7.3	Diagramm der Laufzeiten des Programms für zwei verschiedene 3D-Modelle. . . . .	101
7.4	Nicht texturierte Dreiecke, markiert durch eine rote Umrandung. . . . .	103
7.5	Ergebnisse der Texturierung eines 3D-Modells mit 429.668 Dreiecken für verschiedene Z-Buffer-Größen. . . . .	103
7.6	Ergebnis der Texturierung für einen Würfel. . . . .	104
7.7	Versetzte Kanten an den Texturübergängen. . . . .	105
8.1	Tracking mit dem Polaris-System. [19e] . . . . .	109
8.2	Konzeptzeichnung der Kombination von NIR und Optischer Kohärenztomographie (OCT). . . . .	111

# **Tabellenverzeichnis**

5.1 Auszug der Ergebnisse der normalisierten Mutual Information (MI) Berechnung.	79
.1 Alle Ergebnisse der Berechnung der normalisierten MI.	118



# **Verzeichnis der Listings**

5.1	Berechnung der normalisierten MI aus dem gemeinsamen Histogramm zweier Bilddaten. . . . .	78
6.1	Modulübersicht des entwickelten Prototypen. . . . .	85
6.2	XML-Format für die Kameraparameter. . . . .	88
6.3	Datenstruktur des Z-Buffers. . . . .	90
6.4	Überprüfung, ob sich ein Punkt innerhalb eines Dreiecks befindet, anhand von baryzentrischen Koordinaten. . . . .	92
6.5	Auszug aus dem Programmcode, der für die Pixel eines Dreiecks überprüft, ob das Dreieck im Vordergrund liegt und entsprechend den Z-Buffer aktualisiert. . . . .	93
6.6	Auszug aus der exportierten Collada-Datei. . . . .	97



# Abkürzungsverzeichnis

- API** Application Programming Interface. 30
- ASCII** American Standard Code for Information Interchange (ASCII) ist eine 7-Bit-Zeichenkodierung. 35
- BMP** Windows Bitmap. 45
- CLI** Command-line interface. 82
- CMOS** Complementary metal-oxide-semiconductor. 23
- Collada** COLLABorative Design Activity. 37
- CSG** Constructive Solid Geometry. 27
- CT** Computertomografie. 109
- DC** Diagnocam. 23
- FIR** Ferninfrarot. 19
- in vitro** Vorgänge, die außerhalb eines lebenden Organismus stattfinden.. 7
- in vivo** Prozesse, die im lebendigen Organismus ablaufen.. 7
- ISO** International Organization for Standardization. 37
- KLD** Kullback-Leiber Divergenz. 57
- LED** Leuchtdiode. 22
- LMU München** Ludwig-Maximilians-Universität München. 17
- MI** Mutual Information. 9
- MIR** Mittleres Infrarot. 19
- MMI** Maximizing Mutual Information. 56
- NIR** Nahinfrarot. 7
- NumPy** Numerisches Python. 77
- OCT** Optische Kohärenztomografie. 109
- Open Source** Software, deren Quelltext öffentlich ist und von Dritten eingesehen werden kann.. 34
- OpenGL** Programmierschnittstelle (API) zur Entwicklung von 2D- und 3D-Computergrafikanwendungen.. 30
- PyPI** Python Package Index. 81

## Abkürzungsverzeichnis

---

- Quaternion** Zahlensystem mit vier komplexen Einheiten.. 33
- RMS** Root-mean-square. 109
- SLD** Superlumineszenzdiode. 22
- STL** STereoLithography. 35
- TDD** Test-driven Development. 83
- Vertex** Eckpunkt eines grafischen Primitive.. 28
- VR** Virtuelle Realität. 105
- WebGL** JavaScript-Programmierschnittstelle (API) zur Entwicklung von 2D- und 3D-Computergrafikanwendungen, die hardwarebeschleunigt im Webbrowser dargestellt werden können.. 30
- XML** Die Extensible Markup Language (XML) ist ein Datenformat, das für die Darstellung von hierarchisch strukturierter Daten genutzt wird.. 88

# 1 Einleitung

Die frühzeitige Erkennung von Karies ist in der Zahnmedizin essenziell, um betroffene Zähne so minimal-invasiv wie möglich behandeln zu können. Traditionell werden die Zähne visuell oder mithilfe von Röntgenstrahlen untersucht. Aufgrund der Röntgenschutzverordnung dürfen Röntgenstrahlen nicht für routinemäßige Untersuchungen eingesetzt werden, sondern nur dann, wenn eine sogenannte rechtfertigende Indikation vorliegt. Der Grund hierfür ist, dass Röntgenstrahlen potenziell ionisierend wirken und somit bleibende Gewebeschäden verursachen können.

Seit ca. zehn Jahren stehen Diagnoseverfahren zur Verfügung, die anstelle von Röntgenstrahlen Licht verwenden. Die Eindringtiefe von Licht in Zähne ist zwar begrenzt, eine Frühdiagnostik von Schmelzkaries ist jedoch in vielen Fällen möglich. Optische Diagnoseverfahren haben den Vorteil, dass sie beliebig oft angewendet werden können, ohne einen Schaden zu verursachen.

Die Informatik spielt dabei vor allem für die Datenverarbeitung der Bilddaten eine wichtige Rolle und ist für optische Diagnoseverfahren ebenso wie auch für die bildgebenden Verfahren im Allgemeinen unabdingbar.

## 1.1 Motivation

Bei einer zahnärztlichen Vorsorgeuntersuchung beschreiben die Zahnärzte bei einer Sichtprüfung den gesundheitlichen Zustand der Zähne, während die zahnärztlichen Fachangestellten diese Befunde in einem standardisierten Zahnschema dokumentieren. Zahnärztliche Befunde können sowohl analog als auch digital archiviert werden. In jedem Fall stellt diese Dokumentation eine massive Datenreduktion dar, da die zahnmedizinische Fachangestellten die Befunde nicht wortwörtlich, sondern in Form von Kürzeln und Farben festhalten. Beispielsweise wird ein als nicht erhaltungswürdig diagnostizierter Zahn mit dem Buchstaben „x“ und ein kariöser Zahn mit dem Buchstaben „c“ notiert. Eine genaue Begründung, warum der Zahn nicht erhaltungswürdig ist oder eine örtliche Beschreibung, wo sich die Karies befindet und wie fortgeschritten die Erkrankung des Zahnes ist, wird in dem von den Krankenkassen vorgesehenen Befundschema nicht festgehalten. Selbst wenn das Standardbefundsystem um die betroffenen Zahnoberflächen erweitert wird, fehlen genaue Ortsinformationen sowie Aussagen zum Schweregrad der kariösen Erkrankung der Zahnoberfläche.

Ein Problem der rein visuellen Befundung stellen Bereiche dar, die der direkten Sicht nicht zugänglich sind. Das gilt besonders für den sogenannten Approximalbereich, d. h. dem Bereich einer Zahnreihe, an dem zwei Zähne in direktem Kontakt sind. Durch Röntgenaufnahmen ist es zwar möglich, Defekte im Approximalbereich aufzuzeigen, jedoch selbst diese sind aufgrund von Artefakten und dem Informationsverlust, der durch die zweidimensionale Projektion von sich überlagernden Strukturen entsteht, oft nicht eindeutig [AES17].

Eine vielversprechende Alternative zu den zweidimensionalen Röntgenaufnahmen, die ohne die ionisierende Strahlung auskommt, ist die Illumination oder die Transillumination mittels NIR. Dabei wird der Zahn mit nicht sichtbarem Licht in einem Wellenlängenbereich zwischen  $0,78\mu m$  bis  $2,0\mu m$  angestrahlt bzw. durchleuchtet [JHF03]. Hierbei dringt das Licht (abhängig von der Wellenlänge des NIR) nahezu ohne Streuung durch den gesunden Zahnschmelz. Bei kariösen Läsionen wird das Licht aufgrund des erhöhten Porenvolumens, das bei der Auflösung der Zahnhartsubstanz durch Stoffwechselprodukte von Bakterien entsteht, stark gestreut. Somit ergeben sich unterschiedliche optische Eigenschaften, die über eine Nahinfrarotkamera digital als Aufnahme sichtbar gemacht werden können.

Mehrere Aufnahmen mit NIR aus verschiedenen Blickwinkeln haben das Potential, dass die Tiefe und die genaue Position der Karies bestimmt werden können [AES17]. Durch die Möglichkeit, die Aufnahmen aus verschiedenen Blickwinkeln machen zu können, können dabei approximale Kariesläsionen mit hoher Wahrscheinlichkeit und ohne schädigende Nebenwirkungen erkannt werden.

Die Illumination mit NIR ist demnach potentiell eine geeignete und vor allem nichtinvasive Methode, um die Diagnose der Zahnärzte zu unterstützen. Durch eine Archivierung dieser Aufnahmen bei der Diagnose (beispielsweise bei der halbjährlichen Vorsorgeuntersuchung) könnte der Datenreduktion entgegengewirkt werden. Durch die Archivierung wäre es zudem möglich, den Krankheitsverlauf der Karies zu dokumentieren und auf diese Weise die Kariesaktivität zu bestimmen, die ihrerseits Einfluss auf die Wahl des richtigen Zeitpunkts einer invasiven Kariestherapie hat.

Im Rahmen einer Therapie können heute Füllungen und Kronen computergestützt hergestellt werden. Zur Digitalisierung der Zahndaten stehen für diese computergestützte Therapie intraorale 3D-Scanner zur Verfügung, mit denen die Zahnoberflächen digitalisiert und als 3D-Modell ebenfalls

abgespeichert werden können. Durch eine Kombination der NIR-Aufnahmen mit einem solchen erstellten 3D-Modell wäre es zudem möglich, die Vorteile einer dreidimensionalen Darstellung der NIR-Aufnahmen bei der Archivierung zu nutzen.

## 1.2 Ziele der Arbeit

In dieser Arbeit soll gezeigt werden, dass Nahinfrarotaufnahmen als eine geeignete Methode zur Unterstützung der zahnmedizinischen Diagnostik eingesetzt werden könnte. Ziel der Arbeit ist es, eine prototypische Implementierung einer Software für die 3D-Visualisierung von Kariesläsionen aus Nahinfrarotaufnahmen zu entwickeln. Die Software soll als anschauliche Grundlage dienen, um die Hypothese zu fundieren. Dabei soll der aktuelle Stand der Forschung untersucht und reflektiert werden.

Die Arbeit entsteht in Kooperation mit der Ludwig-Maximilians-Universität München (LMU München) und beabsichtigt, deren Ergebnisse und entwickelte Technologien der Forschungsarbeit in diesem Bereich zu nutzen und auf diesen aufzubauen.

Für die Arbeit sollen ausschließlich frei verfügbare Technologien eingesetzt werden.

## 1.3 Inhaltlicher Aufbau der Arbeit

Die Arbeit ist in folgender Weise gegliedert: In Kapitel 2 werden zunächst die Grundlagen für die Entwicklung der Software beschrieben. Hierfür wird zunächst auf die Nahinfrarot-Bildgebung in der Zahnmedizin in Abschnitt 2.1 eingegangen. Anschließend werden die Grundlagen der 3D-Computergrafik in Abschnitt 2.2 erläutert. Insbesondere wird hier auf die Eigenschaft von 3D-Modellen und deren Datenformate eingegangen. Darauf aufbauend wird in Abschnitt 2.3 anhand des Kameramodells erklärt, wie Texturen den Oberflächenpunkten von 3D-Modell zugewiesen werden können.

Anknüpfend an die Grundlagen wird in Kapitel 3 zunächst eine Analyse der Daten in Abschnitt 3.1 durchgeführt. Daraufhin werden die Anforderungen an die Software in Abschnitt 3.2 erläutert. Aus den Anforderungen und den Erkenntnissen, die aus den Grundlagen gewonnen wurden, werden in Abschnitt 3.3 die Aufgaben für die Recherche definiert.

In Kapitel 4 wird der Stand der Technik beschrieben. Zu Beginn des Kapitels werden in Abschnitt 4.1 verschiedene Techniken zur Kamerakalibrierung aufgezeigt. Zudem wird in Abschnitt 4.2 beschrieben, wie mehrere Texturen auf ein 3D-Modell appliziert werden können.

Mit den bisher gewonnenen Erkenntnissen wird in Kapitel 5 die durchgeführte Machbarkeitsanalyse zur Kamerakalibrierung erläutert. Hierfür werden zunächst in Abschnitt 5.1 die bekannten Verfahren zur Kamerakalibrierung analysiert. Daraufhin werden die Ähnlichkeitsmaße für Silhouetten und Kanten in Abschnitt 5.2 sowie für die Mutual Information (MI) in Abschnitt 5.3 untersucht. Die gewonnenen Erkenntnisse und die Konsequenzen werden daraufhin in Abschnitt 5.4 vorgestellt.

## 1 Einleitung

Die praktische Implementierung des Prototyps wird in Kapitel 6 erläutert. Zunächst werden in dem Kapitel die verwendete Software und die verwendeten Softwarekomponenten in Abschnitt 6.1 und Abschnitt 6.2 aufgezeigt. Danach werden die gewählten Maßnahmen zur Qualitätssicherung in Abschnitt 6.3 aufgelistet. Anschließend wird in Abschnitt 6.4 eine Übersicht über das Programm vermittelt. Die Details zur Implementierung werden im Anschluss in Abschnitt 6.5 beschrieben.

Die erzielten Ergebnisse und eine erste Evaluierung des entwickelten Prototyps werden in Kapitel 7 aufgezeigt. Hierbei werden zunächst die Ergebnisse in Abschnitt 7.1 präsentiert. Anschließend wird eine Laufzeitanalyse in Abschnitt 7.2 durchgeführt. Zudem werden in Abschnitt 7.3 auf bekannte Probleme, deren Ursachen und mögliche Lösungsansätze hingewiesen. Zum Schluss wird in Abschnitt 7.4 der durchgeführte Anwendertest erläutert.

Schließlich fasst Kapitel 8 die Erkenntnisse der Arbeit in Abschnitt 8.1 zusammen und stellt Anknüpfungspunkte in Abschnitt 8.2 vor.

## 2 Grundlagen

In diesem Kapitel wird auf die zum weiteren Verständnis notwendigen Grundlagen eingegangen. Hierfür werden zunächst die Charakteristiken und Anwendungsfälle von Nahinfrarot in der Zahnmedizin dargestellt. Anschließend werden die Grundlagen der 3D-Computergrafik aufgezeigt. Abschließend wird der Vorgang des Texture Mapping anhand des Kameramodells erklärt.

### 2.1 Nahinfrarot-Bildgebung in der Zahnmedizin

In der Zahnmedizin werden traditionell Röntgenstrahlen für die Bildgebung bei der Diagnose verwendet. Die Röntgenstrahlen sind jedoch aufgrund der ionisierenden Eigenschaft schädlich und können Gewebe nachhaltig verletzen. Deshalb gewinnen bildgebende Verfahren, die ohne der Röntgenstrahlung auskommen immer mehr an Bedeutung. Eines dieser Diagnoseverfahren verwendet Nahinfrarot für die Bildgebung.

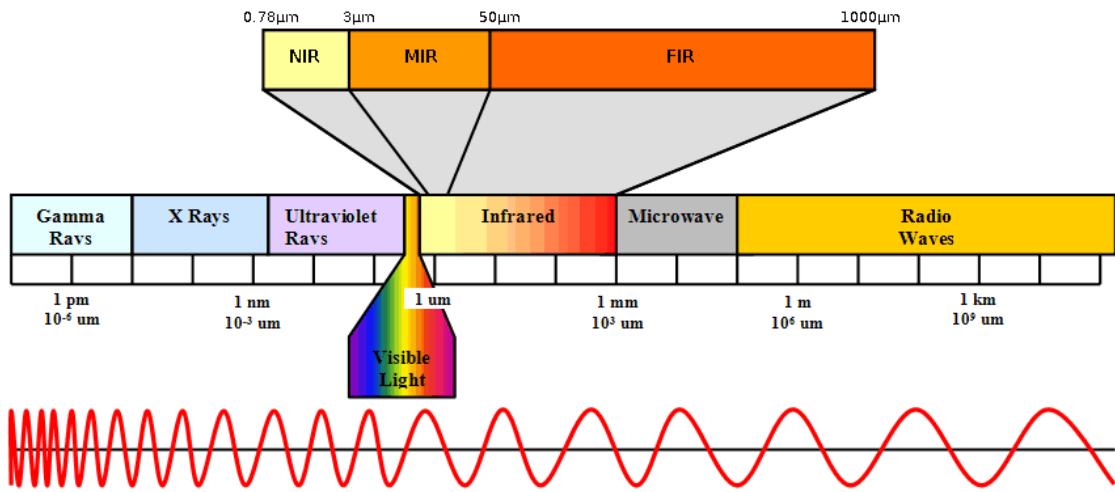
Dieses Verfahren soll in diesem Kapitel untersucht werden. Hierfür wird in diesem Abschnitt zunächst erklärt, was Nahinfrarot ist und welche optischen Eigenschaften es in den Zahnsubstanzen hat. Anschließend wird auf die Grundlagen der NIR-Bildgebung und die wesentlichen Unterschiede zu anderen bildgebenden Verfahren in der Zahnmedizin eingegangen.

#### 2.1.1 Nahinfrarot

Bei NIR handelt es sich, wie bei dem sichtbaren Licht, um elektromagnetische Wellen. Das nicht sichtbare NIR unterscheidet sich dabei aufgrund der unterschiedlichen Wellenlänge vom sichtbaren Licht.

Die Wellenlänge  $\lambda$  berechnet sich aus der Frequenz und der Phasengeschwindigkeit. Unter Frequenz  $f$  versteht man die Anzahl der Schwingungen pro Sekunde. Die Ausbreitungs- oder Phasengeschwindigkeit  $c$  ist die Geschwindigkeit, mit der sich ein monochromatischer Wellenberg bewegt, bzw. ausbreitet [Pri15]. Dabei gilt:  $\lambda = \frac{c}{f}$ . Die Wellenlänge ist demnach der Abstand zweier benachbarter Orte gleicher Phase.

Die Sortierung elektromagnetischer Wellen nach ihrer Wellenlänge  $\lambda$  wird als elektromagnetisches Spektrum bezeichnet. Die für das menschliche Auge nicht sichtbare Infrarotstrahlung (IR-Strahlung) liegt dabei im Spektralbereich zwischen dem sichtbaren Licht ( $\lambda = 0,38 - 0,78\mu m$ ) und den langwelligeren Mikrowellen (siehe Abbildung 2.1) [18a]. Dies entspricht einem Wellenlängenbereich zwischen  $0,78\mu m$  und  $100\mu m$  [18a]. Innerhalb dieses Spektralbereichs des Infrarot unterscheidet man zudem zwischen dem Ferninfrarot (FIR), Mittlerem Infrarot (MIR) und dem NIR.



**Abbildung 2.1:** Elektromagnetisches Spektrum.

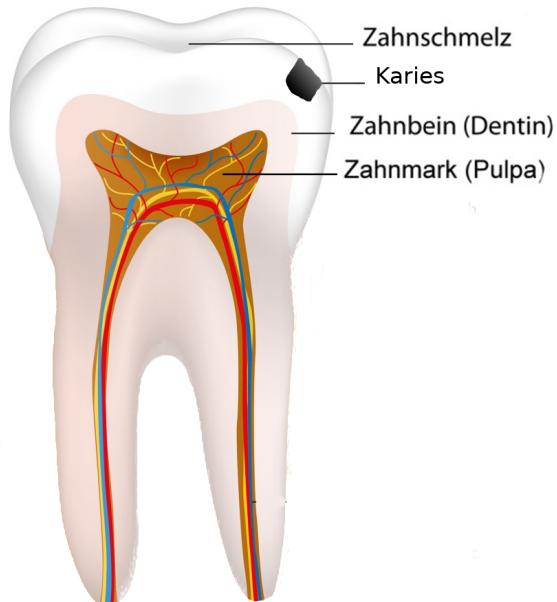
Der Spektralbereich für die jeweilige Infrarotstrahlung (FIR, MIR, NIR) wird von verschiedenen Institutionen unterschiedlich beschrieben. Diese Arbeit richtet sich deshalb an den ISO 20473 Standard der International Organization for Standardization vom April 2007 [18a]. Der ISO 20473 Standard spezifiziert die Einteilung von optischer Strahlung in ein magnetisches Spektrum. Für NIR wurde ein Spektralbereich zwischen  $0,78\mu\text{m}$  und  $3\mu\text{m}$  festgelegt, MIR bewegt sich zwischen  $3,0 - 50\mu\text{m}$  und FIR zwischen  $50 - 100\mu\text{m}$  [18a].

### 2.1.2 Optische Eigenschaften der Zahnsubstanzen im Nahinfrarotbereich

Bei der NIR-Bildgebung werden die optischen Eigenschaften der Zahnhartsubstanzen im NIR-Bereich ausgenutzt, um Kariesläsionen sichtbar zu machen. Um zu verstehen, wie die verschiedenen Kontraste bei der NIR-Aufnahme zustande kommen, muss zunächst der Aufbau und die Anatomie eines Zahnes untersucht werden. Wie in Abbildung 2.2 illustriert besteht ein Zahn aus dem Zahnschmelz, dem Dentin und dem Zahnmark, das zudem die Nerven und Blutgefäße enthält. Die Karies ist eine multifaktoriell bedingte Erkrankung und erzeugt an den betroffenen Stellen eine Demineralisierung der Zahnhartsubstanzen durch die Stoffwechselprodukte von Bakterien. Wird die Karies nicht behandelt, so dringt sie bis in das Zahnmark vor und wirkt somit destruktiv für den betroffenen Zahn.

Betrachtet man die linken zwei NIR-Aufnahmen aus Abbildung 2.5, so erkennt man, dass die verschiedenen Zahnsubstanzen unterschiedliche Kontraste aufweisen. Während das Dentin einen hohen Kontrast aufweist, wird der Zahnschmelz hingegen nahezu transluzent dargestellt. Die Pulpa ist in den Aufnahmen nicht sichtbar. Auffällig sind zudem die Kariesläsionen, die als dunkle Kontraste sichtbar sind.

**Dentin** Das Dentin erscheint in der NIR-Aufnahme als dunkler Kontrast unter dem Zahnschmelz. Der Grund hierfür ist die Streuung, die aufgrund der mikroskopisch kleinen Hohlräume im Dentin, den Dentinkanälchen (Dentintubuli), entsteht und selbst für unterschiedliche Wellenlängen die



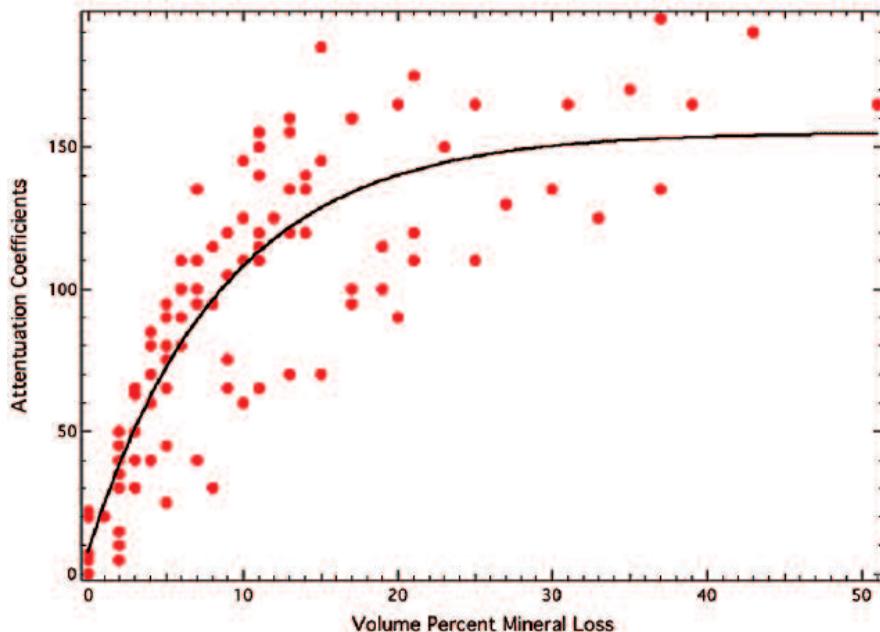
**Abbildung 2.2:** Aufbau und Anatomie eines Zahns.

Strahlung gleichermaßen stark streut [Zij01] [JHJF03]. Die NIR-Strahlen durchdringen demnach das Dentin nicht. Somit ist das Dentin und die darunterliegende Pulpa in der NIR-Aufnahme nicht sichtbar.

**Zahnschmelz** Die Streuung des sichtbaren Lichts für Zahnschmelz ist mit einem Streuungskoeffizienten ( $\mu_s$ ) von  $\mu_s = 60\text{cm}^{-1}$  bei einer Wellenlänge von  $632\text{nm}$  sehr stark. Bei steigender Wellenlänge nimmt der Streuungskoeffizient jedoch ab. Im Nahinfrarotbereich beträgt der Streuungskoeffizient nur noch  $\mu_s = 2 - 3\text{cm}^{-1}$ . Deshalb erscheint der Zahnschmelz bei NIR transluzenter als beim sichtbaren Licht [JHJF03]. Der Zahnschmelz wird daher als heller Kontrast dargestellt. Strukturen, die unter dem Zahnschmelz liegen sind sichtbar.

**Kariesläsionen** Durch die Demineralisierung der Zahnhartsubstanzen, die bei einer Kariesinfektion entstehen, werden die Zahnhartsubstanzen aufgelöst und das Porenvolumen an diesen Stellen erhöht. Strahlung, die in Form von Licht oder NIR auf diese kariöse Läsionen trifft, wird durch die vergrößerten Poren gestreut. Abbildung 2.3 zeigt diese optische Dämpfung im Nahinfrarotbereich bei einer Wellenlänge von  $1310\text{nm}$  in Abhängigkeit des Volumens des Mineralverlusts durch eine Karies.

Der Dämpfungsfaktor ist bei dem demineralisierten Gewebe, das durch die Karies entsteht, 20-50 mal größer als das des gesunden Zahnschmelzes [JHJF03]. Dadurch entsteht ein signifikanter Kontrastunterschied im Vergleich zu dem umliegenden gesunden Gewebe. Bei einer NIR-Aufnahme ist somit der kariöse Bereich als abweichender Kontrast vom Zahnschmelz deutlich auszumachen.



**Abbildung 2.3:** Optische Dämpfung von Kariesläsionen.

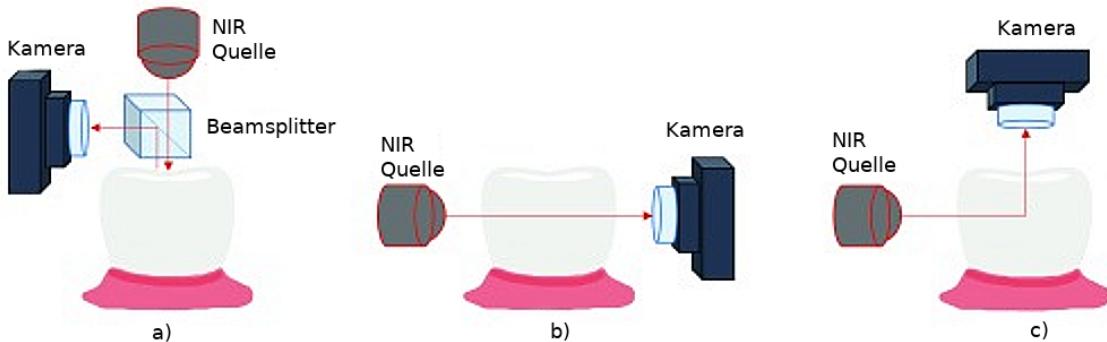
Dargestellt in  $cm^{-1}$  von 100 Punkten aus 10 natürlichen Kariesläsionen des Zahnschmelzes (10 Punkte pro Läsion) bei einem Wellenbereich von  $1310nm$ . Die Kurve beschreibt die beste exponentielle Annäherung an die Dämpfungskoeffizienten und dem prozentualen Volumenverlust durch die Demineralisierung. [JHJF03]

**Absorption der Zahnhartsubstanzen** Die Absorption der NIR-Strahlung und die des sichtbaren Lichts durch die Zahnhartsubstanzen unterscheiden sich nur geringfügig. Der Absorptionskoeffizient ( $\mu_a$ ) für Zahnschmelz beträgt weniger als  $\mu_a = 1cm^{-1}$ . Für Dentin beträgt der Absorptionskoeffizient geringe  $\mu_a = 4cm^{-1}$  [JHJF03]. Die Absorption ist demnach für die NIR-Bildgebung zu vernachlässigen.

### 2.1.3 Bildgebende Verfahren

Die Zahmedizin nutzt die Eigenschaften des NIR, um Kariesläsionen sichtbar zu machen und somit die Diagnose zu unterstützen. Dafür werden spezielle Diagnosewerkzeuge benutzt, deren Aufbau, Funktionsweise und Ergebnisse in diesem Kapitel beschrieben werden.

**Aufbau und Funktionsweise** Der Aufbau der bildgebenden Verfahren, die digitale Nahinfrarotaufnahmen erzeugen, besteht aus einem NIR-Emitter und einer Kamera, die Bilder im NIR-Bereich aufnehmen kann. Der Emittier ist je nach Entfernung zu dem Zahn entweder eine Superlumineszenzdiode (SLD), Leuchtdiode (LED) oder ein Laser [AES17]. SLDs und LEDs werden auf kurze Entferungen verwendet. Laser kommen bei größeren Distanzen zum Einsatz. Bei der NIR-Bildgebung kann entweder die NIR-Strahlung, die vom Zahn reflektiert wird, oder die, die den Zahn durchdringt, von der Kamera erfasst werden.



**Abbildung 2.4:** Herkömmliche NIR Bildgebungsverfahren.

- (a) Das von der Oberfläche des Zahns reflektierte Licht wird über einen Beamsplitter an die Kamera weitergeleitet.
- (b) Transillumination des Zahns mit gegenüberliegender NIR-Lichtquelle und Kamera.
- (c) Okklusale Aufnahme des Zahns mit orthogonal platziert Kamera und NIR-Emitter.

Abbildung 2.4 zeigt den Aufbau herkömmlicher bildgebender Verfahren im NIR. In Abbildung 2.4 (a) wird die, von der Oberfläche des Zahns, reflektierte Strahlung über eine Kamera erfasst. Dabei kommen üblicherweise Strahlungsteiler (Beamsplitter) zum Einsatz.

In Abbildung 2.4 (b) und (c) wird der Zahn durchleuchtet (Transillumination) und die nicht gestreute oder absorbierte Strahlung durch eine Kamera erfasst. Bei der Transillumination wird die Kamera meistens gegenüber des Emitters platziert (Abbildung 2.4 (b)), um den Strahlungsinput der Kamera möglichst zu maximieren. Jedoch ist die gegenüberliegende Positionierung der Kamera zum Emitter nicht zwingend notwendig. Bei einer okklusalen (auf die Kaufläche des Zahnes bezogene) Aufnahme wird die Kamera beispielsweise okklusal und die NIR Lichtquelle orthogonal zur Kamera positioniert (Abbildung 2.4 (c)).

**NIR-Diagnosewerkzeuge in der Zahmedizin** Für die Zahnmedizin wurden bereits einige bildgebende Diagnosewerkzeuge entwickelt, die NIR nutzen, beispielsweise die Diagnocam (DC) der Firma KaVo oder die VistaCam der Firma Dürr Dental. Bei der DC durchleuchten zwei Laserdioden, die sich an den Armen des zangenförmigen Handstücks befinden, den Zahn aus bukkaler (zu der Wange hin) und lingualer (zu der Zunge hin) Richtung. Ein Complementary metal-oxide-semiconductor (CMOS) Sensor nimmt dann das Bild aus okklusaler Richtung auf [LKHL18].

Die VistaCam emittiert das NIR und erfasst dessen Reflektion direkt am Ende des Handstücks. Zwei NIR Laserdioden befinden sich orthogonal zum Handstück und benachbart zum zentral gelegenen CMOS Sensor. Die Reflektion des emittierten NIR wird dann vom CMOS Sensor verarbeitet und als Bild dargestellt. Die Bilder können bei beiden Systemen in Echtzeit über einen Monitor dargestellt und gespeichert werden. Während die DC detailliertere Bilder liefert, können mit der VistaCam Bilder aus jedem beliebigen Winkel aufgenommen werden.

**Ergebnisse** Abbildung 2.5 zeigt die Ergebnisse der verschiedenen beschriebenen Modalitäten. Während die linken zwei Aufnahmen durch Transillumination entstanden, wurden im rechten Bild die Reflektionen des NIR erfasst. Deutlich zu erkennen sind die Kariesläsionen in jeder Aufnahme.



**Abbildung 2.5:** Approximale Kariesläsion eines Zahnes dargestellt mit allen herkömmlichen NIR-Modalitäten der bildgebenden Verfahren.

Die Läsion wurde in der Darstellung eingekreist. (A) Approximale Transilluminationsaufnahme. (B) Okklusale Transilluminationsaufnahme. (C) Okklusus Reflektionsaufnahme.

**Artefakte bei der NIR Bildgebung** Bei der NIR-Bildgebung kommt es häufig zu über- oder unterbelichteten Bereichen in den Aufnahmen. Gründe dafür sind intensive Lichtquellen in der Umgebung oder metallische Restaurationen im Zahn, die zu Reflexionen führen, sowie direktes Licht, das auf den NIR Sensor aufgrund von falscher Ausrichtung des Emitters zu dem Sensor trifft. Zudem können Schmelzbildungsstörungen wie Hypomineralisation (im Volksmund Kreidezähne genannt) oder die Morphologie der okklusalen Oberfläche des Zahns die Bildgebung negativ beeinflussen. [LKHL18]

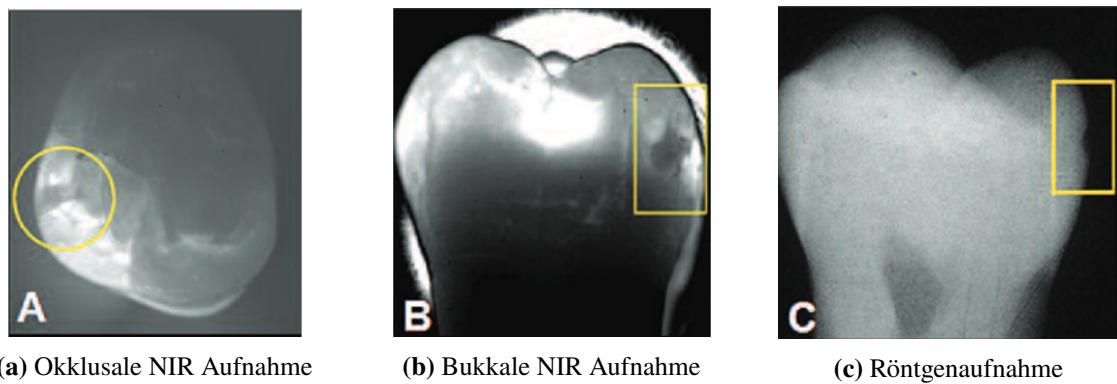
#### 2.1.4 Vergleich zu anderen diagnostischen bildgebenden Verfahren

Die NIR-Bildgebung liefert oftmals detailliertere Aufnahmen und somit mehr Informationen als herkömmliche bildgebende Verfahren, wie die Röntgenaufnahme oder die Aufnahme mit sichtbaren Licht. Zudem ist die NIR-Bildgebung, im Gegensatz zu den Röntgenaufnahmen, ein nichtinvasives Verfahren, das ohne gesundheitsschädlicher Strahlung auskommt. Dieses Kapitel stellt die verschiedenen bildgebenden Verfahren gegenüber.

**Vergleich von NIR- und Röntgenaufnahmen** Auf NIR-Aufnahmen ist der klinische Zustand der Zähne gut sichtbar. Häufig sind Karies, Demineralisierung, Frakturen und Risse im Zahn, die in NIR-Aufnahmen sichtbar sind, in Röntgenaufnahmen kaum oder gar nicht zu erkennen [JHJF03] [AES17].

Abbildung 2.6 zeigt den Unterschied einer NIR- und Röntgenaufnahme. Zu sehen ist eine Approximalkaries, die in den Bildern gelb eingekreist, beziehungsweise durch ein Rechteck markiert wurde. Wohingegen die Karies bei der NIR-Aufnahme als deutlicher Kontrastunterschied dargestellt wird, lässt sich die Karies bei der Röntgenaufnahme nur durch ein geschultes Auge diagnostizieren.

Ein weiterer Vorteil der NIR-Bildgebung ist die Möglichkeit (je nach Modalität), Aufnahmen aus verschiedenen Blickwinkeln zu erstellen. Dies ermöglicht eine stereoskopische Sicht und somit mehr Ortsinformationen über eine Karies [JHJF03]. In Abbildung 2.6 kann beispielsweise durch eine Kombination der NIR-Aufnahmen aus okklusaler und bukkaler Richtung die genaue Position der Karies bestimmt werden.



**Abbildung 2.6:** In vivo Aufnahmen der selben approximalen Kariesläsion als NIR- und Röntgenaufnahme.

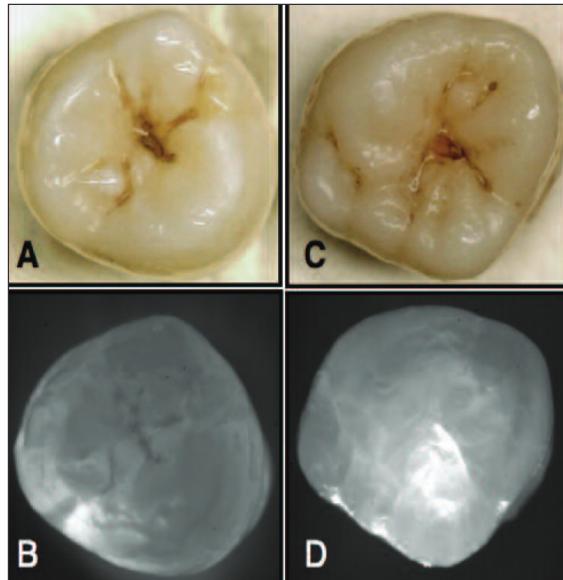
Während bei den herkömmlichen bildgebenden Verfahren mittels NIR nur Aufnahmen von einem Zahn oder den Zwischenräumen zweier Zähne auf einmal erstellt werden können, liefert die Röntgenmethode eine schnellere Methode, viele Zähne auf einmal zu untersuchen [JHJF03]. Des Weiteren liefert die Pulpa beim Röntgen eine Referenz zur Tiefe einer Kariesläsion. Dies ist bei der NIR-Bildgebung aufgrund der Streuung der NIR-Strahlung des Dentin durch die Dentinkanälchen nicht möglich (siehe Abschnitt 2.1.2) [ZB93] [LKHL18].

**Vergleich von NIR-Aufnahmen und Aufnahmen mit sichtbarem Licht** Der Hauptunterschied zwischen einer NIR-Aufnahme und einer Aufnahme, die mit sichtbarem Licht erstellt wurde, ist der signifikant höhere Kontrastunterschied zwischen dem gesunden Zahnschmelz und der Kariesläsion bei der NIR-Aufnahme. Der Grund hierfür ist die geringere Streuung der NIR-Strahlung im Zahnschmelz, wie in Abschnitt 2.1.2 beschrieben. Des Weiteren kann der Zahnschmelz deshalb mit NIR besser durchleuchtet werden, als mit sichtbarem Licht. Defekte und Strukturen weiter unterhalb der Oberfläche des Zahnes können deshalb beim sichtbaren Licht oftmals nicht ausgemacht werden.

