HOCHSCHULE
MITTWEIDA
University of Applied Sciences

Fakultät Angewandte Computer- und Biowissenschaften

Professur Medieninformatik

Masterarbeit

Anwendung von MVCNN-Verfahren auf synthetisch generierte 3D-OCT-Abbildungen der menschlichen Netzhaut

Noah Stolz

Mittweida, den 23. November 2021

Erstprüfer:	Prof. Dr. rer. nat. Marc Ritter
Zweitprüfer:	Dipl. Inf. Holger Langner

Stolz, Noah
Anwendung von MVCNN-Verfahren auf synthetisch generierte 3D-OCT-Abbildungen der menschlichen Netzhaut
Masterarbeit, Fakultät Angewandte Computer- und Biowissenschaften
Hochschule Mittweida — University of Applied Sciences, November 2021

Zusammenfassung

Ziel dieser Arbeit ist das Evaluieren der Klassifikationsfähigkeit eines MVCNN-Verfahrens am Teilproblem der Klassifikation von prozedural generierten, idealisierten Darstellungen von OCT-Scans. Zu diesem Zweck wird ein Tool für das Erstellen solcher Szenen entwickelt sowie ein Algorithmus zur Volumenberechnung von sich überschneidenden Meshes, welcher für das automatische Labeling dieser Szenen verwendet wird.

Name: Stolz, Noah
Studiengang: Medieninformatik und Interaktives Entertainment
Seminargruppe: MI19w1-M

English Title: Application of MVCNN-Processes on synthetically generated 3D-OCT-representations of the human retina

Inhaltsverzeichnis

1 Einführung

1.1 Researchproposal

Im Bereich des Deep Learning Verfahren wurden in den letzten Jahren große Fortschritte dabei erzielt, medizinische Bilddaten durch Convolutional Neural Networks (CNN) zu klassifizieren. Die Anwendung auf 3D-Daten zeigt im Vergleich dazu erst allmähliche Fortschritte. In einem vorangegangen Forschungsmodul, welches die Grundlage für diese Arbeit darstellt, wurde ein Multi View Convolutional Neural Network (MVCNN)-Framework in einen Workflow zur Generierung annotierter 3D-Modelle aus 2D-OCT-Daten der menschlichen Netzhaut integriert. Der MVCNN-Ansatz kombiniert dazu mehrere Kameraperspektiven und verknüpft diese mit 2D-CNNs, um 3D-Daten zu klassifizieren. Anwendungsfall ist die Klassifikation von Krankheitssymptomen aus OCT (Optic Coherence Tomography)-Scans der menschlichen Netzhaut. OCT-Scans zeigen Retinagewebe über mehrere benachbarte Querschnittsbilder, in welchen Ödeme oder Drusen erkannt werden können, die typische Anomalien für degenerative Erkrankungen der Netzhaut darstellen. Um die Unterscheidung verschiedener Krankheitssymptome zu erleichtern, wurden diese im Rahmen der Vorarbeiten über mehrere zweidimensionale Bilder eines OCT-Scans hinweg grafisch annotiert und in ein meshbasiertes 3D-Modell überführt, welches dann mit dem beschriebenen Framework klassifiziert werden kann. Die Evaluation im Rahmen der Vorarbeiten zeigte die prinzipielle Anwendbarkeit dieses Vorgehens. Jedoch ist die Menge an eindeutig annotierbaren und qualitativ hochwertigen Realwelt-Bilddaten für weitere Untersuchungen zu klein, um den Ansatz systematisch für komplexere Problemstellungen zu untersuchen. Daher ist es nun erforderlich, darüber hinaus weitere ergänzende Analysen zunächst über synthetisch generierten Bildern durchzuführen.

Im Rahmen dieser Masterarbeit soll das MVCNN-Framework auf komplexere Klassifikationsprobleme übertragen werden. Zugleich soll untersucht werden wie sich verschiedene Parameter der 3D-Szene, wie unterschiedliche Shader und Belichtungen

des Objektes, die Präsenz zusätzlicher 3D-Modelle, um die relative Position des Objektes zu erfassen usw. auf die erreichbare Klassifikationsgüte auswirken. Dazu soll zunächst ein Workflow für die prozedurale Generierung von synthetischen OCT-3D-Modellen typischer Befundkonstellationen entwickelt und praktisch eingesetzt werden. Dieser erlaubt es schrittweise immer komplexere Krankheitsbilder in illustrativer, idealisierter Form abzubilden und daraus größere Mengen von Trainings- und Testbildsätzen für die Anwendung Maschineller-Lernverfahren parametrisiert und randomisiert zu generieren. In einer abschließenden Evaluation wird die Klassifikationsgüte des MVCNN-Frameworks für ausgewählte exemplarische Klassifikationsprobleme und unterschiedliche Parametrisierungen der synthetischen Bilddaten vergleichend untersucht.

