

Bachelorarbeit

zur Erlangung des Grades
Bachelor of Science in Informatik

Automatische Segmentierung von Micro-CT Bildern zur Untersuchung zahnmedizinischer Strukturen

erstellt von Lukas Konietzka

Lukas Konietzka

Sebastian-Kneipp-Gasse 6A
86152 Augsburg
T +49 172-2728-376
lukas.konietzka@tha.de

Matrikelnummer:
2122553

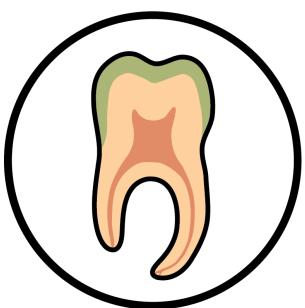
Technische Hochschule
Augsburg

An der Hochschule 1
D-86161 Augsburg
T +49 821 5586-0
F +49 821 5586-3222
www.tha.de
info@tha.de

| | |
|----------------------------|--------------------------|
| Erstprüfer | Prof. Dr. Peter Rösch |
| Zweitprüfer | Prof. Dr. Gundolf Kiefer |
| Eingereicht am | November 14, 2024 |
| Verteidigung am | März 20, 2025 |
| Geheimhaltungsvereinbarung | Nein |

Kurzfassung

3D Slicer ist eines der größten Open Source Projekt mit einer der stärksten Gemeinschaften. Es hat sich über die Jahre hinweg in den unterschiedlichsten Kliniken etablierte. So kommt es, dass auch die Klinik für Zahnerhaltung und Parodontologie in München diese Plattform bereits in ihren Forschungsalltag integriert hat. Ziel dieser vorliegenden Arbeit war die Erstellung eines Moduls für die Software 3D Slicer um so die Ärzte in der Zahnklinik zu unterstützen. Dabei sollte ein bereits prototypisch bestehender Algorithmus überarbeitet und in das Modul integriert werden. Im Laufe der Entwicklung entstand so der *Tooth Analyser*, der es möglich macht den Algorithmus mit nur wenigen Klicks über eine UI zu starten und so die Benutzerfreundlichkeit erheblich zu steigern. Neben dem Mehrwert, den die Anwendung für die praktizierenden Ärzte liefert, konnten auch viele Erfahrungen mit der 3D Slicer Entwicklung gesammelt werden. So wird beispielsweise erläutert, wie die Plug-in-Infrastruktur der Plattform funktioniert und wie schließlich sichergestellt wird, wie ein erstelltes Modul ein Softwareupdate von Slicer überlebt. Die detaillierte Evaluation des *Tooth Analyser* wird zeigen, das die geforderte Integration des Verfahrens in die Plattform 3D Slicer durchaus als Erfolg betrachtet werden kann.



ToothAnalyser

A microCT Research Tool

Inhaltsverzeichnis

| | |
|---|------|
| Abbildungsverzeichnis | vii |
| Abkürzungsverzeichnis | viii |
| Quellcodeverzeichnis | ix |
| Tabellenverzeichnis | 1 |
| 1. Einleitung | 2 |
| 1.1. Ziel der Arbeit | 2 |
| 1.2. Relevanz der Arbeit | 3 |
| 1.3. Fokus der Arbeit | 4 |
| 1.4. Aufbau der Arbeit | 4 |
| 2. Theoretische Grundlagen | 5 |
| 2.1. Domänspezifisch | 5 |
| 2.2. Bildgebung | 7 |
| 2.2.1. Computertomografie | 7 |
| 2.2.2. Datenformate | 8 |
| 2.3. Bildbearbeitung | 10 |
| 2.3.1. Filter | 10 |
| 2.3.2. Segmentierung | 12 |
| 2.4. Verwandte Arbeit | 16 |
| 2.5. Interaktive Bildbearbeitung mit 3D Slicer | 17 |
| 2.5.1. Extension Manager und Plugin Infrastruktur | 18 |
| 2.5.2. Python Umgebung | 20 |
| 2.5.3. MRML Datenstruktur | 21 |
| 2.5.4. Benutzerschnittstelle | 22 |
| 3. Forschungsfrage | 24 |
| 4. Methodik | 25 |
| 4.1. Forschungsdesign | 25 |
| 4.2. Anforderungsanalyse | 26 |

| | |
|--|----|
| 4.3. Recherche zum Stand der Kunst | 28 |
| 4.4. Zerlegung in Teilprobleme | 29 |
| 4.5. Entwicklungsumgebung | 30 |
| 4.6. Forschungsevaluation | 31 |
| 5. Ergebnisse | 32 |
| 5.1. Tooth Analyser | 32 |
| 5.2. Tooth Analyser Bibliothek | 37 |
| 5.3. Konzeptionen | 39 |
| 5.4. Technische Umsetzung | 41 |
| 5.5. Evaluierung | 46 |
| 5.5.1. Softwaretests | 46 |
| 5.5.2. Performance | 49 |
| 5.5.3. Anwendungsszenarien | 51 |
| 5.5.4. Limitierungen | 52 |
| 6. Diskussion und Fazit | 54 |
| 7. Schlussfolgerung und Ausblick | 55 |
| Literaturverzeichnis | 56 |
| A. Anhang | 58 |
| A.1. Klassendiagramm | 58 |

Abbildungsverzeichnis

| | | |
|-------|--|----|
| 1.1. | CT-Aufnahme eines Zahns nach Heck u. a. (2024) | 2 |
| 2.1. | Aufbau eines Zahnes nach K. M. Lehmann u. a. (2012) | 5 |
| 2.2. | Darstellung von Pulpa, Dentin und Schmelz auf einer CT-Aufnahme nach Heck u. a. (2024) | 6 |
| 2.3. | Einordnung der Röntgenstrahlung (X-Ray) nach Zwinkels (2015) | 7 |
| 2.4. | Maske eines lokalen Operators nach Handels (2000, Seite 52) | 10 |
| 2.5. | Interpretation einer CT-Aufnahme nach T. Lehmann u. a. (2013, Seite 360) | 12 |
| 2.6. | Ergebnis eines einfachen Schwellwertverfahrens nach Handels (2000, Seite 96) | 13 |
| 2.7. | Histogramm einer Zahnaufnahme nach Hoffmann | 14 |
| 2.8. | Reproduzierte Ergebnisansicht der anatomischen Segmentierung | 16 |
| 2.9. | Algorithmische formulierung der anatomischen Segmentierung nach Hofmann (2020) | 16 |
| 2.10. | 3D Slicer Ökosystem nach Fedorov u. a. (2012, Seite 1326) | 17 |
| 2.11. | Funktionsweise der Plugin Infrastruktur von 3D Slicer nach 3D Slicer Index (2024) | 19 |
| 2.12. | 3D Slicer High Level Architektur nach Fedorov u. a. (2012, Seite 1326) | 20 |
| 2.13. | 3D Slicer High Level Architektur nach 3D Slicer Community (2024) | 22 |
| 2.14. | Dynamische Eigenschaft einer Komponenten im Qt-Designer nach 3D Slicer Community (2024) | 23 |
| 4.1. | UML-Domänenmodell des gesamten Softwaresystems | 27 |
| 4.2. | Umggebung während der Entwicklung mit 3D Slicer und PyCharm | 30 |
| 5.1. | Logo der 3D Slicer Erweiterung "Tooth Analyser", welche im Rahmen dieser Arbeit entwickelt wurde. Logodesign: Dr. Elias Walter | 32 |
| 5.2. | Startansicht der Erweiterung Tooth Analyser nach dem ersten Aufruf | 33 |
| 5.3. | Ergebnisansicht der Erweiterung Tooth Analyser, nachdem die Analysen und die anatomische Segmentierung erstellt wurden. | 34 |
| 5.4. | Ansicht des Moduls Tooth Analyser während der Ausführung eines Verfahrens | 36 |
| 5.5. | Ausschnitt aus dem Klassendiagramm für den Tooth Analyser | 39 |

| | |
|---|----|
| 5.6. Hinzufügen neuer Funktionen zum Tooth Analyser | 40 |
| 5.7. Ausschnitt des Moduls SampleData in 3D Slicer | 47 |
| 5.8. Verteilung der Laufzeit über den gesamten Bearbeitungszeitraum. 100% entsprechen 16:27 Minuten | 49 |
| 5.9. Konstruktion des Laufzeitverhaltens bei einer Bearbeitung mehrere Bilder | 50 |
| 5.10. Rekonstruktion eines Zahns aus einer CT-Aufnahme mittels der Erweiterung Tooth Analyser. | 51 |
| 5.11. Segmentbetrachtung eines rekonstruierten Zahns auf Basis einer CT-Aufnahme. | 51 |
| 5.12. Klassifizierung von Karies mittels der medialen Flächen. | 52 |
| 5.13. Fehlerhafte Segmentierung einer CT-Aufnahme im Format 8 not unsigend integer (8UInt) | 53 |
| A.1. Klassendiagramm des Tooth Analyser | 58 |

Abkürzungsverzeichnis

| | |
|-----------------|--|
| UI | User Interface |
| CT | Computertomografie |
| MVC | Model View Controller |
| MRML | Medical Reality Modeling Language |
| GUI | Grafical User Interface |
| SEM | Slicer Extension Module |
| 3D | dreidimensonale |
| UX | User Experience |
| X-Ray | Röntgenstrahlung |
| mhd | Meta Header |
| ISQ | Industrial Scan Quality |
| 16Int | 16 bit sigend integer |
| 8UInt | 8 not unsigend integer |
| VTK | Visualization Toolkit |
| ITK | Insight Toolkit |
| ITK-SNAP | Insight Toolkit Snake Automatic Partitioning |
| JSON | JavaScript Object Notation |
| LMU | Ludwig-Maximilians-Universität München |
| GB | Gigabyte |

Quellcodeverzeichnis

| | |
|--|----|
| 2.1. Ausschnitt des Inhaltes einer mhd-Datei | 8 |
| 2.2. Auslesen der Informationen aus den verschiedenen nodes | 22 |
| 5.1. Projektstruktur des Moduls Tooth Analyser mit Fokus auf die ToothAnalyserLib | 37 |
| 5.2. Importieren von Funktionen aus der Bibliothek des Tooth Analyser | 38 |
| 5.3. Modifizierte Methode zum erstellen einer Meta Header (mhd) Datei aus einem Industrial Scan Quality (ISQ) Format | 38 |
| 5.4. Grober Aufbau der Klasse ToothAnalyser nach der Slicer Dokumentation | 41 |
| 5.5. Grober Aufbau der Widget-Klasse und deren Felder | 42 |
| 5.6. Laden der User Interface (UI) in das Modul ToothAnalyser | 42 |
| 5.7. Methode zum Beobachten von Änderungen in der Benutzerschnittstelle | 43 |
| 5.8. Die Logik-Schnittstelle des Tooth Analyser | 44 |
| 5.9. Die Parameter des Tooth Analyser, die als Attribut in der Widget-Klasse liegen. | 45 |
| 5.10. Starten des Algorithmus durch den Aufruf der <code>execut()</code> Methode in der Widget-Klasse. Die Parameter werden mit übergeben. | 45 |
| 5.11. Ein Ausschnitt der Methode <code>execute()</code> , welche die Pipeline für das Verfahren startet und in der Widget-Klasse durch den Apply-Button aufgerufen wird. | 45 |
| 5.12. Ausschnitt der Testklasse zum ausführen der Unitests | 46 |
| 5.13. Implementierung eines Tests zum überprüfen einer Funktion | 48 |

Tabellenverzeichnis

5.1. Wichtige Klassen und Methoden im Tooth Analyser 41

1. Einleitung

Die Computertomografie (CT) hat die Medizintechnik revolutioniert und ist bis heute einer der wichtigsten Methoden für die Bildanalyse. Sie ist eine der führenden Erweiterungen der klassischen Röntgentechnik. Für die Entwicklung dieser Technologie wurden Godfrey Newbold Hounsfield und Allan McLeod Cormack im Jahre 1979 mit dem Nobelpreis für Medizin ausgezeichnet (Handels 2000, Seite 12).

Die Computertomografie wird in den verschiedensten Bereichen und im wahrsten Sinne des Wortes von Kopf bis Fuß eingesetzt. So kommt es, dass auch im Dentalbereich CT Aufnahmen von größter Wichtigkeit sind. Abbildung 1.1 zeigt eine solche CT-Aufnahme. Eine konkrete Anwendung in diesem Kontext ist die Zahnkaries Forschung der Poliklinik für Zahnerhaltung und Parodontologie des LMU Klinikums in München.

Die vorliegende Arbeit soll genau diese Forschung unterstützen. In welchem Umfang und zu welchem Grund ist in den folgenden Abschnitten beschrieben.



Abbildung 1.1.: CT-Aufnahme eines Zahns nach Heck u. a. (2024).

1.1. Ziel der Arbeit

Diese Arbeit beschreibt eine Technik, mit der dreidimensionale (3D) Micro-CT Bilder zur Untersuchung zahnmedizinischen Strukturen automatisch mittels der Software *3D Slicer* segmentiert und analysiert werden können. Was genau unter einer Segmentierung verstanden wird, darüber informiert das Kapitel 2.3.2 Segmentierung. Die algorithmische Formulierung einer konkreten Segmentierung ist bereits vorhanden und prototypisch implementiert. Dieser Algorithmus hat jedoch Schwachstellen. So muss beispielsweise das Verfahren umständlich über ein ipython Notebook im Terminal ausgeführt werden,

was die Benutzerfreundlichkeit deutlich beeinträchtigt. Ziel dieser Arbeit ist es in erster Linie das bereits existierende Verfahren in der Klinik für Zahnerhaltung zu analysieren und für die Mitarbeiter der Klinik benutzbar zu machen. Dabei soll auf etablierte und vertraute Lösungen zurückgegriffen werden.

Es stellt sich nun die Frage, zu welchem Zweck eine automatische und interaktive Segmentierung überhaupt notwendig ist. Für die Zahnklinik an der LMU in München gibt es hierfür viele Gründe. Über den wichtigsten gibt das nächste Kapitel Aufschluss.

1.2. Relevanz der Arbeit

Der wohl relevanteste Punkt wurde bereits im vorherigen Kapitel 1.1 diskutiert, Zahnärzte sind keine Softwareentwickler sonder reine Anwender von Software. Darüber hinaus verfolgt die Klinik für Zahnerhaltung und Parodontologie der LMU in München einen sehr interessanten Forschungsansatz, welche eine Segmentbetrachtung der CTs rechtfertigt.

Über viele Jahre hinweg wurden in der Zahnklinik sehr viel Bilddaten von Zähnen gesammelt. Hierbei wurden Aufnahmen der unterschiedlichsten Arten gemacht. Darunter fallen zum Beispiel einfache Bilddateien, Infrarotbilder und die für diese Arbeit so relevanten dreidimensionalen Micro-CT Aufnahmen. Dieser große Schatz an Bildmaterial soll verwendet werden, um in ferner Zukunft ein neuronales Netzwerk zu trainieren, welches statistische Aussagen über das Verhalten von Karies treffen kann. Jedoch gibt es hier ein Problem, bei dem das Ergebnis dieser Arbeit Unterstützen kann. Karies auf CT-Bildern zu lokalisieren ist nicht trivial. Er ist ohne weitere Bearbeitung des Bildes nur sehr schwer auf eine Stelle einzugrenzen. So kommt es vor, dass drei verschiedene Ärzte auf dem selben Micro-CT Bild drei unterschiedliche Stellen mit Karies identifizieren. Eine Segmentierung des dreidimensionalen CTs kann hier Wunder wirken lassen. Durch die Aufteilung des Micro-CTs in seine zwei Zahnhauptsubstanzen, kann eine sehr gute visuelle Darstellung des Zahnes gewährleistet werden. Für Ärzte bietet diese Darstellung einen sehr großen Mehrwert (vgl. Walter u. a. 2025, S. 1).

Mit dieser klaren und eindeutigen Identifizierung von Karies, sind die Ergebnisse, die ein neuronales Netz generieren würde viel genauer und brauchbarer. Konkret wird mit einer automatischen Segmentierung ein *Ground Truth* gewonnen, der eine eindeutige Basiswahrheit liefert. Hierbei sei gesagt das diese Anwendung nur ein von vielen Möglichkeiten ist. Konkrete Daten über die Ausbreitung einer Krankheit im Menschlichen Körper zu besitzen kann in den verschiedensten Fällen und Institutionen von größtem Nutzen sein. So zeigen es auch Crespigny u. a. (2008) in ihrem Paper.

Anhand dieser Argumente wird deutlich, dass eine automatische Segmentierung durchaus einen Mehrwert für Ärzte bilden kann. Nicht zuletzt auch durch die enorme Zeitsparung.

Für eine automatische Segmentierung von Micro-CT Bildern gibt es einige Softwarelösungen am Markt, die alle eine gut optionen sind. Aus diesem Gurnd wird im folgenden Kapitel ein mögliches Framework diskutiert werden.

1.3. Fokus der Arbeit

Dieser Arbeit setzt den Fokus auf die Open Sorce Plattform *3D Slicer*, da diese ohnehin bereits eine breite Anwendung in der Zahnklinik in München findet. Durch die Modul und Plug-In Infrastruktur dieser Plattform kann die Softwar auch anderen Institutionen bereitgestellt werden. Hierzu muss diese einfach als *3D Slicer Extetion* bereitgestellt werden. *3DSlicer* bietet einen Extension Manager, der ähnlich wie ein App Store betrachtet werden kann. So bleibt die vorerst konkret entwickelte Software nicht nur einer Einrichtung vorbehalten. Eine tiefere Einführung in die Open Sorce Plattform bietet der Abschnitt 2.5. Das weitere Optimieren des bereits bestehenden Verfahrens wird in dieser Arbeit nicht weiter thematisiert. Es werden lediglich Anpassungen vorgenommen, sodass eine Benutzerschnittstelle verwendet werden kann.

Mit diesem Umfang, der Motiavtion und dem gesetzten Fokus, ergibt sich für dies Arbeit eine konkrete Struktur, die einen hohen detailgrad aufweisen wird. Um einen Überblick zu gewähren, sei diese Struktr hier kurz erläutert.

1.4. Aufbau der Arbeit

Die Arbeit ist in sieben Kapitel unterteilt. Nach der Einführung in Kapitel 1, in der die Relevanz und der Fokus beschrieben werden, werden in Kapitel 2 die theoretischen und technischen Grundlagen behandelt, welche zum Verstehen der Ergebnisse essenziell sind. Als Ergebnis der theoretischen Grundlagen bildet das Kapitel 3 eine konkrete Forschungsfrage. Während sich Kapitel 4 darum kümmert mit welchen Methodiken und Lösungsansätzen an die Forschungsfrage herrangegangen wird, erläutert das Kapitel 5 was die konkreten Ergebnisse der Arbeit sind. In Kapitel 6 erfolgt eine kritische Diskussio der Resultate einschließlich möglicher Limitationen. Das Abschließende Kapitel 7 fasst die wichtigsten Erkenntnisse zusammen und gibt einen Ausblick auf zukünftige Forschungsfragen.

Die theoretischen Grundlagen die wie beschrieben direkt nach der Einleitung folgen, sind zentral für das Versetehen der Fragestellung und der später folgenden Ergebnisse der Arbeit.

2. Theoretische Grundlagen

Dieses Kapitel führt in die theoretischen Grundlagen ein, die in dieser Arbeit benötigt werden. Den ersten Teil bilden die domänenspezifischen Grundlagen 2.1, welche genauer darauf eingehen, welchen Inhalt die zu bearbeitenden Bilder bieten und wie dieser zu verstehen ist. Abschnitt 2.2 geht anschließend genauer auf die Bildgebung ein, die eine wichtige Rolle spielen. Der Abschnitt 2.4 geht auf die Arbeit von Hofmann (2020) welche das zugrundeliegende Verfahren beschreibt. Die Abschnitt 2.5 führt in Softwareentwicklungsthemen ein, die zum Erstellen einer *3D Slicer Extension* wichtig sind.

2.1. Domänenspezifisch

Wie bereits aus dem Kapitel 1 Einleitung klar wurde, handelt es sich bei den Micro-CT Bilder um Zahnbilder. Um zu verstehen, wie eine CT-Aufnahme technisch segmentiert und damit zerlegt werden kann, muss zunächst die Zahnstruktur selbst verstanden werden.

Die Abbildung 2.1 zeigt den groben Aufbau eines Zahnes nach K. M. Lehmann u. a. (2012, Seite 17). Zu sehen ist, dass das Dentin oder auch Zahnbein genannt, den Großteil eines Zahnes einnimmt. Im Bereich der Zahnkrone wird das Dentin von Zahnschmelz überzogen. Der Zahnschmelz ragt in die Mundhöhle und ist nach K. M. Lehmann u. a. (2012, Seite 41) das härteste Material im menschlichen Körper. In der Mitte des Zahnes befindet sich Weichgewebe, welches als Pulpa bezeichnet wird vgl. (K. M. Lehmann u. a. 2012, Seite).

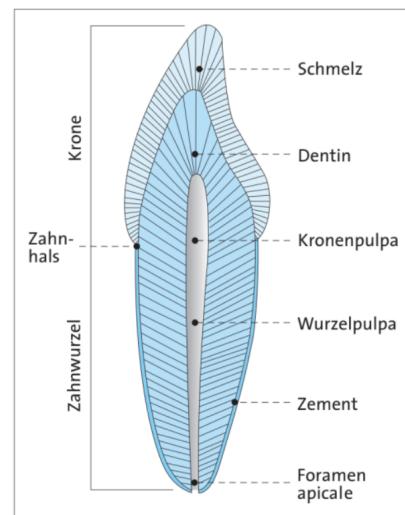


Abbildung 2.1.: Aufbau eines Zahnes nach K. M. Lehmann u. a. (2012)

Für die Bearbeitung von Micro-CT Aufnahmen sind die Bereiche Schmelz Dentin und Pulpa von besonderer Bedeutung. Betrachtet man eine CT wie es zu Beginn in der Abbildung 1.1 gezeigt wurde, so bilden diese drei Gewebearten die unterschiedlichen Grauwerte in einem CT-Bild.