Bei der Sichtprüfung oder bei Aufnahmen mit Dentalkameras bei sichtbarem Licht ist es zudem schwierig, zwischen Plaque und Demineralisierung zu unterscheiden. Oftmals ist es deshalb erforderlich eine Zahnreinigung mit einem Schleifmittel durchzuführen, um eine zuverlässige Diagnose erstellen zu können [JHJF03]. Bei der NIR Aufnahme hingegen kann gut zwischen einer Demineralisierung durch eine Karies und Plaque unterschieden werden, da Plaque auf dem NIR Bild nicht sichtbar ist (siehe Abbildung 2.7).

### 2.1.5 Fazit

Die optischen Eigenschaften der Zahnhartsubstanzen im NIR-Bereich ermöglichen es, Defekte wie Kariesläsionen auch unter der Zahnoberfläche zu visualisieren. Vor allem aufgrund der nichtinvasiven Eigenschaft der NIR-Bildgebung und dem höheren Informationsgehalt, als beispielsweise bei einer visuellen Sichtprüfung bei sichtbarem Licht, stellt die NIR-Bildgebung eine geeignete Alternative zu herkömmlichen bildgebenden Verfahren in der Zahnmedizin dar.



**Abbildung 2.7:** In vitro Aufnahmen der okklusalen Zahnoberfläche mit NIR und sichtbarem Licht.

(A, C) Aufnahmen zweier Zähne mit sichtbarem Licht. Es sind deutliche verfärbte Flecken zu erkennen. (B) Erkennbare Demineralisierung des linken Zahns durch eine NIR Aufnahme (dunkler Kontrast). (C) Die NIR Aufnahme zeigt, dass der Zahn, trotz der Verfärbung auf dem Bild des sichtbaren Lichts, gesund ist. Die Verfärbung entstand demnach nicht durch Demineralisierung sondern durch Plaque.

## 2.2 3D-Computergrafik

Die LMU München stellt für die Implementierung einer dreidimensionalen Darstellung von Kariesläsionen aus NIR-Aufnahmen einen Datensatz, bestehend aus einem 3D-Modell eines Zahnes und mehreren NIR-Aufnahmen desselben Zahnes, für diese Arbeit bereit. Um zu verstehen, wie eine solche dreidimensionale Darstellung mit den gegebenen Daten ermöglicht werden kann, werden zunächst die Grundlagen der 3D-Computergrafik, auf denen die entwickelte Softwarelösung basiert, in diesem Kapitel beschrieben.

### 2.2.1 Einführung

Die 3D-Computergrafik beschäftigt sich mit den Algorithmen, Methoden und Techniken, um eine computergenerierte, dreidimensionale Darstellung aus einem geometrischen Datensatz zu erzeugen. Physikalische Ansätze simulieren dabei reale Begebenheiten, wie Beleuchtung, Materialbegebenheiten und physikalische Einflüsse wie Schwerkrafteinwirkungen bzw. generell Krafteinwirkungen. Das Ziel der 3D-Computergrafik ist die Darstellung einer komplexen 3D-Szene als zweidimensionales Bild. Die Konvertierung des geometrischen Datensatzes in ein zweidimensionales Bild wird als Rendering bzw. Bildsynthese bezeichnet [GCPB14].

Der Prozess der Erstellung einer 3D-Computergrafik kann dabei in drei Phasen abstrahiert werden:

- 3D-Modellierung

- Szenen Layout
- Rendering

**3D-Modellierung** Unter der 3D-Modellierung versteht man die Definition der Form und Farbe eines physikalischen Objektes. Im Allgemeinen kann zwischen der Repräsentation der Oberfläche und des Volumens dieses Objektes unterschieden werden [GCPB14]:

- **Oberflächenbasiert:** Bei diesem Verfahren wird die Oberfläche des Objektes beispielsweise in Form von Polygonnetzen modelliert.
- **Volumenbasiert:** Die Repräsentation des 3D-Modells erfolgt in Form eines Volumens. Voxel und Constructive Solid Geometry (CSG) werden im Allgemeinen für die Darstellung von Volumendaten in der 3D-Computergrafik benutzt (für mehr Informationen zu Voxel und CSG siehe [GCPB14]).

Die Art der Repräsentation hängt dabei von der Methode der Erstellung des geometrischen Datensatzes und des Anwendungsfalls ab. Am häufigsten kommt jedoch die Darstellung der Oberfläche in Form eines Polygonnetzes zum Einsatz.

Die Erstellung des 3D-Modells muss dabei nicht zwangsläufig manuell erfolgen, wie es die Bezeichnung der 3D-Modellierung eigentlich suggeriert, sondern kann auch mit Hilfe von 3D-Scannern erfolgen, welche die Oberflächendaten eines realen Objekts bspw. durch Laserstrahlen erfassen.

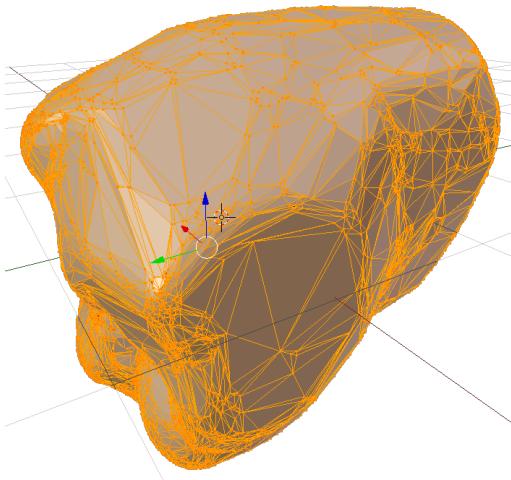
Der Modellierungsprozess umfasst dabei nicht nur die Erfassung oder Modellierung der Form eines Objektes, sondern auch die Bestimmung der Materialeigenschaften und der Zuweisung von Texturen.

**Szenen Layout** Nach der 3D-Modellierung erfolgt die Einrichtung des Szenen Layouts. Das Szenen Layout umfasst die Anordnung der 3D-Modelle, Lichtquellen, Kameras und anderer Entitäten, die später für den Gesamteindruck des gerenderten Bildes beitragen.

**Rendering** Wie bereits erwähnt, ist das Rendering der letzte Schritt, um das zweidimensionale Bild zu erstellen, das in der Szene definiert wurde. Dabei werden alle Daten, die bei der 3D-Modellierung und dem Szenen Layout definiert wurden, verarbeitet, um Effekte, wie Materialeigenschaften, Beleuchtung, usw. in der zweidimensionalen Aufnahme zu visualisieren.

Die Kameraparameter, die in der Szene definiert wurden, geben dabei vor, aus welcher Perspektive die Szene gerendert werden soll. Verbunden damit muss beim Rendering eine Sichtbarkeitsanalyse erfolgen, um auszuschließen, dass Objekte dargestellt werden, die von einem anderen Objekt teilweise oder ganz verdeckt werden.

Für das fotorealistische Rendering gibt es viele verschiedene Ansätze, um die Belichtung einer Szene (bspw. mittels Ray-Tracing) oder die Sichtbarkeit eines Objektes (bspw. mittels Z-Buffer Verfahren oder Ray-Casting) möglichst effizient und akkurat zu berechnen.



**Abbildung 2.8:** Polygonnetz eines 3D-Modells in Form eines Zahns.

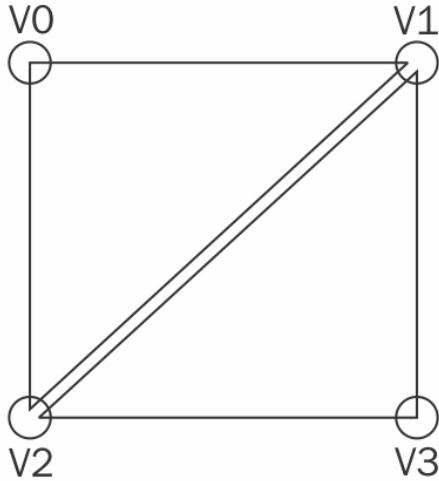
### 2.2.2 Datenstruktur eines 3D-Modells

Wie bereits erwähnt, wird die Form eines 3D-Modells im Allgemeinen anhand von Koordinatenpunkten bestimmt. Die Form kommt zustande, indem die Punkte miteinander verbunden werden und somit Kanten bzw. Flächen der Oberfläche definieren. Man spricht hierbei auch von einem sogenannten Polygonnetz (siehe Abbildung 2.8). Die Datenstruktur dieser Polygonnetze setzt sich aus Vertices, Indices, Normalen und Texturkoordinaten zusammen und bestimmt somit das Aussehen des 3D-Modells.

**Vertices** Ein Polygonnetz besteht aus vielen dreidimensionalen Koordinatenpunkten. Die Koordinatenpunkte bestimmen jeweils einen Eckpunkt eines grafischen Primitivs. Diese Primitive sind beispielsweise Punkte, Linien, Quadrate, Dreiecke, etc. - wobei im Allgemeinen Dreiecke das gebräuchlichste Primitiv ist. Die Eckpunkte des Primitivs werden als Vertex bezeichnet. Ein Vertex wird dabei meist in Form eines Vektors mit drei Komponenten für die drei Koordinaten dargestellt. Die Vertices werden bspw. in Form einer Matrix an den Renderer übergeben, der aus der Datenfolge die Form des 3D-Modell interpretieren kann.

**Indices** Indices beschreiben, wie Vertices miteinander verknüpft werden, um ein Primitiv zu erzeugen. Sie reduzieren die Anzahl der nötigen Vertices, indem die Vertices über Indizes referenziert werden. Das Indexing wird in Abbildung 2.9 illustriert. Die Grafik zeigt ein Quadrat, dessen Fläche mittels zweier Dreiecke dargestellt wird. Ohne Indices wäre eine Beschreibung der Fläche anhand von Vertices folgendermaßen:

```
vertices = [
    [0, 0, 0], #v0
    [1, 0, 0], #v1
    [0, 1, 0], #v2
    [0, 1, 0], #v2
    [1, 0, 0], #v1
    [1, 1, 0] #v3
```



**Abbildung 2.9:** Indices für ein Quadrat.

]

Anstatt alle Koordinaten der Dreiecke als Vertex abzubilden, wird nun für jeden Eckpunkt ein Index in Form einer Nummer zugewiesen, der auf die Koordinaten des Vertex verweist. Koordinaten, die den selben Eckpunkt beschreiben, werden zusammengefasst. Somit kann das Quadrat aus Abbildung 2.9 mit nur vier Vertices beschreiben werden. Die Indices definieren dabei die Dreiecke:

```

vertices = [
    [0, 0, 0], #v0
    [1, 0, 0], #v1
    [0, 1, 0], #v2
    [1, 1, 0] #v3
]

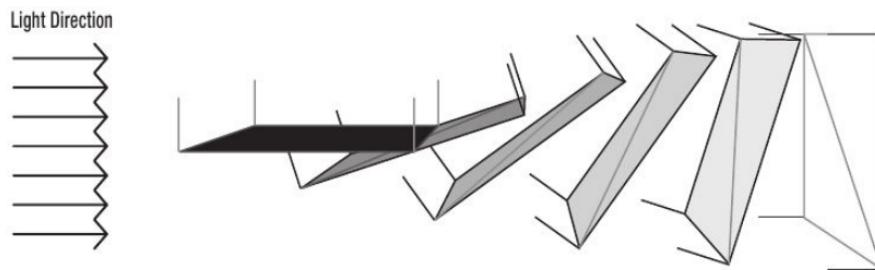
indices = [0, 1, 2, 2, 1, 3]

```

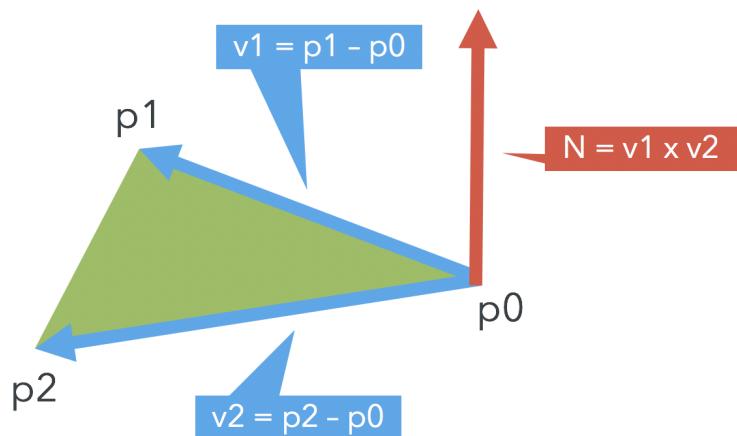
Für komplexe 3D-Modelle, die viele Oberflächenkoordinaten besitzen, kann somit die nötige Datenmenge reduziert werden.

**Normalen** Normalen sind Vektoren, die senkrecht zu einer Fläche stehen und die Orientierung einer Fläche repräsentieren. Normalen werden hauptsächlich für die Berechnung der Beleuchtung benötigt, indem der Winkel zwischen dem eintreffenden Lichtstrahl auf die Oberfläche und der Normalen dieser Fläche bestimmt wird und somit indirekt die Helligkeit berechnet wird, wie in Abbildung 2.10 dargestellt.

Die Normale berechnet sich aus einem gegebenen Dreieck mit den drei Vertices  $p_0, p_1$  und  $p_2$ , indem zunächst der Vektor  $v_1 = p_1 - p_0$  und der Vektor  $v_2 = p_2 - p_0$  ermittelt wird. Anschließend wird die Normale aus dem Kreuzprodukt der beiden Vektoren  $N = v_1 \times v_2$  berechnet. In Abbildung 2.11 wird die Berechnung grafisch illustriert. [GC18a]



**Abbildung 2.10:** Beleuchtung von Flächen abhängig von der Richtung der eintreffenden Lichtstrahlen. [GC18a]



**Abbildung 2.11:** Berechnung des Normalen-Vektors. [GC18a]

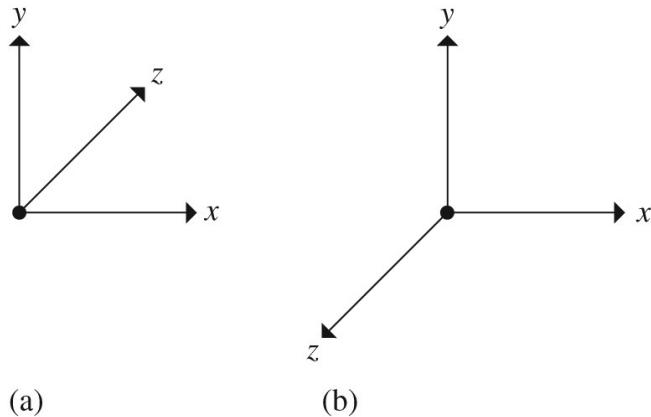
Für Vertices, die für zwei oder mehrere Dreiecke gleich sind, wird die Normale aktualisiert, indem die Vektorsumme der Normale mit den anderen Normalen gebildet und anschließend normalisiert wird.

**Texturkoordinaten** Eine Texturkoordinate beschreibt einen Punkt in der zweidimensionalen Bildebene einer Textur. Sie gibt an, welcher Punkt einer Textur auf welchem Vertex abgebildet wird. Die Texturkoordinaten bestehen aus einem zweidimensionalen float Vektor, der sich im Wertebereich zwischen 0 und 1 bewegt. Die Texturkoordinate (0, 0) befindet sich dabei im Allgemeinen in der linken, oberen Ecke einer Textur und (1, 1) in der rechten, unteren Ecke, wie in Abbildung 2.12 zu sehen. Ein Vertex, der zur Mitte einer Textur zugeordnet werden soll, erhält demnach die Koordinate (0.5, 0.5).

Die Texturkoordinaten haben eine symbolische Repräsentation, die jedoch unter den Grafiksuites und Grafik Application Programming Interfaces (APIs) nicht konsistent ist. Während OpenGL und WebGL die Texturkoordinaten mit den Buchstaben  $s$  für die x-Koordinate und  $t$  für die y-Koordinate bezeichnen, verwenden die meisten anderen Grafiksuites und Grafik-APIs die Buchstabenbezeichnung  $u$  und  $v$  für die x- und y-Koordinate. Deshalb wird oft der Begriff UV-Koordinaten als Synonym für die Texturkoordinaten verwendet. [GC18a]



**Abbildung 2.12:** Repräsentation der Texturkoordinaten einer Textur. [GC18a]



**Abbildung 2.13:** Die verschiedenen Koordinatensysteme der 3D-Computergrafik.

- a) Linkshändiges Koordinatensystem, bei dem die *z*-Achse vom Betrachter wegführt.
- b) Rechtshändiges Koordinatensystem, bei dem die *z*-Achse zum Betrachter hin zeigt.

### 2.2.3 Koordinatensysteme

Für die Definition eines 3D-Modells werden im Allgemeinen dreidimensionale Koordinaten bestimmt, welche Oberflächenpunkte und somit die Form des Modells definieren. Die Koordinaten wären jedoch ohne ein entsprechendes Koordinatensystem, das den Ursprung und die *x*-, *y*- und *z*-Achse des Raumes bestimmt, bedeutungslos. Es gibt zwei unterschiedliche Arten, wie die dreidimensionale Achsen angeordnet werden können, wie in Abbildung 2.13 illustriert. Sie werden als linkshändiges, bzw. rechtshändiges Koordinatensystem bezeichnet, da sie entweder mit den Fingern der linken oder der rechten Hand bestimmt werden können, indem der Daumen in Richtung der *x*-Achse, der Zeigefinger in Richtung der *y*-Achse und der Mittelfinger in Richtung der *z*-Achse zeigt. Stehen die *x*- und *y*-Achsen senkrecht zueinander, so zeigt die *z*-Achse (oder der Mittelfinger) des linkshändigen Koordinatensystems vom Betrachter weg, während die *z*-Achse des rechtshändigen Koordinatensystems zum Betrachter hin zeigt [PJH16].

Die Wahl eines der Koordinatensysteme ist zwar beliebig, hat aber Auswirkungen auf die geometrischen Operationen innerhalb des Systems. Besondere Vorsicht ist geboten, wenn verschiedene Systeme kombiniert werden, beispielsweise bei einem Datenimport eines 3D-Modells mit einem anderen Koordinatensystem, als das eigene. Die beiden Koordinatensysteme müssen in diesem Fall angeglichen werden, indem ein Koordinatensystem in das Andere überführt wird.

#### 2.2.4 Transformationen

Wie bereits erwähnt, werden die Vertices, wie auch die Normalen, im Allgemeinen als Vektoren gespeichert. Jedoch ist es unwahrscheinlich, dass diese Vektoren innerhalb einer Grafikanwendung statisch bleiben. Das 3D-Modell wird in der Szene bewegt, gedreht, vergrößert oder verkleinert. Solche Operationen werden Transformationen genannt. Formal wird das Bewegen eines Objekts als Translation, das Drehen als Rotation und das Ändern der Größe als Skalierung bezeichnet.

**Homogene Koordinaten** Rotationen und Skalierungen können im dreidimensionalen Raum durch eine Multiplikation des Vertex mit einer  $3 \times 3$ -Matrix durchgeführt werden. Bei einer Translation ist dies mit dreidimensionalen kartesischen Koordinaten nicht möglich. Um die Translation im dreidimensionalen Raum als Matrixmultiplikation zu schreiben, und somit auch eine Verknüpfung durch Hintereinanderausführung aller Operationen zu ermöglichen, wurde eine mathematisch elegante Lösung gefunden - die homogenen Koordinaten.

Bei den homogenen Koordinaten wird jeder Vektor um eine zusätzliche Komponente  $h \in \mathbb{R}$  erweitert. Dabei kann  $h$  einen beliebigen Wert ungleich Null annehmen. Für  $h = 1$  sind die ersten drei Komponenten in der Darstellung eines Punktes mit homogenen Koordinaten mit den kartesischen Koordinaten identisch. Die kartesischen Koordinaten berechnen sich durch eine Division der x-, y- und z-Werte durch die zusätzliche Komponente: [BB15]

$$\begin{pmatrix} x \\ y \\ z \\ h \end{pmatrix} \rightarrow \begin{pmatrix} x/h \\ y/h \\ z/h \end{pmatrix} \quad (2.1)$$

Eine  $3 \times 3$ -Translation wird unter Verwendung von homogenen Koordinaten um eine Zeile und Spalte erweitert, sodass eine  $4 \times 4$ -Matrix entsteht:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ h \end{pmatrix} \quad (2.2)$$

Mit Hilfe der homogenen Koordinaten ist es nunmehr möglich, alle Operationen als  $4 \times 4$ -Matrizen darzustellen und zu verknüpfen.

**Translation** Die Translation verschiebt einen Vektor entlang der x-,y- oder z-Achse. Die Translationsmatrix ist definiert als:

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

Eine Verschiebung des homogenen Vektors  $(x, y, z, 1)^T$  um den Vektor  $(t_x, t_y, t_z, 1)^T$  würde demnach folgendermaßen berechnet werden:

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{pmatrix} \quad (2.4)$$

**Skalierung** Eine Skalierung eines Vertex wird durch die Skalierungsmatrix ermöglicht:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

Die Werte entlang der Diagonalen repräsentieren den Skalierungsfaktor in x-,y- und z-Richtung. Die Skalierung eines homogenen Vektors  $(x, y, z, 1)^T$  um den Skalierungsfaktor  $(s_x, s_y, s_z, 1)^T$  erfolgt demnach durch folgende Gleichung:

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \cdot s_x \\ y \cdot s_y \\ z \cdot s_z \\ 1 \end{pmatrix} \quad (2.6)$$

**Rotation** Rotationen können entweder mit Euler-Winkel oder mit Quaternionen durchgeführt werden. Bei der Rotation für den Euler-Winkel  $\omega$  setzt sich eine Rotation aus drei Rotationsmatrizen (eine für jede Achse) zusammen:

$$\begin{aligned}
 R_x &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\omega) & \sin(\omega) & 0 \\ 0 & -\sin(\omega) & \cos(\omega) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 R_y &= \begin{bmatrix} \cos(\omega) & 0 & -\sin(\omega) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\omega) & 0 & \cos(\omega) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 R_z &= \begin{bmatrix} \cos(\omega) & \sin(\omega) & 0 & 0 \\ -\sin(\omega) & \cos(\omega) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
 \end{aligned} \tag{2.7}$$

$$R_{\text{Euler}} = R_x \cdot R_y \cdot R_z \tag{2.8}$$

Ein Quaternion ist eine Erweiterung der komplexen Zahlen. Ähnlich wie bei den komplexen Zahlen, die aus einem Real- und Imaginärteil ( $Z = a \cdot 1 + b \cdot i$ ) bestehen, wird ein Quaternion aus drei Imaginärteilen und einem Realteil konstruiert [Xu09]:

$$q = a \cdot 1 + b \cdot i + c \cdot j + d \cdot k \tag{2.9}$$

Die Rotation kann mit einem Quaternion so interpretiert werden, dass die imaginären Elemente  $b$ ,  $c$ , und  $d$  einen Vektor definieren, um den mit  $a$  gedreht wird.

Die Rotationsmatrix aus Quaternionen lautet folgendermaßen [Xu09]:

$$R_{\text{quat}} = \begin{bmatrix} 1 - 2(c^2 + d^2) & 2(bc - ad) & 2(bd + ac) & 0 \\ 2(bc + ad) & 1 - 2(d^2 + b^2) & 2(cd - ab) & 0 \\ 2(bd - ac) & 2(cd + ab) & 1 - 2(b^2 + c^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

Die Vorteile der Verwendung von Quaternionen sind [Xu09]:

- Die Rotation erfolgt direkt um die gewünschte Achse.
- Bei Euler-Rotationen ist die Reihenfolge der einzelnen Drehungen nicht beliebig. Je nachdem, welche Drehung um welche Achse zuerst ausgeführt wird, erhält man andere Ergebnisse. Dies ist bei Quaternionen nicht der Fall.
- Es tritt keine kardanische Blockade (engl Gimbal Lock) auf. Beim Gimbal Lock liegen zwei der drei möglichen Rotationsachsen parallel zueinander, wodurch das System einen Freiheitsgrad weniger hat.

## 2.2.5 Dateiformate

Für das Abspeichern von 3D-Modellen (und komplette Szenen) existiert eine Vielzahl an verschiedenen Dateiformaten. Die große Anzahl an Dateiformaten kommt nicht zuletzt deshalb zustande, da viele Grafiksuits und Grafik-APIs proprietäre Dateiformate definiert haben. Im Gegensatz zu neutralen bzw. Open Source Dateiformaten können diese Dateiformate nicht beliebig zwischen Grafikprogrammen geteilt werden. In diesem Kapitel werden die bekanntesten Formate beschrieben: STL, Wavefront und Collada.

### 2.2.5.1 STL

STereoLithography (STL) ist ein freies Datenformat, in dem das 3D-Modell nur durch Vertices und Normalen definiert wird. Zusatzinformationen wie UV-Koordinaten sind in diesem Format nicht enthalten.

**ASCII-STL** Eine STL-Datei in ASCII-Code ist folgendermaßen aufgebaut:

```
solid name
facet normal n1 n2 n3
  outer loop
    vertex p1x p1y p1z
    vertex p2x p2y p2z
    vertex p3x p3y p3z
  endloop
endfacet
endsolid name
```

Der Eintrag *name* steht dabei für den Dateinamen. Innerhalb von *facet* bis *endfacet* wird ein Dreieck anhand von Vertices definiert. *n<sub>i</sub>* gibt dabei den Normalenvektor an [18c].

**Binär-STL** Für 3D-Modelle, die viele Dreiecke besitzen, führt das STL-Dateiformat mit ASCII Kodierung zu großen Daten. Deshalb existiert eine kompaktere Binär-Version:

```
UINT8[80] - Header
UINT32 - Number of triangles
foreach triangle
  REAL32[3] - Normal vector
  REAL32[3] - Vertex 1
  REAL32[3] - Vertex 2
  REAL32[3] - Vertex 3
  UINT16 - Attribute byte count
end
```

Der Header weist 80 Bytes auf, wird jedoch von den meisten STL-Readern ignoriert. Anschließend wird über ein 32 Bit Wert die Gesamtanzahl der Dreiecke angegeben, gefolgt von den eigentlichen Dreiecksdaten mit drei 32-Bit float Werten für die Normale und neun 32-Bit float Werten für die drei Vertices. [18c]

**Vor- und Nachteile** Wie bereits erwähnt, kann über das STL-Format keine zusätzliche Information wie beispielsweise die Farbe oder Textur des 3D-Modells mit abgespeichert werden. Jedoch bietet diese Einschränkung einen Vorteil, falls das 3D-Modell keine zusätzlichen Informationen enthalten soll, da die STL-Datei in diesem Falle weniger Speicher verbraucht, als andere Dateiformate und durch die Einfachheit des Datenformats schneller verarbeitet werden kann. [19a]

### 2.2.5.2 Wavefront

Über die Jahre entwickelte sich das Wavefront Dateiformat zu einem Standard um geometrische Daten von 3D-Modellen auszutauschen. Das Wavefront Format besteht aus zwei individuellen Dateitypen: .obj und .mtl.

Während die OBJ-Datei alle geometrischen Eigenschaften eines Objektes, wie die Vertices, Normalen und Texturkoordinaten enthält, beherbergt die MTL-Datei alle Materialdaten (mtl von engl. material template library). Die Materialdaten können dabei Textur-Dateinamen, Informationen zur Beleuchtung oder andere materialspezifische Parameter sein.

**OBJ-Datei** Das OBJ-Dateiformat enthält eine Reihe an Bezeichnungen für Attribute, um ein 3D-Modell zu definieren. In dieser Arbeit werden nur die wichtigsten dieser Attribute erklärt (für mehr Informationen siehe [19h]). Ein Beispiel für die Definition eines Würfels mit dem OBJ-Dateiformat sieht folgendermaßen aus:

```
mtllib cube.mtl
o Cube
v 1.0 1.0 -1.0
v 1.0 -1.0 -1.0
v -1.0 -1.0 -1.0
v -1.0 1.0 -1.0
v 1.0 1.0 1.0
v 1.0 -1.0 1.0
v -1.0 -1.0 1.0
v -1.0 1.0 1.0
vt 0.0 0.0
vt 1.0 0.0
vt 1.0 1.0
vt 0.0 1.0
vn 0.0 1.0 0.0
vn -1.0 0.0 0.0
vn 0.0 -1.0 0.0
vn 1.0 0.0 0.0
vn 1.0 0.0 0.0
vn 0.0 0.0 1.0
vn 0.0 0.0 -1.0
usemtl Material
f 5/1/1 1/2/1 4/3/1
f 5/1/1 4/3/1 8/4/1
f 3/1/2 7/2/2 8/3/2
f 3/1/2 8/3/2 4/4/2
f 2/1/3 6/2/3 3/4/3
f 6/2/3 7/3/3 3/4/3
```

```
f 1/1/4 5/2/4 2/4/4
f 5/2/5 6/3/5 2/4/5
f 5/1/6 8/2/6 6/4/6
f 8/2/6 7/3/6 6/4/6
f 1/1/7 2/2/7 3/3/7
f 1/1/7 3/3/7 4/4/7
```

Für jedes Attribut ( $o, v, vt, vn, \dots$ ) folgt der Wert dieses Attributs. Dabei werden die Attribute folgendermaßen interpretiert [Mar12]:

- mtlLib: Repräsentiert einen Verweis auf die MTL-Datei.
- o: Objektdeklaration
- v: Vertex
- vt: Texturkoordinate
- vn: Normale
- usemtl: Definiert, welches Material benutzt werden soll.
- f: Stellt die Flächen der Dreiecke dar mit jeweils drei Vertex-, Textur- und Normalen-Indizes. Sie agieren wie Indices und bestimmen, welche Vertices ein Dreieck bilden und welche Normalen, sowie Texturkoordinaten zugehörig sind.

**Vor- und Nachteile** Wie bereits erwähnt, ist das Wavefront Format eines der beliebtesten Austauschformate für 3D-Modelle und wird von vielen Grafikprogrammen unterstützt. Zudem unterstützt das Dateiformat, im Gegensatz zum STL-Format, auch das Speichern von Farb- und Texturinformationen. [19a]

Ein Nachteil von Wavefront ist die Komplexität des Formats. Ein weiterer Nachteil ist, dass es zwar eine Binär-Version des Dateiformats gibt, diese jedoch proprietär ist. [19a]

### 2.2.5.3 Collada

COLLAborative Design Activity (Collada) ist ein XML-basiertes Dateiformat, das von Sony Computer Entertainment und der Khronos Group (ebenfalls Entwickler von OpenGL, WebGL, etc.) entwickelt wurde und Open Source bereit gestellt wird. Collada gehört mittlerweile mit Wavefront zu den populärsten Dateiformaten für den Austausch von 3D-Modellen. Die Intention hinter der Einführung von Collada war es, einen Standard für 3D-Dateiformate zu definieren. 2013 wurde Collada von der International Organization for Standardization (ISO) als öffentlich nutzbares Dateiformat aufgenommen. Daher wird das Dateiformat von einer Vielzahl an Grafikprogrammen unterstützt. Die Endung des Dateiformates ist .dae.

Eine Collada-Datei ist aufgrund der XML-Struktur sehr umfangreich, weshalb in dieser Arbeit nicht auf das gesamte Format eingegangen wird (für mehr Informationen siehe [14]). Ein einfaches Dreieck kann mit Collada folgendermaßen geschrieben werden:

## 2 Grundlagen

---

```
<source id=test1>
  <float_array name="values" count="9">
    1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0
  </float_array>
  <technique_common>
    <accessor source="#values" count="3" stride="3">
      <param name="X" type="float"/>
      <param name="Y" type="float"/>
      <param name="Z" type="float"/>
    </accessor>
  </technique_common>
</source>
<triangles count="1">
  <input semantic="POSITION" source="#test1" offset="0"/>
  <p>0 1 2</p>
</triangles>
```

Das Datenformat ist sehr intuitiv zu lesen. Über `<source id=test1>` wird eine neue Datenquelle mit der ID "test1" angelegt. Über `<float_array name="values" count="9">` werden neun Koordinaten als float Werte festgelegt. Das Array hat dabei die Bezeichnung "values". Der Ausdruck `<accessor source="# values" count="3" stride="3">` teilt die Werte der vorher definierten Datenquelle in die x-,y- und z-Koordinate ein und bestimmt so die drei Vertices des Dreiecks. Die Vertices werden schließlich anhand der ID „test1“ als Dreieck, mit der Direktive `<input semantic="POSITION" source= "#test1" offset="0"/>` zusammengefasst. [14]

Der Ausdruck `<p>0 1 2</p>` definiert die Indices und gibt somit an, dass das Dreieck von den Vertices an der Stelle 0, 1 und 2 aufgespannt wird. Wären in dem Collada-Format zusätzlich Normalen definiert, bestünde die Liste der Indices aus doppelt so vielen Einträgen und müsste paarweise interpretiert werden. Beispielsweise würde man die Sequenz `<p>0 0 1 1 2 2</p>` für Vertices und Normalen so interpretieren, dass der Vertex an der Stelle 0 die Normale an der Stelle 0 erhält, der Vertex an Stelle 1 die Normale an der Stelle 1, usw. [14]

**Vor- und Nachteile** Das Collada-Format wird von vielen Grafikprogrammen unterstützt und ist durch die XML-Struktur intuitiv zu interpretieren. Zudem kann eine Collada-Datei aufgrund der XML-Struktur durch ein Collada XML Schema validiert werden. Das Format ermöglicht das Abspeichern von Vertices, Texturkoordinaten, Normalen, etc. sowie Materialeigenschaften. Zusätzlich ist Collada in der Lage, Animationen abzuspeichern. [19a]

Die Speichergröße lässt sich im Vergleich zu STL und Wavefront nicht durch eine Binär-Version komprimieren, wodurch die Collada-Dateien mehr Speicherplatz verbrauchen, als die beschriebenen Konkurrenz-Dateiformate.

## 2.3 Texture Mapping

In diesem Abschnitt wird zunächst eine Einführung in das Texture Mapping gegeben. Anschließend wird das Kameramodell und die zugehörigen Parameter erklärt. Abschließend wird beschrieben, wie die Texturekoordinaten anhand des Kameramodells den Vertices zugeordnet werden können.

### 2.3.1 Einführung ins Texture Mapping

Die Zuweisung der Texturkoordinaten zu den Vertices wird als Texture Mapping oder UV-Mapping bezeichnet. Mit Hilfe des Texture Mapping ist es demnach möglich, die Farben der Oberfläche des 3D-Modells zu bestimmen, die es von der Textur erhält.

In der Praxis wird üblicherweise die Textur an das 3D-Modell angepasst, indem die Oberfläche des 3D-Modells auf eine 2D-Fläche „ausgebreitet“ wird. Diese Methode wird auch als „Unwrapping“ bezeichnet. Das Unwrapping wird dadurch ermöglicht, dass die Polygone des 3D-Modells jeweils schon eine 2D-Fläche beschreiben und somit auf eine 2D-Fläche ausgebrettet werden können. Das ausgebretete Gittermodell des 3D-Modells dient als Vorlage für die Texturierung. Die Textur wird demnach üblicherweise erst nach dem Unwrapping erstellt und auf das 2D-Gittermodell angepasst.

Ist die Textur bereits vorhanden (beispielsweise in Form einer oder mehreren echten Fotoaufnahmen des 3D-Modells), gestaltet sich die Zuweisung der Texturkoordinaten zum 3D-Modell schwieriger. Das Gittermodell müsste manuell auf die Textur angepasst werden, was bei komplexen 3D-Modellen nahezu unmöglich ist. Deshalb werden im Allgemeinen die Texturkoordinaten den Vertices direkt zugewiesen.

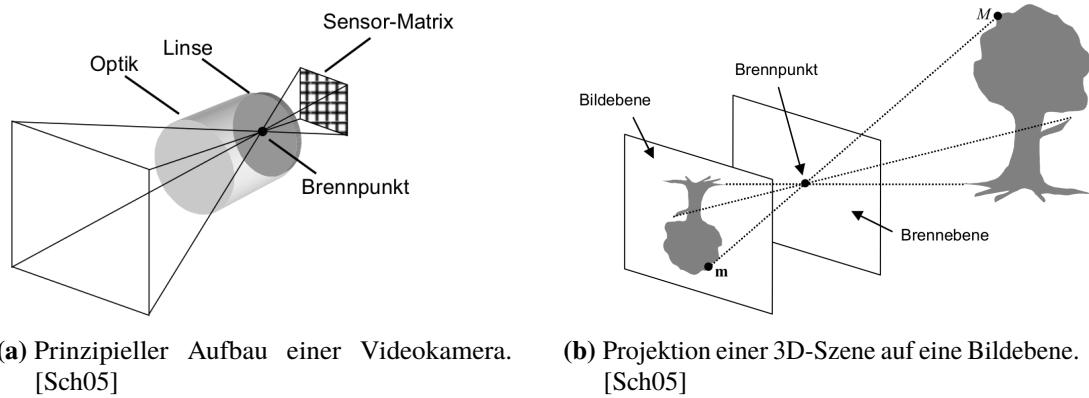
Egal ob die Texturkoordinaten direkt zugewiesen wurden oder ob zuvor ein Gittermodell erstellt wurde; in beiden Fällen muss eine Umrechnung der Texturkoordinaten in die Bilddaten oder anders herum existieren. Diese Rückrechnung erfolgt dabei mit den Kameraparametern des Kameramodells, das im nächsten Abschnitt beschrieben wird.

### 2.3.2 Das Kameramodell

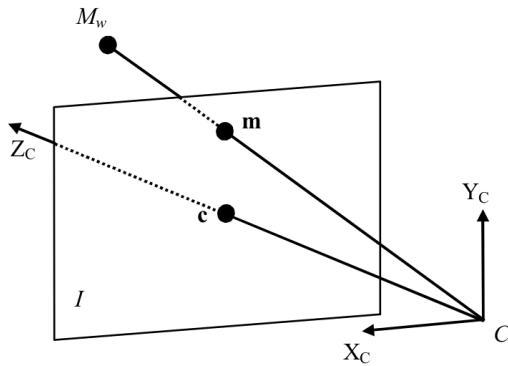
In diesem Abschnitt wird auf die Funktionsweise und die Parameter des Kameramodells eingegangen. Die Erkenntnisse aus diesem Abschnitt werden benötigt, um zu verstehen wie das Texture Mapping funktioniert.

#### 2.3.2.1 Funktionsweise

Abbildung 2.14a zeigt den prinzipiellen Aufbau einer Videokamera. Das einfallende Licht wird über die Optik, welche die Linse enthält, durch deren Brennpunkt auf eine Sensor-Matrix abgebildet. Der Brennpunkt ist der Ort, an dem die parallel einfallenden Lichtstrahlen durch die Linse auf einen gemeinsamen Punkt fokussiert werden. Hinter dem Brennpunkt werden alle Raumpunkte über die Brennebene auf die Bildebene und somit auf die Sensor-Matrix abgebildet. Dadurch erfährt das Bild durch diese Projektion eine Punktspiegelung am Brennpunkt (vergleiche Abbildung 2.14b). Auf der Sensor-Matrix befinden sich Bildpunkte, die das optische Signal in ein elektrisches Signal umwandeln. Durch eine Quantisierung des elektrischen Signals entsteht dadurch für jedes Pixel ein digitaler Wert. Die Anzahl der Pixel wird als örtliche Auflösung bezeichnet und reicht von Standard-TV-Auflösung (720x576) bis hin zu hochauflösendem HD (1920x1080) und Ultra-HD beziehungsweise 4K-Aufnahmen (4096x2048). [Sch05]



**Abbildung 2.14:** Funktionsweise einer Videokamera.

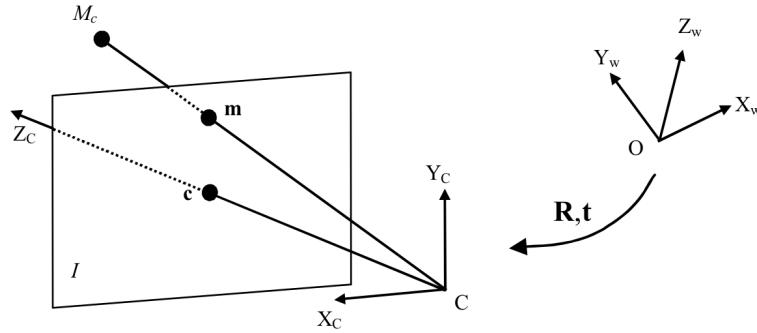


**Abbildung 2.15:** Schematisches Prinzip des Kameramodells. [Sch05]

Als vereinfachtes Modell wird grundsätzlich die Bildebene vor das Projektionszentrum gesetzt [Sch05] [KSS16] [KB17]. Abbildung 2.15 zeigt den geometrischen Zusammenhang eines Punktes im dreidimensionalen Raum  $M_w = (X_w, Y_w, Z_w)^T$  und seiner Projektion  $m = (u, v)^T$  auf die Bildebene  $I$ . Im Gegensatz zu einer realen Videokamera befindet sich dabei die Projektionsebene hinter der Bildebene. Für die mathematischen Herleitungen spielt diese Vereinfachung keine Rolle, außer dass die x- und y-Achsen der Koordinatensysteme gespiegelt werden müssen [Sch05]. Die optische Abbildung wird in eine externe und eine interne Transformation unterteilt [Sch05].