1.2 Aufbau

Einleitung Dieses Kapitel stellt die Fragestellung und das Ziel dieser Arbeit vor.

Grundlagen Die Grundlagen geben das für das Verständnis dieser Masterarbeit notwendige Vorwissen, einerseits über die dieser vorangegangenen Arbeiten und andererseits über die für diese Arbeit relevanten Konzepte.

Konzeption In der Konzeption werden die Anforderungen für das Beantworten diskutiert sowie die Umsetzung dieser durch das Erstellen von Tools.

Implementierung Das Implementierungskapitel befasst sich mit den technischen Details des Erstellens der Tools und den Beschränkungen, welche sich durch die verwendeten Technologien ergeben.

Evaluation In der Evaluation wird das maschinelle Lernverfahren anhand eines Teilproblems getestet.

Fazit In diesem Kapitel werden die erstellten Tools und Ergebnisse der Evaluation mit dem anfänglichen Researchproposal verglichen.

Ausblick Als letztes Kapitel wird ein Ausblick darüber gegeben, wie die erstellten Tools weiter verbessert und die Verfahren getestet werden könnten und wie die Ergebnisse dieser Arbeit zukünftig verwendet werden könnten.

2 Grundlagen

In diesem Kapitel soll das Vorwissen vermittelt werden, welches für ein grundsätzliches Verständnis der Arbeit notwendig ist. Dieses Vorwissen umfasst OCT-Scans, auf welchen die in dieser Arbeit verwerteten Daten beruhen sowie die verwendeten Technologien wie maschinelle Lernverfahren, Mesh-Generierung und Game-Engines. Abschließend wird die dieser Arbeit vorangegangene Vorarbeit erläutert und welche Erkenntnisse der Vorarbeit für diese Arbeit relevant sind.

2.1 OCT-Scans

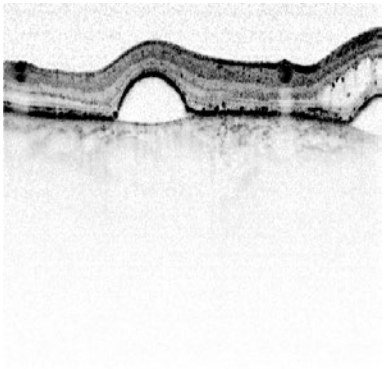

Abbildung 2.1: Ein Bild eines OCT-Scans, in welchem die Form eines Ödems zu erkennen ist.

Der Ursprung des Forschungsthemas liegt in der Optical-Coherence-Tomography (OCT). Hierbei handelt es sich um eine Methode zum Abbilden von Gewebe. Diese zeichnet sich durch ihren nicht invasiven Prozess ab, was bedeutet, dass das Gewebe nicht beschädigt wird, um die Abbildung zu erstellen. Das Erstellen der Bilder basiert auf einem Signal, welches abhängig von der Beschaffenheit des Gewebes mit unterschiedlicher Geschwindigkeit reflektiert wird und dadurch Unterschiede in Bildform erkennen lässt [HSL+91, S.1178f]. Eine solche Abbildung oder OCT-Scan besteht

aus einer Sequenz von Bildern, welche hintereinander in gleichen Abständen erstellt werden. OCT-Scans finden große Anwendung im Bereich der Augenheilkunde, wo sie verwendet werden, um Krankheitssymptome wie Ödeme und Drusen zu erkennen. Augenkrankheiten stellen weltweit eine der häufigsten Erkrankungen im Alter dar. Krankheiten wie die altersbedingte Makula Degeneration (AMD)[Sch06], welche durch Wassereinlagerungen in der Netzhaut erzeugt wird, stellt ein häufiges Problem dar. Projektionen schätzen die möglichen Fälle von AMD bis 2040 auf 288 Mio. [BSW+16]. OCT-Scans sind ein wichtiges Werkzeug zur Analyse solcher Krankheiten. Nach wie vor schwierig ist jedoch das Voraussagen von möglichen Krankheiten durch OCT-Scans von noch gesunden Patienten. Ein Ansatz, welcher hierbei helfen soll, sind maschinelle Lernverfahren (siehe 2.4). Diese sollen zum Beispiel Muster in den Daten von Patienten erkennen, welche an Augenkrankheiten leiden und diese Muster in den Daten von noch gesunden Patienten wieder erkennen. Diese Arbeit ist eine Vorstufe zu solchen Ansätzen und versucht auf OCT-Scan basierende 3D-Daten mit Maschinellen Lernverfahren zu kombinieren.