Die Pulpa unterscheidet sich hierbei nur wenig vom Hintergrund, da sie als einzige der drei Hauptteile eines Zahnes ein Weichgewebe ist und bei einer Röntgenaufnahme nicht absorbiert. Dieser Teil ist in dieser Arbeit weniger relevant und auch nicht Teil des Verfahrens, was das Segmentierungsverfahren eher zu einer Zahnkronensegmentierung macht. Geht man weiter von innen nach außen, so ist der nächste Zahnteil auf einem CT das Zahnschmelz.

Das Dentin ist laut K. M. Lehmann u. a. (2012, Seite 41) eine Hartsubstanz, die dem Kieferknochen sehr nah steht. So kommt es, dass dieser Teil schon deutlich besser auf einem CT zu erkennen ist. Den äußersten Teil in der Mundhöhle bildet das Zahnschmelz.

Der Schmelz ist der härteste Teil im menschlichen Körper und aus diesem Grund auch am hellsten auf dem CT zu erkennen.

Die folgende Abbildung 2.2 sollen durch Gegenüberstellung die einzelnen Zahnsubstanzen auf einer CT Aufnahme lokalisieren.

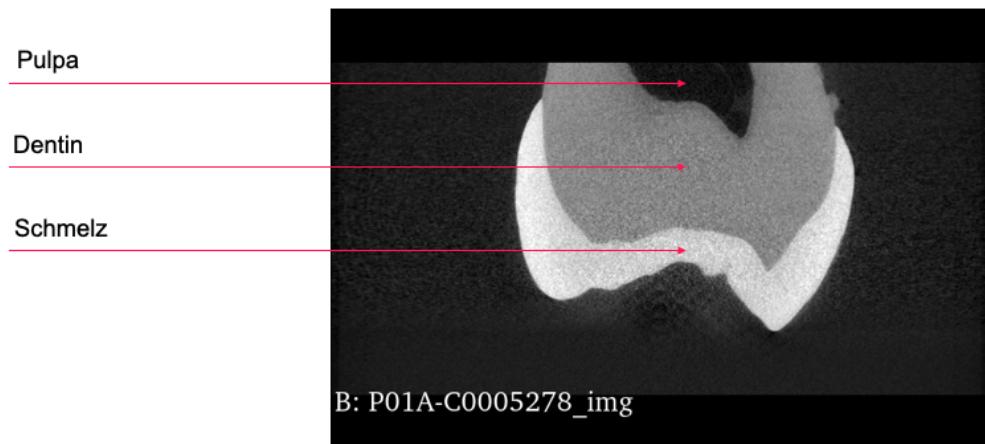


Abbildung 2.2.: Darstellung von Pulpa, Dentin und Schmelz auf einer CT-Aufnahme nach Heck u. a. (2024).

Mit diesem Domänenwissen kann ein Schritt weiter gegangen werden, sodass der Fokus nun auf die CT-Bilder gesetzt wird. Das Kapitel 2.2 Bildgebung führt die Technologie der Computertomografie tiefer ein. Darüber hinaus werden die verschiedenen Formate und statistische Modelle der CT-Aufnahmen vorgestellt.

2.2. Bildgebung

Es gibt die unterschiedlichsten Arten zur Erzeugung dreidimensionaler Bilddaten. Dieser Abschnitt erläutert federführend die Technologie der Micro-CT Aufnahmen und deren Erstellung. Diese sind für eine medizinischen Einsatz besonders interessant. Des Weiteren erfolgt eine Einführung in die Speicherung und Komprimierung von CT-Aufnahmen. Dies sorgt dafür, dass die digitalen Bilddaten deutlich handlicher werden.

2.2.1. Computertomografie

Die Erfindung der Computertomografie (CT) war ein Quantensprung in der Geschichte der Medizin. Sie ist aus heutigen Diagnosen nicht mehr wegzudenken. Ein Micro-CT Bild ist laut Baird und Taylor (2017, Abstract) ein Menge hochauflösender Bilder, die wie ein Stapel zusammengelegt werden. Der Aspekt Micro deutet dabei darauf hin, dass es eine miniaturisierte Ausführung eines üblichen Kegelstrahl-CTs ist, so Buzug (2011, Seite 340). Eine andere Definition erläutert T. Lehmann u. a. (2013). Er beschreibt die Computertomografie als Projektionen einzelner Ebenen im Untersuchungsobjekt. Die Technologie, mit der diese Bilderstapel aufgenommen werden, ist unter der Röntgentechnik oder auch X-Ray bekannt. Die Röntgenstrahlung ist eine Form der elektromagnetischen Strahlung, ähnlich wie das sichtbare Licht, so das National Institute of Biomedical Imaging and Bioengineering (NIBIB) (2024). Anders als das Licht haben die Röntgenstrahlen eine viel höhere Energie. Das führt dazu, dass man mit dieser elektromagnetischen Strahlung viele Objekte durchdringen kann. So auch Gewebeteile eines Zahnes vgl. (National Institute of Biomedical Imaging and Bioengineering (NIBIB) 2024). Die Abbildung 2.3 zeigt diese elektromagnetische Spektrum.

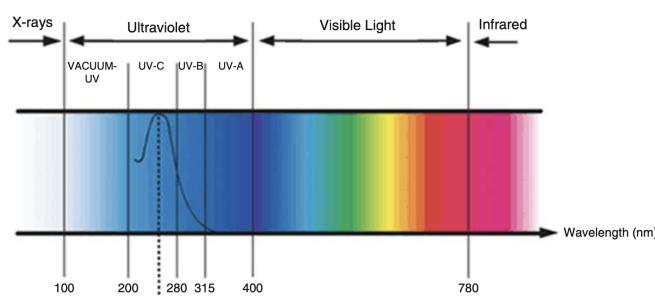


Abbildung 2.3.: Einordnung der X-Ray nach Zwickels (2015)

Durchdringt ein solcher Röntgenstrahl ein Untersuchungsobjekt, werden die Details aufgrund der Wechselwirkung mit Materie auf einer CT-Probe sichtbar. Die bekannteste Wechselwirkung ist die Absorption. Mit der Steigerung der Atomzahl in einem Material

nimmt auch die Absorption eines Materials zu, sodass es leicht ist verschiedenen Materialien in einer CT-Aufnahme zu unterscheiden (vgl. National Institute of Biomedical Imaging and Bioengineering (NIBIB) 2024).

Für eine Micro-CT Aufnahme bedarf es spezieller Technik. Es gibt unterschiedliche Firmen, welche die unterschiedlichsten Modelle anbieten. Im Falle der Zahnklinik an der LMU in München handelt es sich um ein Micro-CT 40 der Firma SCANCO Medical AG (2024). Dieses Gerät erstellt Aufnahmen mittels Röntgenstrahlung und generiert mithilfe der Computertomografie ein dreidimensionales Bild, welches im Format .ISQ abgelegt wird. Wie das nächste Kapitel beschreiben wird, ist der Speicherumfang den solch ein Bild benötigt sehr groß, jedoch dafür auch sehr detailliert. Um ein einfacheres Arbeiten mit den CT Bildern zu ermöglichen, kann eine simple Technik angewendet werden, die hier beschrieben werden soll.

2.2.2. Datenformate

Die rohen Datensätze, welche direkt aus dem Micro-CT Gerät kommen, haben nach SCANCO Medical AG (2024) das Format ISQ. Dieses Format fällt speziell auf die Geräte der Firma SCANCO zurück. Wie das vorherige Kapitel 2.2.1 bereits eingeführt hat, ist dieser Dateityp für eine weitere Bearbeitung nur bedingt geeignet. Unter anderem wegen ihrer Größe. Rösch und Kunzelmann (2018) haben hierfür ein Paket entwickelt. Dieses konvertiert ein ISQ Format in ein mhd Format. Bei einer mhd Datei handelt es sich um ein Metafile, dass auf die eigentliche Datei verweist. Folgender Ausschnitt zeigt die Verwendung des Pakets.

```
python3 isq_to_mhd.py <quelle> <ziel>
```

Diese Meta-Datei kann genutzt werden, um interessante Informationen über das Bild zu erlangen. Wird dieses Kommando ausgeführt, so erstellt das Skript `isq_to_mhd` ein Metafile, das detailliert Daten über die Datei enthält. Ein Ausschnitt dieses Metafiles liefer das Listing 2.1

```
1 ObjectType = Image
2 NDims = 3
3 CenterOfRotation = 0 0 0
4 ElementSpacing = 0.02 0.02 0.02
5 DimSize = 1024 1024 517
6 ElementType = MET_SHORT
7 ElementDataFile = P01A-C0005278.ISQ
```

2.1.Listing.: Ausschnitt des Inhaltes einer mhd-Datei

In der Datei sind Informationen über die Ausprägung, Art und Größe der Datei zu finden. Besonders interessant sind die Punkte `DimSize` und `ElementType`. Über diese Parameter lässt sich die Größe eines Bildes berechnen. Burger und Burge (2009, Seite 10-11) erklärt, dass ein Bild in Zellen aufgeteilt ist, welche Informationen enthalten. Diese Zellen sind im zweidimensionalen Raum als Pixel bekannt. Betrachtet man jedoch ein, wie im Falle der Zahnklinik an der LMU dreidimensionales Bild, so spricht man nicht mehr von einem Pixel, sondern von einem Voxel. Ein Voxel ist demnach das dreidimensionale Äquivalent zu einem Pixel. Burger und Burge (2009, Seite 10-11) beschreibt weiter das jeder dieser Zellen ein binäres Wort der Länge 2^k ist. Die Basis 2 ergibt sich durch das binäre Wort, wo hingegen für k gilt: $k \in \mathbb{N}$. Um für den konkreten Fall aus Listing 2.1 das entsprechende k zu ermitteln, muss der `ElementType` näher betrachtet werden. `MET_SHORT` steht hierbei für Signed short, was eine Größe von 16 Bit entspricht. Damit ergibt sich für die Länge k ein Wert von 4. So können nach Burger und Burge (2009, Seite 10-11) folgende Gleichungen festgehalten werden.

$$\begin{aligned} 1024 \cdot 1024 \cdot 517 &= 542,113,792 \text{ Voxel} \\ 542,113,792 \text{ Voxel} \cdot 2 \text{ Byte/Voxel} &= 1,084,227,584 \text{ Byte} \\ 1,084,227,584 / 1,000,000,000 &= 1.0842 \text{ GB} \end{aligned} \tag{2.1}$$

Die erste Gleichung bestimmt die Gesamtzahl aller Voxel in einem Bild. Gleichung 2 ermittelt die Größe des Bildes in der Einheit Byte. Die letzte Zeile nimmt eine Umrechnung von Byte nach Gigabyte (GB) vor.

Durch die Gleichungen in 2.1 wird klar, dass eine CT-Aufnahme des Typs `.ISQ` direkt nach seiner Aufnahme über einen GB groß ist. Laut Poliklinik (2024) ist dies ein zu großes Format. Es stellt sich also die spannende Frage, wie solch eine Datei komprimiert werden kann, ohne dass es Verluste in der Qualität gibt. Dr. Elisa Walter hat hierfür eine Lösung entwickelt. Betrachtet man den `ElementType` genauer, so fällt auf, dass es noch weitere Typen gibt, die durch eine geringere Länge k deutlich weniger Speicher benötigen. Durch Anwendung simpler Statistik lässt sich herauslesen, dass die 2^4 Byte je Element nicht ausgenutzt werden. Als Werkzeug für die Betrachtung einer solchen Statistik kann das Histogramm eines Bildes genutzt werden. Laut Jähne (2024, Seite 249) ist ein Histogramm die Häufigkeitsverteilung der Grauwerte. Diese zeigt grafisch die unterschiedlichen Grauwerte (X-Achse) zu ihren Häufigkeiten im Bild (Y-Achse). Jähne (2024, Seite 249) macht deutlich, dass das Histogramm jedoch kein Aufschluss über die räumliche Verteilung der Pixel oder Voxel liefert. Werden einige der Argumente nicht verwendet, so kann der `ElementType` verkleinert werden.

2.3. Bildbearbeitung

Nachdem ein CT erzeugt und gegebenenfalls komprimiert wurde, folgt die Bearbeitung eines Bildes. Hierfür bietet das Pipeline-Modell von Handels (2000, Seite 50) eine gute Richtlinie. Er beschreibt mit dieser Visualisierungs-Pipeline Schritte, die bei der Bearbeitung von dreidimensionalen CT-Aufnahmen notwendig sind (vgl. Handels 2000, Seite 50). Die ersten Schritte, *Bildvorverarbeitung* und *Segmentierung*, sind von besonderem Interesse. Dieser Abschnitt orientiert sich an dieser Unterteilung und nimmt sie als Vorbild. Daraus ergeben sich die Abschnitte 2.3.1 Filter und 2.3.2 Segmentierung, welche die Pipelineschritte *Bildvorverarbeitung* und *Segmentierung* wiederspiegeln sollen.

2.3.1. Filter

CT-Aufnahmen rauschen, dies ist ein Fakt und liegt in der Natur einer Röntgenaufnahme. Dies beschreiben auch Diwakar und Kumar (2018, Kapitel 3) in ihrem Paper über CT-Bildrauschen und Entrauschen. Dabei liegt die Ursache des Rauschens nicht an einer Stelle sondern ist auf viele Quellen zurückzuführen. Einen gute Einteilung dieser Quellen liefern ebenfalls Diwakar und Kumar (2018, Kapitel 3). Sie teilen die Rauschquellen auf in *Random noise*, *Statistical noise*, *Electronic noise* und *roundoff noise*.

Unter dem Rauschen eines Bildes versteht man die Streuung der Pixelwerte im Bild. Für eine Segmentierung des Bildes ist dieses Verhalten unerwünscht und führt zu schlechten Ergebnissen (vgl. Handels 2000, Seite 51). Die Bildvorverarbeitung oder auch Filter genannt, hat die Aufgabe dieses Rauschen so gut wie möglich zu redzieren. Hierzu gibt es diverse Möglichkeiten.

Mit Blick auf die folgenden Kapitel sind für diese Arbeit vor allem die lokalen Operatoren relevant. Die lokalen Operatoren sind charakteristisch für die Betrachtung der lokalen Nachbarschaft. Jeder Pixel betrachtet also seine Umgebung und führt auf Basis darauf eine Berechnung des jeweils betrachteten Pixels durch. in Abbildung 2.4 ist der aktuelle Pixel der mit der Position $P = (0/0)$ (vgl. Handels 2000, Seite 52).

| -2 | -1 | 0 | 1 | 2 |
|----|----|---|---|---|
| -2 | | | | |
| -1 | | | | |
| 0 | | ■ | | |
| 1 | | | | |
| 2 | | | | |

Abbildung 2.4.: Maske eines lokalen Operators nach Handels (2000, Seite 52)

Für die konkrete Betrachtung der Nachbarschaft eines Pixels empfiehlt Handels (2000, Seite 52) eine Maske (Ausschnitt) hinzuziehen, die mit einer Matrix interpretiert werden kann und die Nachbarschaft eines Pixels abbildet. Abbildung 2.4 zeigt eine solche Maske und soll das Verfahren so verdeutlichen. Der grau hinterlegte Mittelpunkt - $P = (0/0)$ - ist das aktuell betrachtete Pixel. die Felder um die Mitte herum die Nachbaren. Es fällt jedoch auf, dass durch dieses Schema nicht jede mögliche Ausprägung einer Maske in Frage kommt. Um ein Mittelpunkt und damit einen aktuellen Pixel betrachten zu können, bedarf es eines ungeraden Grades für M . Diese Eingränderung lässt sich in Gleichung 2.2 generisch fassen.

$$M_{(2m+1) \times (2m+1)} = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & x & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \quad m \in \mathbb{N} \quad (2.2)$$

Die Gleichung 2.2 beschreibt die mögliche Ausprägung eines lokalen Operators als Matrix. Dabei sei $m \in \mathbb{N}$. Die Variable x beschreibt das aktuell betrachtete Pixel, während k_{nn} die Nachbarn illustrieren soll. Durch die Gleichung ist auch zu erkennen, dass die Maske des lokalen Operators beliebig groß werden kann. Eine hohe Ordnung der Operatormatrix ist jedoch nicht immer von Vorteil, sodass es letzten Endes auf den Anwendungsfall ankommt.

Mit der Technik der lokalen Operatoren können nun unterschiedliche Arten angewendet werden. Handels (2000, Seite 54 - 55) unterscheidet hier in Glättungsfilter, Mittelwertfilter, Medianfilter, Gaußfilter und Binomialfilter. Alle dieser Filter bedienen sich einer Operatormaske um auf Basis der Nachbarelementen ein statistischen Wert für den Bildpunkt zu erhalten. Um einen genaueren Einblick in alle Filter zu erlangen, sei an dieser Stelle auf Handels (2000, Seite 54 - 55) verwiesen.

Wie zu Anfang dieses Kapitels beschrieben, ist eine Bildvorbearbeitung (Filterung) für eine gute Segmentation des Bildes unerlässlich. So kommt es das auch in der Visualisierungs-Pipeline nach Handels (2000, Seite 50) der zweite Schritt bereits die Segmentierung einführt. Warum dies so ein wichtiger Bestandteil der Bildanalyse ist und welche Methoden sich hier bieten, erläutert das folgende Kapitel.

2.3.2. Segmentierung

Die Bildsegmentierung oder auch Bildaufteilung genannt, ist ein wichtiges Teilgebiet der Bildverarbeitung und beschäftigt sich mit der Bildanalyse. Ihr Ziel ist es, detailirtere beschreibende Bilder aus dem vorliegenden Orginalbild zu berechnen. Dies kann im Falle eines CTs der Zahnklinik an der LMU München die hervorgehobene Darstellung der Zahnsubstanzen Schmelz und Dentin sein. (vgl. T. Lehmann u. a. 2013, Seite 359). Konkret teilt ein Segmentierungsverfahren also ein Bild in Teilbereiche auf. Dabei sind die Teilbereich in sich bemerkenswert homogen. Ramesh u. a. (2021, Seite 1) beschreiben, dass der Prozess der Segmentierung zur Gewinnung wichtiger Informationen dient wie zum Beispiel die Zahnekaries Ausbreitung. So kommt es, dass Handels (2000, Seite 50) in seiner Visualisierungs-Pipeline die Segmentierung als zweiten Schritt und damit als zentrales Problem darstellt. Handels (2000, Seite 95) und T. Lehmann u. a. (2013, Seite 360) beschreiben beide, dass die Bildsegmentierung eines CTs für eine gute und eindeutige Ärzliche Diagnose nicht mehr wegzudenken ist. Warum dem so ist, verdeutlicht die Abbildung 2.5.

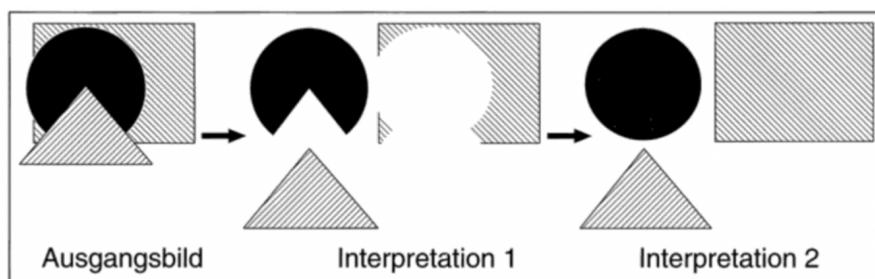


Abbildung 2.5.: Interpretation einer CT-Aufnahme nach T. Lehmann u. a. (2013, Seite 360)

Zu erkennen ist das orginale Bild (Ausgangslage) und mögliche Interpretationsschritte (Interpretation1 und Interpretation2). T. Lehmann u. a. (2013, Seite 360) verdeutlichen mit dieser Abbildung 2.5, dass mittels Segmentierung die einzige mögliche Interpretation die erste ist. Auch wenn die zweite Interpretation die deutlich logischere ist, kann diese ohne weitere Forschung nicht bewiesen werden, so T. Lehmann u. a. (2013, Seite 360). Außerdem ist zu erkennen, dass die Abbildung 2.5 die Definition einer Segmentierung belegt. Die Erzeugung inhaltlich zusammengehöriger Regionen werden hier durch die verschiedenen Formen visualisiert (vgl. T. Lehmann u. a. 2013, Seite 360).

Um ein Bild zu segmentieren gibt es unzählige Möglichkeiten. Für die Auswahl eines Verfahrens spielt unter Anderem der Anwendungsbereich eine wichtige Rolle. Die Verfahren, die in dieser Arbeit von Wichtigkeit sind, sind die Schwellwertverfahren (vgl. T. Lehmann u. a. 2013, Seite 361).

Schwellwertverfahren (engl.: thresholding) gehören zu den Standardwerkzeugen einer Segmentierung, sodass sie die Basis vieler weiterer Verfahren legen. Bei einer Schwellwertbasierten Segmentierung werden die Pixel eines Bildes anhand von Schwellwerten eingruppiert (vgl. Handels 2000, Seite 96). Die nachfolgende Gleichung 2.3 soll dies verdeutlichen.

$$B(x, y, z) = \begin{cases} 1, & \text{falls } t_{\text{unten}} \leq f(x, y, z) \leq t_{\text{oben}}, \\ 0, & \text{sonst.} \end{cases} \quad (2.3)$$

$B(x, y, z)$ beschreibt einen Pixel in einem dreidimensionalen Bild, demnach ein Voxel. Liegen die Werte eines Voxels, also $f(x, y, z)$, innerhalb der beiden Schwellwerte t_{oben} und t_{unten} , dann wird eine 1 zugewiesen. Liegt der aktuell betrachtete Voxel nicht zwischen den Schwellwerten, so wird eine 0 zugewiesen. Das Ergebnis einer solchen primitiven Schwellwertsegmentierung ist in Abbildung 2.6 zu sehen

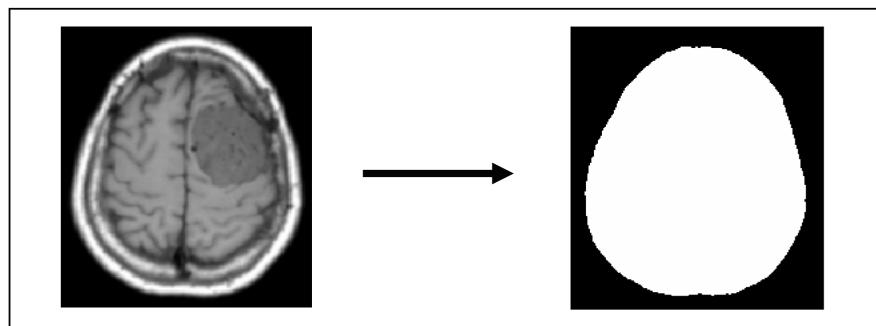


Abbildung 2.6: Ergebnis eines einfachen Schwellwertverfahrens nach Handels (2000, Seite 96)

Zu erkennen ist, dass nach einem einfachen Schwellwertverfahren das Bild nur noch aus zwei unterschiedlichen Graustufen besteht. Betrachtet man das Ergebnis in 2.6 genauer, so ist abgesehen von der Sinnhaftigkeit, diese einfache Segmentierung durchaus erfolgreich verlaufen. Der Grund dafür ist die gute Wahl des Schwellwertes.