### 2.3.2.2 Extrinsische Transformation

Für die optische Abbildung müssen die Koordinaten des abzubildenden Objekts, die sich in einem Weltkoordinatensystem befinden, in das Kamerakoordinatensystem überführt werden. Wie in Abbildung 2.15 ersichtlich, besitzt jede Kamera ein eigenes Koordinatensystem  $(X_c, Y_c, Z_c)$ , dessen Ursprung im Brennpunkt  $C$  liegt. Eine euklidische Transformation überführt nun durch Translation  $t$  und Rotation  $R$  das Weltkoordinatensystem mit dem Ursprung  $O$  in das Kamerakoordinatensystem mit dem Ursprung  $C$  (siehe Abbildung 2.16). Die optische Achse bezeichnet die Symmetriearchse der Kamera und zeigt in Z-Richtung. Sie schneidet die Bildebene im Kamerahauptpunkt  $c$ .



**Abbildung 2.16:** Extrinsische Transformation der Weltkoordinaten in Kamerakoordinaten. [Sch05]

Für die Umrechnung eines 3D-Punktes  $M_w$  von Weltkoordinaten in Kamerakoordinaten gilt somit Gleichung (2.11). Erweitert man die Formel mit homogenen Koordinaten, erhält man Gleichung (2.12). Dabei bezeichnet  $\tilde{M}$  einen homogenen Vektor, der durch eine zusätzliche Komponente erweitert wurde:  $\tilde{M}_c = [M_c^T, 1]^T$ .

$$M_c = \mathbf{R}M_w + \mathbf{t} \quad (2.11)$$

$$\tilde{M}_c = \mathbf{D}\tilde{M}_w \quad \text{mit} \quad \mathbf{D} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_3^T & 1 \end{bmatrix} \quad \text{und} \quad \mathbf{0}_3 = [0, 0, 0]^T \quad (2.12)$$

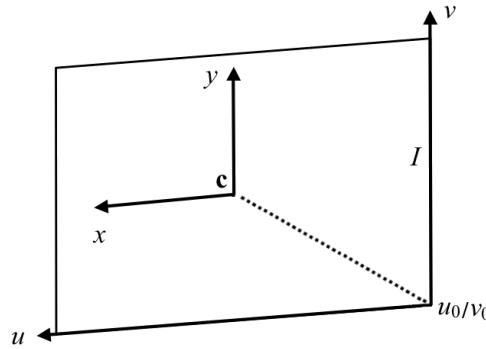
Die Matrix  $D$  enthält die externen Parameter der Kamera und wird somit als extrinsische Matrix bezeichnet. Insgesamt hält die Matrix sechs Freiheitsgrade, die aus den drei Freiheitsgraden der Drehmatrix  $R$  und den drei Freiheitsgraden der Translation  $t$  zusammengesetzt sind.

### 2.3.2.3 Intrinsische Transformation

Nachdem das Weltkoordinatensystem auf das Kamerakoordinatensystem transformiert wurde, ist der nächste Schritt die perspektivische Transformation des 3D-Punktes  $M_w$  mit  $(X_c, Y_c, Z_c)$  auf einen Punkt  $m$  mit  $(x, y)$  der Bildebene (vergleiche Abbildung 2.15 und Abbildung 2.16). Diese Transformation ist in Gleichung (2.13) beschrieben, die aus den Gleichungen der Zentralprojektionen hervor geht. Dabei bezeichnet  $f$  die Brennweite. Die Brennweite ist der Abstand der Bildebene zum Brennpunkt.

$$\begin{bmatrix} U \\ V \\ S \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad \text{mit} \quad x = U/S, \quad y = V/S \quad \text{für} \quad S \neq 0 \quad (2.13)$$

Für die Darstellung im projektiven Raum ergibt sich demnach Gleichung (2.14):



**Abbildung 2.17:** Verschiebung des Kamerahauptpunktes in den Ursprung der diskreten Bildmatrix.

$$s\tilde{m}' = P' \tilde{M}_c \quad \text{mit} \quad P' = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad \text{und} \quad s = S \quad (2.14)$$

Somit transformiert sich der 3D-Punkt in homogenen Koordinaten über die sogenannte perspektivische Projektionsmatrix  $P'$  auf den Punkt  $\tilde{m}'$  in der Bildebene. Aus Gleichung (2.14) und Gleichung (2.12) ergibt sich dann die Transformation von 3D-Weltkoordinaten in 2D-Sensordaten:

$$s\tilde{m}' = P' D \tilde{M}_w \quad (2.15)$$

Der Punkt  $m'$  wird über Gleichung (2.15) metrisch, beispielsweise in mm, berechnet. Für die Umrechnung der Sensorkoordinaten in Bildkoordinaten muss eine Transformation, bestehend aus einer horizontalen und vertikalen Skalierung  $k_u, k_v$ , vorgenommen werden. Die diskrete Bildmatrix wird im Allgemeinen beginnend in einer Ecke durchnummeriert. Deshalb muss zudem eine Verschiebung des Kamerahauptpunktes  $c$  in den Ursprung des Bildkoordinatensystems  $(u_0, v_0)$  erfolgen (siehe Abbildung 2.17). Eine schiefsymmetrische Ausrichtung der Achsen des Bildsensors wird mit dem sogenannten *skew*-Parameter  $s$  beschrieben. Dieser kann jedoch aufgrund der hohen Genauigkeit in der Fertigung von modernen Bildsensoren vernachlässigt werden und wird deshalb  $s = 0$  gesetzt [Sch05]. Zusammengefasst ergibt sich aus den Parametern die Transformationsmatrix  $H$  für die Umrechnung des Punktes von Sensorkoordinaten in Bildkoordinaten:

$$\tilde{m} = H \tilde{m}' \quad \text{mit} \quad H = \begin{bmatrix} k_u & s & u_0 \\ 0 & k_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.16)$$

Führt man die perspektivische Transformation aus Gleichung (2.15) und die Umrechnung der Sensorkoordinaten in Bildkoordinaten aus Gleichung (2.16) zusammen, ergibt sich Gleichung (2.17). Die Matrix  $A$  hält somit die internen Parameter der Kamera und wird deshalb als intrinsische Matrix bezeichnet.

$$s\tilde{m} = HP'\tilde{M}_c = A\tilde{M}_c \quad \text{mit} \quad A = HP' = \begin{bmatrix} \alpha_u & s & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{und} \quad \alpha_u = fk_u, \quad \alpha_v = fk_v \quad (2.17)$$

### 2.3.2.4 Gleichung der perspektivischen Projektion

Die vollständige Transformation eines 3D-Punktes in Weltkoordinaten auf seine Abbildung in der Bildebene in Kamerakoordinaten lautet demnach:

$$s\tilde{m} = A\tilde{M}_c = A \cdot D\tilde{M}_w = A[Rt]\tilde{M}_w = P\tilde{M}_w \quad \text{mit} \quad P = A[Rt] \quad (2.18)$$

Die Matrix  $P$  wird als allgemeine Projektionsmatrix der perspektivischen Projektion bezeichnet und weist elf Freiheitsgrade auf. Diese sind zusammengesetzt aus den Freiheitsgraden der extrinsischen und der intrinsischen Matrix.

Durch eine Division durch die dritte Komponente erhält man die Gleichungen (2.19) und (2.20) der perspektivischen Projektion, für die  $u$  und  $v$  Bildkoordinaten. In Gleichung (2.19) und Gleichung (2.20) stellen  $q_{ij}$  die Komponenten der Projektionsmatrix  $P$  dar.

$$u = \frac{q_{11}X_w + q_{12}Y_w + q_{13}Z_w + q_{14}}{q_{31}X_w + q_{32}Y_w + q_{33}Z_w + q_{34}} \quad (2.19)$$

$$v = \frac{q_{21}X_w + q_{22}Y_w + q_{23}Z_w + q_{24}}{q_{31}X_w + q_{32}Y_w + q_{33}Z_w + q_{34}} \quad (2.20)$$

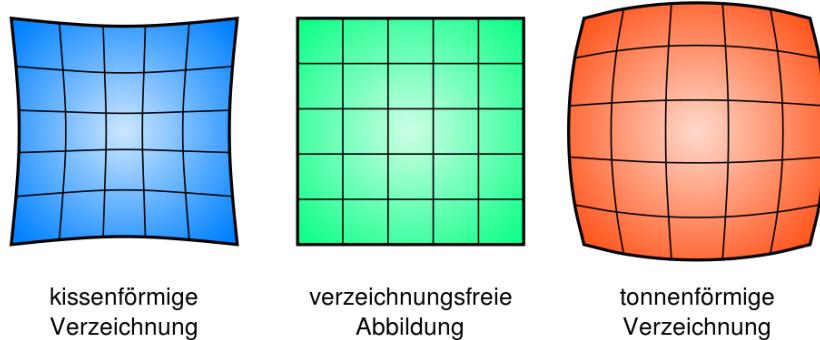
Die Division durch die dritte Komponente eliminiert den Freiheitsgrad des Skalierungsfaktors im projektiven Raum. Das Ergebnis ist eine eindeutige Darstellung der  $u$  und  $v$  Bildkoordinaten in Gleichung (2.19) und Gleichung (2.20).

### 2.3.2.5 Berücksichtigung von Verzeichnungen

In der Optik kann es zu Abbildungsfehlern, den sogenannten Aberrationen kommen. Dies sind Abweichungen von der idealen optischen Abbildung. Die am häufigsten auftretende Aberration ist die Verzeichnung.

Die Verzeichnung ist ein geometrischer Abbildungsfehler, der besonders bei kleiner Brennweite auftritt. Dabei handelt es sich um eine nichtlineare Verzerrung, die mit dem zunehmenden Radialen Abstand zum Brennpunkt auftritt [Sch05]. Somit erfolgen nichtlineare Abbildungen auf die Bildebene. Gerade Kanten werden z.B. somit gekrümmmt dargestellt. Insgesamt gibt es zwei Arten von Verzeichnung: die kissenförmige Verzeichnung und die tonnenförmige Verzeichnung (siehe Abbildung 2.18).

Durch eine Einführung einer nichtlinearen Transformation der  $u$  und  $v$  Koordinaten in die verzerrten Koordinaten  $u_d, v_d$  kann diesem Effekt entgegen gewirkt werden [Sch05].



**Abbildung 2.18:** Die Arten der Verzeichnung. [Vol19]

### 2.3.3 Zuweisung der Texturkoordinaten

Die Zuweisung der Texturkoordinaten erfolgt mit Hilfe der perspektivischen Projektion aus Gleichung (2.18). Sind die Kameraparameter, mit der eine Aufnahme gemacht wurde bekannt, so können Punkte im dreidimensionalen Raum auf die Bildebene zurückgerechnet werden. Für ein Dreieck können so z.B. die drei Vertices auf die Bildebene zurückgerechnet werden. Durch eine Division durch die Dritte Komponente erhält man die genauen Bildkoordinaten  $u$  und  $v$ , wie in Gleichung (2.19) und Gleichung (2.20) beschrieben.

Das Ergebnis sind die Pixel, die jeweils die Ecken des Dreiecks definieren. Um letztendlich die Texturkoordinaten für die Texturierung zu erhalten, müssen die Bildpunkte lediglich noch durch Auflösung der Textur dividiert werden. Die  $u$  Bildkoordinate wird dabei durch die Breite und die  $v$  Bildkoordinate durch die Höhe in Pixel der Textur dividiert. Das Ergebnis sind die Texturkoordinaten, die sich im Bereich zwischen 0 und 1 bewegen, wie in Abschnitt 3.1.1 beschrieben. Somit kann für jedes Primitiv eines 3D-Modells die zugehörigen Texturkoordinaten bestimmt und zurückgerechnet werden, was eine vollständige Texturierung des 3D-Modells ermöglicht.

# 3 Analyse

Wie bereits erwähnt, stellt die LMU München für eine prototypische Implementierung der in dieser Arbeit entwickelten Software die Daten bereit. Diese Daten sollen verwendet werden, um das gesetzte Ziel der Arbeit, die 3D-Visualisierung von Kariesläsionen, zu erreichen.

Nachdem die Grundlagen erarbeitet wurden, können in diesem Kapitel, durch eine Analyse der Daten, erste Erkenntnisse gewonnen werden. Hierfür werden zunächst die Daten und die Anforderungen analysiert. Die im vorherigen Kapitel beschriebenen Grundlagen helfen dabei, durch eine Gegenüberstellung der Ergebnisse der Analyse der Daten und der Anforderungen, erste Erkenntnisse zu gewinnen. Diese Erkenntnisse sollen als Orientierung für die Recherche zum Stand der Technik dienen.

## 3.1 Datenanalyse

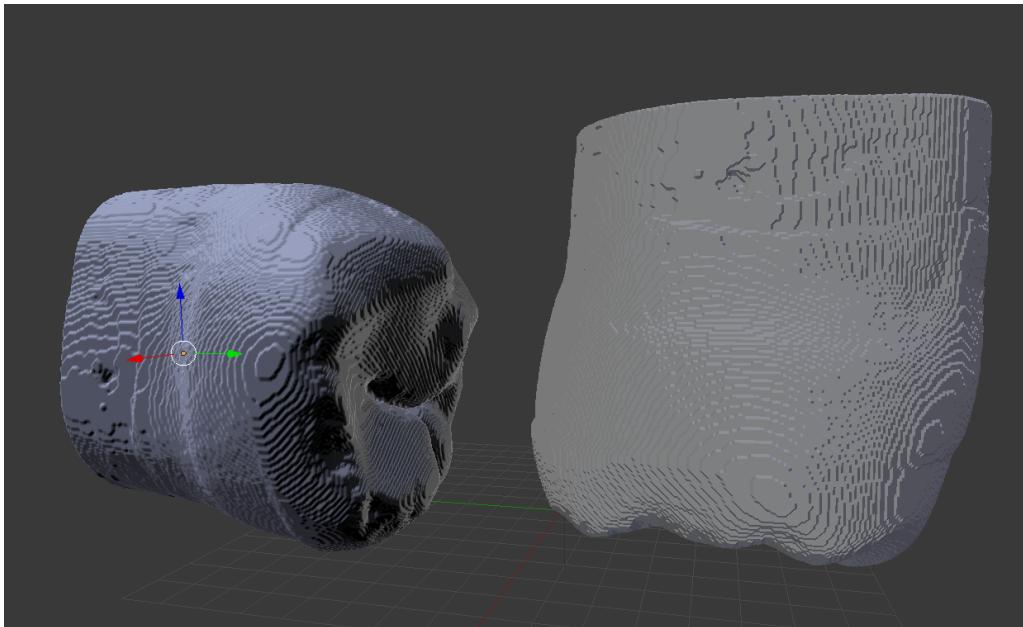
Wie bereits erwähnt, stellt die LMU München einen Datensatz, bestehend aus einem 3D-Modell eines Zahnes und mehreren NIR-Aufnahmen desselben Zahnes, für diese Arbeit bereit. Um die Anforderungen an die zu entwickelnde Software definieren zu können, werden diese Daten zunächst in diesem Kapitel analysiert.

### 3.1.1 3D-Modell

Zunächst werden die Daten des 3D-Modells analysiert. Das 3D-Modell liegt sowohl im STL (.stl) als auch im Wavefront (.obj) Format vor, wobei es sowohl eine Binäre STL als auch eine normale STL gibt. Die Daten des 3D-Modells sind dabei für alle Formate (.stl und .obj) gleich. Die Formate unterscheiden sich hauptsächlich im Speicherverbrauch. Folgende Auflistung gibt Auskunft über diesen:

- **STL:** 113,0 MB
- **Binary STL:** 21,5 MB
- **Wavefront:** 14,4 MB

Das 3D-Modell des Zahns ist in Abbildung 3.1 zu sehen. Auffällig hierbei ist die Oberfläche des Zahnes, die aufgrund der Datenerfassungsmethode nicht glatt, sondern stufenförmig dargestellt wird. Zudem ist die Ausrichtung der 3D-Modelle je nach Dateiformat unterschiedlich.



**Abbildung 3.1:** Bereitgestelltes 3D-Zahnmodell.  
Links: Import des Wavefront Modells; Rechts: Import des STL Modells.

#### 3.1.2 NIR-Daten

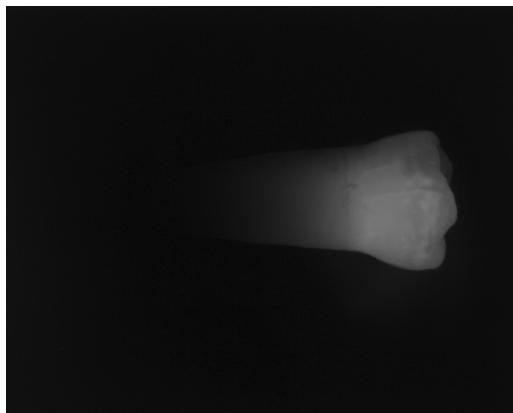
Die NIR-Aufnahmen wurden von der gleichen Zahnprobe erstellt, für die auch das 3D-Modell erstellt wurde. Dabei handelt es sich um eine Aufnahme, die durch die Reflektion der NIR-Strahlen in den Zahnsubstanzen erzeugt wurde (siehe Abschnitt 2.1.3). Der Datensatz besteht aus 22 verschiedenen Aufnahmen mit jeweils einer Auflösung von 1280x1024 Pixeln. Dabei wurden 16 Aufnahmen von der Seite, zwei von oben (Okklusalaufnahme) und vier von schräg oben aufgenommen. Das Datenformat der Aufnahmen ist Windows Bitmap (BMP). Die Kameraparameter, von der aus die Aufnahmen getätigt wurden, sind dabei nicht bekannt - man spricht daher davon, dass die Aufnahmen „unkalibriert“ sind.

Ein Auszug aus dem Datensatz ist in Abbildung 3.2 sichtbar. Auf den ersten Blick auffällig sind die starken Unterschiede in der Belichtung der Aufnahme und die unscharfen bzw. verwischten Ränder.

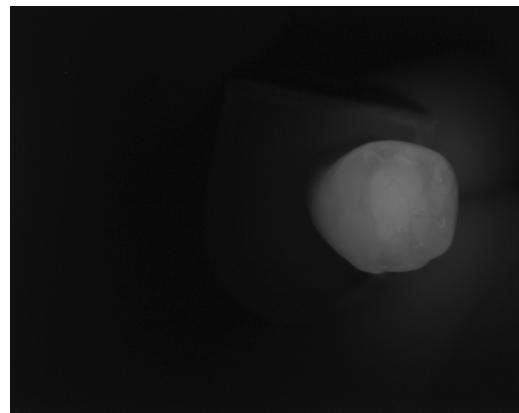
## 3.2 Anforderungsanalyse

Nachdem die Daten untersucht wurden, können die Anforderungen an den, in dieser Arbeit entwickelten Software-Prototypen, durch eine Reflektion der Motivation unter Berücksichtigung der Erkenntnisse der Grundlagen und der Datenanalyse, bestimmt werden.

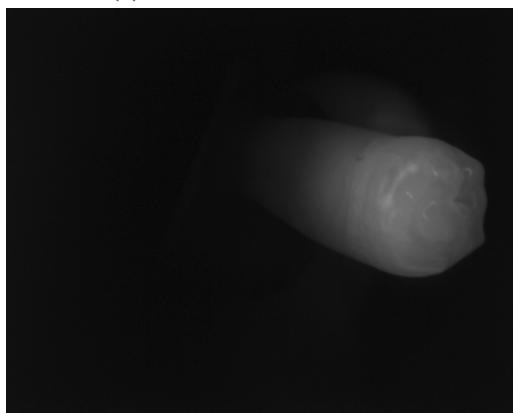
Wie in der Motivation aus Abschnitt 1.1 erwähnt, sind NIR-Aufnahmen eine potenziell geeignete Methode um die Diagnostik der Zahnärzte zu unterstützen. Erweitert wird dieser Ansatz durch die Überlegung, die NIR-Aufnahmen auf ein dreidimensionales Modell anzuwenden, um einen Zahn bzw.



(a) Seitliche NIR-Aufnahme



(b) Bukkale NIR-Aufnahme



(c) NIR-Aufnahme von schräg oben

**Abbildung 3.2:** Auszug aus dem NIR-Datensatz.

eine Zahnreihe interaktiv im dreidimensionalen Raum aus unterschiedlichen Perspektiven betrachten zu können. Durch eine Archivierung des texturierten 3D-Modells kann der Datenreduktion bei der Diagnose entgegengewirkt werden.

Das in dieser Arbeit erstellte Programm soll demnach in der Lage sein, die gegebenen NIR-Aufnahmen auf das 3D-Modell zu applizieren und dreidimensional darzustellen. Des Weiteren soll das texturierte 3D-Modell für die Archivierung abspeicherbar sein.

Aus der Datenanalyse und der Reflektion lassen sich nunmehr funktionale Anforderungen an den Prototypen definieren. Funktionale Anforderungen beziehen sich darauf, was das System leisten soll.

### 3.2.1 Datenvorverarbeitung

Um einen guten visuellen Eindruck des 3D-Modells nach der Texturierung zu erhalten, sollte das 3D-Modell vorverarbeitet werden. Betrachtet man das 3D-Modell, so ist eine Glättung der Oberfläche unerlässlich (siehe Abbildung 3.1). Bei der Texturierung mit den NIR-Aufnahmen würde sonst kein gutes Ergebnis erzielt werden.

#### 3.2.2 Kalibrierung

Die NIR-Aufnahmen sind unkalibriert. Das bedeutet, dass die intrinsischen und extrinsischen Kameraparameter nicht gegeben sind (vgl. Abschnitt 2.3.2). Diese müssen jedoch für die Texturierung zwingend gegeben sein, um eine Zuweisung von Punkten auf der Bildebene der Textur, zu Oberflächenpunkte des 3D-Modells, zu ermöglichen. Geeignete Verfahren, wie die Kameraparameter für Bilder bestimmt werden, müssen demnach recherchiert werden.

#### 3.2.3 Texturierung

Eine weitere Anforderung ist der eigentliche Texturierungsvorgang. Das System muss in der Lage sein, das 3D-Modell mit den Aufnahmen in Bezug zu den Kameraparametern zu texturieren. Die Kameraparameter müssen dabei für jede Aufnahme bestimmt sein. Als Ergebnis wird ein 3D-Modell definiert, dessen Texturkoordinaten für die Texturierung bestimmt sind. Hierbei muss untersucht werden, welche Möglichkeiten es für die Texturierung anhand mehrerer Texturaufnahmen gibt.

#### 3.2.4 Archivierung

Für eine Archivierung muss ein Format bestimmt werden, das die Daten des 3D-Modells und der Texturierung vereint. Das System muss in der Lage sein das 3D-Modell und dessen Texturkoordinaten sowie eine Referenz auf die Textur oder die Textur als solche in einem Format zu exportieren. Die Daten des 3D-Modells und die Texturkoordinaten müssen dabei gegeben sein. Das System soll die Daten des texturierten 3D-Modells in ein entsprechendes Format schreiben und abspeichern. Nach dem Vorgang ist das Ergebnis eine Datei, welche das texturierte 3D-Modell enthält.

#### 3.2.5 Visualisierung

Um das texturierte 3D-Modell ansehen zu können, ergibt sich als letzte funktionale Anforderung die Visualisierung des Modells. Das Programm muss dem Benutzer die Möglichkeit geben das texturierte 3D-Modell visuell und durch interaktive Kameraführung anzusehen. Der Benutzer kann durch Interaktion mit der Tastatur und/oder Maus die Kameraperspektive ändern und somit das Modell aus unterschiedlichen Perspektiven betrachten.

### 3.3 Fazit

Die Kernelemente der Anwendung bilden die Kalibrierung und die Texturierung anhand mehrerer Texturaufnahmen. Hierbei muss untersucht werden, wie die Kameraparameter für die gegebenen NIR-Aufnahmen bestimmt werden können, um eine Zuweisung der Texturkoordinaten für das 3D-Modell tätigen zu können. Des Weiteren muss bei der Texturierung recherchiert werden, welche geeigneten Methoden es unter Berücksichtigung der vorhandenen Daten gibt. Beide Rechercheaufgaben werden im nächsten Kapitel zum Stand der Technik ausgeführt.

# 4 Stand der Technik

Aus der Analyse der Daten und den Anforderungen lassen sich die zwei Kernelemente der Anwendung, die Kalibrierung und Texturierung, extrahieren. In diesem Kapitel wird der Stand der Technik für Verfahren zur Kamerakalibrierung und der Texturierung von 3D-Modellen mit mehreren vorhandenen Texturaufnahmen beschrieben.

## 4.1 Kamerakalibrierung

In dieser Arbeit sollen NIR-Aufnahmen auf ein 3D-Zahnmodell appliziert werden. Die NIR-Aufnahmen dienen somit als Textur für das 3D-Modell. Wie in Abschnitt 2.3 erwähnt, werden dabei die Kameraparameter benötigt, um eine Zuweisung der Texturkoordinaten zu den Oberflächenpunkten des 3D-Modells tätigen zu können. Der Vorgang, bei dem die Kameraparameter berechnet oder angenähert werden, wird als Kamerakalibrierung bezeichnet. Dieses Kapitel stellt die allgemeinen Ansätze zur Bestimmung der extrinsischen und intrinsischen Kameraparameter vor.

### 4.1.1 Korrespondierende Punkte

Der herkömmliche Ansatz, um die Kameraparameter anzunähern erfolgt mit sogenannten korrespondierenden Punkten. Dabei wird eine Reihe von Punkten in jeder Aufnahme bestimmt, die zu bekannten Punkten der Oberfläche des 3D-Modells korrespondieren. Anhand dieser korrespondierenden Punkte können die Kameraparameter festgelegt werden [Tsa87].

**Bestimmen der Kameraparameter** Wie in Abschnitt 2.3.2 beschrieben, enthält die Abbildung der dreidimensionalen Welt auf die zweidimensionale Bildebene die extrinsischen und intrinsischen Kameraparameter, die in der Projektionsmatrix aus Gleichung (2.18) zusammengefasst werden:

$$\tilde{s}m = P\tilde{M}_w \quad \text{mit} \quad P = A[Rt] = \begin{bmatrix} \alpha_u r_1^T + u_0 r_3^T & \alpha_u t_x + u_0 t_z \\ \alpha_v r_2^T + v_0 r_3^T & \alpha_v t_y + v_0 t_z \\ r_3^T & t_z \end{bmatrix} \quad (4.1)$$

Aus Gleichung (4.1) ist ersichtlich, dass sich die Elemente der Projektionsmatrix aus bekannten 3D-Punkten und den entsprechenden korrespondierenden Punkten in der Abbildung berechnen lässt.

Bei der Verwendung des linearen Modells, bei dem die Linsenverzerrung ignoriert wurde, kann so eine ausreichende Genauigkeit bei der Schätzung der Kameraparameter erreicht werden [Sch05]. In den folgenden Gleichungen sei  $q_{jk}$  das (j,k)-Element der Projektionsmatrix. Die ersten drei Elemente jeder Zeile werden zu dem Vektor  $\mathbf{q}_i = (q_{i1}, q_{i2}, q_{i3})^T$  zusammengefasst. Die Projektionsgleichung kann demnach folgendermaßen geschrieben werden:

$$s\tilde{\mathbf{m}} = s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = P\tilde{\mathbf{M}}_w = \begin{pmatrix} q_1^T & q_{14} \\ q_2^T & q_{24} \\ q_3^T & q_{34} \end{pmatrix} \tilde{\mathbf{M}}_w \quad (4.2)$$

Eine Gleichung, welche die Beziehung eines 3D-Punktes und seine korrespondierende Abbildung in der Bildebene  $\mathbf{m} = (u, v)^T$  darstellt, kann durch eine Division durch die dritte Komponente  $s$  bestimmt werden:

$$\begin{aligned} (q_1 - u \cdot q_3)^T \cdot M + q_{14} - u \cdot q_{34} &= 0 \\ (q_2 - v \cdot q_3)^T \cdot M + q_{24} - v \cdot q_{34} &= 0 \end{aligned} \quad (4.3)$$

Für  $N$  Messpunkte erhält man ein Gleichungssystem mit  $2N$  Gleichungen (siehe Gleichung (4.5)). Die Matrix A enthält dabei  $2N \times 12$  Einträge. Der Vektor x hat  $12 \times 1$  Komponenten.

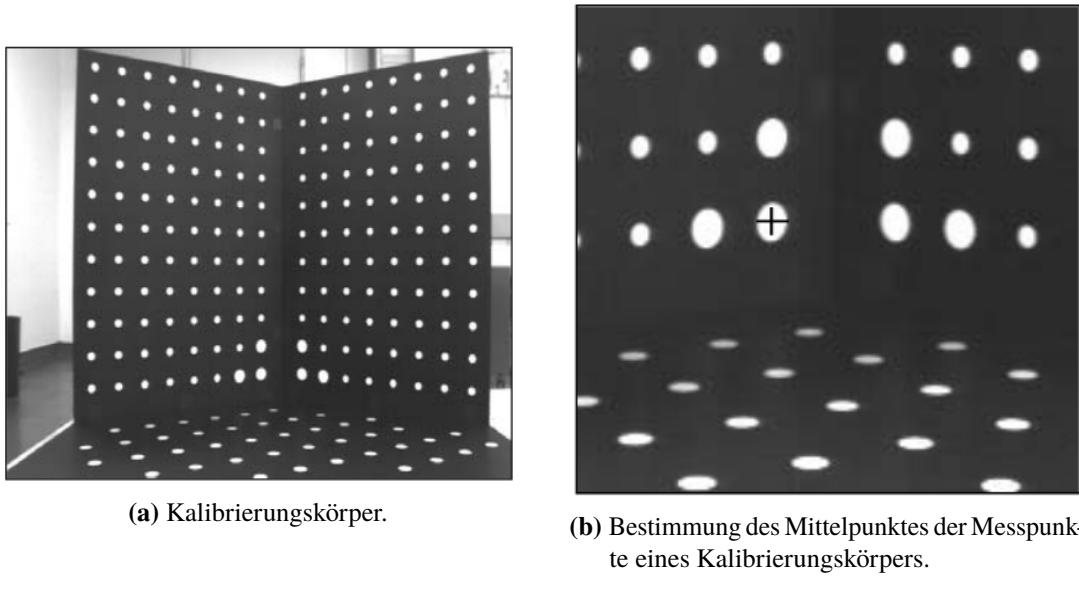
$$A \cdot x = 0 \quad \text{mit} \quad x = [q_1^T, q_{14}, q_2^T, q_{24}, q_3^T, q_{34}]^T \quad (4.4)$$

Um die triviale Lösung auszuschließen, kann folgende Nebenbedingung definiert werden:  $\|q_3\| = 1$ . Durch die Verwendung der Lagrangschen Multiplikation kann somit folgender Ausdruck, mittels Bestimmung der Eigenvektoren, minimiert werden (siehe [Sch05]):

$$\|A \cdot x\| \rightarrow \min \quad (4.5)$$

Als Ergebnis der Berechnung erhält man die Komponenten der Projektionsmatrix und implizit daraus die extrinsischen und intrinsischen Kameraparameter. Durch Verfahren wie dem Gauss-Newton-Verfahren oder dem Levenberg-Marquardt-Optimierungsverfahren kann die Schätzung zudem verbessert werden [Sch05].

**Kalibrierungskörper** Für die Bestimmung der korrespondierenden Punkte wird im Allgemeinen ein Kalibrierungskörper verwendet, der Messpunkte enthält, dessen 3D-Koordinaten bekannt sind (siehe Abbildung 4.1a). Durch geeignete Bildverarbeitungsverfahren, wie zum Beispiel im Bereich der Mustererkennung, können die Koordinaten der Messpunkte in der Aufnahme definiert werden (siehe Abbildung 4.1b).



**Abbildung 4.1:** Bestimmung korrespondierender Punkte anhand eines Kalibrierungskörpers. [Sch05]

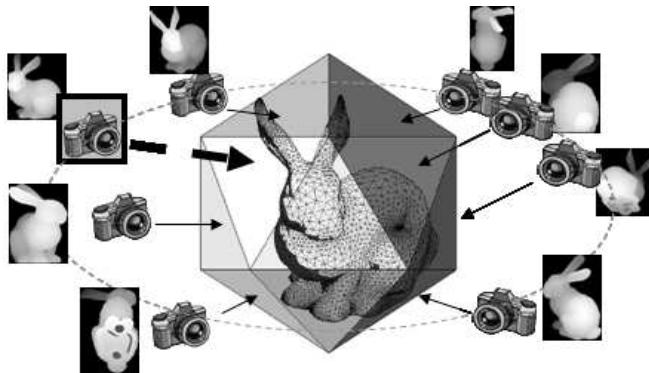
#### 4.1.2 3D/2D-Registrierung

Die Kalibrierung anhand von korrespondierenden Punkten kommt im Allgemeinen nicht ohne Kalibrierungskörper aus. Zwar ist es möglich, bei bekannten 3D-Punkten die korrespondierenden Punkte in der Textur aufgrund von Merkmalen im Bild zu bestimmten, jedoch ist dies bei Röntgenaufnahmen oder auch Infrarot- und NIR-Aufnahmen schwierig [UDR+09].

Deshalb wurden Verfahren entwickelt, die eine Annäherung an die Kameraparameter ermöglichen, ohne die Bestimmung korrespondierender Punkte mit Hilfe von Markern, Kalibrierungskörpern, oder Ähnlichem. Diese Verfahren benutzen 3D/2D-Bildregistrierungsverfahren, die im Grunde für ein zugehöriges 2D-Bild das 3D-Modell so lange aus unterschiedlichen Kamerapositionen aus untersuchen, bis eine möglichst hohe Ähnlichkeit mit dem 2D-Bild gefunden wurde und somit die extrinsischen Kameraparameter angenähert werden können (siehe [UPDR09] und Referenzen, sowie [LHS01] [MK99] [NK99]). Die intrinsischen Kameraparameter lassen sich meist direkt aus der Spezifikation der Kamera herauslesen und werden demnach bei diesem Verfahren als gegeben angenommen.

Im dreidimensionalen Raum sind die Möglichkeiten der Standorte und Ausrichtungen für die Kamera, welche eine bestimmte Texturaufnahme erzeugt hat, unendlich. Eine geeignete Strategie zur Erstellung der perspektivischen Aufnahmen des 3D-Modells ist daher unerlässlich. Dieses Kapitel beschreibt zwei bekannte Verfahren hierfür, die von Uccheddu et al. und Lensch et al. entwickelt und in deren Publikationen vorgestellt wurden.

**Fiktiver Ikosaeder** Uccheddu et al. beschreiben in [UPDR09] (und Referenzen) ein Vorgehen, bei dem die Kameraparameter mit Hilfe eines fiktiven Ikosaeders angenähert werden können. Wie in Abbildung 4.2 zu sehen, wird zunächst ein fiktiver Ikosaeder definiert, in dessen Zentrum sich das zu texturierende 3D-Modell befindet. Das 3D-Modell wird vom Ikosaeder vollständig



**Abbildung 4.2:** Erste Iteration zur Generierung der 2D-Aufnahmen für den Vergleich mit der Texturaufnahme. [UPP11].

eingeschlossen. Der Ikosaeder beschreibt 20 Flächen in Form von Dreiecken um das 3D-Modell herum. Die Normalen dieser Flächen, die vom Zentrum der Flächen ausgehen und in Richtung des Zentrums des Ikosaeders zeigen, beschreiben dabei die Position und Ausrichtung der Kamera, mit der das 3D-Modell in die Bildebene projiziert werden soll.

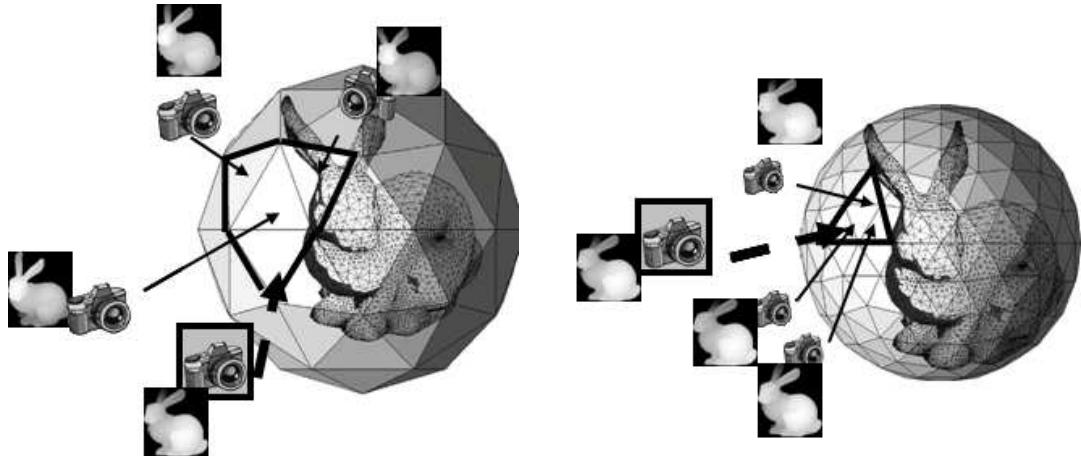
Zunächst werden in der ersten Iteration des Algorithmus für die 20 Normalen der Flächen des Ikosaeders 20 2D-Aufnahmen gerendert. Anschließend werden die Ähnlichkeitsmaße dieser Aufnahmen im Vergleich zu der Texturaufnahme berechnet. Somit wird unter den generierten Aufnahmen die am besten angenäherte Kameraposition ermittelt.

Anschließend wird durch weitere Iterationen dieses Verfahrens aus benachbarten Perspektiven zu der bestimmten Perspektive (mit dem bisherig höchsten Ähnlichkeitsmaß) die Kameraposition noch besser angenähert. Dazu werden die Flächen des Ikosaeders gleichmäßig weiter unterteilt, sodass das fiktive Gebilde sich einer Sphäre immer weiter annähert. Dadurch entstehen mehr Flächen, verbunden mit mehr Normalen und somit mehr potentielle Kamerapositionen. Untersucht werden dabei nur die Flächen und deren Normalen, die sich innerhalb der Fläche, mit der als bisher am besten angenäherte Perspektive befinden (siehe Abbildung 4.3). Dieser Iterationsschritt kann mehrmals wiederholt werden, um die Kameraposition besser anzunähern.