2.2 Meshes

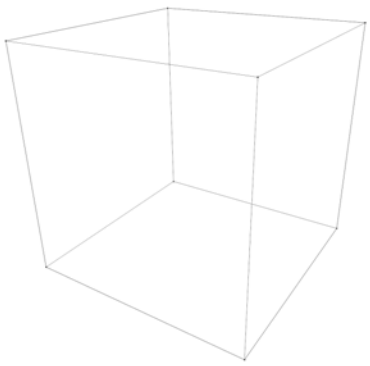

Abbildung 2.2: Das Mesh eines Würfels. Es besteht aus 8 Vertices verbunden mit 12 Kanten, zwischen welchen 6 Flächen aufgespannt sind.

In der Computergrafik ist die Darstellung von drei dimensionalen Formen eine der grundlegendsten Herausforderungen. Hierfür gibt es verschiedene Ansätze wie zum Beispiel Pointclouds, welche die Form eines Objektes durch Punkte beschreiben oder Voxel, welche eine Art von drei dimensionalen Pixeln darstellen. In dieser Arbeit werden sogenannte Meshes verwendet. Dies ist eine der verbreitetsten Methoden

zur Darstellung von 3D-Formen und findet Anwendung in Animationsfilmen, Video-Spielen, 3D-Visualisierungen, etc. Ein Mesh besteht aus Vertices, welche die "Ecken" des Meshs darstellen, Kanten, welche diese Ecken verbinden und Flächen, welche zwischen diesen aufgespannt werden [HDD+93, S.3f] (siehe Abbildung 2.2). Eine Besonderheit des Meshes ist, dass es nur die äußere Form eines Objektes darstellt. Dies bedeutet, dass es keinerlei Informationen über das Innere eines Objektes geben kann, aber macht das Mesh gleichzeitig zu einer äußerst effizienten Methode der Darstellung.

Meshes können auf zwei Arten erstellt werden. Die erste ist die meistverwendete Methode, welche auf dem intellektuellen Erstellen beruht. In diesem Fall fängt ein 3D-Artist meist mit einer grundlegenden Form an und fügt dieser neue Kanten und Flächen hinzu, bis diese der gewünschten Form entspricht. Dieser Vorgang erlaubt das Hinzufügen von spezifischen Details, aber ist zeitaufwendig und kann während der Laufzeit eines Programms nicht mehr angepasst werden. In dieser Arbeit findet die zweite Methode Mesh-Generierung Anwendung. Hierfür wird das Mesh durch einen Algorithmus prozedural generiert. Dieser Algorithmus muss im ersten Schritt die Vertices positionieren. Je nach gewünschtem Mesh kann dies eine vorgegebene Form wie ein Würfel sein oder eine Form mit einem Zufallselement, welches eine randomisierte Form ergibt. Die Vertices werden im zweiten Schritt mit Kanten verbunden. Diese Kanten werden mit Flächen verbunden, um die Form des Meshes erkennbar zu machen.

Ein wichtiger Aspekt für diese Arbeit ist das Bestimmen des Volumens eines Meshes. Zhang und Chens Methode [ZC01] findet hier Anwendung. Diese Methode stellt mehrere Anforderungen an das Mesh. Die erste Anforderung ist, dass das Mesh geschlossen sein muss. Das heißt das Mesh darf keine Löcher aufweisen, welche nicht von Flächen gefüllt sind. Die zweite Anforderung ist, dass das Mesh sich nicht selbst schneidet, da der Algorithmus nicht in der Lage ist die sich überschneidenden Volumen akkurat zu berechnen. Ein anlehnendes Problem ist das Bestimmen des Volumens von Meshes welche sich mit einem anderen Mesh überschneiden. Es ist zwar möglich die einzelnen Volumen dieser Meshes zu bestimmen, aber nicht das gesamte Volumen, welches nicht den überschneidenden Bereich doppelt zählt. Diese letzte Einschränkung ist ebenfalls ein Thema dieser Arbeit.