Die interessanteste Frage bei den Schwellwertverfahren ist die Wahl des Schwellwertes t . Dieser entscheidet zwischen einer guten und einer schlechten Segmentierung. Für die Wahl eines Schwellwertes empfiehlt sich der Blick auf das Bildhistogramm. Dieses gibt Aufschluss über die Grauwertverteilung eines Bildes (vgl. T. Lehmann u. a. 2013, Seite 361). Verfahren, welche eine gute Schwellwertwahl gewährleistet, ohne das zu viele Informationen verloren gehen, sind die Verfahren *Otsu* und *Renyi*.

Das Verfahren nach Otsu gehört zu den Schwellwertverfahren und bestimmt den Schwellwert t durch ein statistische Gütekriterium. Hierzu bedient sich das Verfahren des Bildhistogrammes. Die räumliche Anordnung der Voxel und damit das tatsächliche Bild, benötigt dieser Algorithmus nicht (vgl. T. Lehmann u. a. 2013, Seite 264).

Ein solches Histogramm, welches die Grundlage für das Verfahren nach Otsu liefert sei in Abbildung 2.7 gezeigt. Dies gibt Aufschluss über die unterschiedlichen Grauwerte und wie oft sie in einem Bild vorkommen (vgl. T. Lehmann u. a. 2013, Seite 264). Für eine genauere Beschreibung eines Histogrammes, sei an dieser Stelle auf Burger und Burge (2009, Seite 42) verwiesen.

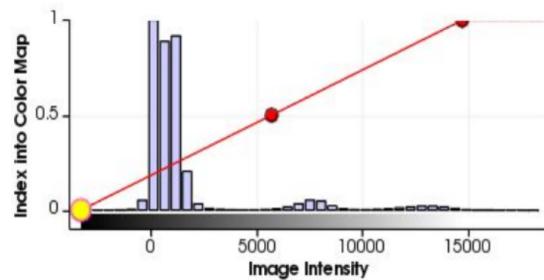


Abbildung 2.7.: Histogramm einer Zahnaufnahme nach Hoffmann

Das Otsu-Verfahren teilt die Grauwerte eines Bildes in verschiedenen Klassen ein, die durch Schwellwerte getrennt werden. Die Klassen können beispielsweise mit K_0 bis K_n bezeichnet werden, wobei sich dieses konkrete Beispiel auf die Klassen K_0 und K_1 beschränkt. Otsu wählt den Schwellwert t , der die Varianz zwischen den Pixelklassen maximiert und gleichzeitig die Varianz innerhalb jeder Klasse minimiert (vgl. T. Lehmann u. a. 2013, Seite 264). Matematisch lässt sich dies wie folgt ausdrücken.

$$t = \max (\sigma_{zw}^2 / \sigma_{in}^2) \quad (2.4)$$

σ_{zw} bildet die Varianz zwischen den beiden Klassen K_0 und K_1 und wird gebildet aus der Wahrscheinlichkeiten mit denen jeder einzelne Grauwert auftritt. σ_{in} hingegen, ist die Varianz innerhalb einer Klasse und entsteht durch die Addition der Varianzen der einzelnen Klassen. Der Schwellwert t ist nun der, für den das Verhältnis maximal wird (vgl. T. Lehmann u. a. 2013, Seite 264).

Laut T. Lehmann u. a. (2013, Seite 264) fällt auf, dass dieses Verfahren vor allem bei bimodalen Bildern zum Einsatz kommt. Ein Bild ist bimodal, wenn es zwei lokale Maxima aufweist. Das Otsu-Verfahren ist jedoch nicht auf eine Bimodalität beschränkt und kann auf beliebig viele Klassen erweitert werden (vgl. T. Lehmann u. a. 2013, Seite 264).

Eine ganz ähnliche Technik für die bestimmung des Schwellwertes liefert das Verfahren der Renyi Entropie. Auch hier ist eine einteilung der Voxel in Klassen vorgesehen.

Das Verfahren nach Rényi ist ein weiteres Verfahren, das im Laufe dieser Arbeit noch eine wichtige Rolle spielt. Wie bereits Beschriftet gehört es ebenfalls zu der Gruppe der Schwellwertverfahren und generiert demnach einen Schwellwert t . Wie auch das Verfahren nach Otsu, benötigt Rényi keine Information über die räumliche Anordnung der Bilder, es genügt das Bildhistogramm. Dabei ist der optimalen Schwellwert t der, der eine maximale Entropie der Bildverteilung erzeugt. Unter einer Entropie wird ein Konzept verstanden, das eine Unordnung, Unsicherheit oder den Informationsgehalt innerhalb eines Systems beschreibt, so Bein (2006). Die Rényi-Entropie ist eine Verallgemeinerung der Shannon-Entropie und hängt von einem Parameter q ab. Die Entropie misst die Unsicherheit oder den Informationsgehalt einer Wahrscheinlichkeitsverteilung, welche sich wie folgt ausdrücken lässt (vgl. Bromiley u. a. 2004, K. 2).

$$H_q(P) = \frac{1}{1-q} \ln \left(\sum_{i=1}^N p_i^q \right) \quad (2.5)$$

Besonderes Augenmerk verdienen hierbei die Parameter p_i und q , welche die charakteristischen Eigenschaften der Rényi-Entropie beschreiben. Der Parameter p_i ist die Wahrscheinlichkeit eines jeden Grauwertes im Bild. i symbolisiert hierbei jeden Grauwert. Wie viele Grauwerte genau betrachtet werden sollen definiert N . Die Variable q hingegen beeinflusst die Gewichtung der Wahrscheinlichkeit p_i für jeden Grauwert. Setzt man den Parameter q auf $q = 1$ so lässt sich so mittels Algebra die Shannon-Entropie zeigen (vgl. Bromiley u. a. 2004, K.2). Um nun mit der Rényi-Entropie den optimalen Schwellwert für ein Bild zu berechnen sieht Rényi ähnlich wie Otsu eine Einteilung in Klassen vor. Die Einteilung erfolgt mittels des Parameters N . So kann nun für jede gebildete Klasse die Gleichung ... angewendet werden. Die Gesamtentropie des Systems wird aus den beiden Teilentropien der jeweiligen Klassen bestimmt(vgl. Bromiley u. a. 2004, K. 2).

$$H_q(T) = H_q(P)^{(1)} + H_q(P)^{(2)} \quad (2.6)$$

Um nun den optimalen Schwellwert t bestimmen zu können muss der Wert genommen werden, bei dem die Gesamtentropie des Systems maximal ist. Dieser Sachverhalt lässt sich wie folgt ausdrücken (vgl. Bromiley u. a. 2004, K. 2).

$$t = \max(H_q(T)) \quad (2.7)$$

Neben unzähligen weiteren Segmentierungstechniken, ist eine für diese Arbeit von ganz besonderer Bedeutung. Diese Technik wurde speziell zum Segementieren der Micro-CT bilder des Zahnklinikums an der LMU in München entwickelt und bildet die Basis dieser Arbeit. Konkret ist damit das Hoffmann-Verfahren gemeint (vgl. Hofmann 2020).

2.4. Verwandte Arbeit

Wie bereits in der Einleitung dieser Arbeit klar wurde verfügt die Poliklinik für Zahnerhaltung und Parodontologie des LMU-Klinikums München über einen breiten Schatz an Bilddaten. Im Rahmen einer Bachelorarbeit an der Hochschule für angewandte Wissenschaften in Augsburg unterstützte Hofmann (2020) die Verarbeitung dieser Bilddaten mit Methoden der 3D-Bildverarbeitung. Konkret sollte die Arbeit die Kariesklassifizierung unterstützen. Hierzu entwickelte er ein Verfahren, das auf Basis adaptiver Schwellwertverfahren die Zahnsubstanzen Schmelz und Dentin aus dem Orginalbild herauslöst. Konkret kann diese Segmentierung mit verschiedenen Verfahren durchgeführt werden. Man spricht hier von einer anatomischen Segmentierung der Zahnkrone.

Durch die Segmentbetrachtung der beiden Zahnhauptteile Schmelz und Dentin konnte Hofmann (2020) eine gute Hilfe für die Befundung kariöse Stellen liefern. Ein Ergebnis aus der Arbeit von Hoffman sei in Abbildung 2.8 gezeigt. Hofmann (2020) entwickelte hierfür ein Prototypisches Verfahren, mit dem es gelang ca. 250 Datensätze der Zahnklinik automatisch aufzubereiten.



Abbildung 2.8.: Reproduzierte Ergebnisansicht der anatomischen Segmentierung

Die anatomische Segmentierung des Zahns umfasst ein ganze Reihe algorithmischer Schritte, sodass sich eine Pipeline an Verarbeitungsschritten ergibt. Die Abbildung 2.9 zeigt den groben Ablauf des Verfahrens. Kleinere Zwischenschritte wurden nicht berücksichtigt.

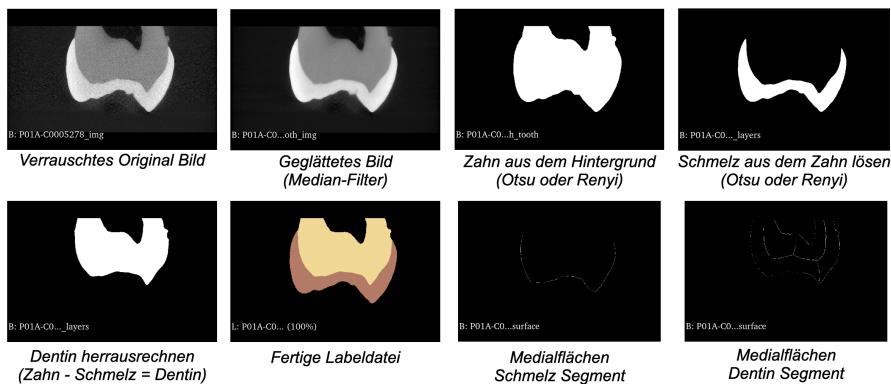


Abbildung 2.9.: Algorithmische formulierung der anatomischen Segmentierung nach Hofmann (2020)

Hofmann (2020) beschreibt, dass dieses Verfahren bis zu einem gewissen Vortschritt des Karies durchgeführt werden konnte, da der Algorithmus diesbezüglich seine Grenzen hat. Außerdem ist das Verfahren für die orginalen ISQ-Bilder erstellt worden, deren Daten im Format 16 bit sigend integer (16Int) vorliegen. Für die Spätere Darstellung der Ergebnisse kann eine überlappende Ansicht in einer Visualisierungssoftware verwendet werden. So ergibt sich die Situation, dass der Algorithmus ein gutes Ergebnis liefert, jedoch nicht benutzerfreundlich zu bedienen ist. Für das Starten und Visualisueren des Verfahrens sind aufwendige Befehle über das Terminal zu tippen (vgl. Hofmann 2020, Seite 53). Genau an dieser Stelle soll die vorliegende Arbeit anknüpfen und das Verfahren nach Hofmann (2020) so interaktiv und benutzerfreundlich gestalten.

Für eine interaktive Verarbeitung von 3D Bilddaten bieten sich einige Möglichkeiten. Die wohl beste Lösung liefer 3D Slicer. Warum die Wahl auf diese Plattform viel und welche Vorteile daraus entstehen wird im folgenden Abschnitt 2.5 erläutert.

2.5. Interaktive Bildbearbeitung mit 3D Slicer

3D Slicer ist eine Open Source Plattform, die speziell für die Verarbeitung von Bilddaten im medizinischen Kontext eingesetzt wird. Dabei wird sie von einer aktiven Community regelmäßig gewartet und weiterentwickelt (vgl. 3D Slicer Community 2024), (vgl. Fedorov u. a. 2012). Für Slicer gibt es offiziell keine Nutzungsbeschränkung. Jedoch sei auch gesagt, dass 3D Slicer nicht für die klinische Nutzung zugelassen ist. Fedorov u. a. (2012) macht deutlich, dass 3D Slicer ausschließlich für die Forschung gedacht ist. Um einen ersten Überblick über die Komponenten von Slicer zu erlangen, soll die Abbildung 2.10 betrachtet werden.

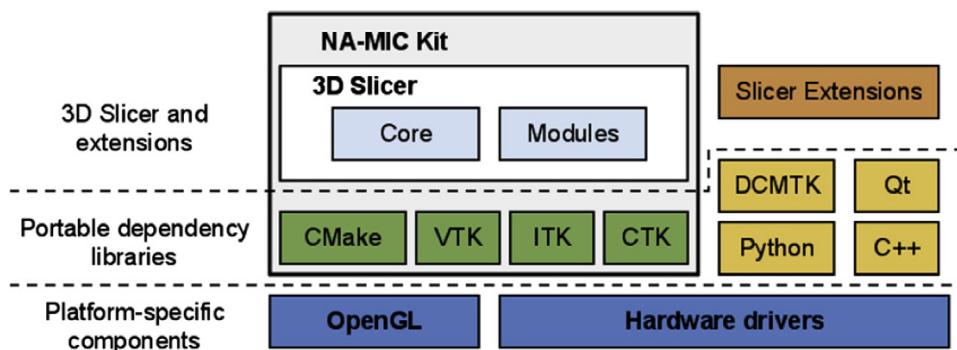


Abbildung 2.10.: 3D Slicer Ökosystem nach Fedorov u. a. (2012, Seite 1326)

Fedorov u. a. (2012, Seite 1326) teilt mit der Abbildung 2.10 die Plattform in drei Schichten auf. Auf der obersten wird klar, dass 3D Slicer aus der Kernanwendung und

den Installierbaren Modulen besteht. Neben den bereits vorhandenen Modulen können von externen Entwicklern Module über die Slicer Extentions entwickelt und bereitgestellt werden. Um eine Weiterentwicklung möglich zu machen hat Slicer eine Reihe von Abhängigkeiten, die jedoch portabel gehalten werden. Auf der untersten Schicht sind die Platformspezifischen Anforderungen zu sehen, die Slicer erfüllen soll.

So kommt es, dass das 3D Slicer Ökosystem sich durch einige Kriterien besonders auszeichnet. Die wohl wichtigsten seien hier Stichpunktartig genannt (vgl. 3D Slicer Community 2024), (vgl. Fedorov u. a. 2012).

- Kostenfreie Software
- Plug-In Infrastruktur durch den Extension Manager
- Ausführen von Skripten in der integrierten Python Konsole
- Verarbeitung von medizinischen Bildern von Kopf bis Fuß
- Interaktive Benutzerschnittstelle

3D Slicer hat für alle diese Punkte jeweils eine Lösung entwickelt, wobei der erste Punkt durch die Open Source Philosophie schon gegeben ist. Die folgenden Abschnitte decken diese Lösungen ab und bilden so eine erste Grundlage für die Entwicklung mit 3D Slicer(vgl. 3D Slicer Community 2024), (vgl. Fedorov u. a. 2012).

2.5.1. Extension Manager und Plugin Infrastruktur

Der wohl bedeutsamste Punkt ist die Plug-In Infrastruktur, welche Slicer von sich aus mitbringt. Um dieses Konzept genau zu beläuchten Teilt man die Platform am besten in zwei Teile auf. Die Kernanwendung und die Module, welcher jeder User personalisiert installieren oder deinstallieren kann. Diese Module werden als *Slicer loadable module* bezeichnet (vgl. Fedorov u. a. 2012, Seite 1332). Slicer realisiert die Struktur durch den Extension Manager, welcher durchaus vergleichbar ist mit einer Art AppStore. Über diesen können bequem und mit wenig Klicks die gewünschten Erweiterungen in das Kernsystem installiert werden.

Neben der Möglichkeit Module zu installieren bietet Slicer noch die Möglichkeit eigenen Module zu bauen und Sie im Extension Manager zu veröffentlichen. Diese werden als Slicer Extension Module (SEM) bezeichnet. Hierzu verfolgt Slicer den Ansatz, dass jeder Entwickler eines Moduls selbst verantwortlich für Wartung und Weiterentwicklung ist. Auch nachdem ein Paket veröffentlicht wurde (3D Slicer Community 2024).

Slicer realisiert dies, indem die Plattform über ein zusätzliches Repository verfügt, dass sich *ExtensionIndex* nennt. Dieses öffentliche Repository ist eine Auflistung aller Slicer Extentions. Die Auflistung erfolgt durch eine Reihe an JavaScript Object Notation (JSON) Datein, die auf die Repositories der einzelnen Entwickler verweisen. Dieser *ExtensionIndex* ist über die Slicer Factory an den Extension Server und damit auch an den Extension Manager angebunden. Die Slicer Factory ist ein System, das aus einem Slicer Extension Repository ein lauffähiges Build erstellt, welches in den Extension Katalog eingebunden werden kann. Ist eine Extension in dem Extension Katalog gelistet, so sorgt der Extension Manager dafür, dass die von der Slicer Factory erstellt build-Datei installiert werden kann. Abbildung 2.11 soll diesen Vorgang verdeutlichen (vgl. 3D Slicer Community 2024).

Die Kernanwendung von 3D Slicer folgt einem Softwarepattern, dass sich Model View Controller (MVC) nennt. Bei der Erstellung einer SEM soll dieser Ansatz ebenfalls gepflegt werden. Eine High Level Betrachtung der Softwarearchitektur von 3D Slicer bietet Fedorov u. a. 2012, Seite 1332 mit der Abbildung 2.12.

Das Zusammenspiel zwischen Medical Reality Modeling Language (MRML), Grafical User Interface (GUI) und der Logic bilden das MVC-Pattern in der Kernanwendung. Das identische Pattern spiegelt sich auch in den einzelnen Modulen von Slicer wieder. So wird sichergestellt, dass ein Softwareentwicklungsparadigma eingehalten wird, was sich *separation of concerns* nennt. Die Kapselung von zusammengehöriger Logik. Bei der Erstellung einer eigenen Extension ist die Idee, dass nur die Logic implementiert werden muss und die komplexe Architektur von Slicer erstmal nicht relevant ist.

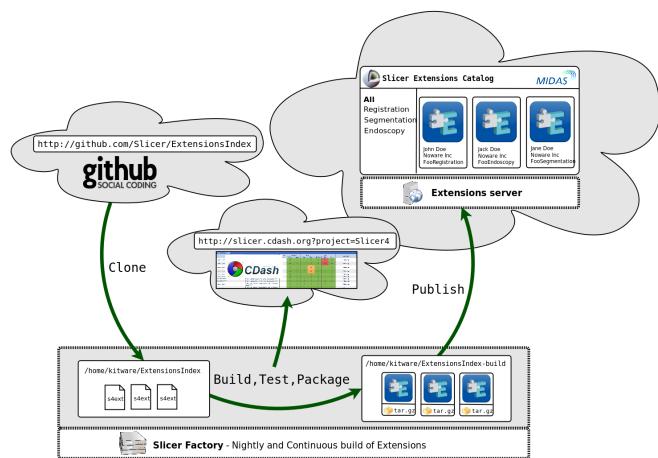


Abbildung 2.11.: Funktionsweise der Plugin Infrastruktur von 3D Slicer nach 3D Slicer Index (2024)

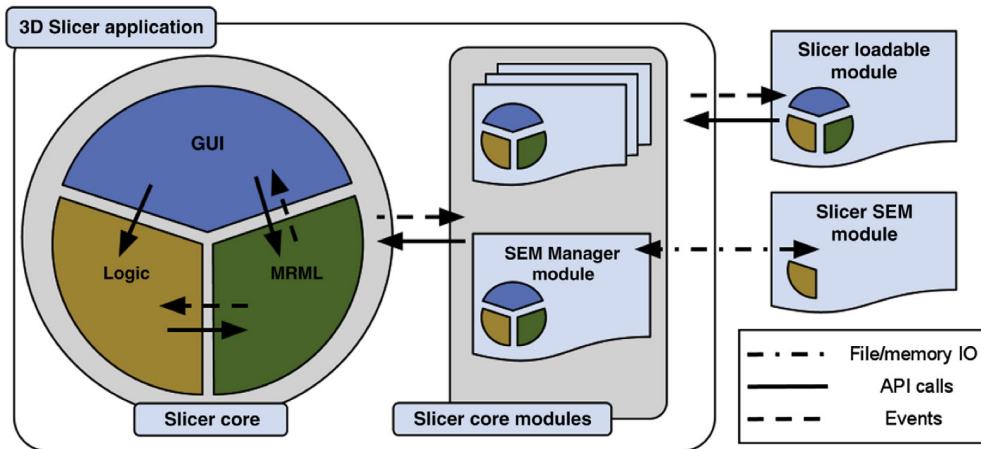


Abbildung 2.12.: 3D Slicer High Level Architektur nach Fedorov u. a. (2012, Seite 1326)

Jedoch bietet sich in Slicer nicht nur die Möglichkeit eigenen Erweiterungen zu erstellen. Es lässt sich hierfür auch die integrierte Python Konsole nutzen.

2.5.2. Python Umgebung

3D Slicer bringt eine integrierte Python Konsole mit, über die mit der Datenstruktur interagiert werden kann. So ist es möglich Python Skripte direkt in der Konsole auszuführen. Um dies zu realisieren bringt Slicer mit der Installation im jeweiligen Betriebssystem eine eigenen Python Umgebung mit. Dieses sieht wie folgt aus.

```
./Slicer/bin/PythonSlicer
```

Diese Python Umgebung verfügt über alle notwendigen Abhängigkeiten und Pakete. Bei der Entwicklung eines SEM kann dann auf die Pip-Pakete in der integrierten Python Umgebung zurückgegriffen werden. So kommt es das für eine Entwicklung mit Slicer keine eigenen Python Umgebung auf der lokalen Maschine installiert sein muss. Slicer bringt hier alls mit.