Obwohl dieser Ansatz vielversprechend erscheint, wird in den Publikationen nicht beschrieben, wie die Größe des Ikosaeders gewählt wird. Das Ausmaß beschreibt jedoch implizit, aus welcher Entfernung die echte Aufnahme erstellt wurde. Auch die Orientierung (Rotation) der Kamera wird in diesem Verfahren vernachlässigt. Es wird stets angenommen, dass die Kamera in Richtung des Zentrums des 3D-Modells gerichtet ist.

**Rotation um Gravitationspunkt** Lenzsch et al. stellen in [LHS01] ein weiteres Verfahren vor, mit dem die Kameraparameter durch eine entsprechende Strategie schnell angenähert werden sollen. Im Gegensatz zu dem im vorhergehenden Kapitel beschriebenen Verfahren wird in diesem Verfahren jedoch nicht die Kamera selbst, sondern das 3D-Modell um das Zentrum der Gravitation  $g$  des 3D-Modells transformiert.

Die Umrechnung der Weltkoordinaten eines Punktes im dreidimensionalen Raum zu Kamerakoordinaten aus Gleichung (2.11) ändert sich somit entsprechend Gleichung (4.6).



(a) Generierung der 2D-Aufnahmen nach der zweiten Iteration - nur die Flächen, die sich aus der vorherig am besten geeigneten Fläche ergeben werden untersucht.

(b) Dritter Iterationsschritt - die Kameraposition wird weiter angenähert.

**Abbildung 4.3:** Iterative Annäherung an die Kameraperspektive. [UPP11]

$$M_c = R(M_w - g) + Rg + t_c = R(M_w - g) + t_{neu} \quad (4.6)$$

Der Startwert wird in [LHS01] zunächst über das Sichtfeld definiert. Das Sichtfeld beschreibt den Bereich innerhalb der 3D-Szene, der von der Kamera erfasst wird [GC18b]. Das Sichtfeld wird über die Brennweite berechnet:

$$F = \cot\left(\frac{f}{2}\right) \quad (4.7)$$

Mit Hilfe des Sichtfelds wird der Abstand der Kamera zum 3D-Modell, also  $t_z$  aus  $t_c = [t_x, t_y, t_z]^T$  berechnet. Angenommen, das 3D-Modell ist auf der Texturaufnahme vollständig sichtbar, so kann die Distanz mithilfe des Sichtfelds und der Größe des Objekts berechnet werden. Der  $t_x$  und  $t_y$  Verschiebungsvektor wird dabei für die Startwertbestimmung vernachlässigt ( $t_x = 0, t_y = 0$ ). [LHS01]

Anschließend werden unterschiedliche perspektivische Aufnahmen erzeugt, indem das 3D-Modell in drei verschiedene x-, und jeweils vier verschiedene y- und z-Richtungen gedreht wird. Somit entstehen insgesamt 48 Aufnahmen, die über ein Ähnlichkeitsmaß mit der Texturaufnahme verglichen werden und somit einen geeigneten Startwert liefern. Durch ein iteratives Vorgehen werden daraufhin die Kameraparameter und Rotationsparameter angepasst, die perspektivische Aufnahme zu den neuen Parametern erstellt und die Ähnlichkeit zu der Texturaufnahme erneut verglichen. [LHS01]

In [LHS01] wird jedoch nicht spezifiziert, wie genau die Distanz berechnet werden kann. Ebenfalls unspezifiziert sind die gewählten Rotationswinkel für die Generierung der perspektivischen Aufnahmen.

### 4.1.3 Ähnlichkeitsmaße der 3D/2D-Registrierung

Wie bereits erwähnt, verwendet das Verfahren der Kamerakalibrierung mittels 3D/2D-Registrierung Ähnlichkeitsmaße, um die extrinsischen Kameraparameter anzunähern. Die korrekte Kameratransformation projiziert das 3D-Modell so in die Bildebene, sodass die Merkmale des projizierten 3D-Modells, bis auf eine kleine Fehlerrate, den Merkmalen der Texturaufnahme gleicht.

Für die 3D/2D-Registrierung wird die Ähnlichkeit der erstellten perspektivischen Aufnahme mit der Texturaufnahme im Allgemeinen anhand von globalen Bildmerkmalen berechnet. Globale Bildmerkmale beschreiben ein Bild als Ganzes und geben somit Auskunft über eine bestimmte Eigenschaft aller Pixel. Diese Eigenschaften können beispielsweise Farben, Histogramme, Texturen, Kanten etc. sein [HAA16a].

Die am häufigsten verwendeten globalen Bildmerkmale bei der Ähnlichkeitsbestimmung der 3D/2D-Registrierung sind Kanten, Silhouetten und die Mutual Information, die in diesem Kapitel vorgestellt werden.

#### 4.1.3.1 Kantenmerkmale

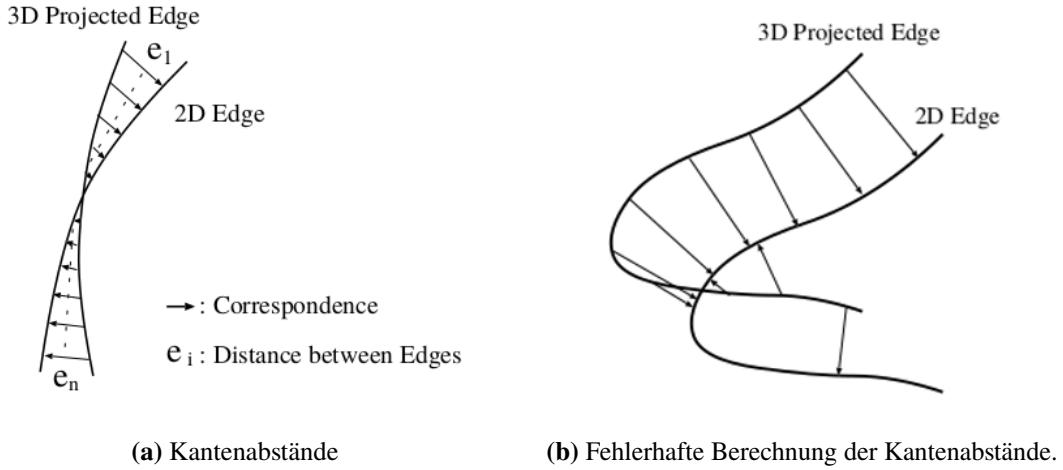
Matsushita et al. verwenden für die Bestimmung der Ähnlichkeit der Bilder Kantenmerkmale der Objektruisse [MK99]. Hierfür werden zunächst die Kantenprofile für die projizierten Aufnahme des 3D-Modells und der Texturaufnahme erstellt. Die Kantenprofile können mit entsprechenden Verfahren wie bspw. durch eine Gradienten-basierte Kantendetektion, dem Canny-Kantenoperator oder (falls sich die Kontraste zwischen Vorder- und Hintergrund genügend abheben) durch eine einfache Schwellwertfunktion erstellt werden (siehe [BB15]).

Als Ähnlichkeitsmaß wird die Summe der Distanzen zwischen den beiden Profilen berechnet (siehe Abbildung 4.4a):

$$E = \frac{1}{n} \sum_{j=1}^n e_j \quad (4.8)$$

Für jeden Kantenpixel der Texturaufnahme wird der geometrisch nächste Kantenpixel der projizierten Aufnahme für die Berechnung der Distanzsumme verwendet. Bei der Kamerakalibrierung mittels korrespondierender Punkte muss vor jeder Aufnahme der Textur ein Bild von dem Kalibrierungskörper aufgenommen werden. Die Kamera darf dabei während der Aufnahme der beiden Bilder nicht bewegt werden. Die von der LMU München bereitgestellten NIR-Aufnahmen wurden ohne Kalibrierungskörper aufgenommen. Eine Kalibrierung anhand von korrespondierenden Punkten wäre demnach nur durch eine erneute Datenerfassung, unter Verwendung eines Kalibrierungskörpers möglich.

Die berechnete Distanzsumme gibt demnach Auskunft, inwiefern sich die Konturen der vergleichenden Objekte im Bild unterscheiden und definiert somit ein Ähnlichkeitsmaß.

**Abbildung 4.4:** Korrespondierende Kanten. [MK99]

#### 4.1.3.2 Silhouetten

Lensch et al. verwenden ein Verfahren, bei dem die Silhouette, des in die Bildebene projizierten 3D-Objekts, mit der Silhouette des Objekts, aus der Texturaufnahme, verglichen wird Lensch2001Aug [LHS00].

Die Segmentierung der Silhouette kann bei den Aufnahmen über ein Schwellwertverfahren bestimmt werden. Bei dem Verfahren werden die Intensitätswerte der Pixel eines Bildes  $f$  anhand zweier Schwellwerte  $t_{min}$  und  $t_{max}$  separiert und in einem Binärbild bzw. einer Binärfolge  $B$  markiert: [Han09a]

$$B(x, y) = \begin{cases} 1 & \text{falls } t_{min} \leq f(x, y) \leq t_{max} \\ 0 & \text{sonst} \end{cases} \quad (4.9)$$

Die Bestimmung der Schwellwerte wird in der Praxis meist manuell vom Anwender vorgenommen. Alternativ können die Schwellwerte häufig auch über die Informationen des Histogramms bestimmt werden. In Hinsicht auf die Ähnlichkeitsbestimmung zwischen einem 3D-Modell und NIR-Aufnahmen, wäre diese automatisierte Bestimmung möglich, insofern das 3D-Modell mit einem Kontrastunterschied zwischen dem Objekt und dem Hintergrund gerendert wird und die Texturaufnahmen einen Kontrastunterschied zwischen dem Objekt und dem Hintergrund aufweisen. In NIR-Aufnahmen ist dies üblicherweise der Fall. Falls der Kontrast zwischen dem Hintergrund und dem Objekt nicht groß genug ist, muss eine andere Bildverarbeitungstechnik wie beispielsweise eine Kantenerkennung für die Segmentierung herangezogen werden [LHS01] [LHS00].

Nach der Segmentierung der Silhouetten werden über eine einfache, pixelweise XOR-Operation der zwei Binärbilder die Pixel zwischen den Umrissen der beiden Aufnahmen in einem Bild bestimmt und in einem weiteren Binärbild markiert.

Es kann nun ermittelt werden, inwiefern die Silhouetten der zwei Objekte aus den Vergleichsbildern übereinander liegen, indem die Werte ungleich Null des XOR verknüpften Bildes gezählt werden. Während bei einer exakten Überlagerung ein Wert nahe an Null zurückgegeben wird, wird der Wert immer größer, je weiter die Objekte in den Aufnahmen voneinander abweichen. Dieser Wert dient somit als Maß zur Ähnlichkeitsbestimmung bei der Registrierung.

#### 4.1.3.3 Mutual Information

Ein weiterer Ansatz wird in von Uccheddu et al. in [UPDR09] (siehe auch Referenzen) vorgestellt. Dabei wird die Histogrammverteilung des projizierten 3D-Modells mit der Histogrammverteilung der Texturaufnahme verglichen. Das Ergebnis dieses Vergleichs ist ein Maß, das besagt, wie viel Informationen das projizierte Bild mit der Texturaufnahme teilt. Dieses Maß wird als MI bezeichnet. Die korrekte Kameratransformation projiziert das 3D-Modell so in die Bildebene, sodass die MI für das projizierte Bild und der Texuraufnahme maximal ist (Maximizing Mutual Information (MMI)).

**Histogramme** Histogramme sind Bildstatistiken, die Auskunft über die Häufigkeitsverteilung einzelner Intensitätswerte geben. Das Histogramm  $H$  eines Grauwertbilds  $X$  mit möglichen Intensitätswerten im Bereich  $X(u, v) \in [0, K - 1]$  enthält genau  $K$  Einträge. Beispielsweise enthält ein 8-Bit-Grauwertbild  $K = 2^8 = 256$  Einträge. Der Wert des Histogramms an der Stelle  $i$  ist die Anzahl der Pixel von  $X$  mit dem Intensitätswert  $i$ : [BB15]

$$h(i) = |X(u, v) = i| \quad fr \quad 0 \leq i \leq K \quad (4.10)$$

Somit enthält der Eintrag  $h(0)$  für ein 8-Bit-Grauwertbild die Anzahl aller Pixel mit dem Wert 0 und  $h(255)$  die Anzahl aller weißen Pixel mit dem maximalen Intensitätswert. Aus der Histogrammberechnung resultiert somit ein eindimensionaler Vektor  $h$  der Länge  $K$ .

**Entropie** Für die Berechnung der MI ist es zunächst notwendig, die Datenverteilung, auch Entropie genannt, zu berechnen. Claude Shannon definierte die Entropie folgendermaßen [CRB18]:

$$H(X) = - \sum_{i=1}^n P(x_i) \log_2 P(x_i) \quad 0 \leq P(x_i), H(x) \leq 1 \quad (4.11)$$

Dabei beschreibt  $P(x_i)$  die Wahrscheinlichkeit mit der das Ereignis  $x_i$  der diskreten Zufallsvariable  $X$  mit  $n$  Ereignissen eintritt. Betrachtet man beispielsweise einen Münzwurf, so gilt:  $P(x_1) = P(x_2) = 0.5$ . Als Entropie erhält man dabei für den Münzwurf eine Entropie von  $H(x) = -P(x_1)\log_2 P(x_1) - P(x_2)\log_2 P(x_2) = 1.0$ . Es handelt sich demnach um eine Gleichverteilung der Wahrscheinlichkeiten. Bei einer Umverteilung der Wahrscheinlichkeiten zu  $P(x_1) = 0.6$  und  $P(x_2) = 0.4$  ergibt sich aus der Entropie  $H(x) = 0.97$ , dass die Wahrscheinlichkeiten nicht mehr gleich verteilt sind.

Erweitert man nun Shannons Entropiedefinition um eine weitere diskrete Zufallsvariable  $Y$  ergibt sich die Gleichung der gemeinsamen Entropie:

$$H(X, Y) = - \sum_{i=1}^m \sum_{j=1}^n P(x_i, y_j) \log_2 P(x_i, y_j) \quad (4.12)$$

Dabei ist  $P(X, Y)$  die gemeinsame Wahrscheinlichkeit, also die Wahrscheinlichkeit, dass  $X$  und  $Y$  gemeinsam auftreten.

Die Gleichung der bedingten Entropie gibt Auskunft darüber, wie viel Unwahrscheinlichkeit von  $X$  entfernt wurde, wenn  $Y$  bekannt ist:

$$H(X|Y) = H(X, Y) - H(Y) \quad (4.13)$$

**Kullback-Leiber Divergenz (KLD)** Die MI basiert auf der Kullback-Leiber Divergenz (KLD), auch relative Entropie genannt, die aussagt, wie sehr eine Wahrscheinlichkeitserteilung  $P(X)$  von einer anderen Wahrscheinlichkeitserteilung  $Q(Y)$  abweicht:

$$KLD(P(X)||Q(Y)) = \sum_{i=1}^n P(x_i) \log_2 \frac{P(x_i)}{Q(y_j)} \quad (4.14)$$

**Mutual Information (MI)** Erweitert man die KLD so, dass die Gleichung der Entropie Auskunft darüber gibt, wie viel Unwahrscheinlichkeit von  $X$  entfernt wurde, wenn  $Y$  bekannt ist, ergibt sich die Gleichung der MI:

$$\begin{aligned} I(X; Y) &= KLD(P(X, Y)||P(X)P(Y)) \\ &= \sum_{i=1}^m \sum_{j=1}^n P(x_i, y_j) \log_2 \frac{P(x_i, y_j)}{P(x_i)P(y_j)} \\ &= H(X) - H(X|Y) \end{aligned} \quad (4.15)$$

Aus Gleichung (4.15) wird ersichtlich, dass die MI zweier diskreter Zufallsvariablen  $X$  und  $Y$  den Grad der Abhängigkeit zwischen der gemeinsamen Wahrscheinlichkeit  $P(X, Y)$  und der Wahrscheinlichkeit im Falle der statistischen Unabhängigkeit  $P(X) \cdot P(Y)$  berechnet. Da die Entropie  $H(X)$  ein Maß für die Unsicherheit über die Zufallsvariable  $X$  ist, und  $H(X|Y)$  das Maß der übrigen Unsicherheit für  $X$ , wenn  $Y$  bekannt ist, so ergibt sich daraus, dass  $I(X; Y)$  Auskunft über die Reduktion der Unsicherheit von  $X$  gibt, wenn  $Y$  bekannt ist. Äquivalent kann gesagt werden, dass  $I(X; Y)$  somit Auskunft über den gemeinsamen Informationsgehalt gibt, den  $Y$  über  $X$  enthält. [MCV+96]

Um die MI im Intervall  $[0, 1]$  zu restriktieren, kann die MI anschließen noch durch die Wurzel des Produktes der Entropie  $H(X)$  und  $H(Y)$  geteilt werden (Gleichung (4.15)). Ergebnis ist die normalisierte MI. Sind  $X$  und  $Y$  statistisch unabhängig voneinander, so gilt  $P(X, Y) = P(X) \cdot P(Y)$  und daraus resultierend für  $NI(X; Y) = 0$ . Sind  $X$  und  $Y$  maximal voneinander abhängig, so gilt  $NI(X; Y) = 1$ .

$$NI(X;Y) = \frac{I(X;Y)}{\sqrt{H(X)H(Y)}} \quad (4.16)$$

Bei der Bildregistrierung sagt die MI somit aus, dass zwei Bilder genau dann am besten registriert sind, wenn  $I(X;Y)$  ein Maximum aufweist. Somit ist die Menge der Information, die Bild  $X$  aus Bild  $Y$  enthält, genau dann maximal, wenn sich die Bilder in der bestmöglichen registrierten Position befinden. Mit Hilfe der MI kann demnach eine Aussage über die Güte bzw. Genauigkeit der Registrierung getroffen werden.

**Depth Images** Uccheddu et al. verwenden in deren Arbeit Depth Images für den Vergleich zwischen dem gerenderten Bild und der Texturaufnahme [UPDR09]. Depth Images sind eine bildliche Darstellung des Z-Buffers, wie in Abschnitt 4.2.1.2 beschrieben. Sie entstehen, indem die Werte des Z-Buffers als Grauwerte interpretiert werden. In dem Depth Image ist somit die Distanz des sichtbaren Objekts, in Bezug zu der Kamera durch verschiedene Grauwerte bestimmt.

Objekte oder Oberflächen, die sich näher zu der Kamera befinden, werden heller dargestellt - Objekte, die sich weiter Weg von der Kamera befinden, werden dunkler dargestellt. Durch die Beziehung der Grauwerte zur Distanz können somit bessere Ergebnisse bei der Berechnung der MI erzielt werden [UPP11].

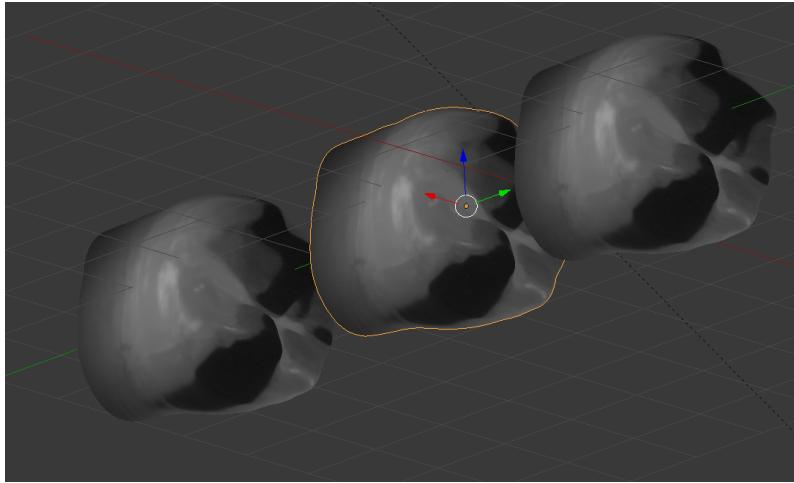
#### 4.1.3.4 Fazit

Der Erfolg bei der Texturierung mittels 3D/2D-Registrierung hängt von dem Ähnlichkeitsmaß ab, das die Ähnlichkeit der gerenderten Aufnahme des 3D-Modells zur Texturaufnahme beschreibt. Für die richtige Perspektive muss dieses Ähnlichkeitsmaß maximal sein. Ist dies nicht der Fall, so gibt es ein oder mehrere andere Texturen für eine perspektivische Aufnahme des 3D-Modells, für die dieses Ähnlichkeitsmaß maximal ist. Somit gibt es auch keine genaue oder eine fehlerhafte Zuweisung einer perspektivischen Aufnahme zu einer bestimmten Textur. Die Folge ist, dass die Kameraparameter, mit diesem gewählten Ähnlichkeitsmaß, nicht bestimmbar sind. Dies muss bei der Verwendung des 3D/2D-Registrierungsverfahrens berücksichtigt werden.

## 4.2 Texturierung anhand mehrerer Aufnahmen

Durch eine Bestimmung der Kameraparameter für die Texturaufnahmen in Bezug zu einem konkreten 3D-Modell wäre es nunmehr möglich, die Vertices des 3D-Modells über die Gleichung der perspektivischen Transformation (siehe Gleichung (2.18)) auf die Bildebene zurückzurechnen und somit die Texturkoordinaten den Oberflächenpunkten des 3D-Modells zuzuweisen.

Jedoch reicht eine einfache Projektion über die Kameraparameter, wie in Abschnitt 2.3 beschrieben, nicht aus, da Bereiche, die eigentlich nicht sichtbar sind, ebenfalls eingefärbt werden würden. Somit würde unter anderem die Rückseite des 3D-Modells ebenfalls eingefärbt werden, wie in Abbildung 4.5 sichtbar.



**Abbildung 4.5:** Das gegebene 3D-Modell des Zahnes projiziert auf eine NIR-Textur. Die Projektion erfolgt anhand der zugehörigen Kameraparameter. Die Textur erscheint sowohl auf der Vorder-, als auch auf der Rückseite.

Deshalb muss eine Sichtbarkeitsanalyse durchgeführt werden, die für jedes Primitiv des 3D-Modells definiert, ob es von einer bestimmten Kameraperspektive aus (potentiell) sichtbar ist. Nicht sichtbare Oberflächen des echten Objekts, also die Flächen, die von anderen Flächen des Objekts verdeckt werden, dürfen nicht mit eingefärbt werden.

In diesem Kapitel wird daher zunächst auf die Sichtbarkeitsanalyse eingegangen. Anschließend wird beschrieben, wie die verschiedenen Texturen kombiniert werden können, um letztendlich ein vollständig texturiertes 3D-Modell zu erhalten.

### 4.2.1 Sichtbarkeitsanalyse

Bei der Sichtbarkeitsanalyse muss zwischen der Sichtbarkeit eines Primitivs an sich und der Verdeckung des Primitiv durch ein anderes Primitiv differenziert werden. Beide Fälle müssen überprüft werden, um eine geeignete Texturierung anhand mehrerer Texturaufnahmen zu ermöglichen. Die beiden Eventualitäten werden in dieser Arbeit unter dem Begriff der Sichtbarkeitsanalyse zusammengefasst. Die Verfahren der Sichtbarkeitsanalyse werden in diesem Kapitel vorgestellt.

#### 4.2.1.1 Normalenüberprüfung

Rocchini et al. verwenden in [RCMS99] einen Ansatz, der die Winkel der Normalen zu der optischen Achse berechnet und vergleicht, um einen Vertex auf seine Sichtbarkeit zu überprüfen.

**Vorgehen** Zunächst wird bei dem Verfahren die Sichtbarkeit eines Vertex  $v$  definiert. Ein Vertex  $v$  ist in einem Bild  $i_k$  sichtbar, insofern er folgende Bedingungen erfüllt:

- Die Projektion von  $v$  in die Bildebene ist in  $i_k$  enthalten.

- Der Normalenvektor von  $v$  zeigt in Richtung des Kameraursprungs. Der Winkel zwischen der optischen Achse und der Normalen muss dabei kleiner als  $\pi/2$  sein.

Dabei wird zunächst der Vertex  $v$  auf einen Punkt  $p$  in der Bildebene zurückgerechnet. Beinhaltet eine Texturaufnahme mit der Bildgröße  $i_k = u \cdot v$  das ganze Objekt, so ist jeder Vertex und somit auch  $v$  in der Texturaufnahme, also  $p$  in  $i_k$  enthalten. Es gilt demnach  $u_0 \leq p_x \leq u$  und  $v_0 \leq p_y \leq v_0$ . Falls das Objekt nicht vollständig auf der Texturaufnahme  $i_k$  sichtbar ist und der Punkt  $p$  außerhalb der Bildgröße  $i_k = u \cdot v$  liegt, so ist  $v$  nicht mehr sichtbar in der Texturaufnahme  $i_k$ .

Anschließend wird für jedes Primitiv, dessen Punkte  $p_j$  in  $i_k$  enthalten sind, der Normalenvektor bestimmt. Der Normalenvektor steht orthogonal auf der Fläche des Vertex und zeigt in die Richtung, aus der die Textur angebracht wird - also die eigentliche Oberfläche des Primitivs. Der Normalenvektor bestimmt somit die Vorder- und Rückseite des Primitivs. Der Normalenvektor eines Vertex kann meist aus dem Datensatz des 3D-Modells ausgelesen werden oder über die Reihenfolge der Vertices aus dem Kreuzprodukt der Vektoren des Primitivs ausgerechnet werden. Um zu bestimmen, ob der Normalenvektor in Richtung der Kamera zeigt, wird daraufhin der Winkel zwischen dem Normalenvektor und dem Vektor der optischen Achse berechnet.

Der Winkel zweier Vektoren ( $P$  und  $Q$ ) im dreidimensionalen Raum berechnet sich aus dem Kreuzprodukt und dem Skalarprodukt der zwei Vektoren (siehe Gleichung (4.17)). Ist der Winkel  $\theta < (\pi/2)$  kleiner als ein bestimmter Winkel (in [RCMS99]  $\theta < (\pi/2)$ ), so wird angenommen, dass der Normalenvektor in Richtung des Kameraursprungs zeigt und somit auf der Textaufnahme  $i_k$  sichtbar ist.

$$\theta = \arccos \left( \frac{P \cdot Q}{|P||Q|} \right) \quad (4.17)$$

**Ergebnis** Das Ergebnis bei diesem Verfahren ist eine Liste von Texturen für einen bestimmten Vertex, auf denen der projizierte Punkt des Vertex potentiell sichtbar ist. Sind alle Vertices eines Primitivs auf einer Textur sichtbar, so ist auch das gesamte Primitiv sichtbar.

**Fazit** Bei der Sichtbarkeitsanalyse mittels Normalenüberprüfung handelt es sich um ein einfaches Verfahren, das nachvollziehbar und einfach zu implementieren ist. Das Verfahren liefert jedoch keine Informationen, ob ein Primitiv von einer Fläche des 3D-Modells verdeckt wird. Das Verfahren müsste demnach mit einer Verdeckungsprüfung erweitert werden, um eine vollständige Sichtbarkeitsanalyse abzudecken.

#### 4.2.1.2 Z-Buffer Verfahren

Neugebauer et al. und Grammatikopoulos et al. verwenden für die Sichtbarkeitsanalyse einen Z-Buffer Algorithmus, der die Tiefeninformation der Oberflächenpunkte an jeder Pixelposition der Bildebene überprüft und mit den anderen Oberflächenpunkten vergleicht.

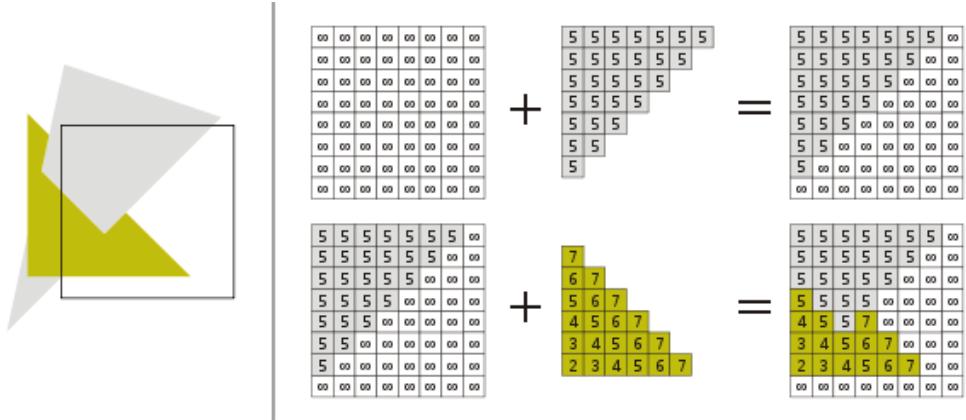


Abbildung 4.6: Illustriertes Z-Buffer Verfahren für zwei Dreiecke. [19j]

**Vorgehen** Die Idee des Z-Buffers ist, für jeden Pixel der Bildebene neben deren Farbe die Tiefeninformation (Z-Information) als ein weiteres Attribut zu speichern. Bei einem Bild mit einer Auflösung von  $1024 \times 768$  wären dies dann im Allgemeinen genau  $1024 \times 768$  Einträge (wobei der Z-Buffer auch größer oder kleiner gewählt werden kann, insofern das Seitenverhältnis des Z-Buffers in Bezug zu der Texturaufnahme eingehalten wird).

Die Tiefeninformation des Pixel hält dabei genau die Tiefeninformation des Punktes eines Primitivs, das diesen Pixel erzeugt hat und ihn somit eine Farbe zugewiesen hat. Werden nun mehrere Primitive gerendert, die den selben Pixel einfärben (also in der Szene hintereinander liegen), so wird die Tiefeninformation der Primitive überprüft und dem Pixel die Farbe des Primitivs mit dem kleineren Tiefenwert zugewiesen. Anschließend wird der Z-Buffer mit diesem neuen Wert aktualisiert.

Durch dieses Verfahren ist es demnach möglich, zwischen Primitiven im Vorder- und im Hintergrund zu unterscheiden und diese entsprechend zu rendern. [Sin15]. Eine Extraktion der Tiefeninformationen in einen eigenen Buffer wird Z-Buffer oder Depth Map (Tiefenkarte) genannt.

Die Z-Informationen der einzelnen Vertices lassen sich über die Gleichung (2.18) der perspektivischen Projektion bestimmen, die den Vertex auf die Bildebene umrechnet. Der Z-Wert der dritten Komponente hält dabei die Z-Information für den Z-Buffer.

Für die Sichtbarkeitsanalyse werden für jede Texturaufnahme, anhand der bekannten Kameraparameter, die Z-Buffer für das 3D-Modell erstellt und mit einem neutralen Wert (meist unendlich) vorinitialisiert. Anschließend werden alle Primitive auf die Bildebene zurückgerechnet. Die Z-Werte der zurückgerechneten Punkte der Bildebene für ein Primitiv werden daraufhin mit den Werten aus dem zuvor erstellten Z-Buffer verglichen. Liegen die Z-Werte des Primitivs unterhalb der Z-Werte des Z-Buffers, so werden die Werte und eine Referenz auf das Primitiv in Form einer ID in den Z-Buffer geschrieben und somit der Z-Buffer aktualisiert. Das Verfahren ist in Abbildung 4.6 illustriert. [NK99]

**Ergebnis** Ist ein Primitiv in einer Texturaufnahme nicht verdeckt und somit sichtbar, so befindet sich dessen bestimmte ID und Z-Werte im Z-Buffer der Textur. Für jede Textur existiert somit ein Z-Buffer mit einer Referenz auf die Primitive, die auf der Textur sichtbar und nicht verdeckt sind.

**Erweiterter Z-Buffer Algorithmus** Das Z-Buffer Verfahren ist im Vergleich zu anderen Algorithmen zur Verdeckungsprüfung zwar einfach zu implementieren, benötigt aber viel Speicher und ist rechenintensiv [AH05]. Grammatikopoulos et al. erweiterten deshalb das Verfahren, indem sie die Texturaufnahme in ein rechteckiges Gitter tessellierten. Die Größe der Zellen der Gitter ist dabei abhängig von der Speichergröße und der Größe des 3D-Modells und des neuen Bildes. Für jedes, auf die Bildebene zurückgerechnetes Primitiv, wird ein umschließendes Parallelogramm definiert. Die Zellen, die im Parallelogramm enthalten sind, werden mit einer Identifikationsnummer (ID) des Vertex versehen.

Das Verfahren wird für alle Vertices wiederholt. Das Ergebnis ist eine Tabelle, die alle IDs der Vertices enthält, die den entsprechenden Zellen zugewiesen wurden. Es muss nunmehr nur noch eine Verdeckungsprüfung mittels Z-Buffer für eine begrenzte Anzahl an Vertices ausgeführt werden, und zwar für jene, die einer entsprechenden Zelle zugeschrieben wurden. Das von Grammatikopoulos et al. vorgestellte Verfahren ist demnach ein beschleunigtes Z-Buffer Verfahren zur Verdeckungsprüfung.

**Fazit** Generell sind die Z-Buffer Verfahren anfällig für Quantisierungsfehler und führen für komplexe Szenen zu schlechten Laufzeiten ( $O(n^2)$ ). Durch Parallelisierung der Berechnungen auf der Grafikkarte oder dem erweiterten Z-Buffer Verfahren kann der schlechten Laufzeit jedoch entgegen gewirkt werden. [AH05]

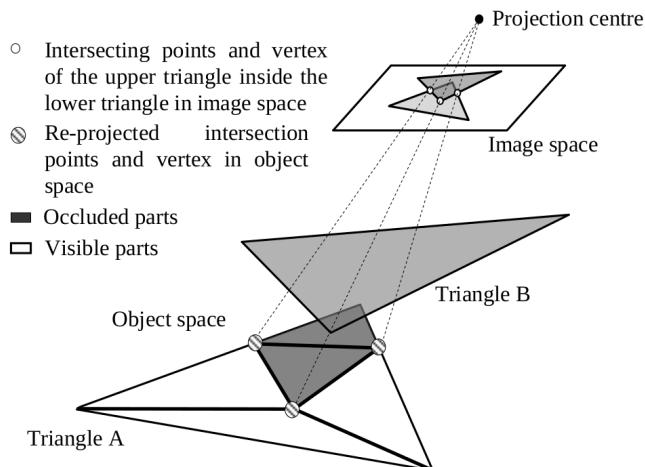
Ein wesentlicher Vorteil des Z-Buffer Verfahrens zur Sichtbarkeitsanalyse ist die integrierte Verdeckungsprüfung. Außerdem spielt die Ausrichtung der Primitive oder Fehler im 3D-Modell, wie bspw. Löcher, keine Rolle.

#### 4.2.1.3 Maler-Algorithmus

Ein weiteres Verfahren für die Sichtbarkeitsanalyse und Verdeckungsprüfung auf Bildebene ist der sogenannte Maler-Algorithmus, der von Hanusch in [Han09b] verwendet wird.

**Vorgehen** Bei dem Maler-Algorithmus werden zunächst alle Dreiecke des 3D-Modells für eine Kameraperspektive auf die Bildebene zurückgerechnet. Zusätzlich wird die Distanz zwischen den Dreiecken und dem Projektionszentrum berechnet. Anschließend vergleicht das Verfahren jedes Dreieck mit jedem anderen Dreieck. Dabei wird das Dreieck mit dem größeren Abstand zum Projektionszentrum als das potentiell verdeckte Dreieck (im folgenden A genannt) und das Dreieck mit dem geringeren Abstand als das potentiell verdeckende Dreieck (im folgenden B genannt) markiert.

Falls alle Vertices von A in B enthalten sind, so wird das Dreieck A vollständig von B verdeckt. Im Gegensatz dazu ist das Dreieck A, wenn es während des Vorgangs von keinem anderen Dreieck verdeckt wird, vollständig sichtbar. Als dritte Möglichkeit, kann A auch teilweise von B verdeckt werden. Das Ergebnis des Maler-Algorithmus sind jedoch nur zwei Listen mit entweder vollständig sichtbaren oder vollständig verdeckten Dreiecken. Demnach muss in diesem Fall das teilweise verdeckte Dreieck A in weitere Dreiecke unterteilt werden, und zwar genau so, dass der verdeckte



**Abbildung 4.7:** Sichtbarkeitsanalyse des Maler Algorithmus. [Han09b]

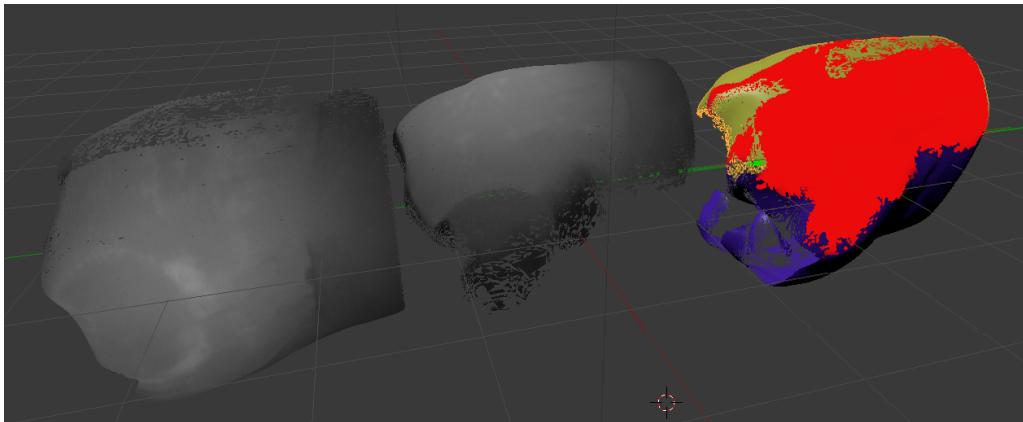
Teil des Dreiecks A in neue Dreiecke und der sichtbare Teil von A in weitere Dreiecke unterteilt wird. Der Vorgang der Unterteilung eines Dreiecks in mehrere kleinere Dreiecke wird Retriangulation genannt [Han09b].

Für die Retriangulation werden zunächst die überschneidenden Punkte der Dreiecke A und B in der Bildebene ermittelt und auf die Objektebene zurückgerechnet. Die Retriangulation erfolgt dann in der Objektebene, indem die originalen Vertices, die Schnittpunkte und die Punkte von B, die in A sind, verarbeitet werden, um die neuen Dreiecke zu erhalten. Dabei wird in [Han09b] der Delunay Triangulation Algorithmus verwendet, der aus einer Punktmenge ein Dreiecksnetz generiert (siehe [19b]). Die Subdivision des Dreiecks A ersetzt das originale Dreieck A im Datensatz. Nach dem Vorgang sind die Dreiecke in vollständig sichtbare und vollständig verdeckte Dreiecke retianguliert. Abbildung 4.7 veranschaulicht das Prinzip dieses Algorithmus grafisch.

**Ergebnis** Das Ergebnis der Sichtbarkeitsanalyse des Maler-Algorithmus besteht aus zwei Listen. Eine Liste hält die vollständig sichtbaren Dreiecke, die andere Liste die vollständig verdeckten Dreiecke. Teilweise verdeckte Dreiecke werden bei dem Verfahren eliminiert.

**Fazit** Dieses Verfahren ist vor allem für „unorientierte“ 3D-Modelle geeignet, also 3D-Modelle bei denen die Vertices nicht in der üblichen Reihenfolge, welche die Vorder- und Rückseite des Dreiecks definiert, vorhanden sind. Auch Löcher in dem Gitter des 3D-Modells beeinträchtigen das Ergebnis des Verfahrens nicht. [Han09b]

Der Maler-Algorithmus liefert zwar eine hohe Präzision und gute Ergebnisse bei komplexen Szenen, ist jedoch Rechenintensiv (Komplexität von  $O(n^2)$ ) [Han09b] [AH05]. Des weiteren handelt es sich bei dem Maler-Algorithmus um einen sehr komplexen Algorithmus, dessen Implementierung sehr aufwändig ist.