2.3 Unity

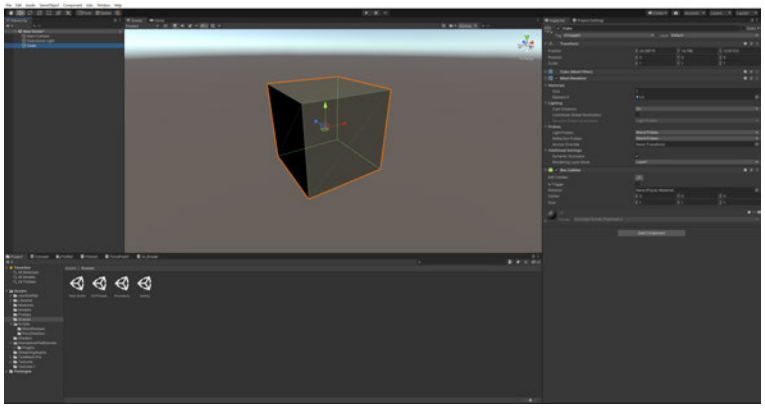

Abbildung 2.3: Die Benutzeroberfläche der Unity-Game-Engine. Das zentrale Fenster gibt einen Blick auf die 3D-Szene, an welcher aktuell gearbeitet wird. Das Fenster soll zeigen, wie die Anwendung in ihrer Benutzung aussieht. Links davon ist die Objekthierarchie, welche alle in der Szene befindlichen Objekte auflistet. Rechts sind die Details des momentan selektieren Objekts zu finden. Unter diesen befindet sich der Asset-Browser, welcher alle vorhandenen Elemente des gesamten Projekts beinhaltet.

Für diese Arbeit wird die Game-Engine Unity verwendet. Eine Game-Engine ist ein Tool, welches unter anderem das Erstellen von Anwendungen mit 3D-Rendering erlaubt [AECM08, S.2]. Darüber hinaus stellen sie aber auch weitere Tools wie Physik-Simulationen und Networking zur Verfügung. Die Game-Engine stellt außerdem die Plattform dar, in welcher die Elemente einer Anwendung zusammengesetzt werden. Dies umfasst Assets wie 3D-Modelle, Texturen und Audio-Dateien, aber auch die Logik der Anwendung durch Skripte. Sobald die Arbeit an der Anwendung abgeschlossen ist, wird ein Build erstellt, welcher ohne die Engine benutzt werden kann. Game-Engines werden meist für die Arbeit an Videospielen verwendet, aber können für jedes Tool, welches 3D-Rendering benötigt, verwendet werden. Die für diese Arbeit gewählte Engine Unity zeichnet sich durch eine weite Verbreitung und detaillierte Dokumentation aus. Das Verwenden einer Game-Engine und spezifischer Unity bringt jedoch auch technische Vorgaben und Einschränkungen mit sich. Die Details dieser werden im Kapitel 4 Implementierung ausgeführt.

2.4 Maschinelle Lernverfahren

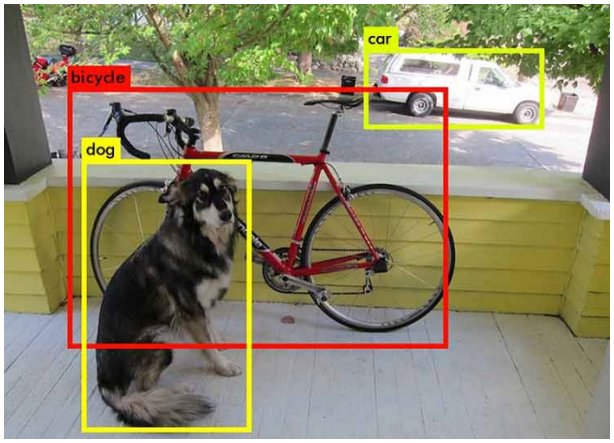

Abbildung 2.4: Das Bild zeigt ein Beispiel für die Verwendung eines maschinellen Lernverfahrens. In diesem Beispiel wurde das Verfahren darauf trainiert Objekte wie Hunde, Autos und Fahrräder in einem Bild zu lokalisieren.[1]

Maschinelle Lernverfahren beschreiben das Feld von Algorithmen, welche sich durch Training in dem Ausführen ihrer Aufgabe verbessern können [GBC16, S.2f]. Solche Algorithmen finden in vielerlei Bereichen Anwendung, wie zum Beispiel beim Identifizieren von Personen in Bildern, dem Erkennen von Spammails oder dem Spielen von Videospielen. Was die maschinellen Lernverfahren auszeichnet, ist die Mustererkennung, welche es dem Algorithmus erlaubt, spezifische Probleme auf nahezu menschlichen Leveln zu lösen und diese in manchen Fällen zu übertreffen. Allgemein können maschinelle Lernverfahren in drei Kategorien eingeteilt werden: Supervised Learning, Unsupervised Learning und Reinforcement Learning. Supervised Learning ist die typischste Form und wird zum Beispiel für die Bildklassifikation verwendet. Hierbei wird das Verfahren mit einem Datensatz trainiert. Dieser Datensatz ist nach im Voraus gewählten Klassen gelabelt. Diese Klassen könnten am Beispiel der Bildklassifikation sein: ”Bild mit Hund” und ”Bild mit Katze”. Der Datensatz besteht nun aus mehreren Bildern, für welche intellektuell festgelegt wurde, ob in dem Bild ein Hund oder eine Katze zu sehen ist. Die Bilder werden in Trainings- und Testbilder geteilt. Die Trainingsbilder werden dem Verfahren übergeben, um