Für den letzten charakteristischen Punkt von Slicer aus Kapitel 2.5 führt der nächste Abschnitt in die durchaus komplexe Datenstruktur MRML ein, die bei einer Entwicklung mit Slicer unausweichlich zu berücksichtigen ist.

2.5.3. MRML Datenstruktur

Die MRML, gesprochen "Murlm" ist ein Datenmodell das dafür entwickelt wurde alle möglichen Bilddaten zu visualisieren und zu speichern, die für einen klinischen Zweck Einsatz finden (vgl. 3D Slicer Community 2024). Laut der 3D Slicer Community (2024) wurde die MRML-Datenstruktur völlig unabhängig von der Slicer Kernanwendung entwickelt. Dies ermöglicht ein portieren der Datenstruktur auf andere Softwareapplikationen. Da Slicer die einzige große Plattform ist, die diese Datenstruktur nutzt, wird der Quellcode für MRML im Repository von 3D Slicer gewartet und weiterentwickelt, so die 3D Slicer Community (2024). Durch den Artikel von Fedorov u. a. (2012, Seite 1327) wird klar, dass MRML mehr ist also nur eine Datenstruktur. Sie beschreiben MRML als Szenenorganisator von Bildern, Annotierungen, Layouts und Anwendungsstaten.

Fedorov u. a. (2012, Seite 1331) beschreiben die MRML-Datenstruktur als Schlüsselkomponenten innerhalb von 3D Slicer. Dies ist auf die Softwarearchitektur von Slicer zurückzuführen, die in Abbildung 2.12 beschrieben wurde. Die Kernanwendung von Slicer arbeitet wie bereits beschrieben nach dem MVC-Pattern. MRML übernimmt hier den Teil des *Models (M)* und bildet damit den Grundstein der Anwendung (vgl. Fedorov u. a. 2012, Seite 1332).

Die 3D Slicer Community (2024) und der Artikel von Fedorov u. a. (2012, Seite 1327) beschreibt MRML als XML-Format. Wird also eine MRML-Szenen gespeichert, so folgt eine Speicherung als .mrml-Datei und damit unter der Haube als XML-Datei. Dabei wird laut 3D Slicer Community (2024) nur eine Referenz auf das Bild gespeichert. Die zu bearbeitende Aufnahme selbst wird nicht innerhalb einer MRML-Datei abgespeichert.

MRML zeichnet sich vor allem dadurch aus, dass es eine Vielzahl an Dateiformaten akzeptiert. Alle Formate, die für einen klinischen Zweck verarbeitet werden, können durch MRML unterstützt werden. Um dies zu gewährleisten, ist die MRML Szene in sogenannte *nodes* aufgeteilt. Die basis Node-Typen folgen der 3D Slicer Community (2024) und sind in der folgenden Aufzählung zu sehen.

- Data nodes
- Display nodes
- Storage nodes
- View nodes
- Plot nodes
- Subject hierarchy node
- Sequence node
- Parameter node

Wird also ein Bild in eine MRML-Szene geladen, so speichert Slicer die unterschiedlichen Eigenschaften eines Bildes in unterschiedlichen nodes. So werden Beispielsweise basis Eigenschaften einer Probe im *Data node* gespeichert, wo hingegen ein *Storage node* beschreibt wie eine *Data node* in einer Datei gespeichert wird. In *Display node* werden die Eigenschaften zur Darstellung eines Bildes hinterlegt. Der Hintergrund für die Speicherung von Probendaten in unterschiedlichen nodes ist, dass beispielsweise das

selbe Bild in unterschiedlichen Formaten vorliegt oder ein und das selbe bild auf zwei unterschiedliche Arten visualisiert werden soll. So kann sich Beispielsweise eine Struktur wie in Abbildung 2.13 ergeben.



Abbildung 2.13.: 3D Slicer High Level Architektur nach 3D Slicer Community (2024)

Die Informationen in einem Bild werden also über diese Typen aufgeteilt und je nach Sinn abgespeichert. Möchte man demnach auf die bestimmte Informationen in einer Probe zugreifen. So kann diese Information über den Aufruf bestimmter Methoden erfolgen

```

1 # data node - vtkMRMLVolumeNode
2 currentVolume.GetImageData()
3 # storage node - vtkMRMLStorableNode
4 currentVolume.GetStorageNode()
5 # display node - vtkMRMLDisplayableNode
6 currentVolume.GetDisplayNode()
  
```

2.2.Listing.: Auslesen der Informationen aus den verschiedenen nodes

Wie die Kommentaren in Listing 2.2 bereits zeigen, gibt es noch eine Besonderheit von MRML. Damit eine Verwaltung aller Dateiformate möglich ist, bedient sich MRML einiger Tools, die sich bereits etabliert haben. Die wichtigsten sind hier bei Visualization Toolkit (VTK) und Insight Toolkit (ITK) (vgl. VTK Development Team 2024), (vgl. ITK Development Team 2024). Diese beiden Tools sind echte Riesen in ihrer Branche. MRML nutzt diese um einige Dateiformate zu lesen und zu schreiben.

Durch das Betrachten der MRML-Szene wird klar, dass Slicer hierdurch viele Möglichkeiten bietet. Speziell für die effiziente Speicherung der Proben in einer Szene durch die unterschiedlichen node Typen. Ein besonderer node, der gleichzeitig auch die Brücke zu der interaktiven Benutzerschnittstelle von Slicer baut, ist der *Parameternode*. Warum dieser eine zentrale Rolle spielt und wie Slicer die Schnittstelle grundsätzlich gestaltet, soll in Kapitel 2.5.4 Benutzerschnittstelle diskutiert werden.

2.5.4. Benutzerschnittstelle

Für das erstellen eines UI, das für eine Slicer Extension nötig ist, nutzt 3D Slicer den Qt-Designer (vgl. Qt Development Team 2024). Die Integration des Qt-Designers als Applikation in eine andere Applikation funktioniert aufgrund der Plattformintegrität, die der Designer mitbringt (vgl. Qt Development Team 2024). Dieser bietet so die Möglichkeit die benötigten Widgets über ein interaktives User Interface zu bauen. Für

dieses UI-Widget gibt es einen Gegenspieler im Quelltext des Programmes, welcher als *ParameterNode* bekannt ist. Der *ParameterNode* ist laut 3D Slicer Community (2024) eine leichte Variante eines MRML-Node um Parametereinstellungen zu speichern. Durch das Zusammenspiel zwischen UI-Widget und *ParameterNode* wird die UI automatisch aktualisiert, wenn sich das Programm ändert und umgeht, so erklärt es die 3D Slicer Community (2024).

Das Erstellen der Verknüpfung zwischen UI-Widget und *ParameterNode* erfolgt über die dynamische Eigenschaft *SlicerParameterName*, die direkt in der Komponentenansicht im Qt-Designer einstellbar ist. Die Abbildung 2.14 soll diesen Vorgang verdeutlichen. Diese Verknüpfung lässt sich laut 3D Slicer Community (2024) auch via Programmcode setzen.

```
widget.setProperty('SlicerParameterName', 'parameterName')
```

Über das Objekt *widget* kann die eigenschaft einer Komponente gesetzt werden, ohne dass sie im Designer berührt werden muss.

Mit dem Ende dieses Abschnittes wurden alle wichtigen Bestandteile von 3D Slicer abgedeckt und diskutiert, sowie alle weiteren Domänen eingeführt. So bleibt nun die Frage nach dem Sinn dieser Arbeit. das Kapitel 3 soll hier Klarheit liefern und die konkrete Fragestellung ausarbeiten.

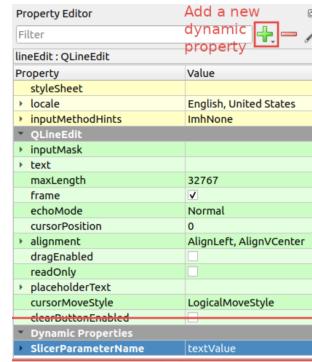


Abbildung 2.14.: Dynamische Eigenschaft einer Komponenten im Qt-Designer nach 3D Slicer Community (2024)

3. Forschungsfrage

Dieses Kapitel beleuchten die zentrale Frage, welche mit Hilfe der Ergebnisse dieser Arbeit beantwortet werden sollen. Dabei kann die Fragestellung in erster Linie als Ergebnis der theoretischen Grundlagen Kapitel 2 interpretiert werden.

Die vorliegende Arbeit befasst sich mit der Erstellung einer Slicer-Extension, die ein spezifisches Segmentierungsverfahren auf Basis der Methode nach Hoffmann integriert. Die Grundlagen für diese Methode bilden die theoretischen Grundlagen.

Die Methode nach Hoffmann nutzt bestehende Segmentierungsmethoden und wurde speziell für die Anforderungen der Zahnklinik in München entwickelt. Sie ist prototypisch implementiert und zeigt in ihrer Funktionalität vielversprechende Ergebnisse. Allerdings hat das aktuelle Verfahren eine wesentliche Einschränkung: Es muss über das Terminal ausgeführt werden. Dies erschwert die Anwendung in der klinischen Praxis und reduziert die Benutzerfreundlichkeit erheblich.

Als interaktive Lösung bietet sich die Software 3D Slicer an, da sie bereits in der Zahnklinik eingesetzt wird und über eine flexible Infrastruktur für die Integration von Erweiterungen verfügt. Diese Eigenschaften machen Slicer zu einer idealen Plattform, um die Methode nach Hoffmann in einer benutzerfreundlichen Form bereitzustellen.

Die zentrale Frage, die so aus den Grundlagen abgeleitet werden kann, ist:

1. Kann das Verfahren der anatomischen Segmentierung als Extension in Slicer implementiert werden?
2. Ist es möglich, die Integration so zu gestalten, dass der zugrunde liegende Algorithmus problemlos austauschbar ist oder zusätzliche Algorithmen eingebaut werden können?
3. Welche Erfahrungen lassen sich bei der Entwicklung einer Slicer-Extension sammeln, und welche Herausforderungen treten dabei auf?

Im folgenden Kapitel wird dargestellt, wie diese Fragestellung methodisch bearbeitet wurde. Es wird gezeigt, wie das Problem strukturiert in Teilaufgaben zerlegt wurde und welche Schritte zur Lösung unternommen wurden.

4. Methodik

Dieses Kapitel beschreibt das methodische Vorgehen, das zur Beantwortung der Forschungsfrage gewählt wurde, um aussagekräftige und reproduzierbare Ergebnisse zu erzielen. Eine nachvollziehbare Methodik ist essenziell, um die Ergebnisse sowohl evaluierbar als auch für zukünftige Arbeiten nutzbar zu machen. Das Hauptziel dieser Arbeit ist die Entwicklung einer stabilen und voll funktionsfähigen Erweiterung für die Software 3D Slicer, die in der Klinik eingesetzt werden kann. Zu Beginn wurde demnach eine umfassende Anforderungsanalyse durchgeführt, um die spezifischen Anforderungen der Domäne zu erfassen und die Ausgangssituation zu klären. Darauf aufbauend folgte eine detaillierte Literaturrecherche, um den aktuellen Stand der Technik zu untersuchen und bestehende Lösungen zu identifizieren. Da das Ziel dieser Arbeit die Entwicklung einer vollständigen Softwarelösung ist, wurde das Problem anschließend in Teilaufgaben zerlegt. Dies ermöglicht eine gezielte Bearbeitung einzelner Komponenten und erleichtert die iterative Entwicklung. Falls für bestimmte Teilbereiche keine passenden Lösungsansätze aus der Literatur ableitbar waren, wurden darauf basierend eigene methodische Ansätze erarbeitet.

Neben der praktischen Anwendung der entwickelten Erweiterung bietet diese Arbeit auch wissenschaftlichen Mehrwert. Daher wird im folgenden Abschnitt die gewählte Methodik detailliert begründet und deren Vorteile herausgearbeitet.

4.1. Forschungsdesign

Das Forschungsdesign dieser Arbeit folgt einem praktischen Entwicklungsansatz mit einem Fokus auf softwaretechnische Methoden. Zum Erreichen der Ziele stützt sich diese Arbeit so am Entwicklungsprozess ab und dokumentiert diesen. Dabei lässt sich der gesamte Zeitraum dieser Arbeit in drei Phasen aufteilen, die jeweils einem unterschiedlichen Zweck dienen. Diese drei Phasen sollen auch eine grobe Orientierung bezüglich der Reihenfolge während der Bearbeitung geben.

Analysephase Diese erste Phase ist bei fast allen Softwareprojekten die wichtigste Phase und gleichzeitig die, die meist zu kurz kommt. Innerhalb der Analysephase werden also alle Anforderungen an die Software gesammelt. Diese basieren zum großen Teil auf der Literaturrecherche. Außerdem werden bestehende Lösungen analysiert und so die Kernfunktionalität herausgefiltert.

Entwicklungsphase Die Entwicklungsphase bildet den größten Teil. Hier findet die konkrete Umsetzung statt. Hierzu wird das System in mehrere Subsysteme unterteilt. Dies ermöglicht eine isolierte Betrachtung. Während der Entwicklung wird ein Phototypenansatz verfolgt.

Evaluationsphase Die letzte Phase dieser Arbeit beschäftigt sich ausschließlich mit der Evaluation der Ergebnisse. Hier soll eine Antwort auf die in 3 formulierten Fragestellungen gefunden werden.

Durch diese Unterteilung ist eine gutes strukturelles vorgehen möglich um mittels einer praktischen Umsetzungsmethodik zu einem guten Ergebnis zu kommen. Die nächsten Kapitel blicken nun in die einzelnen Phasen, beginnend mit einer Anforderungsanalyse.

4.2. Anforderungsanalyse

Nach genauerem Betrachten der Fragestellung aus Kapitel 3 und den Zielen aus 1.1 können bereits einige Anforderungen abgeleitet werden, die für die Erweiterung gelten sollen. Neben diesen Anforderungen wurden auch die Klinik für Zahnerhaltung mit in diesen Prozess eingebunden. Hierzu wurde innerhalb eines Meetings mit dem verantwortlichen Arzt, Dr. Elias Walter, ein Anforderungskatalog ausgearbeitet (vgl. Walter 2025). Diese Anforderungen waren vor allem zu Beginn der Entwicklung sehr wichtig um einen ersten Anhaltspunkt zu gewinnen. Im Laufe des Entwicklungsprozesses wurden Statusberichte eingeplant, die ein Reagieren auf Anforderungsänderungen ermöglichen sollen.

In erster Linie wird klar, dass im Rahmen dieser vorliegenden Arbeit eine Extension für die Plattform 3D Slicer entwickelt werden soll. Die Kernfunktionalität soll dabei die anatomische Segmentierung bilden, wie sie in Kapitel 2.4 beschrieben wurde. Greift man das Ziel dieser Arbeit aus der Einleitung 1.1 nochmals auf, dann kann hierdurch die nächste wichtige Anforderung abgeleitet werden. Die Erweiterung soll gut und einfach über ein User Interface (UI) bedient werden können. Außerdem ist eine stabile Anwendung gefragt, die sich gut in die Kernanwendung von 3D Slicer einfügt. Walter (2025) machte im Interview deutlich, dass die Extension neben einer Einzelbildbearbeitung auch einen Batch-Prozess ermöglichen soll. So können Beispielsweise Parameter an einem Bild erprobt werden und diese anschließend in einen Batch-Prozess für viele Bilder überführt werden. Außerdem soll es möglich sein, verschiedenen Schwellwertverfahren, die in der anatomischen Segmentierung vorgesehen sind, auch in der Extension auszuwählen. Ein

wichtiger Softwaretechnischer Anspruch an die Extension ist die Erweiterbarkeit. Es soll ohne große Mühen möglich sein, ein weiteres Verfahren zu integrieren, ohne das große Anpassungen an der UI oder der Erweiterung selbst unternommen werden müssen. Für ein solides Verständnis dieser Software soll es selbstverständlich eine Dokumentation mit Benutzerhandbuch geben. Zudem wird großer Wert auf die Qualitätssicherung gelegt, weshalb eine Reihe von Unit-Tests (Tests für einzelne Programmeinheiten) vorgesehen ist. Um die Anforderungen an die Software besser zu verstehen und zu strukturieren, ist neben der Sammlung technischer Spezifikationen auch ein solides Verständnis für die zugrunde liegende Domäne essenziell. Die Abbildung 4.1 veranschaulicht dies durch ein UML-Domänenmodell (Unified Modeling Language), das einen visuellen Überblick über die verschiedenen Teile der Software bietet. Dazu sind auch einige der Anforderungen erkennbar (vgl. Walter 2025).

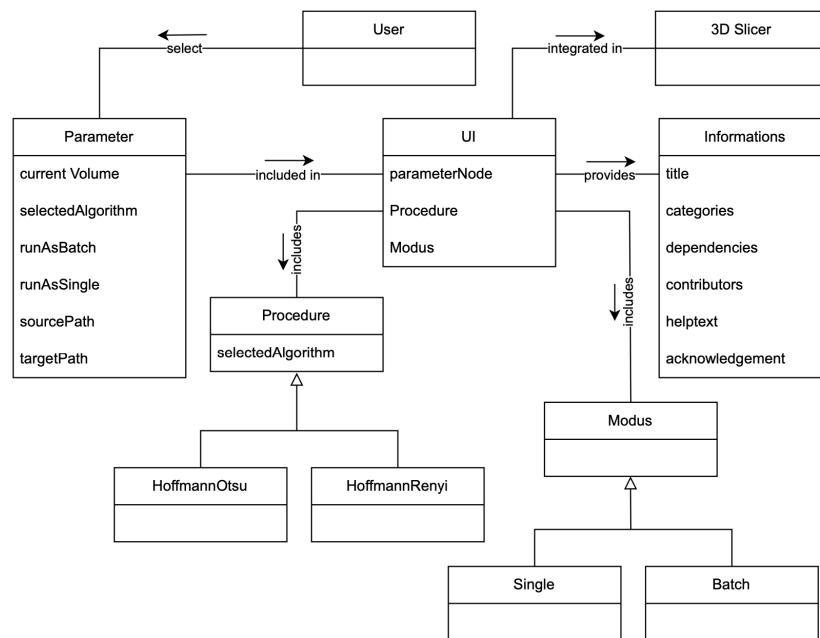


Abbildung 4.1.: UML-Domänenmodell des gesamten Softwaresystems

Durch diese breite Palette an Anforderungen ergeben sich verschiedene Aufgaben für die Implementierung. Bevor jedoch mit der konkreten Umsetzung begonnen werden kann, ist ein noch wichtigerer Schritt erforderlich: die Recherche. Sie dient dazu, den aktuellen Stand der Technik zu erfassen und geeignete Lösungsansätze zu identifizieren.

4.3. Recherche zum Stand der Kunst

Es wäre äußerst ungünstig, erst am Ende eines Projekts festzustellen, dass bereits veröffentlichte Lösungen existieren, in die erhebliche Ressourcen investiert wurden. Um dies zu vermeiden, ist eine umfassende Literaturrecherche essenziell, die den aktuellen Stand der Technik abbildet. Dabei wird auf Fachliteratur sowie domänenspezifische Quellen zurückgegriffen, um alle relevanten Aspekte abzudecken.

Für diese Arbeit spielt eine Quelle eine besonders wichtige Rolle: die offizielle Dokumentation von 3D Slicer Community (2024). Sie bietet wertvolle Anhaltspunkte für die Implementierung und hilft dabei, die technischen Gegebenheiten von 3D Slicer zu verstehen. Zudem enthält sie *Best-Practice-Ansätze*, die bei der Entwicklung berücksichtigt wurden. 3D Slicer stellt außerdem einen Developer Guide zur Verfügung, der Teil der offiziellen Dokumentation ist und den Einstieg in das Framework erleichtert. Ein weiterer zentraler Referenzpunkt ist der 3D Slicer Extension Index, in dem bereits entwickelte Erweiterungen einsehbar sind. Ein konkretes Beispiel ist das Modul *Airway Segmentation*, dessen Analyse dazu beiträgt, bewährte Konventionen für die Entwicklung der eigenen Erweiterung abzuleiten. So kommt es, dass beispielsweise für die Gestaltung einer Benutzerschnittstelle es bereits Lösungen gibt, die sich bewährten und so übernommen werden können.

Neben einer konkreten Implementierungshilfe dient die Literaturrecherche auch dazu, ein fundiertes Verständnis für die Domäne der medizinischen Bildverarbeitung und deren zugrunde liegende Strukturen zu entwickeln. Mithilfe verschiedener domänenspezifischer Publikationen kann ein tieferes Wissen über diesen Fachbereich gewonnen werden. Besonders relevant sind hierbei die verschiedenen Verfahren für die Verarbeitung der Micro CT Aufnahmen. Konkret handelt es sich hier um die unterschiedlichen Algorithmen zur Filterung und Segmentierung von Micro CT Bildern in der Zahnmedizin.

Darüber hinaus ermöglicht die Recherche einen Blick auf alternative Plattformen zur Bildverarbeitung, wie beispielsweise die weit verbreitete Software Insight Toolkit Snake Automatic Partitioning (ITK-SNAP). Ein kurzer Vergleich ergab jedoch, dass diese Lösung aufgrund ihrer Struktur in diesem speziellen Fall nicht mit 3D Slicer konkurrieren kann.

Die Recherche bietet somit einen ersten fundierten Überblick über mögliche Lösungen für die einzelnen Anforderungen. Um nun detaillierter auf die Umsetzung einzugehen, nimmt das nächste Kapitel eine Unterteilung der Gesamtheit der Anforderungen in kleinere Teilsysteme vor.