**Abbildung 4.8:** Überlappungsbereiche zweier Texturen.

Links und in der Mitte sind zwei NIR-Texturbereiche zu sehen, die zu einer Textur kombiniert werden sollen. Rechts ist diese Kombination farblich dargestellt. Eine Textur wurde dabei blau und die andere gelb markiert. Der rote Bereich zeigt an, wo sich die Texturen überlappen würden.

#### 4.2.1.4 Ergebnis der Sichtbarkeitsanalyse

Mit den Ergebnissen der Sichtbarkeitsanalyse können gezielt alle sichtbaren Primitive für eine Perspektive auf die Bildebene der Textur zurückgerechnet und somit ein Texture Mapping vorgenommen werden. Das Ergebnis sind einzelne Bereiche des 3D-Modells, die texturiert wurden. Die Texturen müssen anschließend durch ein geeignetes Verfahren kombiniert werden.

#### 4.2.2 Texturkombination

Bei der Kombination von mehreren Texturaufnahmen auf ein 3D-Modell entstehen zwangsläufig überlappende Bereiche, die von zwei oder mehreren Texturen gleichzeitig eingefärbt werden, wie in Abbildung 4.8 illustriert.

Primitive in diesem Bereich sind somit auf mehreren Texturen sichtbar und werden von diesen potentiell eingefärbt. Diese Bereiche müssen entsprechend behandelt werden, um eine willkürliche Texturierung der Primitiven in diesem Bereich, und somit ein schlechtes Ergebnis der Texturierung, zu vermeiden.

In diesem Kapitel werden die Möglichkeiten beschrieben, diese Bereiche entsprechend zu behandeln und somit eine vollständige Texturierung des 3D-Modells mit den Texturaufnahmen zu erhalten.

##### 4.2.2.1 Texturkombination durch Mittelwertbildung

Bei der Texturkombination durch Mittelwertbildung werden die individuellen Farbwerte, die ein Primitiv von verschiedenen Texturen erhält (also auf diesen sichtbar ist), durch eine Mittelwertbildung kombiniert. Der Mittelwert der Farbwerte wird meist durch Gewichtungen, wie beispielsweise die Auflösung des Primitivs in der Bildebene, die Distanz und dem Winkel des Primitivs zu der Kameraposition der Texturaufnahme, etc., berechnet.

**Gewichtungen nach Neugebauer et al.** Neugebauer et al. führen hierfür in [NK99] drei verschiedene Gewichtungen ein. Im Folgenden wird für einen Punkt  $p$  im Bild  $i$  die Position in der Bildebene als  $q_i$  und dessen Farbwert in Bild  $i$  an Position  $q_i$  als  $c_i$  bezeichnet.

Neugebauer et al. definieren zunächst eine Gewichtung, welche die Auflösungsverteilung beschreiben soll. Hierfür werden zunächst am Oberflächenpunkt  $p$  des 3D-Modells zwei tangentiale Vektoren bestimmt, die bei der Rückprojektion in die Bildebene die Vektoren  $(0, 1)^T$  und  $(1, 0)^T$  liefern. Die Fläche, die von den tangentialen Vektoren in Weltkoordinaten aufgespannt wird ist proportional zu der Auflösungsverteilung auf der Oberfläche. Die Berechnung der euklidischen Länge des Kreuzproduktes dieser tangentialen Vektoren liefert schließlich die Metrik  $w_q$  der Auflösungsverteilung.

Um einen weichen Übergang zwischen den Bildern zu erreichen, führen Neugebauer et al. eine weitere Gewichtung  $w_d$  auf Basis der 2D-Distanz zu dem Punkt, der auf dem Bild nicht sichtbar ist und am nächsten zu  $q_i$  ist, ein (siehe [NK99]).

Als dritten Parameter bestimmen Neugebauer et al. die Gewichtung  $w_t$ , um Fehler zu eliminieren, die beispielsweise bei fehlerhafter Registrierung oder optischen Effekten auftreten. Hierfür wird ein M-Schätzer definiert, der den Durchschnitt  $\mu_c$  und die Varianz  $\omega_c$  der Farbwerte gewichtet nach  $w_i \cdot w_q$  berechnet (siehe [NK99]). Der berechnete Wert bildet dann die Gewichtung  $w_t$ . Die Kombination der drei Gewichtungen für einen Punkt  $q_i$ , der sichtbar in Bild  $i$  mit dem Farbwert  $c_i$  ist, lautet demnach wie folgt:

$$w_q(q_i) \cdot w_d(q_i) \cdot w_t \left( \frac{c_i - \mu_c}{\omega_c} \right) \quad (4.18)$$

Durch diese Gewichtungen erreichen Neugebauer et al., dass sowohl die Kanten geglättet werden, als auch optische Artefakte entfernt werden.

Es ist jedoch möglich, andere geeignete Gewichtungen zu verwenden. Grammatikopoulos et al. verwenden beispielsweise eine Kombination aus der Skalierung der Textur (Distanz der Kamera zum 3D-Modell), den Winkel des Primitivs zum Bild und der Auflösung der Aufnahme.

Der Mittelwert berechnet sich, indem alle individuellen (gewichteten) Farbwerte addiert werden und die Summe durch die Anzahl der Farbwerte dividiert wird.

**Fazit** Das Ergebnis ist ein texturiertes 3D-Modell mit einem weichen visuellen Eindruck. Farb- und Helligkeitsunterschiede der verschiedenen Texturen werden bereits durch die Mittelwertbildung an den Übergängen eliminiert.

Ein Nachteil des Verfahrens ist, dass die hochfrequenten Informationen des Bildes entfernt werden. Unter der Frequenz eines Bildes versteht man die Änderungsrate der Intensität der Pixel im Bild. Die hochfrequenten Bildanteile treten primär an raschen Bildübergängen auf [BB15]. Niederfrequente Bildanteile treten bei weichen Bildübergängen auf. Die Frequenzen eines Bildes sind für den Schärfeeindruck des Bilds verantwortlich [BB15].

Des Weiteren kann es bei diesem Verfahren bei ungenauen Kameraparametern oder Fehler im 3D-Modell zu einer sichtbaren Überlagerungen der Textur kommen.

## 4 Stand der Technik

---

Der Rechenaufwand ist abhängig von den gewählten Gewichtungen, ist aber im Vergleich zu anderen Texturierungsverfahren moderat.

### 4.2.2.2 Iterative Texturkombination

Ein weiterer Ansatz wird in [AH05] beschrieben. Bei diesem Ansatz werden nacheinander alle Texturen für die Texturierung verwendet. Zunächst wird die Textur ausgewählt, auf der die meisten Primitive des 3D-Modells sichtbar sind. Alle sichtbaren Primitive werden anschließend mit dieser Textur texturiert. Daraufhin wird die nächste Textur gewählt und alle, auf dieser Textur sichtbaren Primitive eingefärbt, die im vorherigen Schritt noch nicht eingefärbt wurden. Dieser Vorgang wird dann für alle Texturen wiederholt, um möglichst viele Primitive einzufärben.

Bei diesem Verfahren wird das 3D-Modell zwar durch möglichst viele zusammenhängende Texturen texturiert, jedoch werden Primitive von Texturen eingefärbt, die von anderen Texturen aus besser eingefärbt werden könnten, da die Primitive orthogonaler zu diesen stehen. Somit gehen Bildinformationen verloren.

Dieses Verfahren ist demnach nur für Anwendungsfälle geeignet, bei der nur wenige Texturen vorhanden sind oder nur eine Seite des 3D-Modells texturiert werden soll (bspw. bei Gebäudefassaden). Ein Aspekt, der dennoch für diesen Ansatz sprechen kann, ist der verhältnismäßig geringe Rechenaufwand und eine einfache Implementierung.

### 4.2.2.3 Texturauswahl pro Primitiv

Das Verfahren der Texturauswahl pro Primitiv vermeidet den Verlust der hochfrequenten Informationen des Bildes, indem keine Mittelwertbildung der Texturen vorgenommen wird. Stattdessen wird jeweils nur eine Datenquelle pro Primitiv verwendet. Daraus resultierend werden die hochfrequenten Informationen erhalten.

Dazu wird über alle Primitive, die von mehreren Texturen eingefärbt werden, iteriert und anschließend die Textur als farbgebend für das Primitiv gewählt, die das Primitiv am geeigneten einfärbt.

Hierfür müssen Parameter eingeführt werden, welche die Qualität einer Textur in Bezug zu dem Primitiv beschreibt, das eingefärbt werden soll. Die Textur, für die dieser Parameter maximal ist, wird dann als Datenquelle für die Texturierung des Primitivs verwendet.

Im Allgemeinen werden hierfür zwei Qualitätsparamater verwendet:

- Der Winkel der Normalen zur orthogonalen Achse.
- Die Anzahl der Pixel, die das Primitiv auf der Textur einnimmt.

**Winkel der Normalen zur orthogonalen Achse** Der erste Qualitätsparameter beschreibt die Qualität anhand der Distanz eines Primitivs zum Projektionszentrum und dessen Winkel des Normalenvektors zur orthogonalen Achse. Die Kombination dieser Werte definiert, welche Textur das Primitiv am geeignetsten einfärbt [Han09b]. Diese Textur wird daraufhin als Quelle für dieses Primitiv gewählt.

Der Vorteil dieses Verfahrens ist, dass es sehr gut nachvollziehbar ist. Jedoch spielt die Ausrichtung der Normalen bei diesem Parameter eine Rolle. Vor allem bei 3D-Scannern kommt es häufig vor, dass sich durch Risse oder andere Begebenheiten der Oberfläche des 3D-Modells die Orientierung der Normalen unerwartet ändert. Dies führt dann dazu, dass bei diesem Ansatz die falsche Textur für die Texturierung dieses Primitivs ausgewählt wird. [FSZ04]

Des weiteren färbt die Textur, mit dem besten Winkel und der geringsten Distanz zum Primitiv, dieses nicht immer am geeignetsten ein.

**Anzahl der Pixel, die das Primitiv auf der Textur einnimmt** Der zweite Qualitätsparameter beschreibt die Qualität einer Textur für ein Primitiv anhand der Dimension des projizierten Primitivs in der Bildebene. Dieses Verfahren geht davon aus, dass jene Textur ein Primitiv am geeignetsten einfärbt, welche dem, auf die Bildebene projizierten Primitiv, die meisten Pixel und somit den größten Informationsgehalt liefert.

Der Vorteil dieses Qualitätsparameters, im Vergleich zu dem ersten Qualitätsparameter ist, dass die Ausrichtung der Normalen keine Rolle spielt.

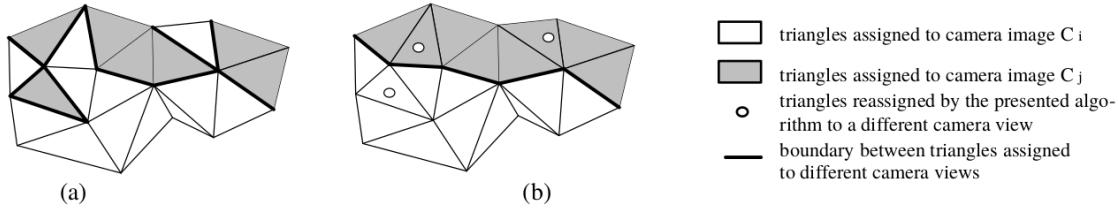
**Fazit** Obwohl dieses Verfahren eine bestmögliche Texturierung einzelner Primitive garantiert, kann das Verfahren jedoch dazu resultieren, dass angrenzende Primitive häufig von unterschiedlichen Texturen texturiert werden. Dies führt bereits bei geringen Ungenauigkeiten der Kameraparameter und unterschiedlichen Belichtungen zu Artefakten, wie sichtbare Kanten oder einzeln sichtbare Primitive, die daraufhin bereinigt werden müssen. [EGP+03]

### 4.2.3 Artefakte

Bei der Texturierung eines 3D-Modells mit mehreren Texturaufnahmen kann es (je nach Verfahren) vorkommen, dass visuelle Artefakte entstehen (siehe Abbildung 4.13). Gründe hierfür sind [NB95]:

- Fehler im 3D-Modell.
- Ungenaue Kameraparameter.
- Unterschiedliche Belichtungen.

Die Artefakte sind dabei, abhängig vom gewählten Verfahren, stärker oder schwächer ausgeprägt. Um ein gutes optisches Ergebnis zu erhalten, müssen diese Artefakte bereinigt werden. Dieses Kapitel beschreibt die häufigsten Artefakte, die auftreten können und wie diese bereinigt werden können.



**Abbildung 4.9:** Illustration der versetzte Kanten.

Dreiecke der Oberfläche des 3D-Modells, die unterschiedlichen Texturen  $C_i$  und  $C_j$  zugeordnet sind. (a) Versetzte Kanten, die bei der Texturzuordnung entstanden sind. (b) Bildung einer zusammenhängenden Kante durch Umgruppierung der Texturzuweisung. [NB95]

#### 4.2.3.1 Versetzte Kanten

Bei den Verfahren aus Abschnitt 4.2.2.3, der Texturauswahl pro Primitiv, kommt es häufig vor, dass die Kanten nicht zusammenhängend, sondern versetzt sind, wie in Abbildung 4.9a) illustriert. Dies führt dazu, dass die Übergänge zwischen den Texturen sichtbar sind und somit kein homogener visueller Eindruck entsteht.

Um einen besseren visuellen Eindruck an den Kanten zu erhalten, kann es deshalb von Vorteil sein, homogene Kanten zu bilden.

Niem et al. stellen hierfür in [NB95] einen Algorithmus vor, der die applizierten Texturen an den Kanten umsortiert, um eine homogene Kante zu erhalten. Der Algorithmus arbeitet dabei wie folgt:

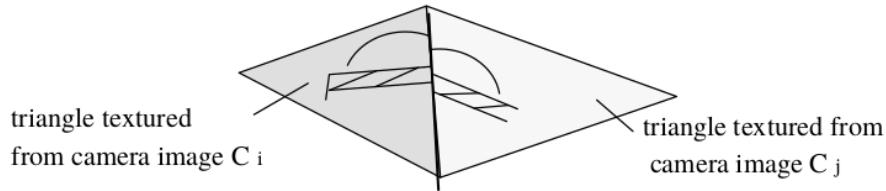
1. Ermitteln eines Dreiecks, das der Textur  $C_i$  zugeordnet ist und mit mindestens zwei Seiten an ein Dreieck angrenzt, das von einer anderen Textur  $C_j$  texturiert wird.
2. Berechnen der Texturauflösung (Pixelanzahl) für das gefundene Dreieck in dem Bild  $C_j$ .
3. Zuweisen der Textur  $C_j$  zu dem Dreieck, falls sich die Reduktion der Auflösung unterhalb einer bestimmten Toleranzgrenze befindet.

Das Ergebnis des Algorithmus ist sichtbar in Abbildung 4.9b). Durch die Umgruppierung wurden die versetzten Kanten von elf auf fünf Kanten reduziert. Die Oberflächenregionen sind somit homogener und die Kanten können besser geglättet werden.

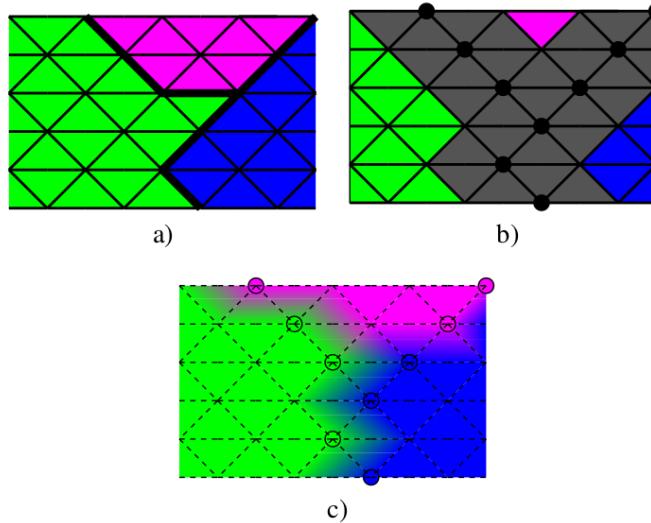
#### 4.2.3.2 Visuelle Diskontinuität

Bei den Übergängen zwischen den Bereichen, die von unterschiedlichen Texturen eingefärbt werden, kommt es häufig zu einer visuellen Diskontinuität. Dies tritt häufig aufgrund von ungenauen Kameraparametern und Fehlern im 3D-Modell ein [Bau03]. Dabei entsteht ein Versatz der Textur an den Kanten der Texturübergänge, wie in Abbildung 4.10 illustriert.

Ein Weg, um einen guten visuellen Eindruck des texturierten 3D-Modells an den Texturübergängen zu erhalten, ist es, an den Kanten weiche Übergänge zwischen den verschiedenen Texturen zu erzeugen. Dies wird dadurch erreicht, dass die Texturübergänge vermischt werden (engl. Blending). Hierfür wurden unterschiedliche Ansätze entwickelt [Bau03] [LHS00] [NB95].



**Abbildung 4.10:** Versatz der Textur an den Kanten von Dreiecken, die von unterschiedlichen Texturen eingefärbt wurden. [NB95]

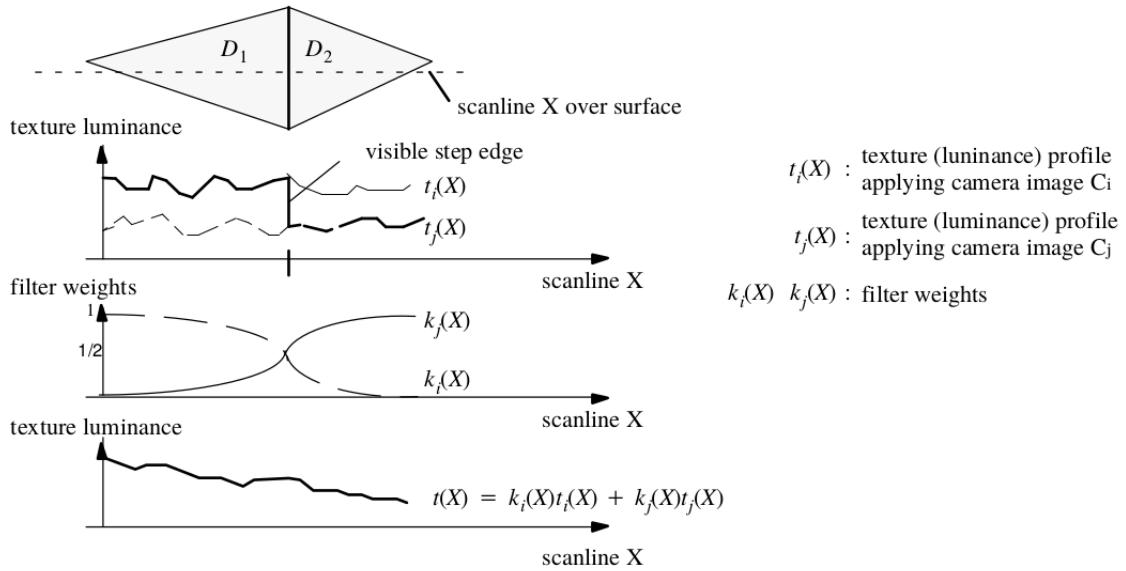


**Abbildung 4.11:** Prinzip des Blending. [LHS00]

- a) Angrenzende Dreiecke, die andere Texturen verwenden. b) Dreiecke, die für das Blending in Frage kommen, wurden grau eingefärbt. c) Ergebnis der Vermischung der Dreiecke.

Üblicherweise wird das Blending nur auf die angrenzende Primitive angewendet, die von unterschiedlichen Texturen eingefärbt wurden (siehe Abbildung 4.11) [Bau03]. Niem et al. projizierten dabei zwei angrenzende Primitive  $D_1$  und  $D_2$  zunächst auf die erste Aufnahme  $C_i$  und anschließend auf die zweite Aufnahme  $C_j$  [NB95]. Das Ergebnis sind zwei Texturen:  $t_i$  für die Projektion von  $D_1$  und  $D_2$  auf das Bild  $C_i$  und  $t_j$  für die Projektion von  $D_1$  und  $D_2$  auf das Bild  $C_j$ . Anschließend wird aus diesen zwei Texturen eine kombinierte Textur für die zwei Primitive  $D_1$  und  $D_2$  mit einem gewichteten Mittelwert erstellt:  $t = k_i(x, y, z) \cdot t_i + k_j(x, y, z) \cdot t_j$ . Die Gewichtungen werden dabei so gewählt, dass ein weicher Übergang entsteht (siehe Abbildung 4.12).

Dieses einfache Verfahren ist zwar effektiv, jedoch ist die Übergangszone, die vermischt wird, abhängig von der Größe und Form der Primitive. Die Dimension der Übergangszone ist jedoch ausschlaggebend für einen guten visuellen Effekt. Falls die Primitive verhältnismäßig klein sind, so ist auch die Übergangszone klein, was dazu führt, dass der Übergang zwischen den Texturen nur leicht vermischt wird und somit immer noch sichtbar ist. Bei großen Übergangszenen resultiert das Verfahren darin, dass hochfrequente Informationen eliminiert werden. Des Weiteren kann es dabei zu einem Überlagerungseffekt der Texturen kommen. [Bau03]



**Abbildung 4.12:** Mitteln der Textur für zwei angrenzende Dreiecke mit unterschiedlichen Texturen, sodass ein weicher Übergang entsteht. [NB95]

#### 4.2.3.3 Sichtbare Kanten aufgrund von Belichtungsunterschieden

Durch unterschiedliche Belichtungen entsteht der Effekt von sichtbaren Übergängen an den Kanten der Texturübergängen, wie in Abbildung 4.13 sichtbar.

Baumberg entwickelte ein Verfahren, um diese Belichtungsunterschiede auszugleichen. Das Verfahren wird von Baumberg als "Multi-band blending in 3D" bezeichnet. Hierfür verwendet Baumberg niederfrequente und hochfrequente Texturdaten.

Der Gesamtablauf des Algorithmus nach Baumberg ist wie folgt:

- Verwischen des aktuelle Kamerabilds, unter Verwendung eines Gaußfilters, um ein niederfrequentes Bild zu erzeugen.
- Subtrahieren des Originalbild, mit dem niederfrequenten Bild, um ein Bild mit den hochfrequenten Bilddaten zu erzeugen.
- Filtern des niederfrequenten Bildes mit einem gewichteten Mittelwertfilter, um die Farbunterschiede zu minimieren und Filtern des hochfrequenten Bildes mit einem nicht-linearen Filter, um die hochfrequenten Informationen zu erhalten.
- Kombinieren der gefilterten Texturen, um die finale Textur zu erhalten.

Das Ergebnis des Verfahrens wird in Abbildung 4.13 gezeigt. Der Ansatz, bei dem die Texturauswahl pro Primitiv erfolgt und keine Belichtungskorrektur vorgenommen wird, ist zwar schärfer, da die hochfrequenten Informationen nicht entfernt wurden, jedoch sind die Übergänge deutlich sichtbar.



**Abbildung 4.13:** Multi-band blending in 3D. [Bau03]

Links: Texturierte Puppe, erstellt mit dem „Texturauswahl pro Primitiv“ Verfahren aus Abschnitt 4.2.2.3. Die Kanten sind aufgrund von Belichtungsunterschieden deutlich zu erkennen. Mitte: Texturierte Puppe mit gewichteten Mittelwertverfahren wie in Abschnitt 4.2.2.1 beschrieben. Die Textur ist leicht verwischt. Hochfrequente Informationen sind verloren gegangen. Rechts: Multiband Blending angewandt auf die Textur aus dem Verfahren aus Abschnitt 4.2.2.3. Die Kanten erscheinen scharf, die Übergänge sind weich - die Belichtung wurde zudem durch das Verfahren angepasst.

#### 4.2.4 Fazit

In diesem Kapitel wurde erklärt, welche Verfahren zur Sichtbarkeitsanalyse es gibt und wie die überlappenden Bereiche, die bei der Projektion der Texturen auf das 3D-Modell entstehen, behandelt werden. Mit diesem Wissen können die Texturen für die Primitiven bestimmt und die Texturkoordinaten, mit Hilfe der Kameraparameter, zugewiesen werden. Als Ergebnis erhält man demnach ein, mit mehreren Texturaufnahmen, texturiertes 3D-Modell mit einem guten visuellen Eindruck.



# 5 Machbarkeitsanalyse

Wie bereits erwähnt, sind die intrinsischen und extrinsischen Kameraparameter für die NIR-Aufnahmen nicht bestimmt (vgl. Abschnitt 2.3.2). Diese müssen jedoch für die Texturierung zwingend gegeben sein und müssen daher durch ein Verfahren, wie in Abschnitt 4.1 beschrieben, ermittelt werden.

Dadurch, dass die Kalibrierungsverfahren die Ähnlichkeit der Aufnahme mit dem 3D-Modell vergleichen, könnte der (in Hinsicht auf die Intensitätswerte) geringe Informationsgehalt der NIR-Aufnahmen, verbunden mit der geringen Auflösung der Aufnahmen problematisch sein.

Bei einem Zahn handelt es sich um ein Objekt mit vergleichsweise wenig Oberflächendetails (bspw. verglichen zu einer Statue, Lebewesen, etc.). Das 3D-Modell bietet daher nur wenige entscheidende Merkmale, mit der die Ähnlichkeit des 3D-Modells, für eine bestimmte Perspektive, zu einer NIR-Aufnahme, im Vergleich zu den anderen Aufnahmen, bestimmt werden könnte. Umso wichtiger ist es, dass die NIR-Aufnahmen einen hohen Informationsgehalt in Form von scharfen Kanten, bzw. deutlich sichtbaren Strukturen bietet.

Das gegebene 3D-Modell weiß jedoch nur wenige Oberflächendetails auf, mit der die Ähnlichkeit einer perspektivischen Projektion des 3D-Modells zu einer NIR-Aufnahme bestimmt werden könnte. Des Weiteren ist es fraglich, ob die NIR-Aufnahmen genug Informationsgehalt liefern, um einen geeigneten Vergleich mit dem 3D-Modell anzustellen. Eine Machbarkeitsanalyse wäre demnach sinnvoll, um zu evaluieren, ob die Kalibrierung mit den aktuellen Erkenntnissen und Methoden realisiert werden kann.

Dieses Kapitel beschreibt die durchgeführte Machbarkeitsanalyse und die Konsequenz aus den Erkenntnissen.

## 5.1 Verfahrensanalyse

Wie in Abschnitt 4.1 beschrieben, können die Kameraparameter entweder anhand von korrespondierenden Punkten oder durch eine 3D/2D-Registrierung angenähert werden. In diesem Abschnitt wird zunächst untersucht, welche der Verfahren für eine Bestimmung der Kameraparameter für die gegebenen Daten geeignet ist.

**Kalibrierung durch korrespondierende Punkte** Betrachtet man die gegebenen NIR-Daten, wie in Abschnitt 3.1.2 geschildert, so kann die Ermittlung der Kameradaten anhand korrespondierender Punkte für diesen Fall ausgeschlossen werden. Der Grund hierfür ist offensichtlich: die Aufnahmen wurden ohne Kalibrierungsmuster aufgenommen. Nur mit dem Kalibrierungsmuster oder einem ähnlichen Bezug wäre es möglich, die korrespondierenden Punkte zu bestimmen, mit denen anschließend die intrinsischen und extrinsischen Kameraparameter berechnet werden könnten.



**Abbildung 5.1:** Darstellung verschiedener Zähne. [19g]  
Links: Eckzahn; Mitte: Backenzahn; Rechts: Schneidezahn.

**Kalibrierung durch 3D/2D-Bildregistrierung** Die Bestimmung der Kameraparameter anhand 3D/2D-Bildregistrierung ist, in Anbetracht der gegebenen Daten, die vielversprechendste Methode. Jedoch muss untersucht werden, welches Ähnlichkeitsmaß für die Bestimmung der Ähnlichkeit einer perspektivischen Projektion des 3D-Modells und einer Aufnahme, in Anbetracht des Anwendungsfalls, herangezogen werden kann.

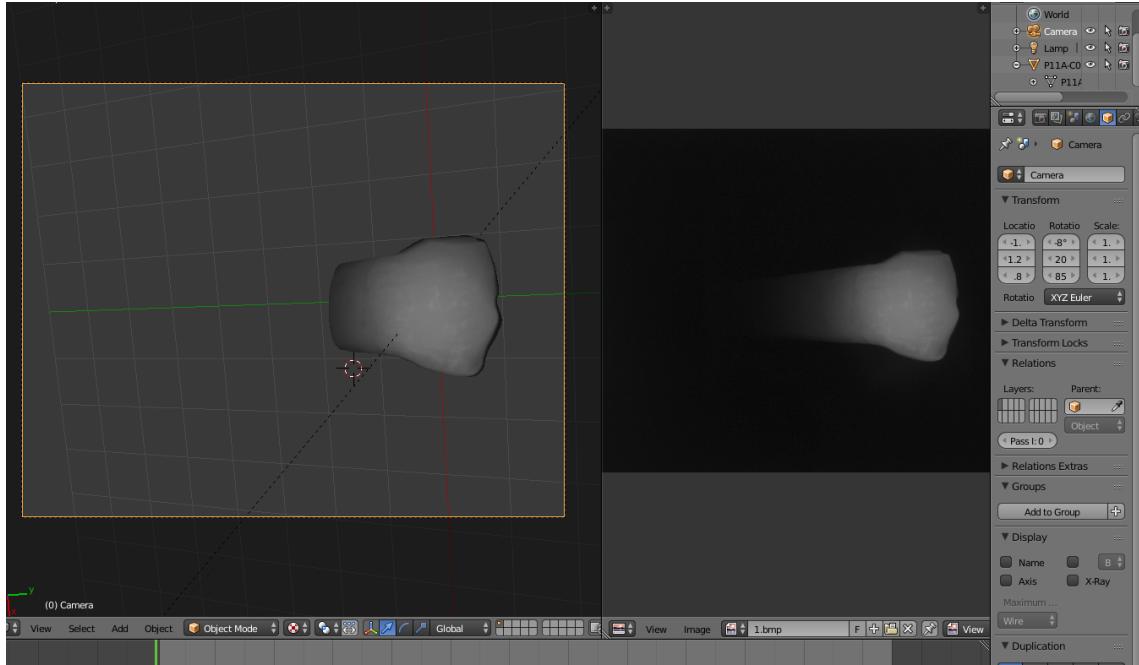
## 5.2 Kanten und Silhouetten

Betrachtet man die Zähne aus Abbildung 5.1 wird klar, dass ein Ähnlichkeitsmaß anhand von Kanten oder Silhouetten in der Praxis nicht möglich ist. Das liegt zum einen daran, dass vor allem bei Schneide- oder Eckzähnen die Silhouetten und Kanten für die Vorder- und Rückseite nahezu identisch sind. Somit kann z.B. für eine Aufnahme der Vorderseite eines Eckzahns, kein eindeutiges Ähnlichkeitsmaß zwischen der perspektivischen Projektion des 3D-Modells von der Vorder- oder Rückseite und der Aufnahme bestimmt werden.

Zum anderen werden die Zähne bei einem späteren, praktischen Anwendungsfall nicht *in vitro*, sondern *in vivo* untersucht. Demnach wäre die Wurzel des Zahnes auf der NIR-Aufnahme nicht sichtbar, was eine Kamerakalibrierung mit 3D/2D-Bildregistrierung anhand von Kanten- oder Silhouettenmerkmalen nahezu unmöglich macht. Diese Ähnlichkeitsmaße können daher in diesem konkreten Fall und somit für diese Arbeit ausgeschlossen werden.

## 5.3 Mutual Information (MI)

Nachdem sich Kanten und Silhouetten als unzureichendes Ähnlichkeitsmaß herausgestellt haben, muss die Mutual Information (MI) als potentielles Ähnlichkeitsmaß untersucht werden. Da die Frage der Eignung nicht ohne einer Überprüfung, anhand einer Implementierung, durchgeführt werden



**Abbildung 5.2:** Manuelle Bestimmung der Kameraposition für eine NIR-Aufnahme mit Blender.

kann, wurde eine prototypische Implementierung entwickelt, welche die MI für ein gerendertes Bild und der gegebenen NIR-Aufnahme berechnet. Dieser Abschnitt beschreibt die Details der Implementierung, sowie die Ergebnisse, die erzielt wurden.

### 5.3.1 Erzeugen der perspektivischen Aufnahmen

Um die aufwändige Implementierung der Generierung der perspektivischen Aufnahmen, wie in ?? beschrieben, zu vermeiden und somit den Fokus auf die Machbarkeit der Ähnlichkeitsbestimmung zu setzen, wurden die Kameraparameter einiger Aufnahmen manuell mit Hilfe der Grafiksuite Blender bestimmt.

Hierfür wurde das 3D-Modell zunächst aus einer bestimmten Kameraperspektive auf die NIR-Aufnahme abgebildet. Durch einen subjektiven Vergleich wurde daraufhin versucht, durch mehrfaches Ändern der Kameraparameter, die Perspektive, aus der die NIR-Aufnahme gemacht wurde, möglichst gut zu bestimmen. Abbildung 5.2 zeigt den Vorgang der manuellen Kalibrierung in Blender.

Nachdem die Kameraperspektive manuell bestimmt wurde, wird das 3D-Modell aus dieser Perspektive (ohne Textur) gerendert. Insgesamt wurden vier NIR-Aufnahmen mit dem 3D-Modell gemappt, die später untersucht werden sollen. Dabei handelt es sich um drei Seitenansichten und eine Draufsicht des Zahnes. Die gerenderten Bilder werden dann jeweils mit allen NIR-Aufnahmen verglichen und die Ähnlichkeit bestimmt.

### 5.3.2 Implementierung

Der nächste Schritt ist die Berechnung der Ähnlichkeit des gerenderten Bildes mit den NIR-Aufnahmen. Die Berechnung der MI ist in Abschnitt 4.1.3.3 beschrieben. Wie dort bereits erwähnt, lässt sich die MI für Bilddaten über deren Histogramm berechnen. Dieses Kapitel beschreibt zunächst, wie der Ansatz für die Bilddaten adaptiert und in Python implementiert wurde.

#### 5.3.2.1 Gemeinsame Wahrscheinlichkeit

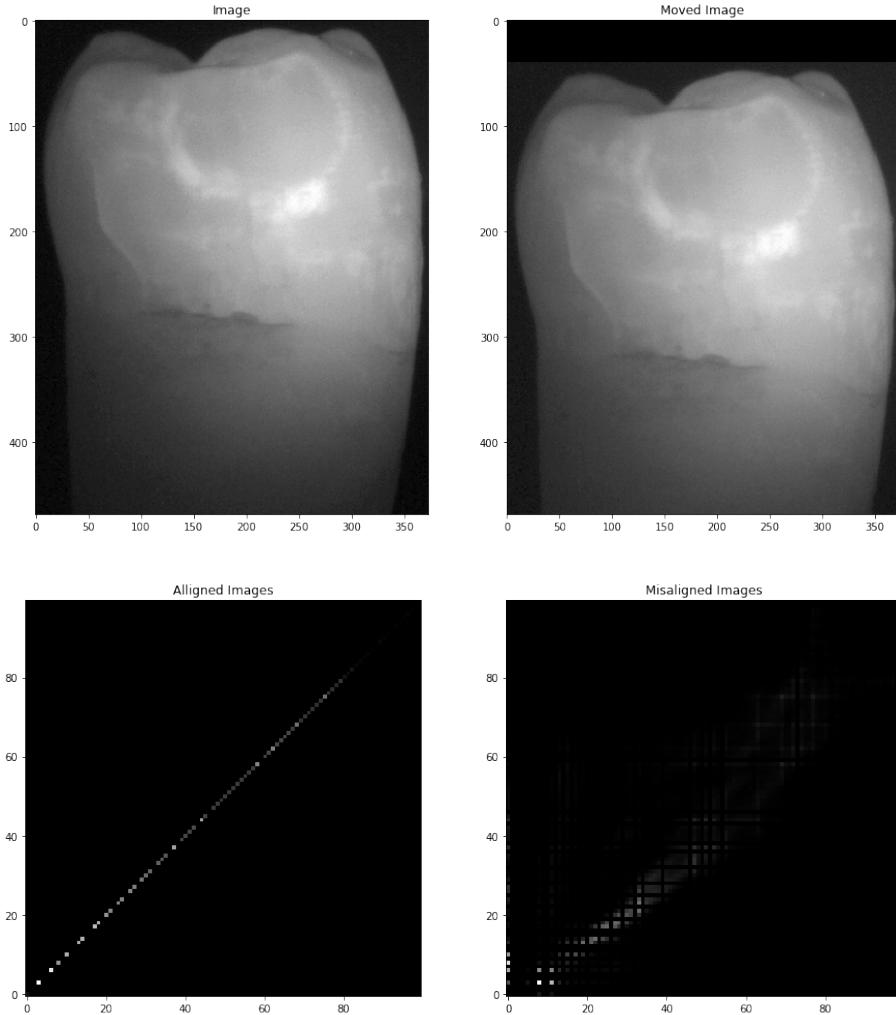
Für die Berechnung der MI  $I(X; Y)$  (Gleichung (4.15)) für zwei Bilddaten  $X$  und  $Y$  müssen zum einen die Wahrscheinlichkeiten  $P(X)$  und  $P(Y)$  sowie die gemeinsame Wahrscheinlichkeit  $P(X, Y)$  der Bilder berechnet werden. Wahrscheinlichkeit im Kontext von Bilddaten bedeutet, mit welcher Wahrscheinlichkeit die Intensitätswerte der Bilder auftreten. Diese Wahrscheinlichkeiten werden über Histogramme berechnet.

Die Wahrscheinlichkeitsmatrix  $P(X)$  eines Bildes  $X$  lässt sich über das Histogramm  $H$  des Bildes berechnen, indem jeder Eintrag  $h(x)$  von  $H$  durch die Summe der Einträge (Pixelanzahl) des Bildes geteilt wird.

Um die gemeinsame Wahrscheinlichkeit zweier Bilder  $X$  und  $Y$  zu berechnen, muss zunächst ein gemeinsames Histogramm (engl. Joint Histogram) erstellt werden. Das gemeinsame Histogramm ist ein 2D-Histogramm, bei dem die erste Dimension die Intensitätswerte des ersten Bildes und die zweite Dimension die Intensitätswerte des zweiten Bildes enthält. Bei zwei 8-Bit-Grauwertbildern  $X$  und  $Y$  besteht das gemeinsame Histogramm  $H$  demnach aus  $K = 256 \times 256$  Einträgen. Der Wert des gemeinsamen Histogramms beträgt an der Stelle  $h(i, j)$  die Anzahl der Pixel mit dem Intensitätswert  $i$  des ersten Bildes  $X$  und dem Intensitätswert  $j$  des zweiten Bildes  $Y$ . Beispielsweise würde der Eintrag  $h(7, 3) = 2$  bedeuten, dass es zwei Pixel an der gleichen Stelle in den zwei Bildern gibt, für die der Intensitätswert im ersten Bild sieben ist im zweiten Bild drei beträgt.

Die Berechnung des gemeinsamen Histogramms für die zwei 8-Bit-Grauwertbilder  $X$  und  $Y$  der selben Größe folgt folgenden Schritten:

1. Erstellen einer Matrix  $H$  mit  $K = 256 \times 256$  Einträgen. Die Einträge werden dabei mit Nullen initialisiert.
2. Iteration über alle Pixel der beiden Bilder, ausgehend von dem oberen linken Pixel. Für jeden Pixel werden folgende Schritte ausgeführt:
  - a) Herauslesen des Intensitätswert  $i$  für den Pixel des ersten Bildes. Dieser Wert beschreibt den Zeileneintrag von  $h$  an der Stelle  $i$ .
  - b) Herauslesen des Intensitätswert  $j$  für den Pixel des zweiten Bildes. Dieser Wert beschreibt den Spalteneintrag von  $h$  an der Stelle  $j$ .
  - c) Inkrementieren des Wertes an der Stelle  $h(i, j)$  der Matrix  $H$  um 1.