[1] https://towardsdatascience.com/deep-learning-method-for-object-detection-r-cnn-explained-ecdadd751d22

den Algorithmus zu trainieren. Diese passt sich über mehrere Trainingsdurchläufe an, um die Bilder zuverlässiger klassifizieren zu können. Das Durchlaufen aller Bilder entspricht einer sogenannten Epoche. Die Testbilder bleiben dem Algorithmus unbekannt aber werden verwendet, um die Qualität des Algorithmus zu überprüfen. Nach einer gewünschten Anzahl von Epochen wird das Training beendet und das Verfahren verwendet, um Bilder zu klassifizieren. Unsupervied Learning wird häufig für Clustering verwendet. Dies bedeutet, dass ungelabelte Daten dem Verfahren übergeben werden und dieses versucht, die Daten nach gemeinsamen Eigenschaften in Gruppen einzuteilen. Reinforcement Learning beschreibt einen Algorithmus, welcher versucht, durch einen Ablauf von Befehlen eine Erfolgsformel zu optimieren. Ein Beispiel hierfür wäre ein Videospiel, in welchem das Verfahren alle, dem Spieler mögliche, Inputs tätigen kann. Die Erfolgsformel könnte in diesem Fall eine Bestzeit im Spiel sein. Der Algorithmus würde nun versuchen, die Inputs in einer Reihenfolge ausführen, welche in der höchsten Bestzeit resultiert.

Für diese Arbeit wird ein Convolutional Neural Network (CNN) verwendet. Dies ist ein Supervised Learning Verfahren. CNN werden vor allem für die Bildklassifikation verwendet, welche sie durch mehrere Dimensionsreduktionen effizienter bewältigen können als die meisten Verfahren [ON15, S.3f].

2.5 Vorhergehende Arbeit

In dem dieser Arbeit vorausgegangenen Forschungsmodul wurde nach einer Methode gesucht, wie auf OCT-Scans basierende 3D-Modelle mit maschinellen Lernverfaren klassifiziert werden könnten. Wie genau diese 3D-Modelle gestaltet werden sollten, wurde als sekundäre Priorität gesetzt. Der Fokus der Arbeit lag auf dem Finden einer Methode zur Klassifizierung solcher 3D-Visualisierungen. Aus diesem Grund wurden nach maschinellen Lernverfahren für 3D-Daten gesucht. Der größte Unterschied dieser Verfahren liegt in der Art von 3D-Darstellung, welche verwendet wird [ASS+18, S.6ff]. Die unterschiedliche Beschaffenheit von Darstellungen wie zum Beispiel Voxel und Pointclouds machen unterschiedliche Ansätze zu deren Klassifizierung notwendig. Da die in Unity bevorzugte Darstellung Meshes sind und dar für das Generieren dieser bereits Vorwissen bestand, wurde eine auf Meshes basierende Methode bevorzugt. Laut Jong-Chyi Su et al. [SGWM18, S.7] war die vielversprechendste Methode

hierfür ein Multi-View-Convolutional-Neural-Network (MVCNN). Dieses wurde von Hang Su et al. [SMKLM15, S.2f] vorgeschlagen und erläutert.

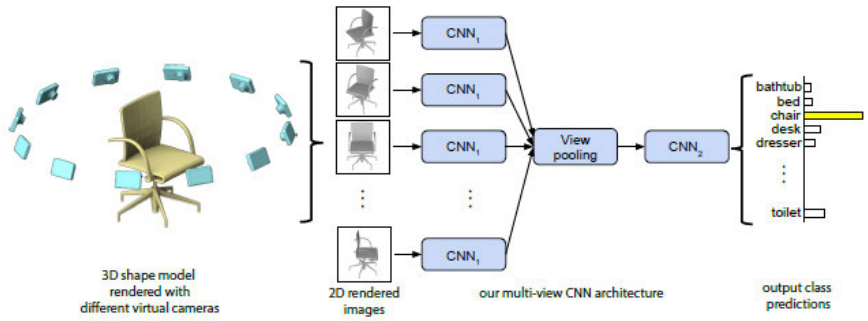

Abbildung 2.5: Der Aufbau eines Multi-View-Convolutional-Neural-Networks. Mehrere Kameras um ein 3D-Modell herum nehmen Bilder auf. Jedes diese Bilder wird jeweils einem CNN pro Kamera-Perspektive übergeben. Die Zwischenergebnisse dieser CNNs werden kombiniert und einem letzten CNN übergeben. Dieses CNN führt die Klassifikation durch.[2]