4.4. Zerlegung in Teilprobleme

Durch die Aufteilung des Gesamtsystems in mehrere kleine Teilaufgaben wird die Software für den Entwicklungsprozess übersichtlicher. Die einzelnen Domänen können so schneller und besser verstanden werden. Es gibt viele Möglichkeiten ein Softwaresystem in kleine Teile aufzuteilen, sodass es am Ende auf den konkreten Anwendungsfall ankommt. Diese Arbeit sieht folgenden Teilaufgaben für das Gesamtsystem vor:

Architektur- und UI-Design: Mithilfe von UML Diagrammen soll die Architektur dieses Systems abgebildet werden und sukzessive immer detaillierter beschrieben werden. Es soll dann verglichen werden, welche Entwurfsmuster für dieses System infrage kommen. Durch die Bearbeitung dieses Teilproblems kann die Anforderung an eine flexible Architektur erfüllt werden. Für einen UI entwurf bedient sich diese Arbeite der Wireframes

Kappselung anatomische Segmnetierung: Das bereits bestehende Segmentierungsverfahren muss in das Modul integriert werden. Hier soll das Verfahren von einem Python Notebook in eine Bibliothek überführt werden, sodass dieses Verfahren in der Extention ausführbar ist.

Parameter Node: Der Benutzer steuert das Verfahren über die Parameter in der UI. Für die Speicherung der Parametereinstellungen hat Slicer den Mechanismus ParameterNode entworfen. Diese wurde bereits in Abschnitt 2.5.4 erwähnt. Dieser Mechanismus ist nicht trivial, erhöht die Benutzerfreundlichkeit des Systems erheblich und soll demnach auch in diese Extention Anwendung finden.

Single Prozess: Sobald alle notwendigen Vorbereitungen getroffen sind, kann der Algorithmus nun eingebettet werden. Hierzu betrachtet man isoliert den Single Prozess. Auch die UI wird erst nur so weit entwickel, wie es für den einfachen Prozess nötig ist. Hierbei wird auf das erstellte Paket der anatomischen Segmenteirung zurückgegriffen.

Batch Prozess: Ist das einfache Verfahren fertig implementiert und funktioniert, so kann der Batch Prozess hinzukommen. Hier bedarf es einer zusätzlichen Arbeit in der UI, da der Benutzer über das Verwenden dieser Funktion gewarnt werden muss. Der Batch Prozess bedarf nämlich erheblicher Ressourcen. Hinzukommt die Implementierung einer Fortschrittsanzeige, sodass zu erkennen ist, dass ein Hintergrundprozess läuft.

Ausführungsmodus: Während der Ausführung des Algorithmus soll das Modul in einem Aufschriftrungszustand wechseln. Hierbei ist wichtig, dass die UI in dieser zeit gesperrt bleibt.

Dokumentation und Test: Abschließend ist eine ausführliche Dokumentation der Architektur erwünscht, sodass zukünftige Entwickler wissen, wo sie ansetzen müssen. Hinzu kommt ein Benutzerhandbuch für eine Verwendung der Erweiterung. Das Benutzerhandbuch und die Architekturdokumentation erfolgen in einer README.md

innerhalb der Extension. An letzter Stelle sollen noch Softwaretests implementiert werden, um die Richtigkeit der Extension sicherzustellen. 3D Slicer sieht hier Unitests vor

Die Ordnung dieser Punkte gibt eine grobe Orientierung bezüglich der Reihenfolge während der Umsetzung an. Damit eine Umsetzung überhaupt realisiert werden kann, sind unterschiedliche Werkzeuge und Mittel notwendig. Diese sollen im nächsten Kapitel kurz erläutert werden.

4.5. Entwicklungsumgebung

Die slicer library noch ERWÄHENEN Da bereits ein Framework feststeht, mit dem gearbeitet werden soll, ist keine weitere Forschung nötig, um die richtige Programmiersprache auszuwählen. Jedoch gibt es eine kleine Auswahl zu treffen. 3D Slicer unterscheidet zwischen zwei Arten von Modulen, die CLI-Module (Command Line Interface), welche in der Sprache C++ geschrieben werden und die Scripted Moduls, die eine Python Implementierung verlangen. Da die anatomische Segmentierung ohnehin in einem Python Notebook bereitliegt, fiel die Wahl hier auf die Scripted Moduls. So kann auch die breite Palette der Python Pakete genutzt werden. Für eine detaillierte Beschreibung des Frameworks selber sei an dieser Stelle auf das Kapitel 2.3f verwiesen, indem das Framework und alle zugehörigen Eigenheiten noch genauer beschrieben wurden. Um den Entwicklungsprozess etwas zu vereinfachen, wurde während der Entwicklung auf ein Modul von Slicer zurückgegriffen, das speziell für Entwickler entworfen wurde. Die Abbildung 4.2 verdeutlicht dieses Tool.



Abbildung 4.2.: Umgebung während der Entwicklung mit 3D Slicer und PyCharm

Mit den Debugging Tools lässt sich eine gewohnte Umgebung reproduzieren, in der der Quellcode Schritt für Schritt analysiert werden kann. Speziell bei der Fehlersuche ist dieses Tool eine sehr gute Unterstützung. Die Abbildung beschreibt weiter, dass als Umgebung für das Erstellen des Programmcode die Software Pycharm verwendet wird. Pycharm ist eine Lösung der Firma Jetbrains, für das Erstellen von Python Quellcode. Dieses Tool bietet eine breite Palette an Funktionalitäten, die das Erstellen von Software vereinfachen und kann als *State of the Art* bezeichnet werden.

Damit während der Entwicklung auch Teilbereiche der Software bereits getestet werden können, werden Testdaten benötigt. Bei diesen Testdaten handelt es sich um orginale Micro CT Aufnahmen von Zahnstrukturen. Diese wurde auf einem Server an der LMU in München bereitgestellt und konnten über den *x2goclient* heruntergeladen werden. Mit dem Zugriff aufen Galadrielrechhner in München konnten auch diverse Pythonumgebungen zum verarbeiten von Daten genutzt werden. Dies war in erster Line für ein Nachfolziehen der anatomischen Segmentierung hilfreich.

Neben der eigentlichen Umgebung und den Entwicklerwerkzeugen steht zur Entwicklung auch ein Python Paket zur Verfügung, das von Herrn Prof. Rösch speziell für die Klinik für Zahnerhaltung an der LMU in München erstellt wurde. Dieses Tool beinhaltet diverse Funktionalität für das Verarbeiten von medizinischen Bilddaten. Speziell für die Micro CT Aufnahmen der Klinik. Das Paket ist ebenfalls über den Server in München erreichbar.

Nachdem die Anforderungen, die Recherche, die konkreten Aufgaben und die verfügbaren Werkzeuge erläutert wurden, bleibt noch die Evaluation der Arbeit. Das Kapitel Forschungsevaluation erläutert die Methodik, mit dem das Erreichen des Forschungsziels messbar gemacht werden kann.

4.6. Forschungsevaluation

Die Evaluation kann grob in zwei Teile unterteilt werden. Der erste Teil ist der wohl wichtigste und beschäftigt sich mit dem Testen der Anwendung durch die Benutzer. Hier kann also die Benutzerfreundlichkeit und die UI der Erweiterung gut analysiert werden. Dabei werden Verbesserungen gesammelt und als möglicher Ausblick zur Verfügung gestellt. Wichtig für die Benutzbarkeit der Software ist auch das Benutzerhandbuch. Dies ist auch Teil der Ergebnisse und muss mittels Benutzertest evaluiert werden.

Der zweite Teil der Evaluation soll prüfen ob der Softwaretechnische Ansatz erfolgreich umgesetzt wurde. Um dies gewährleisten zu können, müssen zusätzlich zur Funktionalität auch Softwaretests bereitgestellt werden. Wie eine der Teilaufgaben aus Kapitel 4.4 bereits zeigt, handelt es sich hierbei um Unitests. Außerdem ist für diesen Teil auch die technische Dokumentation notwendig. Abschließend soll die Performance des Systems noch gemessen und analysiert werden.

Die in diesem Kapitel beschriebenen methodischen Schritte bildeten die Grundlage für die Entwicklung der Erweiterung. Nach der Konzeption und Umsetzung folgt nun die Präsentation der erzielten Ergebnisse. Im nächsten Kapitel wird das Plugin detailliert vorgestellt, seine Funktionen erläutert und anhand von Anwendungsszenarien sowie Tests bewertet.

5. Ergebnisse

Dieses Kapitel präsentiert alle Ergebnisse, die in dieser Arbeit erzielt wurde. Dabei spielen nicht nur die erfolgreichen Ziele eine Rolle, sondern auch die Misserfolge. Zunächst wird die entwickelte Erweiterung in seiner finalen Form beschrieben, gefolgt von einer Darstellung der zentralen Funktionen. Anschließend wird auf die Konzeptionen und Umsetzungen der verschiedenen Teile eingegangen. Abschließend soll die Performance und die verschiedenen Anwendungsszenarien genauer analysiert werden. Daraus ergeben sich auch Limitierungen für die Software. Mit diesen erstellten Analysen kann unter Berücksichtigung eine Aussage bezogen auf die Forschungsfrage gestellt werden.

5.1. Tooth Analyser

Im Rahmen dieser vorliegenden Arbeit ist eine 3D Slicer Extension entstanden, die den Namen Tooth Analyser trägt und für die Forschung im Dentalbereich eingesetzt wird. In erster Linie können mit diesem Plugin Micro CT Aufnahmen anatomisch segmentiert werden. Das Modul schmiegt sich wie alle anderen Module gut in die Kernanwendung ein und ist auch für die aktuelle stabile Version von Slicer (v5.8.0) verfügbar. Neben der eigentlichen Implementierung ist auch ein Logo für das Plugin entstanden, das es nach außen repräsentiert. Die Abbildung 5.1 zeigt dies.



Abbildung 5.1.: Logo der 3D Slicer Erweiterung "Tooth Analyser", welche im Rahmen dieser Arbeit entwickelt wurde. Logodesign: Dr. Elias Walter

Des Emblem des Tooth Analyser bildet einen Zahn ab, dessen Hauptsegmenten (Schmelz, Dentin, Pula) mit den unterschiedlichen Farben (grün, gelb, orange) visualisiert werden. Dies verdeutlicht die Analogie zur anatomischen Segmentierung und lässt gleich vermuten, dass sich dieses Modul mit einer Segmentierung beschäftigt. Der Untertitel deutet konkret auf Micro CT Aufnahmen hin. Wurde der Tooth Analyser installiert, so ist er über den Menüpunkt Module in Slicer auswählbar. Hier wird er in dem Unterpunkt *Segmentierung* eingruppiert, was ein weiteres Indiz auf die grobe Funktionalität liefert. Wird also der Tooth Analyser gestartet so erhält man die Ansicht der Kernanwendung mit der entsprechenden UI. Die Abbildung 5.2 soll genau diese Ansicht verdeutlichen

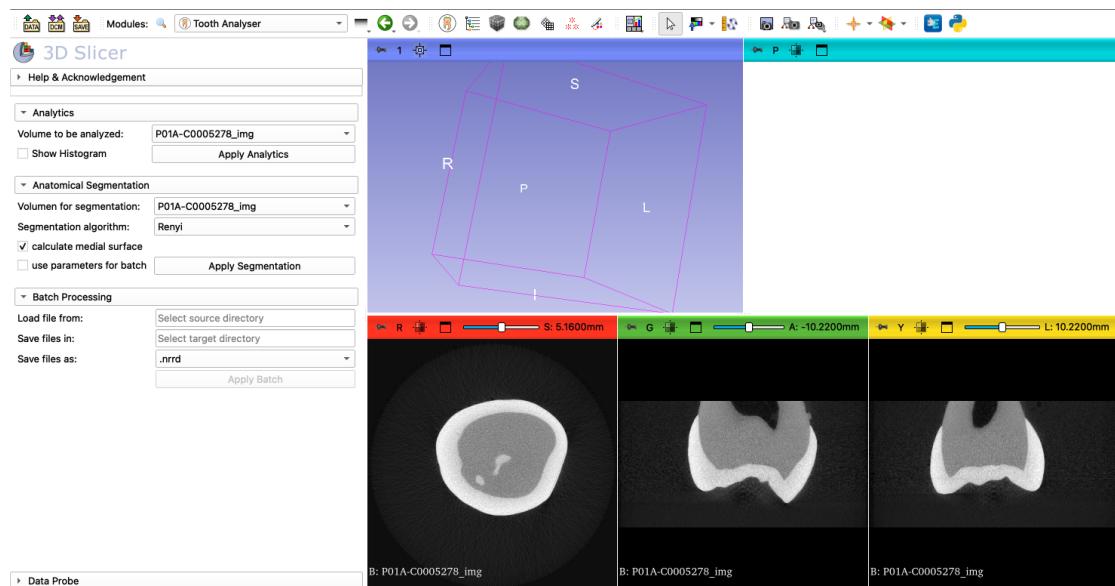


Abbildung 5.2.: Startansicht der Erweiterung Tooth Analyser nach dem ersten Aufruf

Die Ansicht zeigt die Kernanwendung (rechts die verschiedenen Fenster) und die UI des jeweiligen Moduls. Die Kernanwendung kann auch als Szene beschrieben werden und übernimmt alle generischen Handhabungen der Bilder. Neben den Szenen ist auch immer eine Sidebar zu sehen, welche die UI des jeweiligen Moduls abbildet. Im Falle der Abbildung 5.2 ist es die UI des Tooth Analyser. Das manuelle Laden eines Bildes in die Szene ist Teil der Slicer Kernanwendung und nicht teil der Modullogik. Das bereits geladene Bild ist demnach unabhängig von der Slicer Erweiterung entstanden. Betrachtet man die Benutzerschnittstelle genauer, so fällt sofort auf, dass diese in vier Bereiche unterteilt ist. Diese Aufteilung in Bereiche ist ein Ergebnis der Literaturrecherche und eine gute Konvention in der Welt von 3D Slicer. Der Bereich *Help and Acknowledgement* stellt Hilfen und Informationen über das Modul bereit. Über diesen Abschnitt ist auch die offizielle Dokumentation über dieses Modul erreichbar. Zu Beachten ist, dass dieser Bereich nicht eigens für den Tooth Analyser entwickelt wurde. Es handelt sich hier um

eine Funktionalität, die automatisch allen SEM zur Verfügung steht. Bei den übrigen Abschnitten handelt es sich im Features die spezifische für den Tooth Analyser entwickelt wurden. Bevor genauer auf die Funktionalitäten des Tooth Analyser eingegangen wird, sei zunächst auf die Abbildung 5.3 verwiesen, welche die Ergebnisansicht zeigt.

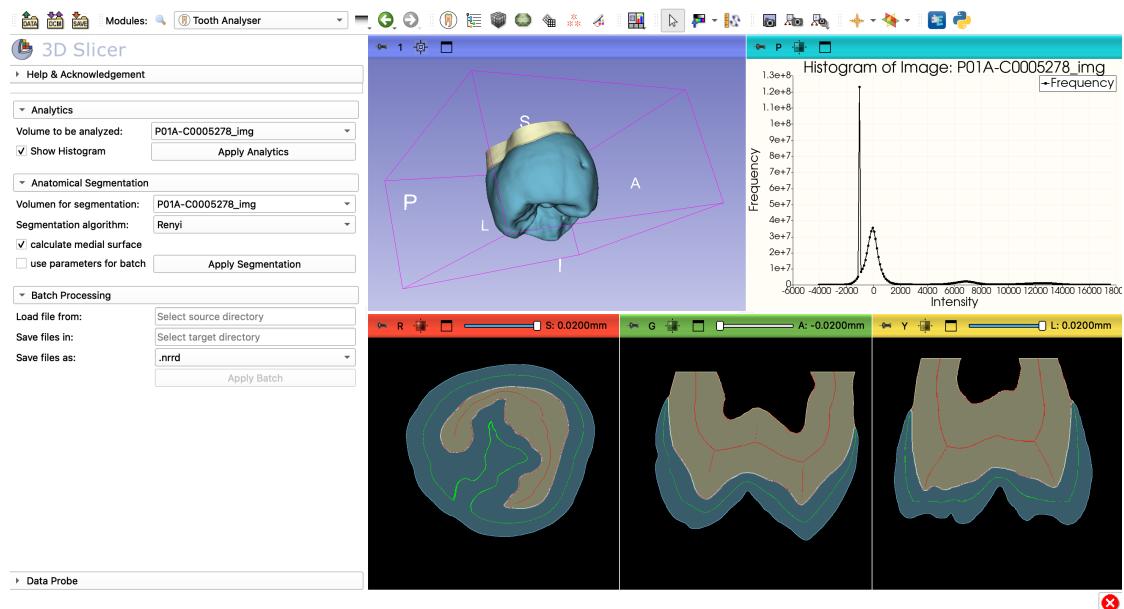


Abbildung 5.3.: Ergebnisansicht der Erweiterung Tooth Analyser, nachdem die Analysen und die anatomische Segmentierung erstellt wurden.

Der Analysebereich des Tooth Analyser ermöglicht es das Histogramm eines gegebenen Bildes zu erstellen. Dies ist besonders interessant, wenn ein Algorithmus für die anatomische Segmentierung ausgewählt werden muss. Diese Algorithmen sind Schwellwertverfahren, die auf das Histogramm eines Bildes basieren, um es zu segmentieren. Das erstellte Histogramm ist rechts oben in der Abbildung 5.3 zu erkennen. Es wird in einem Plot-Node dargestellt und kann über diesen auch verändert werden. Hierzu ist die Pinnadel im Fenster des Plot-Node zu wählen. Durch die Speicherfunktion der Kernanwendung kann der Plot auch problemlos gespeichert werden. Bevor die Analysen erstellt werden können, müssen Parametereinstellungen gewählt werden. Hierbei ist der wichtigste Parameter der, indem das konkrete Bild ausgewählt wird. Bei diesem Parameter handelt es sich um ein Dropdown, indem nur Bilder mit dem Typ `vtkMRMLScalarVolumeNode` ausgewählt werden können. Dies trägt zur Stabilität und Ausfallsicherheit des Systems bei und sorgt dafür, dass nicht jedes beliebige Bild geladen werden kann. Ist keine CT Aufnahme ausgewählt, so bleibt der Button zum Starten der Analysen deaktiviert. Wird ein Bild in die Szene geladen, während der Parameter für das zu analysierende Bild leer ist, wählt der Tooth Analyser automatisch das Bild aus, dass als Erstes in die Szene geladen wurde. So spart der Benutzer einige Klicks. Durch die Checkbox `Show Histogram`

wird der Erweiterung signalisiert das beim Starten der Analysen ein Histogramm des übergebenen Bildes erstellt werden soll.

Die Hauptfunktionalität des Tooth Analyser ist die anatomische Segmentierung welche in Kapitel 2.4 detailliert erläutert wurde. Die konkreten Ergebnisse dieser Segmentierung sind in der Abbildung 5.3 in den Fenstern (blau, rot, grün, gelb) zu sehen. Neben der eigentlichen Segmentierung sind auch hier die medialen Flächen für die Segmente Dentin (rot) und Schmelz (grün) gut sichtbar. Hinzu kommt ein 3D Modell, das auf Basis der erstellten Segmentierung generiert wurde und nur der Visualisierung dient. Ein Abspeichern dieses 3D Modells als Netz ist nicht möglich. Um überhaupt eine anatomische Segmentierung eines Zahnes erstellen zu können, sieht der Algorithmus zunächst drei Parameter vor, die eingestellt werden müssen. Um die Komplexität gering zu halten, wurde bewusst auf viele Parameter verzichtet. Ähnlich wie bei den Analysen ist auch hier die Wahl des zu segmentierenden Volumens der entscheidende Parameter. Dessen Bedeutung gleicht der Analysen, insbesondere in Bezug auf das Verhalten. Damit schnell ein Ergebnis generiert werden kann, wurde für die übrigen zwei Parameter eine Vorauswahl definiert, die zum vollen Ergebnisumfang des Tools führt. So ergibt sich die Situation, dass nach dem Laden eines CTs in die Szene nur auf den Button für das Ausführen gedrückt werden muss, damit eine anatomische Segmentierung erstellt wird. Dies nimmt dem Benutzer viel Arbeit ab und sorgt für eine gute User Experience (UX). Sind jedoch Einstellungen in den Parametern gewünscht, so können diese natürlich getätigt werden. Über den Parameter *Segmentation algorithm* kann das entsprechenden Schwellwertverfahren gewählt werden, mit dem der Zahn segmentiert werden soll. Dies mag nur geringfügig eine Änderung auf die Ergebnismenge ausmachen, kann aber dennoch wichtig sein. Die Checkbox *calculate medial surface* ermöglicht eine optionale Erstellung der medialen Flächen. Wird diese Funktion ohnehin nicht gebraucht, so kann diese hier ausgelassen werden und damit Laufzeit eingespart werden.

Die letzte Funktionalität, die geboten wird, stellt kein neues Verfahren dar, sondern nur eine andere Art der Ausführung. Die Rede ist hier von einem Batch Modus, der nicht nur ein Bild segmentiert, sondern das Verfahren der anatomischen Segmentierung auf eine ganze Reihe an Bildern anwendet. Um diesen Modus aktiv zu schalten, muss neben den Parametern im Abschnitt *batch* auch die Checkbox *use parameters for batch* aktiviert werden. Der Tooth Analyser überträgt dann die aktuellen Parametereinstellungen der anatomischen Segmentierung an den Batch Modus. Um dann den Batch Modus ausführen zu können, müssen noch zwei Pfade angegeben werden, die jeweils zu einem Ordner führen. Diese beiden Pfade teilen sich auf in *source* und *target* und geben an, wo die Daten liegen, die segmentiert werden soll und wo die segmentierten Daten auf der Festplatte gespeichert werden. Abschließend ist hier noch zu wählen in welchem Format die erstellten Daten abgespeichert werden sollen. Bei diesem Feature ist zu beachten, dass nach Erfolgreichem ausführen keine Daten in der Slicer Szene geladen werden.

Der Prozess läuft im Hintergrund und ist bis auf die Parametereinstellung über die UI komplett getrennt von Slicer. Nach Ende des Batch Modus wird in dem angegebenen Zielordner ein Unterordner erstellt, der alle Dateien der segmentierten Bilder enthält. Hierfür sieht der Tooth Analyser weitere Unterordner für jedes segmentierte Bild vor. Ein automatisches Laden aller segmentierten Bilder nach dem Batch Prozess ist nicht implementiert, sodass der erstellte Ordner mit allen Segmentierungsdaten manuell über den Import geladen werden muss.