(a) Gemeinsames Histogramm eines Bildes (b) Gemeinsames Histogramm des Originalbildes mit dem verschobenen Bild.

**Abbildung 5.3:** Gemeinsames Histogramm von 8-Bit-Grauwertbildern.

Das gemeinsame Histogramm gibt Auskunft, inwiefern die Intensitätswerte der Bilder verteilt sind. Abbildung 5.3a zeigt das gemeinsame Histogramm für zwei Bilder, die perfekt registriert sind - also im Grunde identisch sind. Die Werte sind in diesem Fall im gemeinsamen Histogramm diagonal verteilt. In Abbildung 5.3b wurde das Bild verschoben. Dadurch werden die Werte weiter verstreut.

Die Erstellung des gemeinsamen Histogramms übernimmt bei der Implementierung das Framework Numerisches Python (NumPy), durch einen Aufruf der Funktion `np.histogram2d(img.ravel(), img_moved.ravel(), bins=100)`. Der Parameter `bins=100` beschreibt dabei, dass die Werte, die in einem bestimmten Intervall liegen, für eine bessere Visualisierung zusammengefasst werden sollen. Die Größe eines Intervalls ist dabei 100.

## 5 Machbarkeitsanalyse

---

### **Listing 5.1** Berechnung der normalisierten MI aus dem gemeinsamen Histogramm zweier Bilddaten.

```
def normalized_mutual_information(hgram):
    """ Normalized Mutual information for joint histogram
    """
    # Convert bins counts to probability values
    pxy = hgram / float(np.sum(hgram))
    px = np.sum(pxy, axis=1) # marginal for x over y
    pxh = - np.sum(px*np.log(px))
    py = np.sum(pxy, axis=0) # marginal for y over x
    pyh = - np.sum(py*np.log(py))
    px_py = px[:, None] * py[None, :] # Broadcast to multiply marginals
    # Calculate MI
    nzs = pxy > 0 # Prevent zero pxy values
    mi = np.sum(pxy[nzs] * np.log(pxy[nzs] / px_py[nzs]))
    return mi / np.sqrt(pyh * pyh)
```

---

#### 5.3.2.2 MI

Aus dem gemeinsamen Histogramm kann somit die gemeinsame Wahrscheinlichkeitsmatrix  $P(X, Y)$  berechnet werden, indem jeder Eintrag des gemeinsamen Histogramms durch die Summe der Elemente (Pixel) eines der beiden Bilder geteilt wird. Mit den Wahrscheinlichkeitsmatrizen  $P(X)$ ,  $P(Y)$  und  $P(X, Y)$  ist es nunmehr möglich, die MI  $I(X; Y)$  für die beiden Bilder  $X$  und  $Y$  zu bestimmen. Die Implementierung der Funktion für die Berechnung der normalisierten MI ist in Listing 5.1 zu sehen.

Die normalisierte MI beträgt beispielsweise für die Bilder aus Abbildung 5.3a durch die exakte Überlagerung  $NI(X; Y) = 1.0$  und für Abbildung 5.3b durch die Verschiebung  $NI(X; Y) \approx 0.043732$ .

#### 5.3.3 Ergebnisse

Tabelle 5.1 zeigt einen Auszug der Ergebnisse für die Berechnung der MI<sup>1</sup>. Die grünen Felder zeigen dabei das Ergebnis der MI für das gerenderte Bild und die zugehörige NIR-Aufnahme, die wie in Abschnitt 5.3.1 beschrieben manuell aufeinander gemappt wurden. Die roten Felder markieren die MI, die für andere NIR-Aufnahmen einen höheren (oder gleich hohen) Wert erzielt haben.

#### 5.3.4 Auswertung

Der Wert der MI sollte für alle NIR-Aufnahmen und der zugehörigen gerenderten Aufnahme maximal sein, da (trotz manueller Kalibrierung) die Perspektive des gerenderten Bildes für diese NIR-Aufnahme am besten übereinstimmt und somit die Ähnlichkeit der beiden Aufnahmen am

---

<sup>1</sup>Tabelle .1 im Anhang zeigt sämtliche Ergebnisse der MI-Berechnung

	0,589	0,373	0,392
	0,415	0,333	0,339
	0,315	0,379	0,362
	0,278	0,379	0,333
	0,328	0,378	0,385
	0,335	0,358	0,376
	0,341	0,436	0,413

**Tabelle 5.1:** Auszug der Ergebnisse der normalisierten MI Berechnung.

Zeilen: NIR-Aufnahmen. Spalten: Gerendertes 3D-Modell. Grüne Felder: MI der NIR-Aufnahme zur zugehörigen gerenderten Ansicht. Rote Felder: Fälschlicherweise höhere oder gleich hohe MI für andere Ansichten.

## 5 Machbarkeitsanalyse

---

größten sein sollte. Dies ist jedoch für einige Aufnahmen nicht der Fall, wie die rot markierten Felder in Tabelle 5.1 zeigen. In diesen Fällen ist eine eindeutige Bestimmung der Ähnlichkeit einer NIR-Aufnahme zu einer bestimmten Perspektive nicht möglich.

Die MI ist demnach aufgrund von nicht eindeutigen Maxima, als Ähnlichkeitsmaß bei der Kalibrierung der Kameraparameter nicht geeignet. Des Weiteren kann argumentiert werden, dass vor allem in Hinsicht darauf, dass diese Analyse im Rahmen eines Testszenarios, mit optimalen Bedingungen durchgeführt wurde, ein praktischer Einsatz unter erschwerten Bedingungen nicht möglich ist.

### 5.4 Erkenntnisse und Konsequenzen

Aus der durchgeführten Machbarkeitsanalyse der Kamerakalibrierung wurde die Erkenntnis gewonnen, dass durch die allgemein bekannten und in dieser Arbeit vorgestellten automatisierten Verfahren keine eindeutige Kalibrierung vorgenommen werden kann. Dies liegt daran, dass sich die untersuchten Ähnlichkeitsmaße, sowohl für diesen definierten Testfall, als auch bei einem Einsatz in der Praxis, nicht eignen.

Eine Kalibrierung der Kameraparameter wäre demnach beispielsweise durch ein vorhandenes Kalibrierungsmuster und durch das Verfahren der korrespondierenden Punkte wie in Abschnitt 4.1.1 beschrieben möglich. Alternativ könnten im produktiven Einsatz durch Trackingsysteme, wie dem Polaris System, die Kameraparameter bereits bei der Datenerfassung bestimmt werden. Das Polaris System ist in der Lage, die Position und Orientierung eines Objektes (in diesem konkreten Fall der NIR-Kamera) anhand von aktiven oder passiven Markern zu bestimmen (siehe [19e]).

Da die Kalibrierung in dem gegebenen Datensatz nicht vorgenommen werden kann, müssen in dieser Arbeit die Kameraparameter, wie in Abschnitt 5.3.1 geschildert, manuell bestimmt werden, um eine Texturierung zu ermöglichen. Dadurch kommt es zwar für diesen konkreten Anwendungsfall, mit den gegebenen Daten, zwangsläufig zu ungenauen Ergebnissen - für eine Demonstration des prototypisch entwickelten Verfahrens ist dies jedoch ausreichend.

# 6 Implementierung

In diesem Kapitel wird auf die konkrete Implementierung des Prototyps für die 3D-Visualisierung von Kariesläsionen eingegangen. Dazu werden zunächst die nicht-funktionalen Anforderungen an die Software definiert. Daraufhin werden die verwendete Software und die externen Software-Komponenten, die in den Prototyp eingebunden wurden, aufgeführt. Anschließend werden die bei der Entwicklung durchgeführten Maßnahmen zur Sicherung der Qualität aufgezeigt. Abschließend werden Details zur Implementierung einiger ausgewählter Komponenten erläutert.

## 6.1 Verwendete Software

Folgende Software wurde im Rahmen der Entwicklung des Prototyps genutzt:

**Blender** Für die Datengenerierung wurde die Grafiksuite Blender benutzt [Ble18]. Mit Hilfe der Software konnten die NIR-Aufnahmen manuell auf das 3D-Modell gemappt werden und somit die Kameraparameter grob bestimmt werden. Zudem wurden vom Prototyp generierte 3D-Modelle in Blender für eine Überprüfung importiert und visualisiert.

**Vizard** Für die Visualisierung und die Darstellung des Ergebnisses in Virtueller Realität (VR) wurde das Programm Vizard von WorldViz verwendet. Vizard bietet die Möglichkeit, in einer eigenen Entwicklungsumgebung Python Skripte zu erstellen, die auf die Vizard-API zurückgreifen. Somit können einfach komplexe dreidimensionale Szenen definiert und über einen Renderer visualisiert werden. Vizard unterstützt die Anbindung verschiedener Hardwarekomponenten, wie VR-Brillen, Powerwalls, Tracker für Motion-Capturing, etc.

**Anaconda** Das Anaconda Modul bietet einen einfachen Weg, um Data-Science und Machine-Learning für die Programmiersprachen Python und R zu nutzen. Der Fokus von Anaconda liegt auf der Vereinfachung des Paketmanagements und der Softwareverteilung. Hierfür stellt Anaconda das Paketmanagement-Tool „conda“ und die Möglichkeit, virtuelle Umgebungen aufzusetzen, bereit.

**pip** „Pip“ ist ein Paketverwaltungsprogramm für Python. Da die Module PyWavefront und pycollada über den Paketmanager von Anaconda nicht direkt verfügbar sind, wurden diese über „pip“ in die Umgebung eingebunden.

## 6 Implementierung

---

**PyPI** Python Package Index (PyPI) ist eine Verwaltungsplattform für Software, die mit der Python Programmiersprache geschrieben wurden [19f]. Mit Hilfe von PyPI können Entwickler ihre Software verteilen.

**twine** Das Ziel von twine ist es, eine einfache und sichere Interaktion mit PyPI zu gewährleisten. Mit Hilfe von twine können somit Python-Pakete, nach einer HTTPS-Authentifizierung, auf PyPI hochgeladen werden.

### 6.2 Verwendete Softwarekomponenten

Für die Implementierung wurde die Programmiersprache Python in der Version 3.5.5, sowie folgende Softwarekomponenten verwendet:

**NumPy** Bei NumPy handelt es sich um ein Framework für Python, das mathematische und numerische Funktionen und Funktionalitäten bereit stellt. Das Erweiterungsmodul wurde zum größten Teil in C geschrieben, was eine größtmögliche Ausführungsgeschwindigkeit der Funktionen garantiert. Numpy zeichnet sich vor allem für dessen multidimensionale Arrays und die bereitgestellten Routinen zur Bearbeitung dieser aus. [18b]

**PyWavefront** PyWavefront ermöglicht das Importieren von Wavefront Dateien und generiert für jedes Material die zugehörigen Daten.

**pycollada** Pycollada ist ein Python-Modul, um Collada-Dateien zu erstellen, bearbeiten und importieren. Das Modul erlaubt es, die Collada-Datei als Python-Objekt zu editieren.

**pydoc** Mit dem pydoc-Modul kann eine Dokumentation von Python Modulen erstellt werden. Die Dokumentation kann sowohl als Text in der Konsole oder in einem Webbrowser angezeigt werden, als auch im HTML-Format abgespeichert werden. Die Dokumentation für Klassen und Funktionen können dabei direkt im Code als Kommentare definiert werden.

**Click** Click steht für Command Line Interface Creation Kit und ist ein Python Paket zur Erstellung von Command-line Interfaces (CLIs) [19i]. Mit Hilfe des Pakets konnte durch eine einfache Annotationsschreibweise das CLI für den Prototypen entwickelt werden.

## 6.3 Qualitätssicherung

Die Maßnahmen zur Qualitätssicherung stützen sich auf das Qualitätsmodell beschrieben im ISO/IEC 9126-1:2001 Standard [19d]. Das Qualitätsmodell unterscheidet die Anforderungen an die Qualität der Software in die Kategorien Funktionalität, Zuverlässigkeit, Effizienz, Benutzbarkeit, Änderbarkeit und Übertragbarkeit. Die durchgeführten Maßnahmen zur Qualitätssicherung, aufgegliedert in das Qualitätsmodell, werden in diesem Kapitel beschrieben.

### 6.3.1 Funktionalität

Die Anforderung an die Funktionalität, im Sinne des Qualitätsmodell beschreibt das Vorhandensein von Funktion mit fest definierten Eigenschaften. Die prototypische Implementierung befreit sich von solchen Konventionen in dem Sinne, dass die Sicherheit, Interoperabilität, Ordnungsmäßigkeit sowie die Richtigkeit (Genauigkeit von berechneten Werten) nicht als feste Anforderung gesehen wird. Bei einer produktiven Implementierung sollte die Anforderung jedoch gesetzt werden, vor allem in Hinsicht auf die Ordnungsmäßigkeit bei der Entwicklung von medizinischen Anwendungen.

### 6.3.2 Zuverlässigkeit

Die Fehlertoleranz wird im Zuge einer ausreichenden Testabdeckung des Codes so gering wie möglich gehalten werden. Dies garantiert eine entsprechende Zuverlässigkeit des Systems.

Für die Qualitätssicherung wurden Tests für einzelne Module, sogenannte Unit-Tests, sowie modulübergreifende Integrationstests geschrieben. Hierfür wurde das Python Modul „unittest“ verwendet.

Bei dem Softwareentwicklungsprozess wurde die Methode der Testgetriebenen Entwicklung (Test-driven Development (TDD)) angewandt, bei der die Software-Tests bereits vor der eigentlichen Implementierung der gewünschten Funktionalität geschrieben wurden und somit der Programmcode auf die Erfüllung des Tests angepasst wird.

### 6.3.3 Effizienz

Um eine gewisse Effizienz zu gewährleisten, wurden bei dem Prototypen Code-Abschnitte auf der CPU parallelisiert. Des Weiteren wurde das Erweiterungsmodul „numpy“ genutzt. Wie bereits erwähnt, werden die Funktionen und Funktionalitäten von numpy mit einer hohen Ausführungsgeschwindigkeit abgearbeitet.

### 6.3.4 Benutzbarkeit

Die Benutzbarkeit beschreibt den Aufwand, der vom Benutzer bei der Bedienung gefordert wird. Die Interaktionen mit dem System erfolgen bei dem entwickelten Prototypen über die Kommandozeile. Hierbei kann das System mit einigen wenigen Befehlen bedient werden.

### 6.3.5 Änderbarkeit

Das System soll einfach veränderbar und erweiterbar sein. Dies wird durch eine entsprechende Modularisierung und definierten Schnittstellen erreicht. Zudem wurde bei der Implementierung darauf geachtet, nur wenige Bibliotheken zu benutzen. Dadurch kann die Logik jedes Programmteils sofort nachvollzogen werden, was eine Änderung an dem Programm vereinfacht. Zudem gibt es keine „versteckten“ Funktionsaufrufe. Des Weiteren wurde der Code mit Hilfe von „pydoc“ kommentiert, um die Lesbarkeit zu erhöhen.

### 6.3.6 Übertragbarkeit

Die Übertragbarkeit des Programms wird insofern gewährleistet, indem das Programm als Paket auf PyPI bereitgestellt wird. Dadurch kann die entwickelte Software als Paket direkt über den Paketmanager „pip“ durch den Aufruf `pip install nirmapper` bezogen werden. Das Programm ist kompatibel mit allen UNIX-basierten Betriebssystemen.

## 6.4 Gesamtübersicht

Um eine Gesamtübersicht über das Programm zu vermitteln, werden in diesem Kapitel der Gesamtablauf des Programms mit Hilfe eines Ablaufdiagramms dargestellt, sowie eine Übersicht über die Module gegeben.

### 6.4.0.1 Ablaufdiagramm

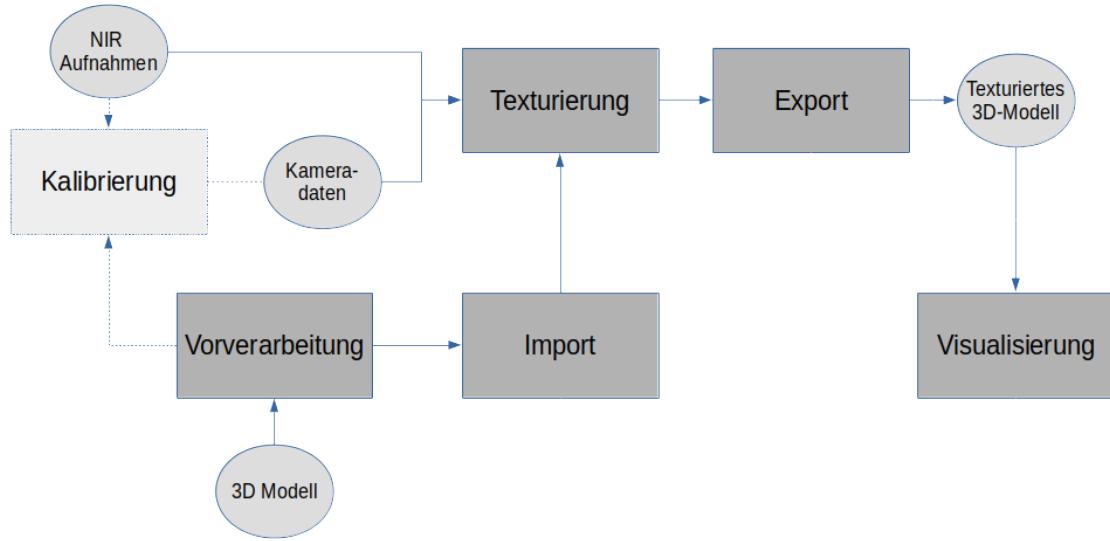
In Abbildung 6.1 ist ein Ablaufdiagramm des Programms illustriert. Die Kalibrierung wird als optionales und nicht implementiertes Modul in der Illustration dargestellt. Bekannte Kameradaten können somit auch direkt dem Texturierungsmodul übergeben werden.

Das 3D-Modell wird zunächst vorverarbeitet und anschließend importiert. Bei dem Import werden die Daten für die Texturierung aufgearbeitet. Zusammen mit den Kameraparametern und den vorverarbeiteten NIR-Aufnahmen werden die Daten an die Texturierung übergeben.

Nach der Texturierung wird das texturierte 3D-Modell in einem entsprechenden Format exportiert. Dieses Format kann anschließend über ein entsprechendes Modul visualisiert werden.

### 6.4.1 Modulübersicht

Wie bereits erwähnt, ist das entwickelte Programm in mehrere Module aufgeteilt, um eine einfache Änderbarkeit zu gewährleisten. Listing 6.1 zeigt die Modulaufteilung des Prototyps. Die Hauptmodule sind in `nirmapper/` beherbergt. Jedes der Module besteht aus ein oder mehreren Klassen mit entsprechenden Funktionalitäten. Das Modul `nirmapper/nirmapper.py` vereint diese Funktionalitäten und bietet einen Einstiegspunkt in die Hauptfunktionalitäten des Programms. Die CLI-Funktionalitäten sind in `bin/nirmapper` untergebracht. Die Modul-Tests sind in dem Paket `tests/` enthalten.

**Abbildung 6.1:** Gesamtlauf des Programms.**Listing 6.1** Modulübersicht des entwickelten Prototypen.

```

3DNIRMapper/
  bin/
    nirmapper
  nirmapper/
    __init__.py
    camera.py
    examples.py
    exceptions.py
    model.py
    nirmapper.py
    renderer.py
    utils.py
  tests/

```

## 6.5 Implementierungsdetails

In diesem Abschnitt wird auf Details der Implementierung eingegangen. Die Hauptfunktionalität des entwickelten Programms bildet die Texturierung. Dieses Kapitel beschreibt die einzelnen Vorgänge, die letztendlich die texturierten Daten liefern.

### 6.5.1 Klassendiagramm

Um ein Gesamtverständnis über die Funktionalität der Texturierung des Programms zu vermitteln, ist in Abbildung 6.2 das Zusammenspiel der einzelnen Klassen in einem UML-Klassendiagramm festgehalten.

## 6 Implementierung

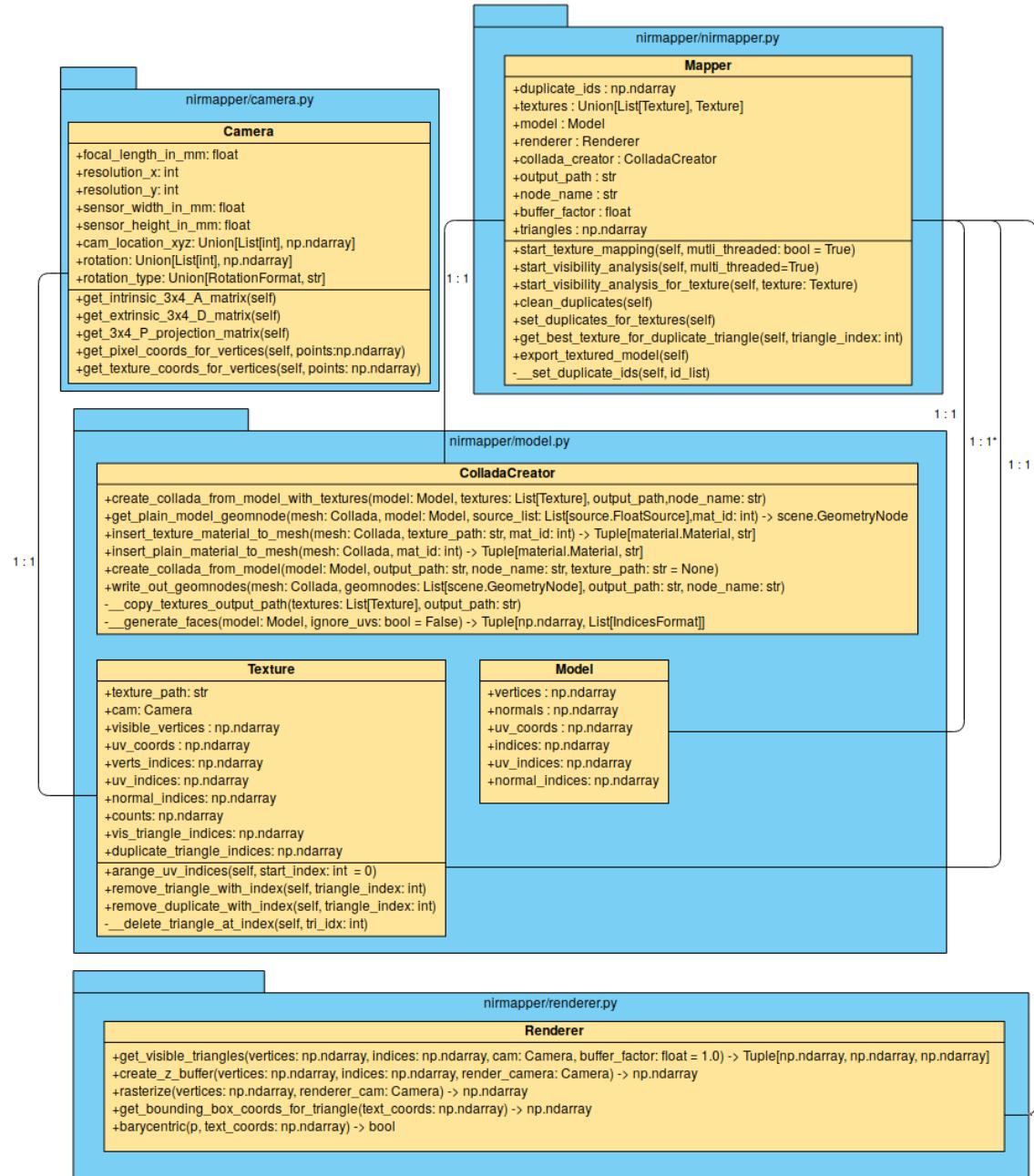


Abbildung 6.2: UML-Klassendiagramm, der für die die Texturierung verantwortlichen Klassen.

Nach dem Importieren des 3D-Modells und der Texturen, werden diese als Objekt dem Mapper übergeben. Der Mapper ist für den Gesamtablauf der Texturierung verantwortlich. Die Klasse beinhaltet hierfür ein Objekt mit dem Typ `Renderer`, der für eine übergebene Textur und einem 3D-Modell eine Sichtbarkeitsanalyse durchführt und die Ergebnisse dem Mapper zurück liefert. Anschließend kombiniert der Mapper die Texturen anhand der Ergebnisse der Sichtbarkeitsanalyse, indem er die überlappenden Bereiche der Texturen bereinigt. Zuletzt ruft der Mapper den `ColladaCreator` auf, der die erzeugten Daten in eine Collada-Datei exportiert.

### 6.5.2 Gewähltes Koordinatensystem

Für die Implementierung wurde das selbe Koordinatensystem wie in Blender gewählt, da die Kameraparameter der Testdaten mittels Blender bestimmt wurden und somit keine Konvertierung dieser Parameter vorgenommen werden musste. Bei dem Koordinatensystem handelt es sich demnach um ein rechtshändiges Koordinatensystem, bei dem die z-Achse nach oben gerichtet ist.

### 6.5.3 Import des 3D-Modells

Das entwickelte Programm unterstützt den Import von 3D-Modellen im Wavefront-Format (.obj) (siehe Abschnitt 2.2.5.2). Dadurch, dass Wavefront aufgrund der Beliebtheit von vielen anderen Programmen unterstützt wird, ist es denkbar, dass 3D-Modelle, die in einem anderen Format vorliegen, ohne viel Aufwand in das Wavefront-Format konvertiert und somit in das entwickelte Programm importiert werden können.

Die Funktionalität für den Import wird in der Klasse `Wavefront` im Modul `model.py` bereitgestellt. Das Modul verwendet die externe Bibliothek „PyWavefront“ für den Import. Dabei werden sämtliche Daten (Vertices, Normalen, Texturkoordinaten, etc.) von der Bibliothek in einer einzigen Liste importiert. Durch ein Format in Form eines Strings wird definiert, wie diese Daten zu interpretieren sind. Die einzelnen Formate sind dabei [19f]:

- V3F: Ein Vertex, bestehend aus drei float Werten.
- N3F: Der Normalenvektor, bestehend aus drei float Werten.
- C3F: Vertex-Farben, bestehend aus drei float Werten.
- T2F: Texturkoordinaten, bestehend aus zwei float Werten.

Das Gesamtformat wird als kombinierten String der einzelnen Formate beschrieben. Demnach würde der String „V3F\_ N3F\_ T2F“ so interpretiert werden, dass in der Liste zunächst drei float Werte einen Vertex definieren, gefolgt von drei float Werten für den Normalenvektor und anschließend zwei float Werten für die Texturkoordinaten. Diese Kombination wiederholt sich gemäß der Gesamtanzahl der Vertices.

Die entwickelte Software interpretiert die importierte Datenliste gemäß des Formats von PyWavefront, teilt die Daten in Listen aus Vertices, Normalenvektoren und Texturkoordinaten auf und erstellt die Indices für die jeweiligen Daten. Anschließend werden die Daten noch in das intern

## 6 Implementierung

---

### **Listing 6.2** XML-Format für die Kameraparameter.

---

```
<?xml version="1.0"?>
<data>
    <focal-length>35</focal-length>
    <resolution>
        <width>1280</width>
        <height>1024</height>
    </resolution>
    <sensor>
        <width>32</width>
        <height>25.6</height>
    </sensor>
    <location>
        <x>-1.2196</x>
        <y>1.2096</y>
        <z>9.8</z>
    </location>
    <!-- <rotation type="EULER">
        <x>-8</x>
        <y>20.2</y>
        <z>85.2</z>
    </rotation>-->
    <rotation type="QUAT">
        <w>0.715</w>
        <x>-0.169</x>
        <y>0.082</y>
        <z>0.674</z>
    </rotation>
</data>
```

---

verwendete Koordinatensystem konvertiert, indem die Vertices und Normalen um 90° um die x-Achse rotiert werden. Zuletzt werden die Daten in ein Objekt der Klasse `Model`, das sich ebenfalls im Modul `model.py` befindet, geschrieben und stehen nunmehr für eine Weiterverarbeitung bereit.

#### **6.5.4 Import der Texturdaten**

Die Texturdaten können über die Angabe eines Dateipfades importiert werden. Unterstützt werden Texturen im `.jpg`, `.png` und `.bmp` Format. Für die Definition der Kameraparameter wurde ein XML-Format definiert, das die intrinsischen und extrinsischen Kameraparameter beherbergt, mit denen ein Bild aufgenommen wurde. Das vollständige Format ist in Listing 6.2 gelistet. Es unterstützt dabei sowohl Euler- als auch Quaternionen-Rotationen.

Das Programm iteriert über sämtliche Bilddaten, die sich im angegebenen Dateipfad befinden und sucht nach den zugehörigen XML-Dateien, mit denselben Dateinamen wie die Bilddaten. Die in der XML-Datei angegebenen Kameraparameter werden ausgelesen und in ein `Camera` Objekt aus dem Modul `camera.py` geschrieben. Zusammen mit einer Referenz auf die Textur wird das `Camera` Objekt einem `Textur` Objekt übergeben, das für die Texturierung verwendet werden kann.

### 6.5.5 Sichtbarkeitsanalyse mit Z-Buffer Verfahren

Der erste Schritt der Texturierung ist die Sichtbarkeitsanalyse der Primitiven des importierten 3D-Modells für die jeweiligen Texturen. Für jede Primitive muss definiert werden, ob es von der Kameraperspektive der Textur aus (potentiell) sichtbar ist. Flächen, die von anderen Flächen des Objekts verdeckt werden, dürfen ebenfalls nicht mit eingefärbt werden und müssen bestimmt werden.

Die Wahl des Verfahrens fiel hierbei auf das Z-Buffer Verfahren aus Abschnitt 4.2.1.2. Gründe für die Auswahl sind zum einen, dass die Verdeckungsprüfung in dem Verfahren inkludiert ist. Es muss demnach keine zusätzliche Verdeckungsprüfung implementiert werden, wie es beispielsweise bei der Normalenprüfung aus Abschnitt 4.2.1.1 der Fall wäre. Des Weiteren ist das Verfahren, im Vergleich zu dem Maler-Algorithmus aus Abschnitt 4.2.1.3, einfacher zu implementieren. Hinzu kommt, dass die Z-Werte, welche die Distanz des Primitivs zu der Kamera angeben, exakt die dritte Komponente aus der Gleichung (2.18) der perspektivischen Projektion sind, die bei der späteren Zuweisung der Texturkoordinaten zu den Vertices ohnehin benötigt wird. Somit kann diese Funktionalität ebenfalls für die Sichtbarkeitsprüfung genutzt werden.

Insgesamt besteht der Vorgang der Sichtbarkeitsprüfung aus folgenden Schritten:

1. Initialisieren des Z-Buffers
2. Rasterisierung
3. Z-Buffer Vergleich

#### 6.5.5.1 Initialisieren des Z-Buffers

Wie aus dem Klassendiagramm aus Abbildung 6.2 ersichtlich, hält der `Mapper` die `Textur` Objekte und das 3D-Modell in Form eines `Model` Objekts als Attribute. Der `Mapper` iteriert über die Texturen und übergibt das `Camera` Objekt der jeweiligen Textur zusammen mit den Vertices und Indices des 3D-Modells und einem Z-Buffer-Faktor (`buffer_factor`) dem `Renderer` über die Methode `get_visible_triangles()` (siehe Abbildung 6.2). Der Z-Buffer-Faktor bestimmt dabei den Multiplikationsfaktor, der auf die Auflösung des übergebenen `Camera` Objekts (also `resolution_x` und `resolution_y`) angewandt wird.

Im Allgemeinen wird die Größe des Z-Buffers so gewählt, dass der Z-Buffer dieselbe Dimension aufweist, wie die Auflösung der Kamera (siehe Abschnitt 4.2.1.2). Dies entspräche einem Z-Buffer-Faktor mit `buffer_factor = 1`. Bei komplexen Szenen, bei denen das 3D-Modell viele Primitive besitzt, kann diese Größe jedoch unzureichend sein, um gute Ergebnisse zu erzielen (siehe Abschnitt 7.3.2). Deshalb, wurde mit dem Z-Buffer-Faktor eine einfache Möglichkeit implementiert, um die Dimension des Z-Buffers anzupassen zu können.

Die aktualisierte Kamera wird zusammen mit den Vertices und Indices der Methode `create_z_buffer()` übergeben, die den Z-Buffer in Form eines dreidimensionalen numpy-Arrays erstellt. Dabei ist die Länge der ersten und zweiten Dimension des Arrays äquivalent zu der, mit dem Z-Buffer-Faktor multiplizierten, Auflösung. Bei einer Auflösung von  $1024 \times 768$  und einem `buffer_factor = 2` beträgt der Z-Buffer demnach eine Größe von  $2048 \times 1536$ . Die dritte Dimension des Z-Buffers hält zwei

### **Listing 6.3** Datenstruktur des Z-Buffers.

---

```
[[[-1, np.inf], [-1, np.inf], [-1, np.inf], ... ],
 [[-1, np.inf], [-1, np.inf], [-1, np.inf], ... ],
 [[-1, np.inf], [-1, np.inf], [-1, np.inf], ... ],
 [[-1, np.inf], [-1, np.inf], [-1, np.inf], ... ],
 ...

```

---

Werte. Der erste Wert steht für den Index des Dreiecks und wird mit dem Wert  $-1$  vorinitialisiert. Der zweite Wert stellt den Z-Wert des Dreiecks dar und wird mit dem Unendlichkeitswert von numpy (`np.inf`) vorinitialisiert. Die Datenstruktur des Z-Buffers ist in Listing 6.3 veranschaulicht.

#### **6.5.5.2 Rasterisierung**

Nachdem der Z-Buffer initialisiert wurde, kann mit der eigentlichen Berechnung begonnen werden. Hierfür iteriert das Programm über alle Dreiecke des 3D-Modells, indem eine Sequenz aus den Vertices und den zugehörigen Indices erstellt wird. Ein Dreieck besteht aus drei Vertices mit jeweils drei Koordinaten. Die Indices geben an, in welcher Reihenfolge die Vertices ein Dreieck bilden. Das Programm erstellt demnach für  $n$  Dreiecke ein Array, entsprechend der vorgegebenen Reihenfolge der Indices, das aus  $n \times 3 \times 3$  Einträgen besteht. Somit kann über die einzelnen Dreiecke iteriert werden.

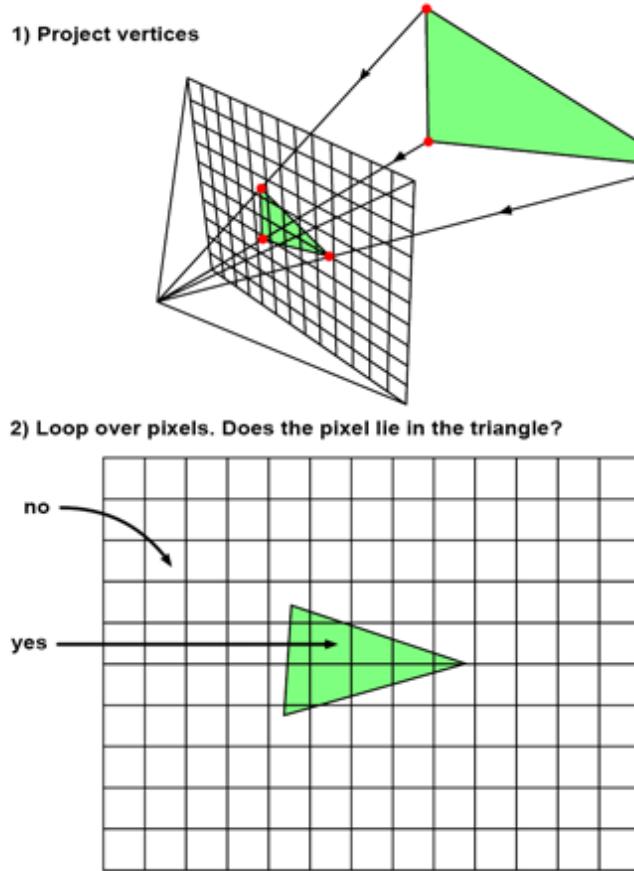
Jedes Dreieck wird zunächst gerastert. Das bedeutet, dass die Vertices des Dreiecks auf die Bildebene zurückgerechnet werden und anschließend geprüft wird, welche Pixel das Dreieck in der Bildebene einnimmt. Der Vorgang ist in Abbildung 6.3 illustriert.

Die Zurückrechnung der Vertices auf Bildebene wird über die Klasse `Camera` ermöglicht. Die Klasse berechnet die intrinsischen und extrinsischen Kameraparameter gemäß der Gleichung (2.18) der perspektivischen Projektion (siehe Klassendiagramm aus Abbildung 6.2) und stellt somit die Projektionsmatrix für die Umrechnung der Vertices in Texturkoordinaten bereit.

Die Überprüfung, ob sich ein Pixel innerhalb des Dreiecks der Bildebene befindet, erfolgt mit den sogenannten Baryzentrischen Koordinaten. Sie dienen dazu, die Lage von Punkten in Bezug zu dem Dreieck zu bestimmen. Baryzentrische Koordinaten bestehen aus drei Zahlen ( $u, v, w$ ). Um zu bestimmen, ob ein Punkt innerhalb des Dreiecks mit den Vertices  $A, B$  und  $C$  liegt, wird folgende Formel angewandt [Scr14]:

$$P = uA + vB + wC \quad \text{mit} \quad u + v + w = 1 \tag{6.1}$$

Der Punkt  $P$  kann somit durch eine Linearkombination von Punkten dargestellt werden, bei der die Summen der Koeffizienten eins ergeben [Scr14]. Baryzentrische Koordinaten können demnach als Flächen von Sub-Dreiecken gesehen werden, wie in Abbildung 6.4 illustriert. Die baryzentrischen Koordinaten berechnen sich wie folgt [Scr14]:

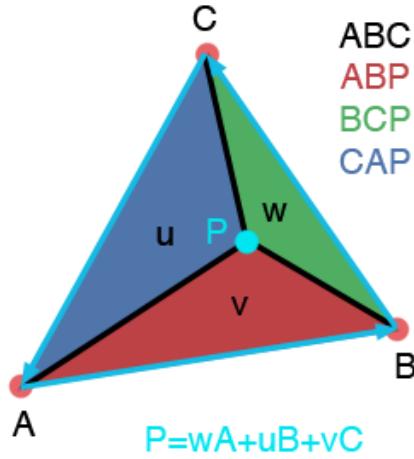


**Abbildung 6.3:** Ablauf der Rasterisierung eines Dreiecks in der Bildebene. [Scr15]

$$\begin{aligned}
 u &= \frac{A_{CAP}}{A_{ABC}} \\
 v &= \frac{A_{ABP}}{A_{ABC}} \\
 w &= \frac{A_{BCP}}{A_{ABC}}
 \end{aligned} \tag{6.2}$$

Sind zwei der baryzentrischen Koordinaten, beispielsweise  $u$  und  $v$ , gegeben, kann zudem die dritte Koordinate direkt mit  $w = 1 - u - v$  bestimmt werden, da bekanntlich  $u + v \leq 1$  gilt. Zusammen mit der Cramerschen Regel wurde diese Erkenntnis genutzt, um eine effektive Implementierung zu realisieren. Listing 6.4 zeigt die implementierte Überprüfung, ob sich ein Punkt  $p$  in einem Dreieck, das auf die Bildebene zurückgerechnet wurde, `text_coords` befindet.