Die Grundidee des MVCNN ist es, mehrere Perspektiven um das Objekt herum mit jeweils einem CNN zu trainieren und das Ergebnis dieser zu kombinieren (siehe Abbildung 2.5). Für diesen Ansatz werden mehrere Kameras um das Mesh platziert, welches klassifiziert werden soll. Im Fall von Objekten, welche eine eindeutige Ausrichtung besitzen, werden zwölf Kameras empfohlen. Ein Beispiel hierfür wäre das 3D-Modell eines Stuhls, dessen Sitzfläche und Lehne die Oberseite klar markieren. Falls die Richtung des Objektes nicht eindeutig festlegbar ist, wie zum Beispiel bei dem 3D-Modell eines Steins, werden stattdessen 20 Kameras empfohlen. Für das Forschungsmodul wurde letztere Option gewählt, da den Formen von Objekten wie Ödemen meist keine Richtung zugewiesen werden kann. Diese Kameras nehmen jeweils ein Bild auf. Es werden Bilder von allen Objekten, welche zum Training des MVCNN benutzt werden sollen, aufgenommen. Alle Bilder einer Perspektive werden anschließend einem eigenen CNN übergeben. Diese 20 CNNs teilen die übergebenen Bilder in Trainings- und Testbilder auf und trainieren mit diesen für die gewünschte Anzahl von Epochen. Ziel ist es hier, die View-CNNs darauf zu trainieren, Objekte aus ihrer jeweiligen Perspektive zu erkennen. Nachdem die View-CNNs trainiert

[2]Hang Su, Subhransu Maji, Evangelos Kalogerakis und Erik Learned-Miller: Multi-view convolutional neural networks for 3d shape recognition

wurden, werden die verwendeten Bilder inklusive der Test-Bilder erneut ihrem jeweiligen CNN übergeben. Der Unterschied liegt darin, dass in diesem Durchlauf das Zwischenergebnis der Bilder vor der letzten Convolutional Layer entfernt wird. Auf dieses Zwischenergebnis wurden bereits mehrere Dimensionsreduktionen angewandt. Die Zwischenergebnisse aller Perspektiven eines Objektes werden kombiniert. Dieses Kombinieren verwendet dieselbe Art von Dimensionsreduktion wie ein CNN nur in diesem Fall über mehrere Bilder angewendet. Das Ergebnis ist ein einziges kombiniertes Bild. Dieser Prozess wird für die Bilder aller Trainingsobjekte wiederholt, sodass ein kombiniertes Bild pro Ausgangsobjekt entsteht. Dieses kombinierten Bilder werden verwendet, um ein letztes CNN, das Final-CNN, zu trainieren. Das Final-CNN gibt die endgültige Klassifikation eines Objektes an. Eine große Stärke des MVCNN ist, dass durch die große Anzahl von Kameras und CNNs ein hohes Level von Redundanz entsteht und die schlechte Accuracy vereinzelter View-CNNs keinen großen Effekt auf das finale Ergebnis hat.

Diese MVCNN-Architektur wurde im Rahmen des Forschungsmoduls innerhalb von Unity implementiert. Hierfür wurde eine C#-Library für CNNs namens "ConvNetSharp" verwendet. Um die implementierte MVCNN-Architektur zu evaluieren und die Machbarkeit des Ansatzes festzustellen, sollte ein simples Klassifikationsproblem erstellt werden. Das gewählte Problem ist die Unterscheidung von stark idealisierten Ödem- und Drusen-3D-Modellen. Hierfür wurden synthetische Daten prozedural generiert. Ödeme wurden mit verformten Kugeln und Drusen mit Pyramiden-ähnlichen Formen dargestellt. Das Ergebnis der Evaluation war eine klare Fähigkeit des MVCNN, die Formen zu unterscheiden. Hier knüpft diese Arbeit an. Es sollen jetzt idealisierte 3D-Nachstellungen eines OCT-Scans klassifiziert werden.

3 Konzept

Dieses Kapitel beschreibt die Konzeption, angefangen mit den aus der Forschungsfrage hervorgehenden Anforderungen und Rahmenbedingungen. Anschließend folgt die Umsetzung dieser Anforderungen. Wie die zu erstellenden Tools zu verwenden sind, wird innerhalb der praktischen Anwendung erläutert.

3.1 Anforderungen

In diesem Unterkapitel werden die Anforderungen, welche aus dem Forschungsfrage hervorgehen, erläutert. Dies beinhaltet Verbesserungen der Ödem-/Drusen-Generierung, ein Tool zur Generierung von 3D-OCT-Szenen und Features für das maschinelle Lernen wie das Berechnen von Volumen.