Wird nun eines der bereitgestellten Verfahren ausgeführt, so wechselt der Tooth Analyser in einen sogenannten *Processing Mode*. Dieser soll in erster Linie dem Benutzer signalisieren, dass aktuell ein Prozess ausgeführt wird. Hierzu sei auf die Abbildung 5.4 verwiesen.

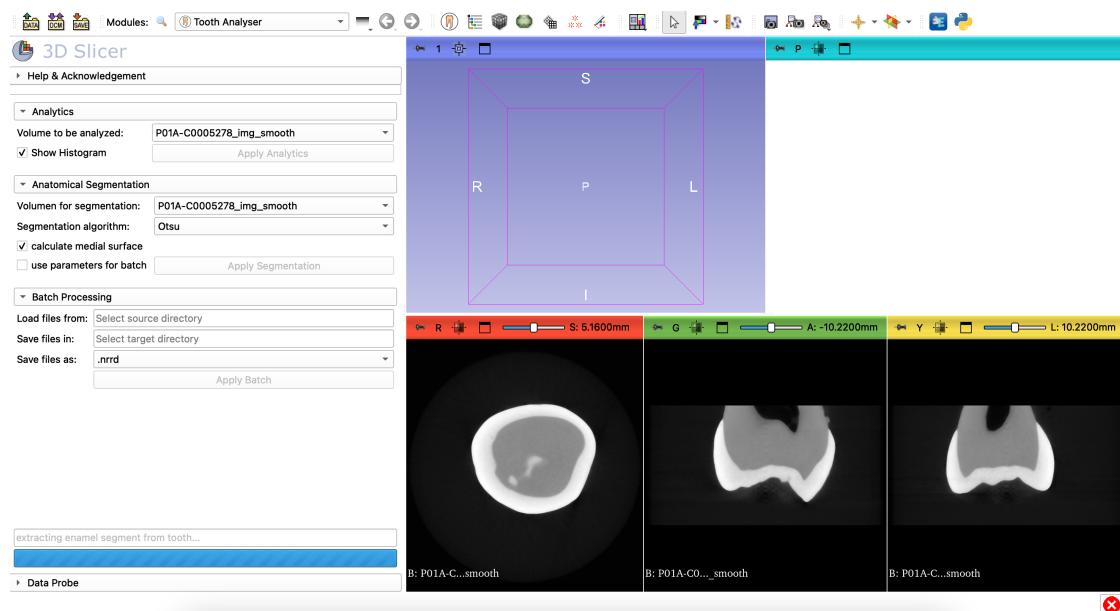


Abbildung 5.4.: Ansicht des Moduls Tooth Analyser während der Ausführung eines Verfahrens

Mit dem Start eines Verfahrens wechselt der Tooth Analyser automatisch in einen Prozess Modus. Dieser deaktiviert die Buttons zum Ausführen eines Algorithmus. Des Weiteren wird am unteren Rand des Moduls eine Fortschrittsanzeige mit Statusleiste angezeigt, um dem Benutzer mitzuteilen, in welchem Schritt er sich befindet. Der Cursor ändert innerhalb der Anwendung dann den Modus auf warten. Hierbei ist zu beachten, dass Slicer selber während der Ausführung nicht bedient werden kann, da die Ressourcen des Prozesses für den laufenden Algorithmus verwendet werden. Die Dokumentation von Slicer empfiehlt hier auch kein parallelisieren der Anwendung, da dies zu GUI Fehlern führen würde. Der Benutzer muss also die Zeit schlichtweg abwarten.

Da neben der reinen Erstellung noch weitere Anforderungen an die Erweiterung gegeben waren, beschäftigt sich die nächsten Kapitel tiefer mit den softwaretechnischen Aspekten des Tooth Analyser. Hierzu sollen auch die einzelnen Teilaufgaben, die zu Beginn definiert wurden, wieder aufgegriffen werden.

5.2. Tooth Analyser Bibliothek

Eine dieser Teilaufgaben beschäftigte sich mit der Integration der anatomischen Segmentierung welches prototypisch als Python Notebook bereitgestellt wurde. Dieses Notebook funktioniert, war aber nicht gut strukturiert. So viel es schwer den Überblick darin zu bewahren und das Verfahren gut nachzuvollziehen. Die gute Quelltextdokumentation kompensiert dies etwas. Für das von Herrn Hofmann entwickelte Verfahren wurde demnach ein eigenes Python-Modul erstellt. Somit konnte der Algorithmus von Aktivitäten in Slicer gut getrennt werden. Das Verfahren selber wurde gut strukturiert in Funktionen gekapselt, sodass schnell erkannt werden kann, in welchem Kontext gerade gearbeitet wird. Die einfache Quelltextkommentierung wurde als Dokumentationsblock an die jeweilige Funktion gehängt. Um die Einordnung des Moduls für die anatomische Segmentierung besser einordnen zu können, sei die Projektstruktur hier gezeigt.

```

1 | -- ToothAnalyser
2 | | -- CMakeLists.txt
3 | | -- Testing
4 | | -- Resources
5 | | -- ToothAnalyser.py
6 | | -- ToothAnalyserLib
7 | | | -- NewFunction
8 | | | | -- AnatomicclasSegmentation
9 | | | | | -- __init__.py
10 | | | | | -- Segmentation.py
11 | | | | | -- isq_to_mhd.py

```

5.1.Listing.: Projektstruktur des Moduls Tooth Analyser mit Fokus auf die ToothAnalyserLib

Zu sehen ist der Ordner `ToothAnalyser`, der alle Datei beinhaltet, die für das Modul relevant sind. Die Datei `ToothAnalyser.py` ist das Hauptskript und stellt die hier die Anbindung an Slicer dar. Außerdem ist Ordner `ToothAnalyserLib` zu finden, der den ganzen externen Quellcode für das Modul beinhaltet. So auch die anatomische Segmentierung. Kommen in Zukunft weitere Funktionen dazu, kann diese Bibliothek problemlos um neue Funktionen erweitert werden. Dies symbolisiert der Ordner `NewFunction`.

Soll eine Funktion aus dieser Bibliothek verwendet werden, so kann dies im Modul `ToothAnalyser.py` einfach über den folgenden Befehl erfolgen.

```
1 from ToothAnalyserLib.AnatomicalSegmentation.Segmentation
2     import (loadImage, isSmoothed)
```

5.2.Listing.: Importieren von Funktionen aus der Bibliothek des Tooth Analyser

Betrachtet man das Pythonpaket der anatomischen Segmentierung genauer, so fällt auf, dass es sich in zwei Module teilt. Im Modul `Segmentation.py` befinden sich alle Funktionen, die zum Ausführen der Pipeline für die anatomische Segmentierung notwendig sind. Darüber hinaus finden auch einige Hilfsfunktionen Platz. Das Modul `isq_to_mhd.py` liefert das Skript, mit dem Dateien im Format ISQ in eine mhd Datei umgewandelt werden können. Für die Anwendung im Tooth Analyser wurde diese Methode leicht modifiziert.

```
1 def isq_to_mhd_as_string(isq_file_name) -> str:
2     mhd_param, offset, grey_range = _read_isq_param(
3         isq_file_name)
4     mhd_param['ElementDataFile'] = isq_file_name
5
6     # Use a StringIO buffer to construct the MHD content
7     mhd_buffer = StringIO()
8     for key, value in mhd_param.items():
9         mhd_buffer.write(f'{key}={value}\n')
9     return mhd_buffer.getvalue()
```

5.3.Listing.: Modifizierte Methode zum erstellen einer mhd Datei aus einem ISQ Format

Der Quellcode aus 5.3 zeigt, dass statt einer Abspeicherung auf der Festplatte ein Pufferspeicher genutzt wird (Zeile acht). Anstatt dann auf die abgespeicherte Datei zu verweisen wird einfach der Pufferspeicher ausgelesen.

Nachdem zu Beginn ein Gesamtüberblick über die Ergebnisse gewonnen wurde und damit das Modul Tooth Analyser eingeführt wurde, folgen nun die Lösungen der restlichen Teilaufgaben in Form von Implementierungsdetails.

5.3. Konzeptionen

Die ausgearbeiteten Konzeptionen bilden überwiegend die Ergebnisse der in 4.4 beschriebenen Teilaufgaben. Konkret soll das bedeuten, dass hier konzeptionell gezeigt wird, wie die softwaretechnischen Aspekte aus den Anforderungen umgesetzt wurden. Hierzu soll zunächst das Design-Klassendiagramm betrachtet werden, das sich aus dem Domänenmodell in Abbildung 4.1 ableiten lässt.

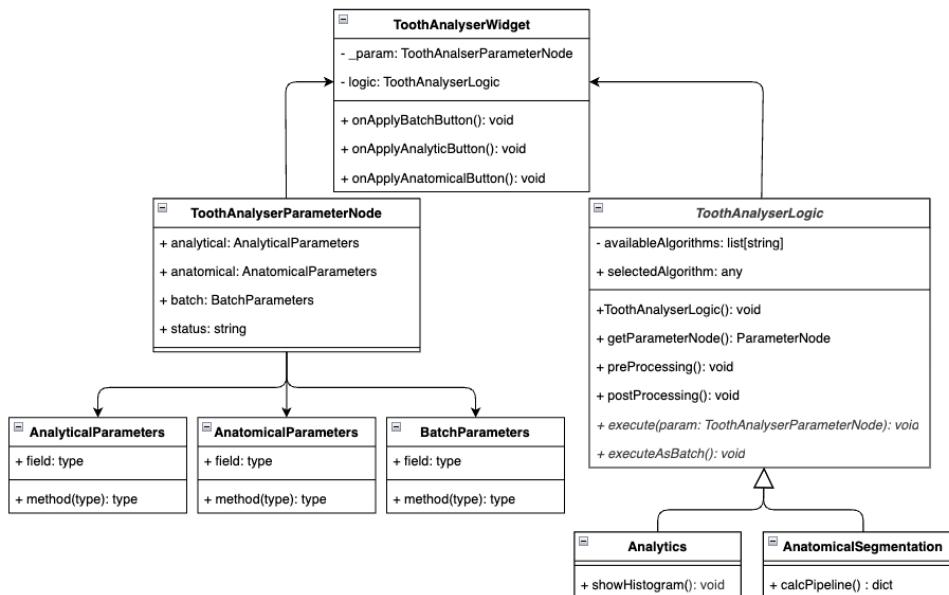


Abbildung 5.5.: Ausschnitt aus dem Klassendiagramm für den Tooth Analyser

Wie das Diagramm zeigt, ist die Anwendung über die Klasse `ToothAnalyserWidget` in das Kernsystem eingebunden. Sie hält auf der einen Seite alle Parameter der verschiedenen Funktionsbereich und auf der anderen Seite die dazugehörigen Logiken. Des Weiteren bildet diese Klasse die gesamt UI des `Tooth Analyser` ab, was unter anderem das Laden der UI mit einschließt. Die UI des `Tooth Analyser` wurde mit der Software QT-Designer erstellt und dann mittels einer Methode in die Anwendung geladen.

Wie auch schon im Kapitel 5.1 beschrieben teilt sich die UI auch hier sichtbar in unterschiedliche Teile auf. Betrachtet man nun Abbildung 5.5 so fällt auf, dass Für jeden Funktionsbereich eine eigene Parameterklasse erstellt wurde, die dann wiederum in der Klasse `ParameterNode` zusammengefasst werden. Dies stellt eine gute Erweiterbarkeit der UI um zusätzliche Parameter sicher. Es wurde an dieser Stelle bewusst keine Generalisierung verwendet, da Slicer genau diesen konzeptionierten Mechanismus vorsieht. Auf der anderen Seite der Struktur finden sich die Logikklassen. Die verschiedenen

Logiken die aktuell und zukünftig den Tooth Analyser ausstatten, verwenden alle dieselbe Schnittstelle, die über die Klasse `ToothAnalyserLogic` bereitgestellt wird. Soll also in Zukunft weitere Funktionen hinzukommen, so kann hier einfach eine weitere Klasse an die Schnittstelle angehängt werden. So entsteht etwas was man in der Fachliteratur als *Strategy Pattern* bezeichnet (vgl. Siebler 2014, S. 99). Der Benutzer der Software wählt also über die verschiedenen Buttons aus, welche Strategie er gerade nutzen will. Der Tooth Analyser sorgt dann dafür, dass auch die richtige Funktion geladen wird. Das Feature für den Batch Modus funktioniert nach einem anderen Schema. Da dieser Modus auch für alle zukünftigen Funktionen gelten soll, wird dieser nicht als eigene Strategie, sondern direkt in der Schnittstelle bereitgestellt. So ist sichergestellt, dass eine Implementierung erfolgen muss, sie jedoch für alle Strategien unterschiedlich sein kann. Um noch etwas genauer zu zeigen, welche Schritte notwendig sind um den Tooth Analyser mit weiteren Funktionen auszustatten sei auf Abbildung 5.6 verwiesen. Hier wird deutlich, welche Klassen und Methoden erstellt werden müssen.

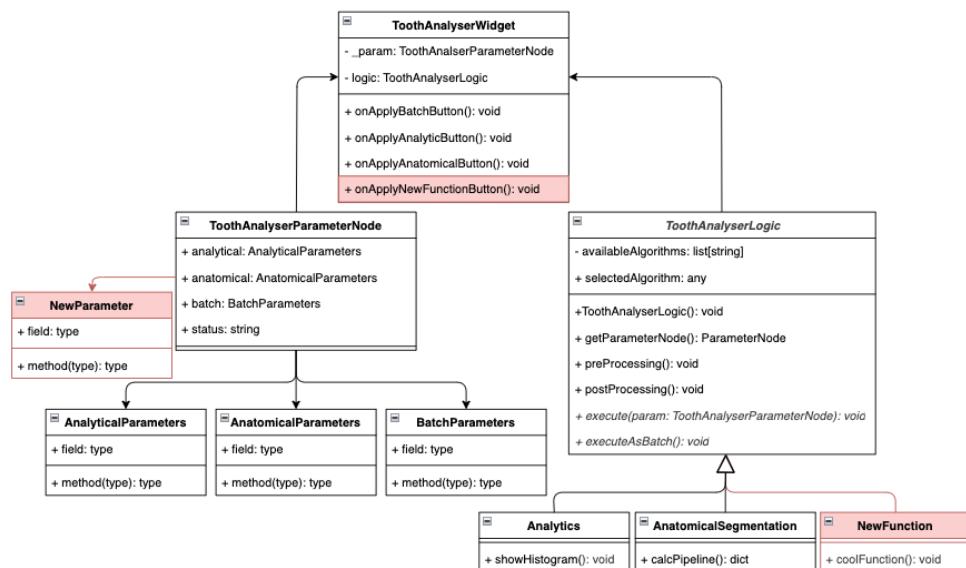


Abbildung 5.6.: Hinzufügen neuer Funktionen zum Tooth Analyser

Wie bereits zu erkennen ist, signalisieren die roten Elemente die Punkte, an denen die Funktionalität erweitert werden kann. Die gestrichelte, rote Linie verdeutlicht nur den Zusammenhang dieser Komponenten und hat keine Auswirkung auf die Konzeption. Um einen noch genaueren Einblick in die Strukturen zu geben, geht das nachfolgende Kapitel noch einen Schritt weiter und geht auf konkrete Implementierungsdetails ein.

5.4. Technische Umsetzung

Zu Beginn sei gesagt, dass dieses Kapitel nicht alle Funktionen detailliert beschreibt, sondern sich auf die wichtigsten Funktionen und Methode des Tooth Analyser beschränkt. Auf Basis des Klassendiagramms aus Abbildung 5.5 lässt sich eine grobe Übersicht über die wichtigsten Klassen gewinnen. Tabelle 5.1 liefert diese Übersicht. Die Reihenfolge gibt eine grobe Orientierung bezüglich der Wichtigkeit.

| Klassen | Beschreibung |
|---------------------|---|
| ToothAnalyser | Klasse für den Abschnitt Hilfe |
| ToothAnalyserWidget | Die UI-Klasse mit Anbindung an das Kernsystem |
| ToothAnalyserLogic | Die Logik-Schnittstelle |

Tabelle 5.1.: Wichtige Klassen und Methoden im Tooth Analyser

Die drei Klassen aus der eben gezeigten Tabelle werden von der Slicer Dokumentation grob vorgeschrieben, und bilden somit den Kern der Erweiterung. Die Klasse ToothAnalyser bildet hierbei den Abschnitt *Help and Acknowledgment*, den jedes Modul mitbringen muss. Dort stehen alle wichtigen Meta-Informationen über das konkrete Modul. Der nachfolgende Ausschnitt eines Quelltextes zeigt grob den Aufbau dieser Klasse.

```

1 class ToothAnalyser(ScriptedLoadableModule):
2     def __init__(self, parent):
3         ScriptedLoadableModule.__init__(self, parent)
4         self.parent.title = _("ExtensionName")
5         self.parent.categories = ["categories"]
6         self.parent.dependencies = ["moduels"]
7         self.parent.contributors = ["author"]
8         self.parent.helpText = _("help")
9         self.parent.acknowledgementText = _("ackn.")

```

5.4.Listing.: Grober Aufbau der Klasse ToothAnalyser nach der Slicer Dokumentation

Direkt in Zeile eins ist zu sehen, dass die Klasse eine Generalisierung der Klasse `ScriptedLoadableModule` ist. Über diese Elternklasse wird die Integration in die Slicer Kernanwendung sichergestellt. Innerhalb des Konstruktors der Klasse werden unterschiedliche Felder angelegt, welche die verschiedenen Texte widerspiegeln sollen. Zu Beachten ist noch, dass die Felder die einen einfachen String erwarten, in einer unscheinbaren Methode `_()` gekapselt sind. Diese Methode ist Teil des Slicer Python Framework und sorgt dafür, dass automatisch der Kontextname gewechselt wird. Über bekannte Html-Tags können auch Bilder oder Hyperlinks eingebaut werden, welche auf Screenshots oder Dokumentationen verweisen.

Die Klasse `ToothAnalyserWidget` bildet die gesamte Benutzerschnittstelle der Erweiterung ab und kümmert sich gleichzeitig um das Zusammenspiel zwischen Logik und Parameter. Hierfür hat die Widget-Klasse Zugriff auf alle Parameter und auf die Logik-Schnittstelle. Das nachfolgende Listing zeigt einen groben Aufbau der Klasse `ToothAnalyserWidget` und deren Felder.

```

1 class ToothAnalyserWidget(ScriptedLoadableModuleWidget):
2     def __init__(self, parent=None) -> None:
3         ScriptedLoadableModuleWidget.__init__(self, parent)
4         self.logic = None
5         self._param = None

```

5.5.Listing.: Grober Aufbau der Widget-Klasse und deren Felder

Wie auch schon die Klasse `ToothAnalyser` ist auch die Widget-Klasse eine Kindklasse. Der Konstruktor dieser Klasse wird jedes Mal aufgerufen, wenn der Benutzer zum ersten Mal nach dem Start von Slicer das Modul aufruft. Neben dem Konstruktoraufgriff der Elternklasse, werden auch die Parameter und die Logik-Schnittstelle initialisiert. So ergibt sich die Situation, dass die UI die aktuellen Parametereinstellungen abfragt und sie an die Logik weiterleitet.

Neben der Steuerungsaufgabe erstellt die Widget-Klasse auch die UI aus der .ui Datei. Da diese unter der Haube ein XML Format aufweist, kann diese einfach in das Modul hineingeladen werden. Der Quellcode aus ... zeigt dies.

```

1 def createUI(self) -> any:
2     uiWidget = slicer.util.loadUI(self.resourcePath("UI/W.ui"))
3     self.layout.addWidget(uiWidget)
4     self.ui = slicer.util.childWidgetVariables(uiWidget)
5     return uiWidget

```

5.6.Listing.: Laden der UI in das Modul ToothAnalyser

Mittels der Bibliothek `slicer` lässt sich hier die Datei für die UI laden. Diese Datei liegt im Ordner *Resources* sodass der Pfad zu diesem Ordner mit der Methode `self.resourcePath()` ausgelesen werden kann. Als Argument kann dann der Name der Datei angegeben werden. In diesem Fall existiert noch ein Unterordner mit der entsprechenden Datei (`UI/W.ui`).

Ein weiterer nennenswerter Punkt in der Klasse `ToothAnalyserWidget` sind die event-basierten Methoden. Diese sorgen dafür, dass auf bestimmte Aktionen in der Anwendung reagiert werden kann. Der `Tooth Analyser` hat in der Widget-Klasse mehrere solcher Event-Methoden, die hier genannt werden.

`enter()` wird immer ausgeführt, wenn der Benutzer das Modul öffnet
`exit()` wird immer ausgeführt, wenn der Benutzer ein anderes Modul öffnet
`onSceneStartClose()` wird immer ausgeführt, kurz bevor das Modul geschlossen wird
`onSceneEndClose()` wird ausgeführt, nachdem das Modul geschlossen wurde
`observeParameter()` wird ausgeführt, wenn der Benutzer Änderungen an der UI macht

Die Methode `observeParameter()` ist hierbei sehr interessant und verdient besondere Beachtung. Die Methode ist eine Beobachtungsmethode, die an ein Event gekoppelt ist, dass sich *ModifiedEvent* nennt. Dieses Event wird immer ausgelöst, wenn der Benutzer Änderungen in der Slicer UI macht. Dies umschließt nicht nur das Modul. Wird also solch ein Event gefeuert, so wird die Methode `observeParameter()` aufgerufen. Das Listing 5.12 zeigt diese Methode:

```

1 def observeParameters(self, caller=None, event=None) -> None:
2     self.handleApplyBatchButton()
3     self.handleApplyAnalyticsButton()
4     self.handleApplyAnatomicalButton()

```

5.7.Listing.: Methode zum Beobachten von Änderungen in der Benutzerschnittstelle

Die Methode verfügt über die Parameter `caller` und `event`. Mittels diesen Parametern kann der Auslöser des Events und die Art des Events bestimmt werden. So könnte man verschiedenen Aktionen auf verschiedene Events ausführen. Innerhalb des Tooth Analyser wird diese Methode genutzt, um die Klick-Logik der Buttons abzubilden. Die Methoden `handleApply<...>Button()` steuern jeweils die Sichtbarkeit der einzelnen Buttons. Immer wenn der Benutzer dann Änderungen an der UI durchführt, wird die Logik innerhalb dieser Methode aus Listing 5.12 aktualisiert.