Das Verfahren zur Überprüfung, welche Pixel sich innerhalb eines Dreiecks befinden, wird zudem beschleunigt, indem nicht alle Pixel der Bildebene überprüft werden, sondern nur diejenigen, die sich innerhalb einer sogenannten „Bounding-Box“ des Dreiecks befinden. Die Bounding-Box ist ein Rechteck, welches das Dreieck minimal umgibt. Das Prinzip ist in Abbildung 6.5 bildlich dargestellt.



**Abbildung 6.4:** Baryzentrische Koordinaten können als Sub-Dreiecke CAP (für u), ABP (für v) und BCP (für w) interpretiert werden. [Scr14]

---

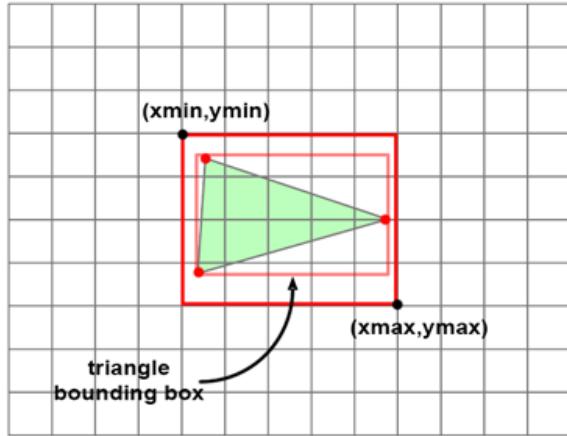
**Listing 6.4** Überprüfung, ob sich ein Punkt innerhalb eines Dreiecks befindet, anhand von baryzentrischen Koordinaten.

```
@staticmethod
def barycentric(p, text_coords: np.ndarray) -> bool:
    """
    Barycentric coordinates help to check if a pixel is inside a triangle.
    :param p: The Pixel to check for inclusion
    :param np.ndarray text_coords: The texture coordinates of the triangle
    :return bool: True if inside - False if outside of triangle
    """

    v0, v1, v2 = text_coords[1] - text_coords[0], \
                 text_coords[2] - text_coords[0], \
                 p - text_coords[0]
    den = v0[0] * v1[1] - v1[0] * v0[1]
    if den == 0:
        return False
    v = (v2[0] * v1[1] - v1[0] * v2[1]) / den
    w = (v0[0] * v2[1] - v2[0] * v0[1]) / den
    u = 1.0 - v - w
    # Make near zero values to zero
    if np.isclose([u], [0])[0]:
        u = 0

    return (u >= 0) and (v >= 0) and (u + v <= 1)
```

---



**Abbildung 6.5:** Beschleunigung des Rasterisierungs-Verfahrens durch die Bestimmung einer Bounding-Box. [Scr15]

---

**Listing 6.5** Auszug aus dem Programmcode, der für die Pixel eines Dreiecks überprüft, ob das Dreieck im Vordergrund liegt und entsprechend den Z-Buffer aktualisiert.

---

```
for pixel in included_pixels:
    uvz_coords = render_camera.get_pixel_coords_for_vertices(triangle, include_z_value=True)
    # mean is ok here because we don't have to check triangles that get cut by others
    z_value = np.mean(uvz_coords[:, -1:])
    if z_value < z_buffer[pixel[0], pixel[1]][1]:
        z_buffer[pixel[0], pixel[1]] = [idx, z_value]
```

---

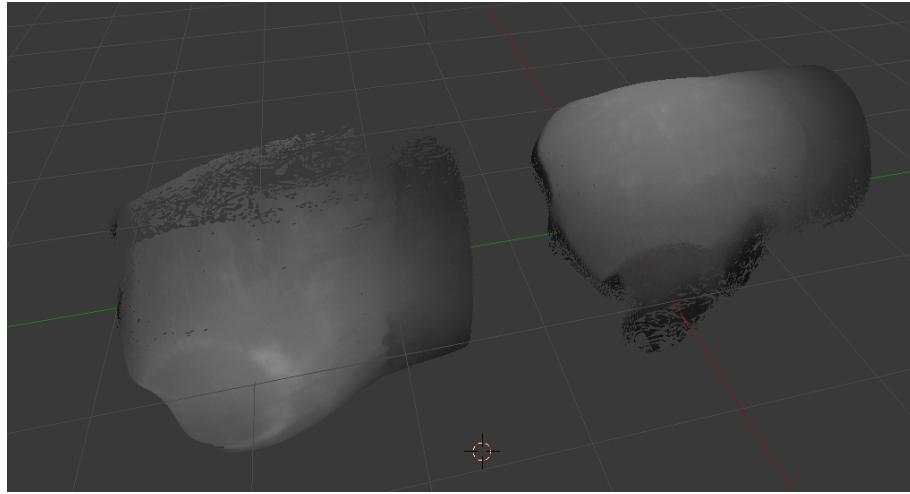
Die Rasterisierung erfolgt in der Methode `rasterize()` des Renderer. Das Ergebnis ist demnach ein numpy-Array mit den Pixeln, die sich innerhalb eines Dreiecks befinden.

#### 6.5.5.3 Z-Buffer Vergleich

Nachdem die inkludierten Pixel eines Dreiecks durch den Vorgang der Rasterisierung herausgefunden wurden, wird der Z-Buffer letztendlich gefüllt. Die Positionen der Pixel für das Dreieck geben zugleich die Positionen zur Überprüfung im Z-Buffer an (bspw. indiziert der Pixel 724×342 den Eintrag des Z-Buffer-Arrays an der Stelle [724][342]).

Verglichen wird jeweils der Mittelwert der Z-Werte für die drei Vertices des Dreiecks, mit dem bereits vorhandenen Wert im Z-Buffer. Liegt der Mittelwert unter dem an der Stelle des Pixels vorhandenen Wert im Z-Buffer, so wird an dieser Stelle im Array der Wert mit dem niedrigeren Z-Wert ersetzt und der Index des Dreiecks, das überprüft wurde, an die Stelle im Array geschrieben (siehe Listing 6.5).

Der Mittelwert, als Gesamtwert für den Z-Wert für alle inkludierten Pixel des Dreiecks, ist in Hinsicht auf den Anwendungsfall valide, da es im 3D-Modell keine sich überschneidenden Dreiecke geben sollte (vergleiche mit Abbildung 4.6).



**Abbildung 6.6:** Ergebnis des Texture Mapping für die sichtbaren Vertices zweier Texturen.

### 6.5.5.4 Multiprocessing

Um die Sichtbarkeitsanalyse zu beschleunigen, wurde das Verfahren auf der CPU mit Hilfe des „multiprocessing“ Moduls von Python parallelisiert. Hierbei wird für jede Textur, für die eine Sichtbarkeitsanalyse durchgeführt werden soll, ein eigener Prozess angelegt, der auf einem Kern der CPU abgearbeitet werden kann. Somit können bei vier vorhanden Texturen und einer CPU mit vier Kernen die Sichtbarkeitsanalyse auf alle vier Kerne aufgeteilt werden, was zu einer Beschleunigung des Verfahrens führt.

Mit Hilfe des Multiprocessing konnte beispielsweise die Rechenzeit für die Sichtbarkeitsanalyse eines 3D-Modells, mit 20.468 Dreiecken, für fünf Texturen und einer Z-Buffer-Größe von  $640 \times 512$  auf einem Mittelklasse-PC, mit einer Intel Core™ i7-2700K CPU mit 3.50GHz  $\times 8$ , 8GB RAM und einem 64-bit Ubuntu 18.04.02 Betriebssystem, von 157.95 Sekunden auf 47.21 Sekunden reduziert werden (Mittelwert aus jeweils drei Durchläufen).

### 6.5.5.5 Ergebnis

Das Ergebnis der Sichtbarkeitsanalyse mit dem gewählten Verfahren der Z-Buffer-Analyse ist demnach eine Liste von den Dreiecken des 3D-Modells, die auf der jeweiligen Textur aus sichtbar und nicht verdeckt sind. Mit diesen Erkenntnissen können die sichtbaren Vertices anhand der Kameraparameter bereits auf die Bildebene zurückgerechnet und ein Textur-Mapping für die jeweiligen Texturen vorgenommen werden, wie in Abbildung 6.6 illustriert. Der nächste Schritt ist die Kombination der einzelnen Texturen, um die überlappenden Bereiche zu behandeln, wie in Abbildung 4.8 bildlich dargestellt und in Abschnitt 4.2.2 beschrieben.

### 6.5.6 Texturkombination

Als nächster Schritt wurde die Texturkombination behandelt, welche die Ergebnisse der Sichtbarkeitsanalyse verwendet, um die bis dahin noch einzelnen Texturdaten zu kombinieren und somit ein, mit allen Texturen, texturiertes 3D-Modell zu erhalten.

Bildvorverarbeitungsverfahren bzw. Bildbearbeitungsverfahren in der Medizin zielen immer darauf hin, den Informationsgehalt für die Diagnostik zu erhöhen [Han09a]. Es wäre demnach konträr, den Informationsgehalt bei der Texturierung zu verringern oder sogar zu verfälschen. Beispielsweise wäre eine Kantenglättung bei den Texturübergängen, wie in Abschnitt 4.2.3.2 erläutert, zwar rein optisch von Vorteil, jedoch besteht die Gefahr, dass durch die Glättung Informationen der NIR-Aufnahme verloren gehen.

Deshalb wurde für die Texturkombination das Verfahren der Texturauswahl pro Primitiv gewählt, wie in Abschnitt 4.2.2.3 beschrieben, da bei diesem Verfahren keine Bildinformationen verfälscht oder verloren gehen.

Der gewählte Qualitätsparameter ist die Anzahl der Pixel, die das Primitiv auf der Textur einnimmt, da der Wert direkt aus dem bereits implementierten Z-Buffer heraus gelesen werden kann. Zudem bietet die Textur, auf der das Dreieck am meisten Pixel einnimmt, potentiell auch den größten Informationsgehalt. Dadurch werden nach der Texturierung möglichst viele Details sichtbar.

#### 6.5.6.1 Vorgehen

Für die Texturkombination hält der Mapper eine Liste (`duplicate_ids`) mit den Indexen der Dreiecke, die von mehreren Texturen eingefärbt werden. Hierfür wird der Parameter `vis_triangle_indices`, der die sichtbaren Dreiecke als Index-Liste enthält, des jeweiligen Textur Objekts ausgelesen und mit den anderen Texturen verglichen (siehe Abbildung 6.2).

Der Mapper iteriert über die `duplciate_ids` Liste und berechnet die Anzahl der Pixel, die das Dreieck auf jeder Textur einnimmt, indem er die Einträge des Z-Buffers zählt, für die der Index der selbe, wie der des zu überprüfenden Dreiecks, ist. Somit wird ermittelt, welche Textur das Dreieck am geeignetsten einfärbt.

Für die Texturen, auf denen das Dreieck weniger Pixel einnimmt, werden die Vertices, Indices und Texturkoordinaten für das Dreieck „gelöscht“. Das mehrfach eingefärbte Dreieck wurde somit bereinigt und der Index des Dreiecks wird aus der `duplciate_ids` Liste des Mapper Objekts entfernt.

Der Vorgang wiederholt sich solange, bis jedes Dreieck, das mehrfach eingefärbt wurde, einer Textur zugeordnet wurde und somit die Übergänge zwischen den Texturen bereinigt wurden.

#### 6.5.6.2 Ergebnis

Das Ergebnis der Texturkombination sind Textur Objekte, die jeweils die Daten für die Dreiecke bzw. Vertices enthalten, die von der Textur, die das Objekt referenziert, endgültig eingefärbt werden. Die Texturen wurden demnach so kombiniert, dass die Bereiche, in denen die Primitiven von zwei oder mehreren Texturen eingefärbt wurden, bereinigt wurden. Der letzte Schritt ist, die Daten der Texturen in einem Format zusammenzuführen und zu exportieren.

### 6.5.7 Zusammenführung und Export ins Collada Format

Der Export erfolgt bei dem Programm im Collada Format. Der Vorteil dieses Formates ist, dass die Texturaufnahmen als einzelne, unveränderte Bilder vorliegen und somit im Nachhinein bearbeitet werden können. Zudem handelt es sich bei Collada um ein sehr bekanntes Format, das deshalb von vielen Programmen importiert werden kann. Der Hauptgrund für die Wahl von Collada, als finales Datenformat anstelle von Wavefront liegt jedoch in der einfacheren und besser dokumentierten Bedienung des externen Frameworks „pycollada“ im Vergleich zu dem, für den Import verwendeten Framework, „PyWavefront“.

Für den Export ruft der Mapper nach der Sichtbarkeitsanalyse und der Texturkombination die Methode `create_collada_from_model_with_textures()` des ColladaCreator auf. Die Methode nimmt das 3D-Modell und die Texturen entgegen und fügt diese gemäß des Collada Formats zum finalen, texturierten 3D-Modell zusammen.

Hierfür definiert der ColladaCreator zunächst drei Datenquellen (siehe Abschnitt 2.2.5.3). Die erste Datenquelle beherbergt die Vertices und die zweite Datenquelle die Normalen des 3D-Modells. Für die dritte Datenquelle werden alle Texturkoordinaten der übergebenen Textur Objekte nacheinander in eine Liste geschrieben. Die dritte Datenquelle beherbergt somit sämtliche Texturkoordinaten der Texturen.

Anschließend legt der ColladaCreator für jede Textur ein Material, mit einem Verweis auf den Pfad der Textur an. Die Materialien erhalten eine Referenz auf die zuvor definierten Datenquellen, die später in der Collada-Datei als `<input>` Attribut, wie in Abschnitt 2.2.5.3 beschrieben, gesetzt werden.

Jedes Material unterscheidet sich dabei (neben dem Pfad zu dem jeweiligen Bild) von den anderen Materialien, durch deren Indices. Wie in Abschnitt 2.2.5.3 erläutert, werden die Indices anhand der Datenquellen interpretiert. Sind Vertices, Normalen und Texturkoordinaten definiert, so sind die Indices so zu interpretieren, dass der erste Index den Vertex, der zweite Index die Normale und der dritte Index die Texturkoordinate definiert. Demnach sind die Indices in Dreierpaaren zu interpretieren.

Dadurch, dass jedes Textur Objekt die Indices der Vertices und Normalen enthält, die auf der Textur sichtbar sind, können diese bereits kombiniert werden (siehe Klassendiagramm aus Abbildung 6.2). Die Indices für die Texturkoordinaten werden dabei für die Materialien immer sequentiell fortlaufend gesetzt, da die Datenquelle für die Texturkoordinaten eine konkatenierte Liste aus allen Texturkoordinaten der Textur Objekte ist. Besitzt demnach die erste Textur vier sichtbare Vertices und die zweite Textur sechs Vertices, so würde das zugehörige Material der ersten Textur die Indices `[0, 1, 2, 3]` und das Material der zweiten Textur die fortlaufenden Indices `[4, 5, 6, 7, 8, 9]` für die Texturkoordinaten bekommen.

Veranschaulicht wird das Prinzip in Listing 6.6. Der Auszug aus der exportierten Collada-Datei zeigt die Indices für die zwei Materialien zweier Textur Objekte. Die erste Textur hält dabei die Indices von zwei Dreiecken mit `vert_indices = [0, 7, 4, 0, 3, 7]` und `normal_indices = [0, 7, 4, 0, 3, 7]`. Die zweite Textur hält die Indices für ein Dreieck mit `vert_indices = [4, 5, 1]` und `normal_indices = [4, 5, 1]`. Kombiniert ergeben sich dann die Indices `[0, 0, 0, 7, 7, 1, 4, 4, 2, 0, 0, 3, 3, 3, 4, 7, 7, 5]` für das erste Material und `[4, 4, 6, 5, 5, 7, 1, 1, 8]` für das zweite Material.

**Listing 6.6** Auszug aus der exportierten Collada-Datei.

```
[...]
<triangles count="2" material="material_0">
    <input offset="0" semantic="VERTEX" source="#verts-array-vertices" />
    <input offset="1" semantic="NORMAL" source="#normals-array" />
    <input offset="2" semantic="TEXCOORD" set="0" source="#uv_array" />
    <p>0 0 7 7 1 4 4 2 0 0 3 3 3 4 7 7 5</p>
</triangles>
<triangles count="1" material="material_1">
    <input offset="0" semantic="VERTEX" source="#verts-array-vertices" />
    <input offset="1" semantic="NORMAL" source="#normals-array" />
    <input offset="2" semantic="TEXCOORD" set="0" source="#uv_array" />
    <p>4 4 6 5 5 7 1 1 8</p>
</triangles>
[...]
```

Der Auszug zeigt die Indices für zwei Materialien.

---

Durch die Indices können somit die entsprechenden Vertices jeder Textur den Texturkoordinaten zugeordnet werden. Die Datenquellen und Materialien werden dann zusammen mit einigen Metainformationen in eine Collada-Datei geschrieben. Grafikprogramme, die das Collada-Format unterstützen, können die Materialien interpretieren und als texturiertes 3D-Modell darstellen.

## 6.6 Zusammenfassung

In diesem Kapitel wurde beschrieben, wie das entwickelte Programm eine Texturierung, anhand mehrerer Texturaufnahmen, vornimmt. Dabei wurden auf die Details der Implementierung, der gewählten Verfahren, zur Sichtbarkeitsanalyse mittels Z-Buffer und der Texturkombination, durch die Texturauswahl pro Dreieck, eingegangen.

Das Ziel des Prototypen lag darin, Kariesläsionen aus NIR-Aufnahmen dreidimensional zu visualisieren. Mit der entwickelten Softwarelösung ist es nunmehr möglich, die NIR-Aufnahmen eines zugehörigen 3D-Modells mit bekannten Kameraparametern auf ein 3D-Modell zu applizieren und durch einen Import der erzeugten Collada-Datei, in einem externen Programm zu visualisieren. Somit wurde das gesetzte Ziel erreicht.



# 7 Ergebnisse und Evaluierung

Zu Beginn dieses Kapitels werden die erzielten Ergebnissen der Implementierung vorgestellt. Anschließend werden Probleme sowie deren Ursache analysiert und Lösungsmöglichkeiten vorschlagen. Zuletzt wird die durchgeführte Evaluierung durch einen Anwendertest beschrieben.

## 7.1 Ergebnisse

Das Ergebnis der Texturierung für ein 3D-Modell, das mit fünf NIR-Aufnahmen, die aus unterschiedlichen Perspektiven aufgenommen wurden, texturiert wurde, ist in Abbildung 7.1 sichtbar. Das 3D-Modell ist dabei dasselbe, das von der LMU München bereitgestellt wurde, jedoch wurde die Oberfläche geglättet und die Dreiecksanzahl auf 20.468 Dreiecke reduziert. Die NIR-Aufnahmen stammen dabei ebenfalls aus dem von der LMU München bereitgestellten Datensatz und haben eine Auflösung von  $1280 \times 1024$ . Die Kameradaten wurden mittels Blender manuell bestimmt. Die Größe des Z-Buffers betrug hierbei  $2560 \times 2048$ . Insgesamt wurde die Textur weitestgehend zusammenhängend auf das 3D-Modell appliziert, was ein gutes visuelles Ergebnis liefert.

Das Ergebnis der Texturzuweisung ist dabei sehr akkurat, wie in Abbildung 7.2 sichtbar. Die Textur wird dabei ohne Versatz auf das 3D-Modell appliziert. Die Ortsinformationen (von beispielsweise Kariesläsionen) gehen daher nicht verloren.

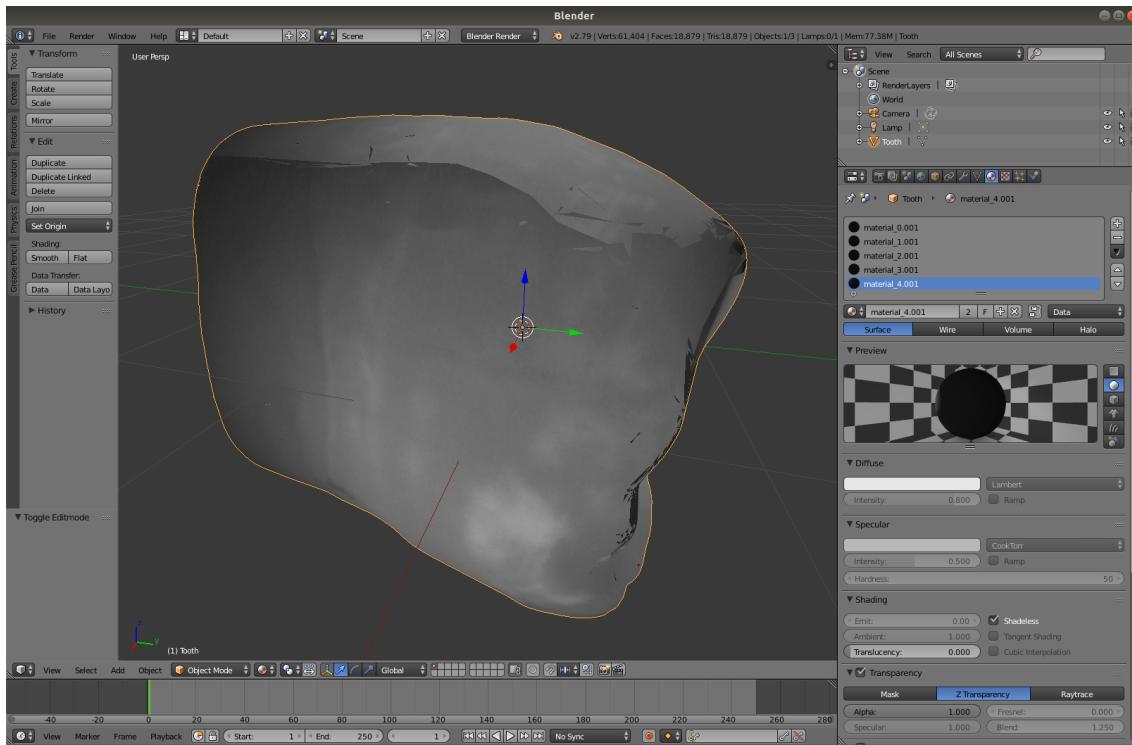
## 7.2 Programmlaufzeit

Die Programmlaufzeit hängt von einigen Faktoren ab. Zum einen steigt die Laufzeit für die Sichtbarkeitsanalyse bei steigender Größe des Z-Buffers und höherer Anzahl der Dreiecke des 3D-Modells. Dies liegt daran, dass der Algorithmus für die Sichtbarkeitsanalyse eine Laufzeit von  $O(n^2)$  aufweist. Somit steigt die Laufzeit exponentiell an.

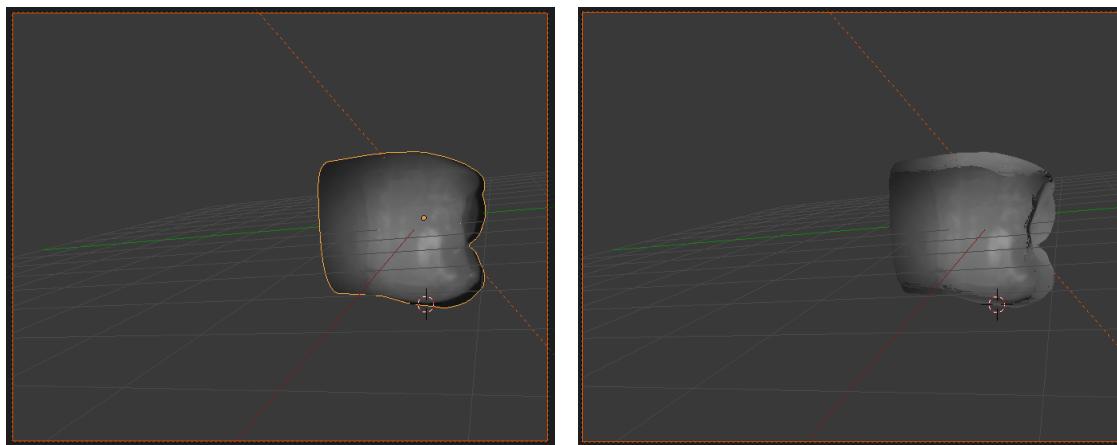
Zum anderen kommt es bei komplexen 3D-Modellen, mit vielen Dreiecken, bei der Texturkombination zu hohen Laufzeiten. Dies liegt daran, dass die Anzahl der Dreiecke steigt, die mehrfach eingefärbt wurden und somit der Aufwand höher wird, die Dreiecke einer finalen Textur zuzuweisen.

Abbildung 7.3 zeigt die Laufzeiten für die Sichtbarkeitsanalyse der Texturkombination und die Gesamtlaufzeit für verschiedene Z-Buffer-Größen von zwei 3D-Modellen. Das erste 3D-Modell besitzt 20.468 Dreiecke und das Zweite 40.668 Dreiecke. Das Programm wurde dabei auf einem Mittelklasse-PC mit einer Intel Core™ i7-2700K CPU mit 3.50GHz  $\times 8$ , 8GB RAM und einem 64-bit Ubuntu 18.04.02 Betriebssystem ausgeführt. Die Zeitwerte sind jeweils die Mittelwerte aus drei Durchläufen für die verschiedenen Z-Buffer-Größen.

## 7 Ergebnisse und Evaluierung



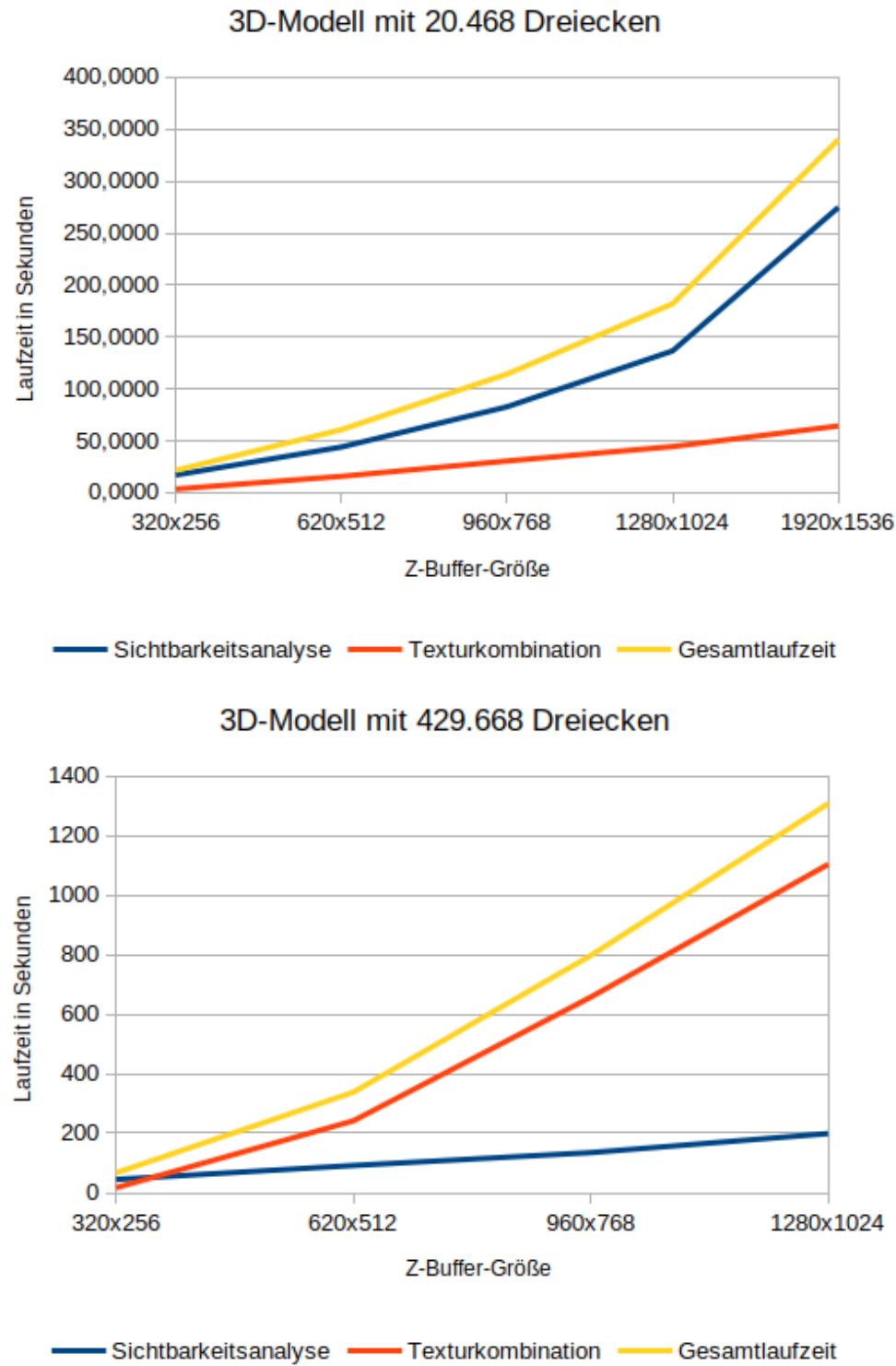
**Abbildung 7.1:** Ergebnis des Texture Mapping für fünf Texturen, die auf ein 3D-Modell mit 20.468 Dreiecken appliziert wurden, dargestellt in Blender.



(a) Manuell gemappte Textur mit einer NIR-Aufnahme.

(b) Ergebnis der Texturierung.

**Abbildung 7.2:** Gegenüberstellung der manuell gemappten Textur mit dem Ergebnis der Texturierung. Die Gegenüberstellung erfolgt von der selben Kameraperspektive aus. Die Textur wurde von dem entwickelten Programm ohne sichtbaren Abweichungen der Position der Farbwerte auf das 3D-Modell appliziert.



**Abbildung 7.3:** Diagramm der Laufzeiten des Programms für zwei verschiedene 3D-Modelle.

Wohingegen bei dem 3D-Modell mit 20.468 Dreiecken bei steigender Z-Buffer-Größe die Laufzeit für die Sichtbarkeitsanalyse stark exponentiell ansteigt, so steigt bei dem 3D-Modell mit der höheren Anzahl an Dreiecken die Laufzeit für die Texturkombination mit zunehmender Z-Buffer-Größe stärker an, als für die Sichtbarkeitsanalyse. Somit wurde die Vermutung bestätigt, dass beide Faktoren eine Auswirkung auf die Laufzeit haben.

Durch eine Auslagerung der Funktionalitäten zur Sichtbarkeitsanalyse und Texturkombination auf die Grafikkarte könnte der exponentiellen Laufzeit entgegen gewirkt werden.

## 7.3 Bekannte Probleme

In diesem Kapitel wird auf erkannte Probleme des Programms, deren Ursache und mögliche Lösungsansätze eingegangen.

### 7.3.1 Sichtbare Übergänge durch Belichtungsunterschiede

Wie in Abbildung 7.1 erkennbar, sind die Übergänge zwischen den Texturen deutlich sichtbar. Bereits bei der Analyse der NIR-Daten wurden starke Unterschiede in der Belichtung festgestellt (siehe Abschnitt 3.1.2). Diese führen nun durch die unterschiedliche Belichtung zu dem Artefakt der sichtbaren Kanten, wie in Abschnitt 4.2.3.3 erläutert.

Durch ein Verfahren wie dem „Multi-band blending in 3D“, das von Baumberg entwickelt wurde, kann diesem Artefakt entgegen gewirkt werden (siehe ebenfalls Abschnitt 4.2.3.3).

### 7.3.2 Teilweise nicht eingefärbte Dreiecke

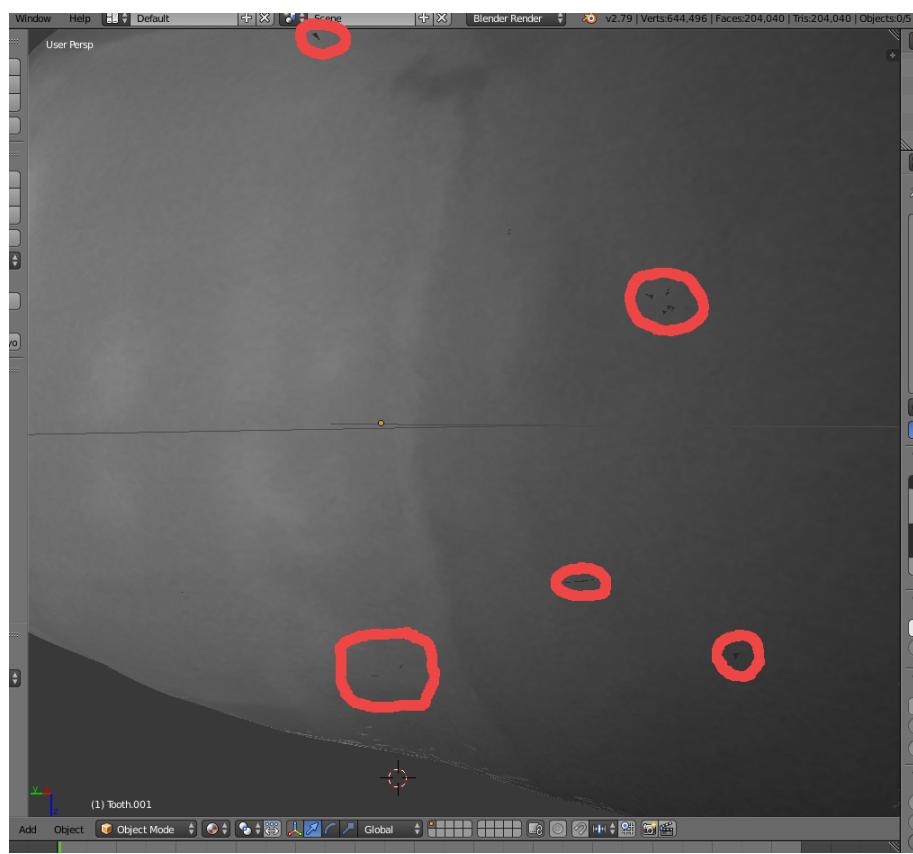
Bei der Texturierung kommt es, vor allem bei 3D-Modellen mit vielen, kleinen Dreiecken zu dem Artefakt, dass manche Dreiecke, die sich im offensichtlich sichtbaren Bereich einer Textur befinden, nicht eingefärbt wurden (siehe Abbildung 7.4).

Der Grund hierfür ist, dass Dreiecke nur dann einer Textur zugeordnet werden können, wenn diese bei der Sichtbarkeitsanalyse ein oder mehrere Pixel einnehmen. Ist das Dreieck jedoch so klein, dass es keine Pixel einnimmt, so kann es keiner Textur zugeordnet werden und es entstehen Dreiecke im sichtbaren Bereich, die nicht eingefärbt wurden.

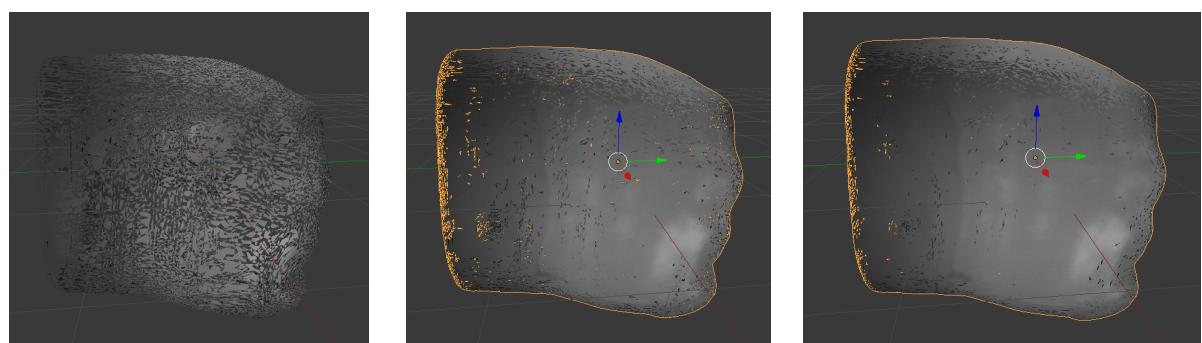
Das Ergebnis der Texturierung ist demnach stark abhängig von der gewählten Größe des Z-Buffers, da die Dreiecke nicht mit der Auflösung der Textur, sondern mit der des Z-Buffers zurückgerechnet werden. Je größer der Z-Buffer gewählt wird, umso höher ist die Chance, dass auch kleine Dreiecke Pixel auf der Textur einnehmen und somit eingefärbt werden. Eine Versuchsreihe mit verschiedenen gewählten Z-Buffer-Größen belegt diese These, wie in Abbildung 7.5 illustriert.

Dadurch, dass die Laufzeit stark abhängig von der gewählten Größe des Z-Buffers ist, empfiehlt es sich jedoch nicht, den Z-Buffer beliebig groß zu wählen. Es muss hier ein geeigneter Kompromiss zwischen der Laufzeit und der Qualität der Texturierung gefunden werden.

Oft ist es auch möglich, die Anzahl der Dreiecke des 3D-Modells durch Grafikprogramme wie Blender zu reduzieren und somit bessere Ergebnisse zu erhalten.



**Abbildung 7.4:** Nicht texturierte Dreiecke, markiert durch eine rote Umrandung.

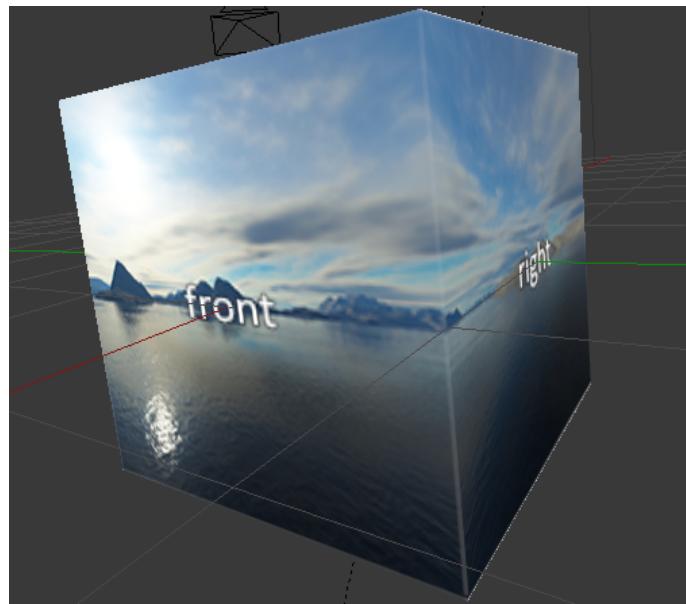


(a) Z-Buffer-Größe: 320 × 256

(b) Z-Buffer-Größe: 960 × 768

(c) Z-Buffer-Größe: 1280 × 1024

**Abbildung 7.5:** Ergebnisse der Texturierung eines 3D-Modells mit 429.668 Dreiecken für verschiedene Z-Buffer-Größen.



**Abbildung 7.6:** Ergebnis der Texturierung für einen Würfel.  
Ein Versatz an den Texturübergängen ist nicht sichtbar.

### 7.3.3 Visuelle Diskontinuität

Durch die manuell bestimmten Kameraparameter kommt es zwangsläufig zu einem Versatz an den Texturübergängen, wie in Abschnitt 4.2.3.2 beschrieben. Dies war bei den Testdaten nicht zu vermeiden, da die echten Kameradaten nicht vorlagen.