3.1.1 Verbesserte Ödem-/Drusen-Generierung

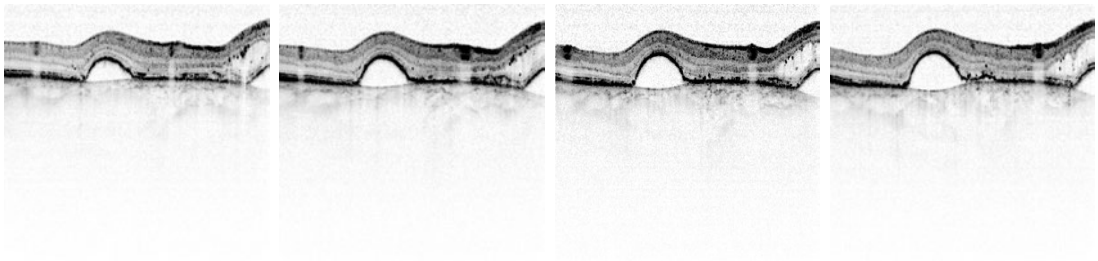

Abbildung 3.1: Eine Bild-Sequenz eines OCT-Scans, welche die Form eines Ödems erkennen lässt

In dem dieser Arbeit vorhergegangenen Forschungsmodul wurden bereits die Krankheitserscheinungen für Drusen und Ödeme als simplifizierte Formen generiert. Im Falle der Ödeme als in verschiedene Dimensionen unterschiedlich skalierte Kugeln,

welche die meist rundliche Form der Ödeme approximieren sollte. Die Drusen wurden mit einer Pyramiden-ähnlichen Gestalt dargestellt, um eine scharfkantige Form zu zeigen, welche sich deutlich von den Ödemen abheben sollte. Im Rahmen dieser Arbeit sollen die Formen für diese Objekte erneut generiert werden, mit dem Ziel, die Darstellung einer OCT-Scans basierten Form anzugleichen. Dies bedeutet, dass die Form der Ödeme und Drusen durch mehrere hintereinanderliegende, verbundene Polygone beschrieben wird. Jedes dieser Polygone stellt den Querschnitt eines Objektes innerhalb des Bildes eines OCT-Scans wieder. Die Ödeme und Drusen sollen hierbei deutlich voneinander unterscheidbar sein, durch simple Eigenschaften wie Form und Größe, aber auch durch die grundlegende Form der 3D-Darstellungen. Diese Darstellung soll in beiden Fällen ein organisches Erscheinungsbild behalten, aber den Ödemen eine rundliches, "schaumartiges" und den Drusen ein scharfkantiges, "kristallenes" Aussehen geben. Von diesen Objekten sollen mithilfe von prozeduraler Generierung Variationen erstellt werden, welche sich deutlich von ihrer Ausgangsform unterscheiden, aber in jedem Fall noch als Ödeme bzw. Drusen erkennbar bleiben.

3.1.2 Toolbasierte Generierung von 3D-OCT-Szenen

Diese Meshgenerierung soll innerhalb eines 3D-Tools verwendet werden, um 3D-Szenen zu erstellen. Diese Szenen sollen 3D-Nachstellungen eines OCT-Scans repräsentieren, mit mehreren Ödemen und Drusen sowie zwei übereinander liegende Epithelschichten. Diese dreidimensionale Visualisierung eines OCT-Scans ist die grundlegende Anforderung an das Tool. Eine simple Benutzeroberfläche soll das Anpassen der Parameter der Meshgenerierung erlauben. Der Nutzer soll hierbei Kontrolle über die Position und Randomisierung der Form der Objekte haben und die grundlegenden Eigenschaften eines Objektes wie Rotation und Skalierung auch nach dem Platzieren dieses anpassen können. Die Epithelschichten sollten mehrere Parameter zum Anpassen ihrer Form bereitstellen. Dazu gehört eine grundlegende Randomisierung der Form, um diese organischer als ein ebenes Quadrat erscheinen zu lassen. Die Form der oberen dieser Schichten soll außerdem durch die Fovea, den Punkt des schärfsten Sehens, beeinflusst werden, welche an beliebiger Position platziert werden sollte und Optionen zur Anpassung ihrer Form benötigt, um verschiedene OCT-Scans möglichst gut darstellen zu können. Die obere Epithelschicht sollte außerdem von Objekten, welche sich direkt unter dieser befinden, sichtlich