Die letzte der wichtigen Klassen bildet die Logik-Schnittstelle, wie sie in Tabelle 5.1 Zeile drei genannt wurde. Diese Schnittstelle ist der zentrale Punkt aller aktuellen und zukünftigen Logik-Klassen. Jede Klasse, die einen speziellen Algorithmus implementiert, muss dieses Interface implementieren. So wird sichergestellt, dass alle Algorithmen nach dem gleichen Aufbau eingebaut wurden. Dies sorgt für ein einheitliches Vorgehen. Das Listing 5.8 zeigt diese eben beschriebenen Schnittstelle und soll diese so genauere beleuchten.

```

1 class ToothAnalyserLogic(ScriptedLoadableModuleLogic):
2     def __init__(self) -> None:
3         ScriptedLoadableModuleLogic.__init__(self)
4
5     def getParameterNode(self) -> ToothAnalyserParameterNode:
6         return ToothAnalyserParameterNode(super().
7             getParameterNode())
8
9     def preProcessing(self) -> None:
10        """Implement pre processing here"""
11        pass
12
13    def postProcessing(self) -> None:
14        """Implement post processing here"""
15        pass
16
17    def execute(self, param: ToothAnalyserParameterNode) -> None:
18        """Abstract method"""
19        raise NotImplementedError("Please_implement_this")
20
21    def executeAsBatch(self, param: ToothAnalyserParameterNode)
22        -> None:
23        """Abstract method"""
24        raise NotImplementedError("Please_implement_this")

```

5.8.Listing.: Die Logik-Schnittstelle des Tooth Analyser

Auch die dritte der wichtigen Klassen erbt von einer übergeordneten Klasse und verfügt so über deren Funktionen. Die Schnittstelle verfügt über drei Methoden, die eine Standardimplementierung haben und direkt im Interface implementiert werden. Die Methode `getParameterNode()` sorgt dafür, dass alle Parameter des Moduls zur Verfügung stehen und verwendet werden können. `preProcessing()` und `postProcessing()` sind für eine Vor- und Nachverarbeitung des entsprechenden Bildes verantwortlich. Die Vorverarbeitung könnte beispielsweise eine Komprimierung des Bildes beinhalten, wohingegen die Nachbereitung Analysen auf dem Ergebnis anstoßen könnte. Im Rahmen dieser vorliegenden Arbeit wurden für die Methoden `preProcessing()` und `postProcessing()` keine Methodenrümpfe gebaut. Eine mögliche zukünftige Implementierung ist im Kapitel 7 zu finden. Die beiden noch übrigen Methoden `execute()` und `executeAsBatch()` sind abstrakte Methoden, die eine Implementierung in den jeweiligen Unterklassen erfordern. Dabei bilden beide Methoden zentrale Punkte. Innerhalb dieser Methoden soll der entsprechende Algorithmus der Klasse ausgeführt und gestartet werden. Einmal mit nur einem Bild und anschließender Visualisierung in der Slicer Szene und einmal als Batch ohne Visualisierung.

Um den Zusammenhang der Klassen im Tooth Analyser noch besser verstehen zu können sollen hier jeweils die nötigen Punkte aus den einzelnen Klassen herausgegriffen werden. Dies verdeutlicht das Zusammenspiel der Methoden um den zugrunde liegenden Algorithmus zu starten.

```

1 @parameterNodeWrapper
2 class ToothAnalyserParameterNode:
3     analytical: AnalyticalParameters
4     anatomical: AnatomicalParameters
5     batch: Batch
6     status: str = ""

```

5.9.Listing.: Die Parameter des Tooth Analyser, die als Attribut in der Widget-Klasse liegen.

```

1 def onApplyAnatomicalButton(self) -> None:
2     self.activateComputingMode(True)
3     with slicer.util.tryWithErrorDisplay(_("Failed")):
4         try:
5             AnatomicalSegmentationLogic.execute(self._param)
6         except:
7             slicer.util.errorDisplay(_("Error"))
8     self.activateComputingMode(False)

```

5.10.Listing.: Starten des Algorithmus durch den Aufruf der `execute()` Methode in der Widget-Klasse. Die Parameter werden mit übergeben.

```

1 @classmethod
2 def execute(cls, param: ToothAnalyserParameterNode) -> None:
3     toothDict = cls.calcPipeline(
4         sourcePath=sourcePath,
5         calcMidSurface=param.anatomical.calcMidSurface,
6         param=param,

```

5.11.Listing.: Ein Ausschnitt der Methode `execute()`, welche die Pipeline für das Verfahren startet und in der Widget-Klasse durch den Apply-Button aufgerufen wird.

Die Parameter, die im Parameter Knoten gespeichert werden liegen wie bereits beschrieben als Attribut und der Klasse `ToothAnalyserWidget`. Wenn ein Button zum Ausführen der entsprechenden Logik gedrückt wird, wird die Methode `execute()` ausgeführt und alle Parameter als Argument mit übergeben. So ist innerhalb der Ausführung ein voller Zugriff auf die Parameter möglich.

Nach der konkreten Umsetzung der Erweiterung kann Schritt für Schritt mit der Evaluation der Ergebnisse begonnen werden. Ein wichtiger Teil sind die Testergebnisse

5.5. Evaluierung

Neben der Implementierung der Features wurden auch diverse Tests durchgeführt, die eine Bewertung der Ergebnisse möglich machen. Hierzu wurden Softwaretests, Benutzertests und Messungen durchgeführt. Dieses Kapitel beschäftigt sich mit der Testauswertung der verschiedenen Testfälle. Begonnen wird mit einer Analyse der Softwaretests, gefolgt von Laufzeitmessungen. Abschließend werden die Anwendungsfälle des Tooth Analyser näher beleuchtet und auf die Limitierungen der Anwendung eingegangen.

5.5.1. Softwaretests

Betrachtet man die Dokumentation von Slicer genauer, so fällt auf, dass dies eine recht starre Struktur für die Implementierung von Tests vorgibt. Dabei ist jedoch nicht festgelegt, welche Art von Tests verwendet werden soll. Im Rahmen des Tooth Analyser wurden hier ausschließlich Unitest implementiert, welche die einzelnen Einheiten und Funktionen im Tooth Analyser abdecken. Der grobe Testaufbau sei hier gezeigt.

```

1 class ToothAnalyserTest(ScriptedLoadableModuleTest):
2     def setUp(self):
3         slicer.mrmlScene.Clear()
4         self.loadSampleData()
5
6     def runTest(self):
7         self.setUp()
8         self.testIsSmoothed()
9         # add more tests here...
10
11    def testIsSmoothed(self):
12        from ToothAnalyserLib.AnatomicalSegmentation.
13            Segmentation import isSmoothed
14        sampleDate = self.getSampleDataAsITK()
15        result = isSmoothed(sampleDate)
16        self.assertFalse(result)
17        self.delayDisplay("Test_1_passed")

```

5.12.Listing.: Ausschnitt der Testklasse zum ausführen der Unitests

Wie gleich zu erkennen ist, wurden alle Softwaretests in der Klasse `ToothAnalyserTest` gekapselt. Diese ist wie auch bei einigen anderen Klassen eine generalisierte Klasse der `ScriptedLoadableModuleTest`. Der grundsätzliche Aufbau der Testklasse ist simpel gehalten. Es gibt eine Methode `setup()` in der die Testumgebung bereitgestellt wird und eine Methode `runTest()` in der die einzelnen Testfälle ausgeführt werden.

Betrachtet man die konkrete Testmethode `testIsSmoothed()` genauer, so fällt die Methode `getSampleDataAsITK()` auf, die hier kurz thematisiert werden soll. Viele der geschriebenen Methoden und Funktionen benötigen für einen guten Test ein konkretes Bild. Hierfür stellt der Tooth Analyser Beispielbilder zur Verfügung, mit denen die Tests ausgeführt werden können. Da diese Bilder eine ausgeprägte Größe haben (ca. 500MB) wurden diese in einem separaten Repository bereitgestellt. So müssen nicht erst einige Gigabyte (GB) an Bildern heruntergeladen werden, wenn das Modul installiert werden soll. Die Bilder werden erst dann heruntergeladen, wenn sie benötigt werden. Um dieses Herunterladen zu ermöglichen, werden die Bilder beim Starten des Moduls erstmals in Slicer registriert, sodass sie dann im Modul `sampleData` zur Verfügung stehen. Damit ist nicht nur gewährleistet, dass zukünftige Entwickler Tests ausführen können, es kann so auch Benutzern Beispielbilder bereitgestellt werden, um erste Erfahrungen mit dem Tool zu machen. Die Abbildung 5.7 zeigt das Modul `SampleData` mit besonderem Augenmerk auf das Bild *ToothCT*.

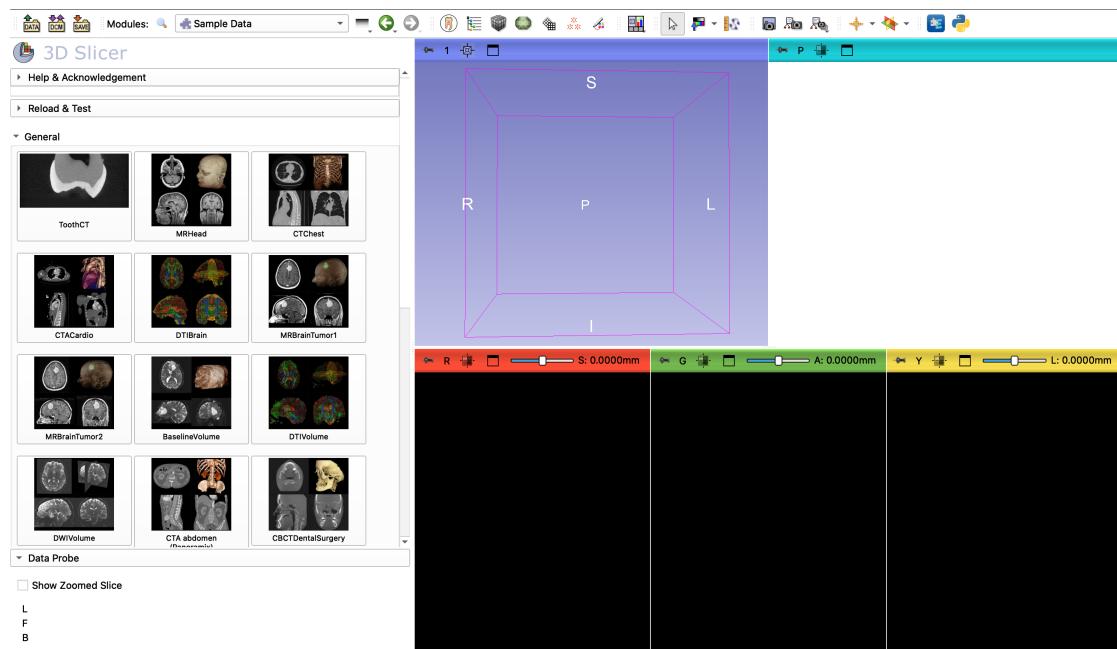


Abbildung 5.7.: Ausschnitt des Moduls SampleData in 3D Slicer

Ein Testfall der vielen soll hier Beispieldhaft genauer betrachtet werden. Hierbei geht es um den Test der Funktion `smoothImage()`. Diese nimmt ein Bild und führt eine Glättung durch. Um solch eine Funktion zu testen, bedarf es etwas mehr als ein simpler Unitest, jedoch liefert der fertige Test eine gute Lösung um den gesamten Umfang der Methode zu Testen. Vergleicht man ein verrausches Bild mit einem geglättetes, dann unterscheiden sie sich bis auf die visuelle Darstellung auch in der Streuung der Pixelwerte. So kann mittels der Standardabweichung kontrolliert werden, ob das Bild nach einer Filterung

eine kleinere Standardabweichung hat als vor dem Bild. Ist dies der Fall, so kann von einer erfolgreichen Filterung ausgegangen werden. Die konkrete Implementierung eines solchen Tests liefert der Quellcode 5.13.

```

1 def testSmoothImage(self):
2     from ToothAnalyserLib.AnatomicalSegmentation.Segmentation
3         import smoothImage
4
5     data = self.getSampleDataAsITK()
6     dataFiltered = smoothImage(data)
7     dataStdDev = np.std(sitk.GetArrayFromImage(data))
8     dataFilteredStdDev = np.std(sitk.GetArrayFromImage(
9         dataFiltered))
10    self.assertTrue(dataFilteredStdDev < dataStdDev)
11    self.delayDisplay("Test_1_passed")

```

5.13.Listing.: Implementierung eines Tests zum überprüfen einer Funktion

Im ersten Schritt wird ein Beispielbild geladen und in ein ITK Format umgewandelt. Anschließend folgt die Glättung des Bildes. Ist diese Glättung fertig, so können die Streuungen der beiden Bilder verglichen werden.

Betrachtet man alle Softwaretests, so lässt sich sagen, dass viele und die wichtigsten Methoden getestet wurde. Jedoch wurden nicht alle Methoden und Funktionen getestet. Viele bilden eine sehr konkrete Lösung, die nicht einfach zu testen ist und deshalb einiges an Entwicklungszeit beanspruchen. Eine ebenso gute Aussage lässt sich anhand der Laufzeit des Moduls treffen. Hierzu wurde die Performance des Tooth Analyser genauer unter die Lupe genommen.

5.5.2. Performance

Die Performance des Systems war nie ein wichtiges Kriterium und stand deshalb zu keiner Zeit im Fokus dieser Arbeit. Dennoch ergaben sich interessante Ergebnisse, die hier kurz erläutert werden sollen. Unter der Performance versteht dieses Kapitel das konkrete Laufzeitverhalten der Anwendung also jene Zeit die zwischen Start und Ende vergeht. Grundsätzlich lässt sich dazu sagen, dass die Laufzeit bei der Bearbeitung von 3D micro CT Aufnahmen sehr stark vom Type des Bildes abhängt. So kommt es beispielsweise darauf an, wie groß das Bild ist, oder ob es bereits eine Filterung erfahren hat. Bei der Verarbeitung der micro CT Bilder aus dem Klinikum für Zahnerhaltung wurde eine konkrete Messung durchgeführt, die hier in Abbildung 5.8 gezeigt wird.

Vorbedingungen:

- 16 bit sigend integer
- Type .ISQ
- Mit Filterung
- Mit Berechnung der medial Flächen

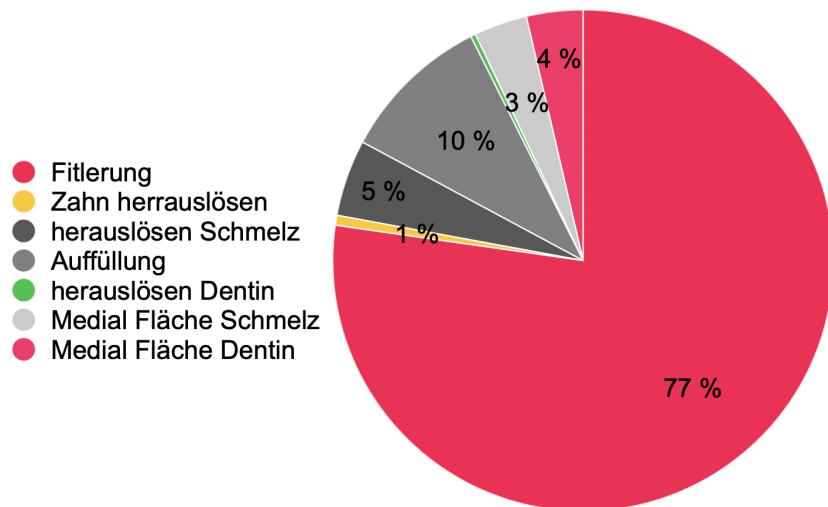


Abbildung 5.8.: Verteilung der Laufzeit über den gesamten Bearbeitungszeitraum. 100% entsprechen 16:27 Minuten

Zu sehen ist, dass unter diesen Bedingungen die Bearbeitung eines einzelnen Bildes ca. 17 Minuten beansprucht. Dabei fallen über drei Viertel der Zeit auf die Filterung zurück. Der Bereich Auffüllung beinhaltet ebenfalls eine Filterung der einzelnen Segmente, weswegen dieser den zweitgrößten Teil ausmacht. Ein weiterer wesentlicher Teil stellen die beiden medialen Flächen dar. Um dieser doch enormen Laufzeit etwas entgegen zu Wirken

wurden zwei Mechanismen implementiert. Das Verfahren kann einerseits erkennen, ob ein Bild bereits gefiltert wurde und andererseits die medialen Flächen optional berechnen. So kommt es das sich ein *Best Case* ergibt der grob nur noch ein Viertel der Zeit benötigt. Dieser *Best Case* tritt ein, wenn ein Bild in den Algorithmus gegeben wird, das bereits gefiltert wurde und keine medialen Flächen erfordert.

Überträgt man das Laufzeitverhalten eines einzelnen Bildes auf die Bearbeitung der Bilder in einem Batch Prozess so lässt sich die Laufzeit mit der Anzahl der zu bearbeitenden Bilder steigend linear ausdrücken. Das Diagramm aus 5.9 zeigt dies.

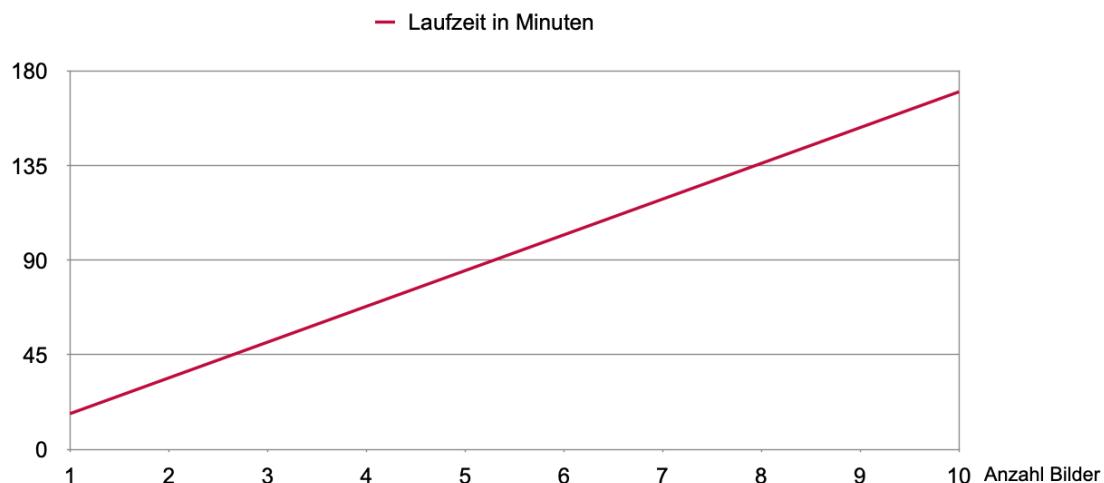


Abbildung 5.9.: Konstruktion des Laufzeitverhaltens bei einer Bearbeitung mehrere Bilder

Wie zu sehen ist, ist die Laufzeit eines Batch Prozesses maßgeblich davon geprägt, wie lange ein einzelnes Bild benötigt. Ist die Zeit eines einzelnen Bildes bekannt, so lässt sich gut prognostizieren, wie lange eine Verarbeitung von n vielen Bildern dauert. Wie bereits angedeutet, hängt die Zeit eines einzelnen Bilds von vielen Faktoren ab. In diesem Fall würde die Bearbeitung von zehn Bildern etwa 170 Minuten beanspruchen. Dies entspricht zwei Stunden und 50 Minuten.

Im Rahmen eines kleinen Stresstests wurden 103 CT Aufnahmen aus dem Tooth Analyser heraus in einem Batch Prozess gestartet. Hierzu wurde ein Server der Ludwig-Maximilians-Universität München (LMU) genutzt, der über große Ressourcen verfügt. Die Bearbeitung eines Bildes benötigt hier etwa neun Minuten. Befolgt man die Verteilung aus der Abbildung 5.9, so lässt sich eine Zeit von 15 Stunden und 27 Minuten prognostizieren. Das Resultat dieses Tests zeigt, das in etwa auch diese Zeit benötigt wurde. Viel wichtiger ist jedoch, dass hierbei alle Bilder erfolgreich anatomisch segmentiert werden konnten.

Nichtsdestotrotz bleibt die Anwendung trotz ihrer doch hohen Laufzeit ein enormes Zeitsparnis für die praktizierenden Ärzte. Laut Dr. Elias Walter würde eine händische Segmentierung ca. 20 Minuten aktive Arbeit bedeuten. Diese Arbeit kann der Tooth Analyser deutlich reduzieren, indem kein aktives Arbeiten erforderlich ist, sondern im Hintergrund läuft. Für welche Anwendungsfälle der Tooth Analyser genau eingesetzt werden kann, darüber berichtet das Kapitel Anwendungsszenarien.

5.5.3. Anwendungsszenarien

In erster Linie bietet der Tooth Analyser eine Visualisierungshilfe, die für Ärzte unterstützend wirken soll. Wie auch von Slicer empfohlen wird dieses Modul von den Ärzten rein zur Forschung eingesetzt. Mit dem Tooth Analyser lässt sich mittels einer micro CT-Aufnahme ein 3D Modell des Zahnes erstellen. Man spricht hier von einer Rekonstruktion des Zahnes. Durch die Segmentierung erlaubt dieser rekonstruierte Zahn auch eine Segmentbetrachtung von Dentin und Schmelz. Die Abbildungen 5.10 und 5.11 zeigen diese Rekonstruktion genauer.

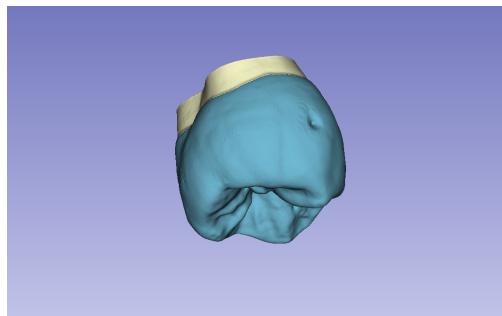


Abbildung 5.10.: Rekonstruktion eines Zahns aus einer CT-Aufnahme mittels der Erweiterung Tooth Analyser.