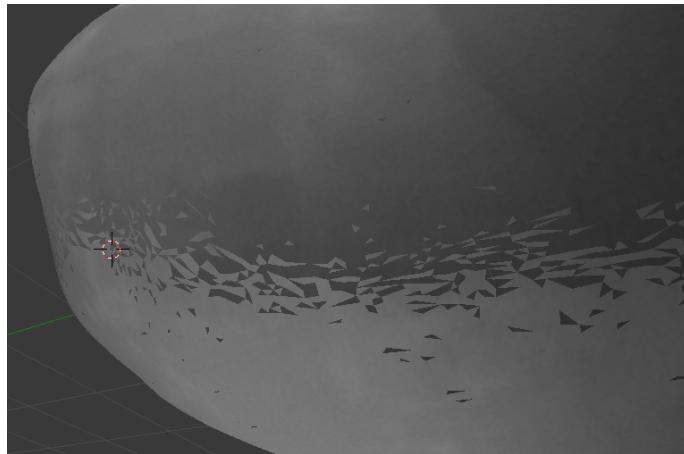
Um zu evaluieren, ob dieses Problem auch bei korrekten Kameradaten auftritt, wurde ein anderes 3D-Modell, mit einer bereits applizierter Textur, in Form eines Würfels verwendet. Anschließend wurden willkürlich Kamerapositionen bestimmt und Bilder des 3D-Modells mit der vorhanden Textur von diesen Kamerapositionen aus gerendert. Die erstellten Bilder und die Kameradaten wurden, zusammen mit dem 3D-Modell des Würfels, dem erstellten Programm übergeben. Nach der Texturierung konnte keine visuelle Diskontinuität festgestellt werden (siehe Abbildung 7.6).

Es kann demnach davon ausgegangen werden, dass durch eine bessere Bestimmung der Kameradaten das Problem der visuellen Diskontinuität in der Praxis verhindert werden kann.

### 7.3.4 Versetzte Kanten

Vor allem bei dem 3D-Modell, dessen Dreiecke nicht reduziert wurden, kommt es bei den Übergängen zu versetzten, zusammenhanglosen Kanten, wie in Abschnitt 4.2.3.1 beschrieben (siehe Abbildung 7.7).

Dies liegt an dem gewählten Verfahren der Texturkombination durch die Texturauswahl pro Dreieck, das keine zusammenhängende Kanten bei den Übergängen der Texturen bildet. Wegen der Unterschiede in der Belichtung wird dieser Effekt verstärkt. Durch den Algorithmus, wie in



**Abbildung 7.7:** Versetzte Kanten an den Texturübergängen.  
Texturiert wurde hierbei ein 3D-Modell mit 429.668 Dreiecken.

Abschnitt 4.2.3.1 erklärt, der die applizierten Texturen an den Kanten umsortiert, kann dieses Artefakt jedoch bereinigt werden. Mithilfe einer Beleuchtungskorrektur der Texturaufnahmen würde dieser Effekt zudem weniger hervorstechen.

## 7.4 Anwendertest

In diesem Abschnitt wird die Evaluierung der Ergebnisse anhand eines Anwendertests beschrieben. Die gewonnenen Erkenntnisse aus der Evaluierung helfen dabei, die Hypothese zu unterstützen, dass NIR-Aufnahmen als eine geeignete Methode zur Unterstützung der zahnmedizinischen Diagnostik eingesetzt werden könnten.

### 7.4.1 Versuchsziel

Bei der Evaluierung soll durch einen Anwendertest in Form einer Einschätzung der visualisierten Ergebnisse durch einen zahnmedizinischen Experten bestimmt werden, ob die entwickelte, dreidimensionale Darstellung der NIR-Aufnahmen einen Mehrwert bei der zahnmedizinischen Diagnostik liefern könnte.

### 7.4.2 Versuchsaufbau

Um einen möglichst guten Eindruck der Ergebnisse bieten zu können, wurde das texturierte, 3D-Modell als dreidimensionales Objekt in virtueller Realität (VR) dargestellt. Hierfür wurde ein Python Skript für das Visualisierungsprogramm „Vizard“ geschrieben, welches das 3D-Modell importiert und in der VR-Brille „HTC-Vive“ in VR darstellt. Somit war es dem Anwender möglich, das 3D-Modell aus unterschiedlichen Perspektiven in VR betrachten zu können.

## 7 Ergebnisse und Evaluierung

Für die Evaluierung wurde ein promovierter Experte der Zahnmedizin aus der LMU München eingeladen, der die Ergebnisse bewerten sollte. Zusätzlich nahmen ein promovierter Experte im Bereich der Computergrafik und insbesondere der medizinischen Informatik sowie ein Informatik-Student bei dem Anwendertest teil.

### **7.4.3 Versuchsergebnisse**

Die Resonanz bei dem Anwendertest viel positiv aus. Die erkannten Probleme und deren Ursachen sowie die möglichen Lösungsansätze wurden aufgezeigt und besprochen. Laut Expertenmeinung sei der Mehrwert für die Diagnostik durch die dreidimensionale Darstellung der NIR-Aufnahmen, trotz der angegebenen Probleme, gegeben. Somit konnte die aufgestellte Hypothese bestätigt werden.

# **8 Zusammenfassung und Ausblick**

In diesem Kapitel wird zunächst eine Zusammenfassung der gesamten Arbeit gegeben und die gewonnenen Erkenntnisse der Arbeit bewertet. Abschließend wird ein Ausblick auf mögliche Fortsetzungen und erweiterte Ansätze gegeben.

## **8.1 Zusammenfassung**

In dieser Arbeit wurde die Hypothese fundiert, dass Nahinfrarotaufnahmen als eine geeignete Methode zur Unterstützung der zahnmedizinischen Diagnostik eingesetzt werden können. Dies wurde durch eine prototypischen Implementierung einer Software erreicht, die vorhandene Nahinfrarotaufnahmen auf ein 3D-Modell appliziert. Das daraus resultierende, texturierte 3D-Modell kann wiederum in einem externen Programm dreidimensional visualisiert werden.

Der Leser wurde in Kapitel 2 mit den Grundlagen der Nahinfrarotbildgebung in der Zahnmedizin vertraut gemacht. Hierfür wurde dem Leser die Eigenschaften des Nahinfrarot, vor allem in Bezug zu der Bildgebung, aufgezeigt. Die verschiedenen bildgebenden Modalitäten wurden in Hinblick auf die technische Funktionsweise beschrieben und aktuelle NIR-Diagnosewerkzeuge vorgestellt. Zudem wurde ein Vergleich zwischen der NIR-Bildgebung und den herkömmlichen bildgebenden Verfahren angestellt. Des Weiteren wurde auf die Grundlagen der 3D-Computergrafik eingegangen, um ein Verständnis bei dem Leser aufzubauen, wie die, von der LMU München bereitgestellten Nahinfrarotaufnahmen, auf ein 3D-Modell digital appliziert werden können. Hierfür wurde, nach einer Einführung in die 3D-Computergrafik an sich, die Charakteristiken und Eigenschaften von 3D-Modellen beschrieben. Anschließend wurde aufgezeigt wie deren Daten manipuliert und in welche Dateiformate abgespeichert werden können. Abschließend wurde der Vorgang der Texturierung an sich erläutert, indem erklärt wurde, wie Texturen, anhand des Kameramodells, dem 3D-Modell zugewiesen werden können.

Kapitel 3 beschäftigte sich mit der Analyse der gegebenen Daten. Diese wurden unter Berücksichtigung der Erkenntnisse aus den Grundlagen untersucht und Anforderungen an das System aufgestellt. Aus der Erkenntnissen der Datenanalyse, konnten Rechercheaufgaben zum Stand der Technik abgeleitet werden.

Der Stand der Technik wurde in Kapitel 4 aufgezeigt. Zu Beginn des Kapitels wurden die Möglichkeiten zur Kamerakalibrierung untersucht. Dabei wurde insbesondere auf die Kalibrierung anhand von 3D/2D-Registrierung und die verschiedenen Möglichkeiten der Generierung der perspektivischen Aufnahmen sowie der möglichen Ähnlichkeitsmaße für dieses Verfahren eingegangen. Zudem wurden in dem Kapitel die Verfahren der Texturierung anhand mehrerer Texturaufnahmen beschrieben. Dabei wurde erläutert, warum die Sichtbarkeitsanalyse dabei eine Rolle spielt und

## 8 Zusammenfassung und Ausblick

---

welche Verfahren es hierfür gibt. Anschließend wurde erläutert, wie die Texturen letztendlich kombiniert werden können und welche Artefakte dabei auftreten können. Für die Bereinigung der Artefakte wurden bereits Lösungsansätze vorgestellt.

In der Machbarkeitsanalyse aus Kapitel 5 wurde untersucht, ob die bekannten Ähnlichkeitsmaße für eine Kamerakalibrierung durch das 3D/2D-Registrierung geeignet sind. Dabei wurde die Erkenntnis gewonnen, dass eine Kalibrierung der zur Verfügung gestellten Daten nur im Vornherein, beispielsweise durch ein Kalibrierungsmuster oder ein Trackingsystem, möglich ist.

In Kapitel 6 wurde die Implementierung des Prototypen dargestellt. Für die Implementierung eingesetzte Software und Softwarekomponenten wurden aufgezeigt und die Maßnahmen zur Sicherung der Qualität erläutert. Anschließend wurde eine Übersicht über die Implementierung gegeben, indem zunächst der Gesamtlauf des Programms illustriert wurde und anschließend die entwickelten Module aufgelistet wurden. Abschließend wurden Implementierungsdetails zum Import der Daten, der Sichtbarkeitsanalyse sowie der Texturkombination und dem Export vermittelt.

Zum Schluss wurden die Ergebnisse in Kapitel 7 präsentiert und evaluiert. Dabei wurde auf bekannte Probleme und deren Ursachen eingegangen sowie mögliche Lösungsvorschläge erbracht. Anschließend wurde der durchgeführte Anwendertest beschrieben. Es konnte gezeigt werden, dass die gesetzten Anforderungen und Ziele erfüllt wurden. Zudem wurde bestätigt, dass die aufgestellte Hypothese, durch das Ergebnis des entwickelten Prototypen, fundiert wurde.

## 8.2 Ausblick

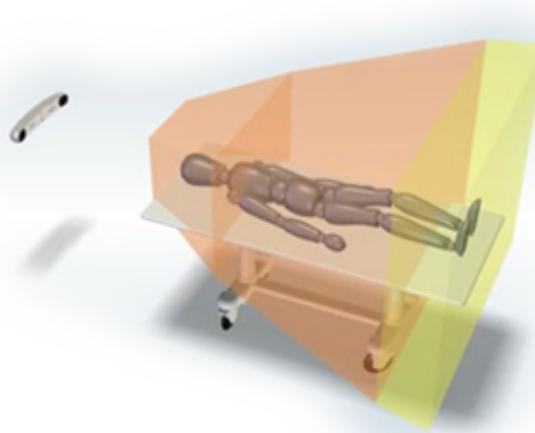
Im Folgenden werden einige Weiterentwicklungsmöglichkeit und erweiterte Konzepte vorgestellt.

### 8.2.1 Verwenden von lokalen Bildmerkmalen als Ähnlichkeitsmaß der Kalibrierung mithilfe von 3D/2D-Registrierung

Auch wenn lokale Bildmerkmale im Allgemeinen nicht für objektive Funktionen zur Ähnlichkeitsbestimmung bei der 3D/2D-Registrierung verwendet werden, so haben diese dennoch Potential.

Lokale Bildmerkmale sind Merkmale, die sich nicht auf das gesamte Bild beziehen, sondern auf einzelne Bildabschnitte bzw. „interessante Punkte“. Dabei gibt es eine Vielzahl an Detektoren, die diese Punkte in Bildern bestimmen können [HAA16b]:

- Laplacian of Gaussian Detektor
- Hessian-Laplace Detektor
- Difference of Gaussian Detektor
- SURF Detektor
- ...



**Abbildung 8.1:** Tracking mit dem Polaris-System. [19e]

Ein mögliches Verfahren, um die Ähnlichkeit der projizierten Aufnahme und der Textur zu bestimmen, wäre demnach, die interessanten Punkte mit einem entsprechenden Verfahren in beiden Bildern herauszufinden und beispielsweise durch eine Distanzfunktion zu vergleichen.

Somit wäre eine Kalibrierung mithilfe 3D/2D-Registrierung potentiell möglich, insofern geeignete Bildmerkmale aus den NIR-Aufnahmen und dem 3D-Modell bestimmt werden können.

### 8.2.2 Kombination mit einem Trackingsystem

Durch eine Kombination mit einem Trackingsystem könnten die Kameraparameter bereits während der Aufnahme der NIR-Bilder ermittelt werden. Wie bereits erwähnt, könnte das Polaris-System von der Firma NDI-Medical hierfür verwendet werden [19e].

Das Polaris-System kann dabei Objekte, wie eine NIR-Kamera, anhand von aktiven oder passiven Markern erfassen und die Position der Objekte im dreidimensionalen Raum bestimmen. Hierfür verwendet das System zwei Positionsensoren, die in einem dreidimensionalen Bereich in Form einer Pyramide, mittels Nahinfrarot die applizierten Marker detektieren (siehe Abbildung 8.1). Die Genauigkeit liegt dabei laut Spezifikation in einem Bereich von 0.3mm Root-mean-square (RMS).

Somit könnten die Kameraparameter direkt, bei der Aufnahme des NIR-Bildes, bestimmt werden. Anschließend könnte das entwickelte Programm die einzelnen NIR-Texturen zu einem 3D-Modell kombinieren.

Jedoch müsste evaluiert werden, wie genau das Trackingverfahren arbeitet. Zudem könnte das verwendete Nahinfrarot, um die Marker zu detektieren, die NIR-Aufnahmen negativ beeinflussen.

### 8.2.3 Tomographie mittels Nahinfrarot

Eine weitere Weiterentwicklungsmöglichkeit stützt sich auf die Idee, die Tomographie mit Nahinfrarot zu verbinden. Bei der Tomographie werden mehrere Schnittbilder eines Objektes (bspw. des Kopfes eines Patienten) erstellt. Die Schnittbilder sind eine überlagungsfreie Darstellung der entsprechenden Objektschicht. Aus den Schnittbildern kann anschließend über ein mathematisches Verfahren ein dreidimensionales Bild rekonstruiert werden. Bei der Tomographie gibt es eine Vielzahl an verschiedenen Verfahren, unter anderem die Röntgentomographie, Computertomografie (CT) und Optische Kohärenz tomografie (OCT).

Die Röntgentomographie und CT verwenden dabei Röntgenstrahlen für die Bildgebung. Dabei rotiert ein Röntgenstrahlen-Emitter um das Objekt und emittiert einen fächerartigen Röntgenstrahl. Die Strahlung wird von einem gegenüberliegenden Detektor erfasst, der ebenfalls um das Objekt rotiert. Aufgrund der unterschiedlichen Eigenschaften der Gewebe und wie diese die Röntgenstrahlen absorbieren, entsteht daraus das Schnittbild.

Eine Transillumination ist auch mit NIR möglich, wie in Abschnitt 2.1.3 erläutert. Somit könnten auch, ähnlich wie bei der Röntgentomographie oder CT, Schnittbilder erzeugt werden, indem das Verfahren adaptiert wird.

Das OCT verwendet Lichtstrahlen für die Konstruktion der Schnittbilder. Dabei wird ein Objekt zunächst mit den Lichtstrahlen beleuchtet. Der Lichtstrahl dringt in das Objekt und wird in unterschiedlichen Tiefen von den verschiedenen Gewebe schichten reflektieren. Diese Reflektion wird von einem Detektor erfasst. Die Abstände, in denen das reflektierte Licht auf den Detektor trifft, geben demnach Auskunft über die Tiefe der Gewebe schichten. Somit kann aus den Abständen des reflektierten Lichts ein Schnittbild erzeugt werden.

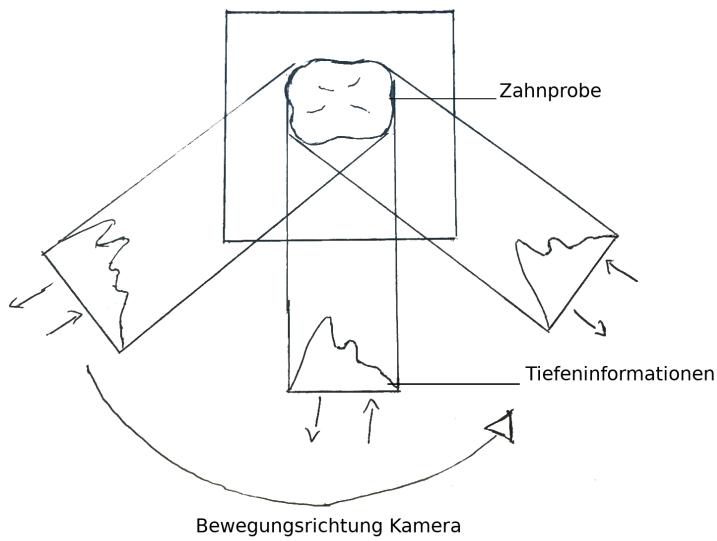
Betrachtet man die optischen Eigenschaften eines Zahnes im Nahinfrarot, wie in Abschnitt 2.1.2 beschrieben, so lassen sich ebenfalls Tiefeninformationen, aufgrund der unterschiedlichen Streuungen des NIR in den Zahnsubstanzen, extrahieren. Es wäre demnach denkbar, die Funktionsweise des OCT durch Reflektionsaufnahmen mit NIR zu adaptieren, wie in der Konzeptzeichnung aus Abbildung 8.2 illustriert.

Mit den erzeugten Schnittbildern könnte ein bereits texturiertes 3D-Modell anschließend durch die selben mathematischen Verfahren, wie bei der Röntgentomographie, CT oder OCT, rekonstruiert werden.

### 8.2.4 Automatisierte Diagnose

Die Vision hinter all diesen Ansätzen ist eine automatisierte Diagnose. Beispielsweise soll dabei das System, während der Zahnarzt oder die Zahnärztin eine Kamera bzw. einen Scanner durch den Mundraum der Patienten führt, eine Diagnose des Gesundheitszustand der Zähne erstellen. Hierfür muss das Programm dazu in der Lage sein, den Zustand der Zähne anhand von Bildmerkmalen zu erkennen.

Durch eine Kombination einer Nahinfrarot-Bildgebung mit einem 3D-Scanner, wie beispielsweise der Omnicam der Firma Dentsply Sirona, wäre es möglich ein 3D-Modell zu erstellen und gleichzeitig NIR-Bilder der Zähne des Patienten aufzunehmen [19c]. Somit könnten die NIR-Aufnahmen bei dem Scannen eines Abschnitts gleichzeitig auf das 3D-Modell appliziert werden. Die automatisierte



**Abbildung 8.2:** Konzeptzeichnung der Kombination von NIR und Optischer Kohärenztomographie (OCT).

Diagnose des Gesundheitszustandes erfolgt dann durch Bilderkennungsverfahren. Die Zahnärzte wären dann in der Lage, die Diagnose des Systems dreidimensional visualisiert auf einen Bildschirm zu untersuchen.



# Literaturverzeichnis

- [14] [Online; accessed 30. Jan. 2019]. Apr. 2014. URL: [https://www.khronos.org/files/collada\\_schema\\_1\\_5](https://www.khronos.org/files/collada_schema_1_5) (zitiert auf S. 37, 38).
- [18a] *ISO - International Organization for Standardization*. [Online; accessed 9. Oct. 2018]. Okt. 2018. URL: <https://www.iso.org/standard/39482.html> (zitiert auf S. 19, 20).
- [18b] *NumPy — NumPy*. [Online; accessed 20. Jan. 2019]. Okt. 2018. URL: <http://www.numpy.org> (zitiert auf S. 82).
- [18c] *STL File Format (3D Printing) - Simply Explained | All3DP*. [Online; accessed 30. Jan. 2019]. Aug. 2018. URL: <https://all3dp.com/what-is-stl-file-format-extension-3d-printing> (zitiert auf S. 35).
- [19a] *8 Most Common 3D File Formats in 2019 | All3DP*. [Online; accessed 31. Jan. 2019]. Jan. 2019. URL: <https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/#collada> (zitiert auf S. 36–38).
- [19b] *Delaunay Triangulation - MATLAB & Simulink*. [Online; accessed 2. Jan. 2019]. Jan. 2019. URL: <https://www.mathworks.com/help/matlab/math/delaunay-triangulation.html> (zitiert auf S. 63).
- [19c] *Dentsply Sirona Deutschland*. [Online; accessed 10. Mar. 2019]. März 2019. URL: <https://www.dentsplysirona.com/de-de> (zitiert auf S. 110).
- [19d] *ISO/IEC 9126-1:2001*. [Online; accessed 17. Jan. 2019]. Jan. 2019. URL: <https://www.iso.org/standard/22749.html> (zitiert auf S. 83).
- [19e] *Polaris Spectra and Vicra*. [Online; accessed 23. Jan. 2019]. Jan. 2019. URL: <https://www.ndigital.com/medical/products/polaris-family> (zitiert auf S. 80, 109).
- [19f] *PyPI – the Python Package Index*. [Online; accessed 27. Feb. 2019]. Feb. 2019. URL: <https://pypi.org> (zitiert auf S. 82, 87).
- [19g] *Schneidezahn, Eckzahn und Backenzahn*. [Online; accessed 10. Mar. 2019]. März 2019. URL: <https://www.amazon.de/Erler-Zimmer-Schneidezahn-Eckzahn-und-Backenzahn/dp/B006IB2E4K> (zitiert auf S. 74).
- [19h] *Wavefront OBJ: Summary from the Encyclopedia of Graphics File Formats*. [Online; accessed 30. Jan. 2019]. Jan. 2019. URL: <http://www.fileformat.info/format/wavefrontobj/egff.htm> (zitiert auf S. 36).
- [19i] *Welcome to Click — Click Documentation (7.x)*. [Online; accessed 7. Mar. 2019]. Feb. 2019. URL: <https://click.palletsprojects.com/en/7.x> (zitiert auf S. 82).
- [19j] *Z-Buffer – Wikipedia*. [Online; accessed 25. Feb. 2019]. Feb. 2019. URL: <https://de.wikipedia.org/wiki/Z-Buffer> (zitiert auf S. 61).

- [AES17] K. Angelino, D. A. Edlund, P. Shah. „Near-Infrared Imaging for Detecting Caries and Structural Deformities in Teeth“. In: *IEEE J. Transl. Eng. Health Med.* 5 (Apr. 2017), S. 2300107. issn: 2168-2372. doi: [10.1109/JTEHM.2017.2695194](https://doi.org/10.1109/JTEHM.2017.2695194) (zitiert auf S. 16, 22, 24).
- [AH05] Y. Alshawabkeh, N. Haala. „Automatic Multi-image Photo-texturing of Complex 3 D Scenes“. In: 2005 (zitiert auf S. 62, 63, 66).
- [Bau03] A. Baumberg. „Blending Images for Texturing 3D Models“. In: *Proceedings of the British Machine Vision Conference* (Jan. 2003). doi: [10.5244/C.16.38](https://doi.org/10.5244/C.16.38) (zitiert auf S. 68–71, 102).
- [BB15] W. Burger, M. J. Burge. *Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java* (X.media.press) (German Edition). Springer Vieweg, 2015. isbn: 3642046037. URL: <https://www.amazon.com/Digitale-Bildverarbeitung-algorithmische-Einf%C3%BChrung-X-media-press/dp/3642046037?SubscriptionId=AKIAIOBINVZYXQZ2U3A&tag=chimbori05-20&LinkCode=xm2&camp=2025&creative=165953&creativeASIN=3642046037> (zitiert auf S. 32, 54, 56, 65).
- [Ble18] Blender Foundation. *blender.org - Home of the Blender project - Free and Open 3D Creation Software*. [Online; accessed 15. Oct. 2018]. Okt. 2018. url: <https://www.blender.org> (zitiert auf S. 81).
- [CRB18] L. P. Coelho, W. Richert, M. Brucher. *Building Machine Learning Systems with Python: Explore machine learning and deep learning techniques for building intelligent systems using scikit-learn and TensorFlow*, 3rd Edition. Packt Publishing, 2018. URL: <https://www.amazon.com/Building-Machine-Learning-Systems-Python-ebook/dp/B079Q7Q9R5?SubscriptionId=AKIAIOINVZYXQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B079Q7Q9R5> (zitiert auf S. 56).
- [EGP+03] S. F. El-Hakim, L. Gonzo, M. Picard, S. Girardi, A. Simoni. *Visualization of Frescoed Surfaces : Buonconsiglio Castle – Aquila Tower , “ Cycle of the Months ”*. [Online; accessed 4. Jan. 2019]. 2003. url: <https://www.semanticscholar.org/paper/Visualization-of-Frescoed-Surfaces-%3A-Buonconsiglio-El-Hakim-Gonzo/8f8c815306023aab029eeccaff85ca6f9994ffff> (zitiert auf S. 67).
- [FSZ04] C. Frueh, R. Sammon, A. Zakhori. „Automated texture mapping of 3D city models with oblique aerial imagery“. In: *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004*. Sep. 2004, S. 396–403. doi: [10.1109/TDPVT.2004.1335266](https://doi.org/10.1109/TDPVT.2004.1335266) (zitiert auf S. 67).
- [GC18a] F. Ghayour, D. Cantor. *Real-Time 3D Graphics with WebGL 2: Build interactive 3D applications with JavaScript and WebGL 2 (OpenGL ES 3.0)*, 2nd Edition. Packt Publishing, 2018. URL: <https://www.amazon.com/Real-Time-Graphics-WebGL-interactive-applications-ebook/dp/B07GVNQLH5?SubscriptionId=AKIAIOINVZYXQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B07GVNQLH5> (zitiert auf S. 29–31).
- [GC18b] F. Ghayour, D. Cantor. *Real-Time 3D Graphics with WebGL 2: Build interactive 3D applications with JavaScript and WebGL 2 (OpenGL ES 3.0)*, 2nd Edition. Packt Publishing, 2018. url: <https://www.amazon.com/Real-Time-Graphics-WebGL-interac>

- 
- tive-applications-ebook/dp/B07GVNQLH5?SubscriptionId=AKIAIOBINVYZXQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B07GVNQLH5 (zitiert auf S. 53).
- [GCPB14] F. Ganovelli, M. Corsini, S. Pattanaik, M. D. Benedetto. *Introduction to Computer Graphics: A Practical Learning Approach (Chapman & Hall/CRC Computer Graphics, Geometric Modeling, and Animation)*. Chapman und Hall/CRC, 2014. URL: <https://www.amazon.com/Introduction-Computer-Graphics-Practical-Geometric-ebook/dp/B00NFODIVQ?SubscriptionId=AKIAIOBINVYZXQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B00NFODIVQ> (zitiert auf S. 26, 27).
- [GKK+04] L. Grammatikopoulos, I. Kalisperakis, G. Karras, T. Kokkinos, E. Petsa. „Automatic multi-image photo-texturing of 3D surface models obtained with laser scanning“. In: *ResearchGate* (Nov. 2004), S. 25–27. URL: [https://www.researchgate.net/publication/228853468\\_Automatic\\_multi-image\\_photo-texturing\\_of\\_3D\\_surface\\_models\\_obtained\\_with\\_laser\\_scanning](https://www.researchgate.net/publication/228853468_Automatic_multi-image_photo-texturing_of_3D_surface_models_obtained_with_laser_scanning) (zitiert auf S. 60, 62, 65).
- [HAA16a] M. Hassaballah, A. A. Abdelmgeid, H. A. Alshazly. „Image Features Detection, Description and Matching“. In: *SpringerLink* (2016), S. 11–45. doi: [10.1007/978-3-319-28854-3\\_2](https://doi.org/10.1007/978-3-319-28854-3_2) (zitiert auf S. 54).
- [HAA16b] M. Hassaballah, A. A. Ali, H. A. Alshazly. „Image Features Detection, Description and Matching“. In: *ReseachGate* 630 (Feb. 2016), S. 11–45. doi: [10.1007/978-3-319-28854-3\\_2](https://doi.org/10.1007/978-3-319-28854-3_2) (zitiert auf S. 108).
- [Han09a] H. Handels. *Medizinische Bildverarbeitung: Bildanalyse, Mustererkennung und Visualisierung für die computergestützte ärztliche Diagnostik und Therapie*. Springer-Verlag, Juli 2009. ISBN: 978-383489571-4. URL: [https://books.google.de/books?id=VGHr\\_Gz4HJIC](https://books.google.de/books?id=VGHr_Gz4HJIC) (zitiert auf S. 55, 95).
- [Han09b] T. Hanusch. „A new texture mapping algorithm for photorealistic reconstruction of 3D objects“. In: (Okt. 2009) (zitiert auf S. 62, 63, 67).
- [JHF03] R. Jones, G. Huynh, G. Jones, D. Fried. „Near-infrared transillumination at 1310-nm for the imaging of early dental decay“. In: *Opt. Express* 11.18 (Sep. 2003), S. 2259–2265. ISSN: 1094-4087. URL: <https://www.ncbi.nlm.nih.gov/pubmed/19466117> (zitiert auf S. 16, 21, 22, 24, 25).
- [KB17] A. Kaehler, G. Bradski. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, 2017. ISBN: 1491937998. URL: <https://www.amazon.com/Learning-OpenCV-Computer-Vision-Library/dp/1491937998?SubscriptionId=AKIAIOBINVYZXQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1491937998> (zitiert auf S. 40).
- [KSS16] J. Kessenich, G. Sellers, D. Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V (9th Edition)*. Addison-Wesley Professional, 2016. ISBN: 0134495497. URL: <https://www.amazon.com/OpenGL-Programming-Guide-Official-Learning/dp/0134495497?SubscriptionId=AKIAIOBINVYZXQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0134495497> (zitiert auf S. 40).

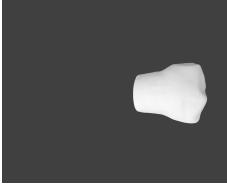
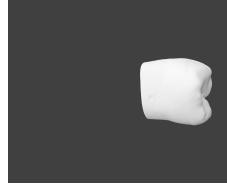
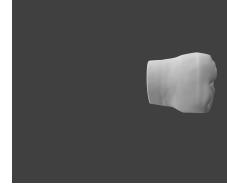
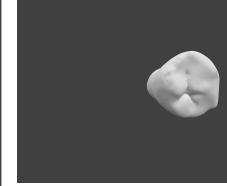
- [LHS00] H. P. A. Lensch, W. Heidrich, H.-P. Seidel. *Automated texture registration and stitching for real world models*. IEEE, Okt. 2000. ISBN: 978-0-7695-0868-9. doi: [10.1109/PCCGA.2000.883955](https://doi.org/10.1109/PCCGA.2000.883955) (zitiert auf S. 55, 68, 69).
- [LHS01] H. P. A. Lensch, W. Heidrich, H.-P. Seidel. „A Silhouette-Based Algorithm for Texture Registration and Stitching“. In: *Graphical Models* 63.4 (Aug. 2001), S. 245–262. ISSN: 1524-0703. doi: [10.1006/gmod.2001.0554](https://doi.org/10.1006/gmod.2001.0554) (zitiert auf S. 51–53, 55).
- [LKHL18] A. Lederer, K. H. Kunzelmann, R. Hickel, F. Litzenburger. „Transillumination and HDR Imaging for Proximal Caries Detection“. In: *J. Dent. Res.* 97.7 (Juli 2018), S. 844–849. ISSN: 1544-0591. doi: [10.1177/0022034518759957](https://doi.org/10.1177/0022034518759957) (zitiert auf S. 23–25).
- [Mar12] R. Marucchi-Foino. *Game and Graphics Programming for iOS and Android with OpenGL ES 2.0*. Wrox, 2012. ISBN: 9781119975915. URL: <https://www.amazon.com/Game-Graphics-Programming-Android-OpenGL/dp/1119975913?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1119975913> (zitiert auf S. 37).
- [MCV+96] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, P. Suetens. *Multi-modality image registration by maximization of mutual information*. IEEE, Juni 1996. ISBN: 978-0-8186-7368-9. doi: [10.1109/MMBIA.1996.534053](https://doi.org/10.1109/MMBIA.1996.534053) (zitiert auf S. 57).
- [MK99] K. Matsushita, T. Kaneko. „Efficient and Handy Texture Mapping on 3D Surfaces“. In: *Comput. Graphics Forum* 18.3 (Sep. 1999), S. 349–358. ISSN: 0167-7055. doi: [10.1111/1467-8659.00355](https://doi.org/10.1111/1467-8659.00355) (zitiert auf S. 51, 54, 55).
- [NB95] W. Niem, H. Broszio. „Mapping Texture From Multiple Camera Views Onto 3D-Object Models For Computer Animation“. In: *in Proceedings of the International Workshop on Stereoscopic and Three Dimensional Imaging*. 1995, S. 99–105 (zitiert auf S. 67–70).
- [NK99] P. J. Neugebauer, K. Klein. „Texturing 3D Models of Real World Objects from Multiple Unregistered Photographic Views“. In: *Blackwell Publishers Ltd and the Eurographics Association* (1999). ISSN: 1467-8659. URL: <https://diglib.eg.org/handle/10.2312/8569> (zitiert auf S. 51, 60, 61, 65).
- [PJH16] M. Pharr, W. Jakob, G. Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 2016. ISBN: 9780128006450. URL: <https://www.amazon.com/Physically-Based-Rendering-Theory-Implementation/dp/0128006455?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0128006455> (zitiert auf S. 31).
- [Pri15] L. Priese. *Computer Vision*. Springer Berlin Heidelberg, 2015. ISBN: 978-366245129-8. URL: [https://books.google.de/books?id=MgycCgAAQBAJ&pg=PA14&dq=nahes+infrarot&hl=de&sa=X&ved=0ahUKEwjV2M\\_JgLdAhVIKVAKHXRCI8Q6AEIPzAE#v=onepage&q=nahes%20infrarot&f=false](https://books.google.de/books?id=MgycCgAAQBAJ&pg=PA14&dq=nahes+infrarot&hl=de&sa=X&ved=0ahUKEwjV2M_JgLdAhVIKVAKHXRCI8Q6AEIPzAE#v=onepage&q=nahes%20infrarot&f=false) (zitiert auf S. 19).
- [RCMS99] C. Rocchini, P. Cignoni, C. Montani, R. Scopigno. „Multiple Textures Stitching and Blending on 3D Objects“. In: *SpringerLink* (1999), S. 119–130. doi: [10.1007/978-3-7091-6809-7\\_12](https://doi.org/10.1007/978-3-7091-6809-7_12) (zitiert auf S. 59, 60).

- 
- [Sch05] O. Schreer. *Stereoanalyse und Bildsynthese (German Edition)*. Springer, 2005. ISBN: 354023439X. URL: <https://www.amazon.com/Stereoanalyse-Bildsynthese-German-Schreer/dp/354023439X?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=354023439X> (zitiert auf S. 39–43, 50, 51).
- [Scr14] Scratchapixel. *Ray Tracing: Rendering a Triangle (Barycentric Coordinates)*. [Online; accessed 2. Mar. 2019]. Aug. 2014. URL: <https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates> (zitiert auf S. 90, 92).
- [Scr15] Scratchapixel. „Rasterization: a Practical Implementation“. In: *Scratchapixel* (Jan. 2015). URL: <https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation> (zitiert auf S. 91, 93).
- [Sin15] P. Singh. *OpenGL ES 3.0 Cookbook*. Packt Publishing, 2015. URL: <https://www.amazon.com/OpenGL-3-0-Cookbook-Parminder-Singh-ebook/dp/B00YN5T48C?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B00YN5T48C> (zitiert auf S. 61).
- [Tsa87] R. Tsai. „A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses“. In: *IEEE Journal on Robotics and Automation* 3.4 (Aug. 1987), S. 323–344. ISSN: 0882-4967. doi: [10.1109/JRA.1987.1087109](https://doi.org/10.1109/JRA.1987.1087109) (zitiert auf S. 49).
- [UDR+09] F. Uccheddu, A. Del Mastio, F. Remondino, A. Pelagotti, V. Cappellini. „Texture mapping of flat-like 3D models“. In: *ResearchGate* (Aug. 2009), S. 1–6. doi: [10.1109/ICDSP.2009.5201210](https://doi.org/10.1109/ICDSP.2009.5201210) (zitiert auf S. 51).
- [UPDR09] F. Uccheddu, A. Pelagotti, A. Del Mastio, F. Remondino. „Automated multispectral texture mapping of 3D models“. In: *ResearchGate* (Jan. 2009). URL: [https://www.researchgate.net/publication/228880147\\_Automated\\_multispectral\\_texture\\_mapping\\_of\\_3D\\_models](https://www.researchgate.net/publication/228880147_Automated_multispectral_texture_mapping_of_3D_models) (zitiert auf S. 51, 56, 58).
- [UPP11] F. Uccheddu, A. Pelagotti, F. Picchioni. „Automated texture registration on 3D models“. In: *Proceedings of SPIE - The International Society for Optical Engineering* 8180 (Okt. 2011), S. 6–. doi: [10.1117/12.898605](https://doi.org/10.1117/12.898605) (zitiert auf S. 52, 53, 58).
- [Vol19] Mag. M. Volgger. *Verzeichnung*. [Online; accessed 10. Mar. 2019]. März 2019. URL: [https://www.univie.ac.at/mikroskopie/1\\_grundlagen/optik/opt\\_linsen/5f\\_verzeichnung.htm](https://www.univie.ac.at/mikroskopie/1_grundlagen/optik/opt_linsen/5f_verzeichnung.htm) (zitiert auf S. 44).
- [Xu09] J. Xu. *Practical WPF Charts and Graphics (Expert's Voice in .NET)*. Apress, 2009. ISBN: 1430224819. URL: <https://www.amazon.com/Practical-Charts-Graphics-Experts-Voice/dp/1430224819?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1430224819> (zitiert auf S. 34).
- [ZB93] J. R. Zijp, J. J. Bosch. „Theoretical model for the scattering of light by dentin and comparison with measurements“. In: *Appl. Opt.* 32.4 (Feb. 1993), S. 411–415. ISSN: 1559-128X. doi: [10.1364/AO.32.000411](https://doi.org/10.1364/AO.32.000411) (zitiert auf S. 25).

- [Zij01] J.R. Zijp. „Optical properties of dental hard tissues“. In: s.n. (2001). URL: [https://www.rug.nl/research/portal/en/publications/optical-properties-of-dental-hard-tissues\(887742f6-ce4b-432a-8e36-d75368b5b851\).html](https://www.rug.nl/research/portal/en/publications/optical-properties-of-dental-hard-tissues(887742f6-ce4b-432a-8e36-d75368b5b851).html) (zitiert auf S. 21).

Alle URLs wurden zuletzt am 07.03.2019 geprüft.

## .1 Tabellen

				
1.bmp	0,589	0,448	0,373	0,392
2.bmp	0,374	0,326	0,242	0,254
3.bmp	0,353	0,341	0,247	0,261
4.bmp	0,415	0,455	0,333	0,339
5.bmp	0,315	0,339	0,379	0,362
7.bmp	0,338	0,339	0,352	0,332
8.bmp	0,278	0,314	0,379	0,333
9.bmp	0,265	0,300	0,371	0,330
10.bmp	0,281	0,319	0,365	0,337
11.bmp	0,329	0,343	0,368	0,362
12.bmp	0,371	0,364	0,313	0,327
13.bmp	0,385	0,328	0,237	0,250
14.bmp	0,351	0,298	0,214	0,226
15.bmp	0,341	0,285	0,209	0,220
16.bmp	0,338	0,292	0,212	0,227
17.bmp	0,404	0,376	0,274	0,285
18.bmp	0,328	0,378	0,378	0,385
19.bmp	0,335	0,355	0,358	0,376
20.bmp	0,284	0,312	0,343	0,332
21.bmp	0,315	0,268	0,304	0,275
22.bmp	0,341	0,381	0,436	0,413
23.bmp	0,258	0,324	0,343	0,323

**Tabelle .1:** Alle Ergebnisse der Berechnung der normalisierten MI.

Zeilen: NIR-Aufnahmen. Spalten: Gerendertes 3D-Modell. Grüne Felder: MI der NIR-Aufnahme zur zugehörigen gerenderten Ansicht. Rote Felder: Fälschlicherweise höhere oder gleich hohe MI für andere Ansichten.