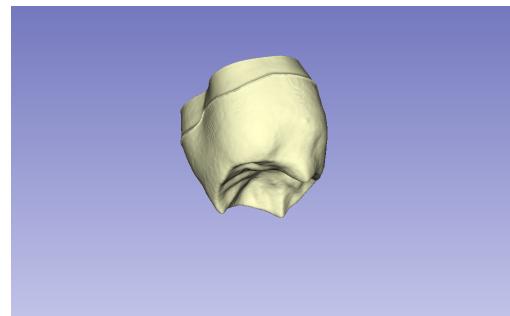


Abbildung 5.11.: Segmentbetrachtung eines rekonstruierten Zahns auf Basis einer CT-Aufnahme.

Das Betrachten der einzelnen Segmente wie sie in Abbildung 5.11 gezeigt wird, erfolgt nicht in der Erweiterung Tooth Analyser. Hierzu wird auf das Modul *Data* verwiesen, das eine hierarchische Darstellung aller Daten in der Szene liefert. Über die Sichtbarkeitseinstellungen der einzelnen Datenelemente können dann die Segmente sichtbar oder unsichtbar geschaltet werden. Einen weiteren Fall, indem die Anwendung unterstützen kann, ist die Klassifizierung von Karies. Die Abbildung 5.12 zeigt diesen Fall.

Hierfür sind die medialen Flächen der einzelnen Segmente nötig. Diese sind im Bild als rote und grüne Linie sichtbar, verteilen sich aber über das ganze 3D Bild, was daraus eine Fläche macht. Legt man nun diese Flächen über das originale Bild, so lässt sich mittels dieser Linie der Karies auf einem CT gut klassifizieren. Diese besagten Linien bilden dann die Grenzen. Ragt der Karies über diese mediale Fläche hinaus hat er bereits einen sehr ausgeprägten Zustand und wird anders eingeordnet als ein Karies, der die mediale Fläche noch nicht überschritten hat.

Da durch die Klinik für Zahnerhaltung nur CT-Bilder von Zahnkronen bereitgestellt wurden, wurde der Tooth Analyser auch nur mittels dieser Aufnahmen getestet. Durch einige Benutzertests in der Klinik zeigte sich aber, dass auch die Segmentierung eines ganzen Zahns möglich ist. Selbst bei einer komplexeren Wurzel. Jedoch bleibt es nicht aus, dass der Tooth Analyser auch Einschränkungen mitbringt und so auch die Verwendung dieses Moduls limitiert.

5.5.4. Limitierungen

Dieses Kapitel soll alle Punkte transparent aufdecken, die im Modul Tooth Analyser noch Probleme machen, oder gar nicht erst umgesetzt wurden. Der limitierende Faktor in der Erweiterung ist das eingeschränkte Format der Bilder. Es können innerhalb dieser Erweiterung nur Bilder verarbeitet werden, die das Format **Int16!** (**Int16!**) haben. Der Grund hierfür liegt in der Implementierung des Algorithmus. Die anatomische Segmentierung wurde speziell für die CT-Aufnahmen im ISQ Typ entwickelt. Aus diesem Grund wurden die Schwellwerte und Parameter für dieses Verfahren fest codiert. Führt man dennoch die Segmentierung mit einem Bild im Format 8UInt durch, so stellt man fest, dass der Algorithmus zwar ein Ergebnis generiert, dieses aber nicht brauchbar ist. Die Abbildung 5.13 zeigt ein solches falsche Ergebnis.

Zu sehen ist ein 3D Modell, das nur aus einem Segment besteht. In diesem Fall wurde der gesamte Zahn mit Pulpa als Dentin markiert. Nach einer etwas detaillierteren Analyse konnte festgestellt werden, dass das Problem wohl im Bereich der Auffüllung liegt. Den Grund dafür liefern die einzelnen Datensätze während der Segmentierung. Betrachtet man diese genauer so fällt auf, das der Zahn zu Beginn richtig segmentiert wird, auch

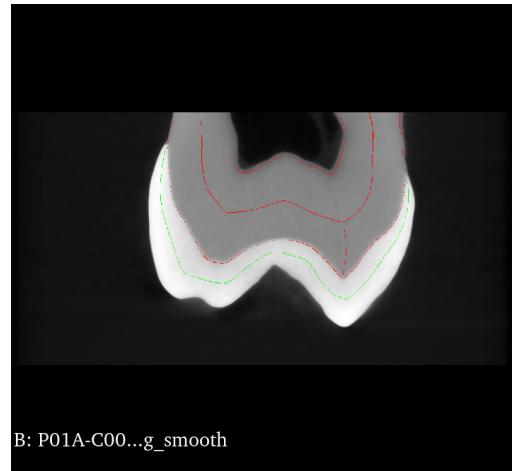


Abbildung 5.12.: Klassifizierung von Karies mittels der medialen Flächen.

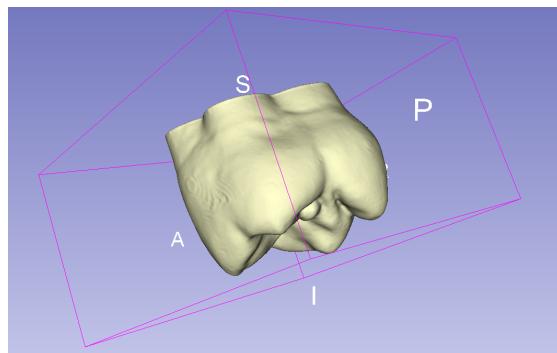


Abbildung 5.13.: Fehlerhafte Segmentierung einer CT-Aufnahme im Format 8UInt

ohne die Pulpa. Nachdem dann die Auffüllung über das Bild läuft, wird diese zuerst richtige Segmentierung wieder verworfen.

Durch diese Limitierung ergibt sich eine weitere. Zu Beginn in der Analysephase des Projektes, war eine Vorverarbeitung der Bilder vorgesehen, dass die Komprimierung eines Bildes vornimmt und so die Bilder deutlich handlicher macht. Das Verfahren hierzu wurde in Kapitel 2.2.2 erläutert. Da dieses Verfahren allerdings einen Formatwechsel von 16Int nach 8UInt bewirkt und diese Bilder nicht richtig segmentiert werden können, scheidet die Vorverarbeitung von Bildern aus.

Eine weitere Limitierung der Software liegt im Batch Modus. Wird ein Batch Modus ausgeführt, so werden im Anschluss die Bilder nicht in die Szene geladen. Dies muss manuell erfolgen über den Import des Kernsystems. Soll für die unterschiedlichen Bilder ein 3D Modell erzeugt werden müssen diese ebenfalls manuell als Segmentierung geladen werden. Diesem Verhalten könnte man mit einer einfachen Checkbox entgegenwirken, indem man die Checkbox setzt, wenn man nach dem Batch alle Bilder laden möchte.

Zuletzt sei noch auf eine Limitierung hingewiesen, die der Erweiterbarkeit des Moduls dient. Soll die Erweiterung um zusätzliche Funktionen erweitert werden, dann sind kleine Änderungen in einer bestehenden Klasse notwendig. Konkret geht es hier um die Klasse `ToothAnalyserWidget`. Hier muss je nach Ausprägung der UI der neuen Funktion, Methoden hinzugefügt werden. Diese Situation ist zwar entgegen vieler Konventionen, jedoch sorgt sie auch dafür, dass die Komplexität des Quellcodes nicht zu groß wird.

Mit der Beschreibung dieser limitierenden Faktoren wurden alle Ergebnisse, die in dieser Arbeit erzielt wurden präsentiert. Es bleibt zum Schluss noch eine Interpretation der Ergebnisse aus kritischer Sicht.

6. Diskussion und Fazit

Betrachtet man gegen Ende dieser vorliegenden Arbeit nochmals die Forschungsfrage aus Kapitel 3 so können noch weitere Punkte hinzugefügt werden. Grundsätzlich lässt sich sagen, dass die Integration der anatomischen Segmentierung aus Kapitel 2.4 gut in das Ökosystem von Slicer integriert werden konnte. Somit ergibt sich die Situation, dass das zuvor aufwendig zu bedienenden Verfahren nach dieser Arbeit mit nur wenigen Klicks ein Ergebnis liefert. Bei der Integration dieses Verfahrens mussten keine Abstriche in Bezug auf die Qualität des Ergebnisses gemacht werden. Die Benutzerfreundlichkeit der Software konnte so um ein Vielfaches verbessert werden. Es steigt auch der Benutzerkreis der Anwendung. Der Entwicklungsprozess selbst kann ebenfalls als durchwegs positiv betrachtet werden. Durch die sehr gute Dokumentation von 3D Slicer konnten schnell erste Ergebnisse erzielt werden, die noch nicht final waren, jedoch in die richtige Richtung gingen. Es lässt sich so sagen, dass einfache Algorithmen ohne viel zusätzliche Abhängigkeiten, mit etwas Vorwissen schnell in einem Slicer Modul integriert werden können. Es ist auch nicht zwingend notwendig, ein Modul über den *Extension Manager* zu veröffentlichen. Soll das Modul nur wenigen ausgewählten Personen zur Verfügung stehen, so kann es auch lokal in die Slicer Anwendung eingebunden werden. Betrachtet man die Erweiterbarkeit der Software, so kann auch diese als weitestgehend erfolgreich angesehen werden. Durch die Kapselung des zugrunde liegenden Verfahrens lässt sich dieses einfach durch ein anderes austauschen. Auch das Hinzufügen neuer Funktionalität lässt sich problemlos realisieren. Es sei jedoch gesagt, dass zu viele unterschiedliche Funktionen zur Überladung des Moduls führen könnte. Zu empfehlen ist dann eine Aufteilung in Submodule. Bei der technischen Umsetzung dieser Ansprüche konnten jedoch nicht alle Prinzipien einer sauberen Softwareentwicklung gewährleistet werden. Diese Entscheidung war jedoch aktiv und hat auch positive Auswirkungen auf das Gesamtsystem. Abschließend lässt sich also sagen, dass diese vorliegende Arbeit der zu Beginn gestellten Forschungsfrage gerecht werden konnte und alle Anforderungen erfüllte. Betrachtet man jedoch die Ziele dieser Arbeit aus Kapitel 1.1 so stellt man fest, dass der Tooth Analyser noch lange nicht den Reifegrad erreicht hat, der alle diese Punkte abdeckt. Um dies endgültig zu gewährleisten, sind unter anderem weitere Benutzertests und gegebenenfalls Fehlerbehebungen notwendig.

7. Schlussfolgerung und Ausblick

Der Tooth Analyser bietet mit dem aktuellen Stand bereits einen großen Mehrwert für die Ärzte in der Klinik für Zahnerhaltung. So lässt sich sagen, dass diese vorliegende Arbeit durchaus von Erfolg gekrönt ist. Durch das Kapitel 5.5.4 wird aber auch klar, dass noch viel Potenzial im Tooth Analyser steckt. Das meiste Potenzial ist hierbei in den Limitierungen zu finden. Eine hervorragende Ergänzung dieser Arbeit würde eine Vorverarbeitung der Bilder bieten. Es soll also in Zukunft möglich sein, die CT-Aufnahmen in einer Vorverarbeitung zu komprimieren und sie dann mittels eines Verfahrens zu bearbeiten. Dies schließt auch eine Verarbeitung aller Formate ein. Ein schöner Nebeneffekt dieser Ergänzung ist, dass durch eine Komprimierung der Bilder die Laufzeit sinkt. Der verlorene Detailgrad dieser Komprimierung ist dabei nicht störend. Somit lässt sich sagen, dass durch das Akzeptieren unterschiedlicher Formate der größte Mehrwert für den Tooth Analyser gewonnen werden kann. Des Weiteren lässt der Bereich Analysen im Tooth Analyser noch viel Spielraum. Hier gibt es diverse Möglichkeiten, das Modul noch mit weiteren Funktionen zu bestücken. Eine gute Idee ist hier eine Integration des Pythonpaketes *radiomics*. Damit lassen sich Radiomics-Merkmale aus Bildern extrahieren und isoliert analysieren. Den letzten Ausblick, der hier gegeben werden soll, bezieht sich wieder auf das Verfahren der anatomischen Segmentierung. In der aktuellen Version nimmt das Verfahren nur eine Aufteilung in Schmelz und Dentin vor. Der Teil der Pulpa wird dem Hintergrund zugeordnet und nicht beachtet. Die zusätzliche Segmentierung der Pulpa würde die Arbeit ebenfalls hervorragend ergänzen. Jedoch sei auch gesagt, dass dies die wohl herausforderndste Aufgabe die hier erwähnten Ausblicke ist. Die Pulpa hebt sich nur sehr leicht aus dem Hintergrund hervor und ist demnach schwer zu segmentieren.

Ordnet man die hier genannten Punkte nach ihrer Wichtigkeit ein, so lässt sich sagen, dass durch das Erweitern des Moduls auf mehrere Bildformate der größte Mehrwert für den Tooth Analyser gewonnen werden kann. Jedoch sei auch gesagt, dass die übrigen Punkte ebenfalls einen großen Mehrwert für den Tooth Analyser und damit für die praktizierenden Ärzte am Klinikum für Zahnerhaltung in München bietet.

Literaturverzeichnis

- 3D Slicer Community (2024). *3D Slicer: A multi-platform, free and open source software package for visualization and image analysis.* <https://www.slicer.org>. Zugriff am 21. November 2024 (siehe S. vi, 17–19, 21–23, 28).
- 3D Slicer Index (2024). *Slicer Extensions Index.* <https://github.com/Slicer/ExtensionsIndex>. GitHub repository (siehe S. vi, 19).
- Baird, Emily und Gavin Taylor (2017). „X-ray micro computed-tomography“. In: *Current Biology* 27.8, R289–R291 (siehe S. 7).
- Bein, Berthold (2006). „Entropy“. In: *Best Practice & Research Clinical Anaesthesiology* 20.1, S. 101–109 (siehe S. 15).
- Bromiley, PA, NA Thacker und E Bouhova-Thacker (2004). „Shannon entropy, Renyi entropy, and information“. In: *Statistics and Inf. Series (2004-004)* 9.2004, S. 2–8 (siehe S. 15).
- Burger, Wilhelm und Mark James Burge (2009). *Digitale Bildverarbeitung: Eine Algorithmische Einführung Mit Java*. Springer-Verlag (siehe S. 9, 14).
- Buzug, Thorsten M (2011). „Computed tomography“. In: *Springer handbook of medical technology*. Springer-Verlag, S. 311–342 (siehe S. 7).
- Crespigny, Alex de, Hani Bou-Reslan, Merry C Nishimura, Heidi Phillips, Richard AD Carano und Helen E D'Arceuil (2008). „3D micro-CT imaging of the postmortem brain“. In: *Journal of neuroscience methods* 171.2, S. 207–213 (siehe S. 3).
- Diwakar, Manoj und Manoj Kumar (2018). „A review on CT image noise and its denoising“. In: *Biomedical Signal Processing and Control* 42, S. 73–88 (siehe S. 10).
- Fedorov, Andrey, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Christophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dylan Jennings, Fiona M. Fennessy, Milan Sonka, John Buatti, Stephen R. Aylward, James V. Miller, Steve Pieper und Ron Kikinis (Nov. 2012). „3D Slicer as an Image Computing Platform for the Quantitative Imaging Network“. In: *Magnetic Resonance Imaging* 30.9, S. 1323–1341. doi: 10.1016/j.mri.2012.05.001 (siehe S. vi, 17–21).
- Handels, Heinz (2000). *Medizinische Bildverarbeitung*. Springer-Verlag (siehe S. vi, 2, 10–13).
- Heck, Katrin, Friederike Litzenburger und Karl-Heinz Kunzelmann (Jan. 2024). *In-vitro caries diagnostic study - 375 microtomographic images of the coronal part of extracted human teeth*. Zugriff am: 2024-11-15. doi: 10.5282/ubm/data.445 (siehe S. vi, 2, 6).
- Hofmann, Simon (Jan. 2020). „Unterstützung der Karies-Klassifizierung in Mikro-CT-Aufnahmen durch 3D-Bildverarbeitung“. Bachelorarbeit. Technische Hochschule Augsburg (siehe S. vi, 5, 15–17).
- ITK Development Team (2024). *ITK: Insight Segmentation and Registration Toolkit.* <https://itk.org>. Entwickelt und gepflegt von Kitware, Zugriff am 21. November 2024 (siehe S. 22).
- Jähne, Bernd (2024). *Digitale bildverarbeitung: und bildgewinnung*. Springer-Verlag (siehe S. 9).

- Lehmann, Klaus M, Elmar Hellwig und Hans-Jürgen Wenz (2012). *Zahnärztliche Propädeutik: Einführung in die Zahnheilkunde; mit 32 Tabellen*. Deutscher Ärzteverlag (siehe S. vi, 5, 6).
- Lehmann, Thomas, Walter Oberschelp, Erich Pelikan und Rudolf Repges (2013). *Bildverarbeitung für die Medizin: Grundlagen, Modelle, Methoden, Anwendungen*. Springer-Verlag (siehe S. vi, 7, 12–14).
- National Institute of Biomedical Imaging and Bioengineering (NIBIB) (2024). *X-Rays - Science Topic*. Zugriff am: 2024-11-15. URL: <https://www.nibib.nih.gov/science-education/science-topics/x-rays> (siehe S. 7, 8).
- Poliklinik (2024). *Poliklinik für Zahnerhaltung und Parodontologie des LMU-Klinikums München*. <https://www.klinikum.uni-muenchen.de/Poliklinik-fuer-Zahnerhaltung-und-Parodontologie/>. Zugriff am 22. November 2024 (siehe S. 9).
- Qt Development Team (2024). *Qt Documentation: Introduction to Qt*. <https://doc.qt.io/qt-6/qt-intro.html>. Entwickelt und gepflegt von The Qt Company, Zugriff am 21. November 2024 (siehe S. 22).
- Ramesh, KKD, G Kiran Kumar, K Swapna, Debabrata Datta und S Suman Rajest (2021). „A review of medical image segmentation algorithms“. In: *EAI Endorsed Transactions on Pervasive Health and Technology* 7.27, e6–e6 (siehe S. 12).
- Rösch, Peter und Karl-Heinz Kunzelmann (2018). „Efficient 3D Rigid Registration of Large Micro CT Images“. In: *International Journal of Computer Assisted Radiology and Surgery*; Bd. 13. June 2018, issue 1 (supplement), S. 118–119. doi: 10.1007/s11548-018-1766-y (siehe S. 8).
- SCANCO Medical AG (2024). *Brochures and Documentation*. Zugriff am: 22. November 2024 (siehe S. 8).
- Siebler, F. (2014). *Design Patterns Lernen M. Java*: Carl Hanser Verlag (siehe S. 40).
- VTK Development Team (2024). *VTK: The Visualization Toolkit*. <https://vtk.org>. Entwickelt und gepflegt von Kitware, Zugriff am 21. November 2024 (siehe S. 22).
- Walter, Elias (Feb. 2025). *Interview zur Anforderungsanalyse für die Slicer Erweiterung*. Persönliches Interview. Geführt von Lukas Konietzka. (siehe S. 26, 27).
- Walter, Elias, Agnes Wolf, Falk Schwendicke und Katrin Heck (2025). *Automatic Detection and Analysis of Carious Lesions in Micro CTs*. Projektbeschreibung. Unveröffentlichtes Dokument (siehe S. 3).
- Zwinkels, Joanne (2015). „Light, electromagnetic spectrum“. In: *Encyclopedia of Color Science and Technology* 8071, S. 1–8 (siehe S. vi, 7).

A. Anhang

A.1. Klassendiagramm

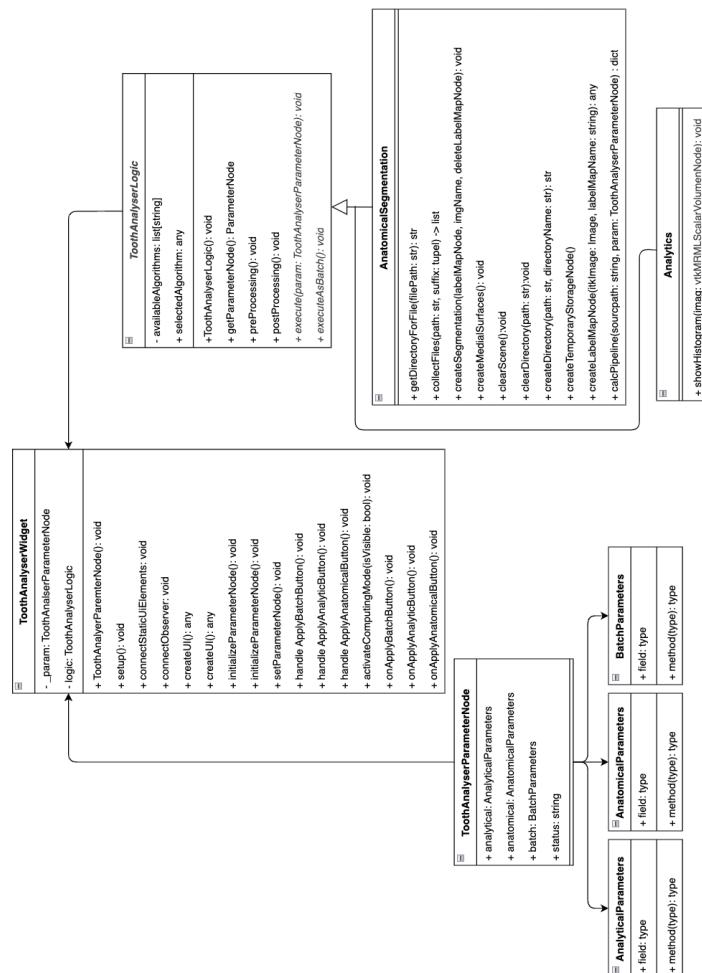


Abbildung A.1.: Klassendiagramm des Tooth Analyser

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig 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.

Augsburg, den 4. März 2025

Lukas Konietzka