beeinflusst werden durch Aufwölbungen über diesen Objekten. Außerdem sollten beiden Epithelschichten die Krümmung des Auges erkennbar darstellen. Zuletzt muss bei den Schichten berücksichtigt werden, dass die Objekte sich zwischen den Schichten befinden werden und es trotzdem möglich sein muss, sie sehen zu können. Wichtig für die Arbeit mit diesen Szenen ist eine Option für das Speichern und Laden von Szenen, einerseits um Zwischenstände festhalten zu können und andererseits um Kopien von Szenen erstellen zu können, welche die Basis für weitere Szenen bilden. Da die für das Trainieren eines maschinellen Lernverfahrens benötigte Datenmenge sehr groß ist, sind Methoden, welche das zeiteffiziente Erstellen einer großen Anzahl von Szenen vereinfachen, ebenfalls Ziel der Umsetzung. Zusätzlich sollen Features implementiert werden, welche mit angemessenem Aufwand zu einer signifikanten Verbesserung der User-Experience und Produktivität führen.

3.1.3 Features für Maschinelle Lernverfahren

Entscheidend für diese Arbeit ist eine Problem-Domäne festzulegen, welche das MVCNN zu lösen versuchen soll. In dieser Arbeit soll das Realweltproblem der Unterscheidung von OCT-Scans, welche zu einem gesunden oder im Sehen beeinträchtigten Patienten gehören, mit künstlichen Daten nachgestellt werden. Die Annahme, auf welcher das Klassifikationsproblem beruht, ist, dass die Einschränkung der Sehfähigkeit abhängig ist von dem Volumen der Drusen und Ödeme, welche sich innerhalb eines spezifischen Bereiches um die foveale Senke befinden. Dieser Bereich soll durch einen Zylinder beschrieben werden, welcher an der Position der fovealen Senke zentriert ist. Die erstellten Szenen sollen dann abhängig des Volumens der jeweiligen Ödeme und Drusen, welches sich innerhalb des Zylinders befindet, klassifiziert werden, in ”Gesund” und ”Beeinträchtigt”. Aus diesem Grund ist eine Methode zur Berechnung der Volumen der Ödeme und Drusen notwendig. Diese Methode sollte auch berücksichtigen, dass es möglich ist für die Objekte, sich zu schneiden, und dass dieses überschneidende Volumen nicht mehrfach gezählt werden sollte. Es sollte außerdem nur das Volumen eines Objektes gezählt werden, welches sich tatsächlich innerhalb des Zylinders befindet, und nicht in jedem Fall das gesamte Objekt.

3.1.4 Evaluation

Die Arbeit soll abgeschlossen werden mit einer Evaluation des MVCNNs mithilfe der erstellten Tools. Für die Evaluation dieses maschinellen Lernverfahrens muss ein Teilproblem gewählt werden, an welchem das Verfahren gemessen werden kann. Maschinelle Lernverfahren benötigen große Datenmengen als Trainingssätze, um sie zu testen. Diese Trainingssätze müssen erstellt und gelabelt werden, was durch ausschließlich intellektuelle Verfahren ressourcenkostspielig ist und eine Automatisierung dieses Prozesses eine Priorität sein sollte. Um unterschiedliche Tests durchzuführen, deren Ergebnisse für tiefere Einblicke in das ML-Verfahren verglichen werden können, müssen die Testsätze parametrisierbar sein, um den Umgang des Verfahrens mit kontrollierten Unterschieden zu erfassen. Zuletzt muss es möglich sein, das Generieren und Labeln der Daten zu überwachen, um ausschließen zu können, dass signifikante Ergebnisse innerhalb des ML-Trainings nicht auf Fehlern innerhalb der Trainingssätze beruhen. Hierfür kann es notwendig sein, weitere Tools zu erstellen, deren Aufgabe das Monitoren der anderen Tools ist.

3.2 Umsetzung

Aus den genannten Anforderungen gehen die notwendigen Schritte für die Umsetzung der benötigten Features hervor. Angefangen wird die Umsetzung mit der Generierung der Ödeme und Drusen. Anschließend wird erläutert, wie diese eines der Kernfeatures des Tools zur Erstellung von 3D-Szenen bilden. Es werden ebenfalls die restlichen Features in ihrem Aufbau beschrieben, welche das Tool benötigt, um eine repräsentative 3D-Visualiserung eines OCT-Scans darzustellen. Um die Volumen der Ödeme und Drusen zu berechnen, wird sich einer Methode für die Berechnung von Meshes bedient. Diese Methode ist jedoch nicht auf sich überschneidende Meshes anwendbar. Aus diesem Grund werden sogenannte Mesh-Boolean-Operationen verwendet, um die Volumen sich überschneidender oder nur Teilweise innerhalb des fovealen Zylinders befindlicher Meshes zu berechnen.

3.2.1 Meshdarstellung von OCT-Objekten

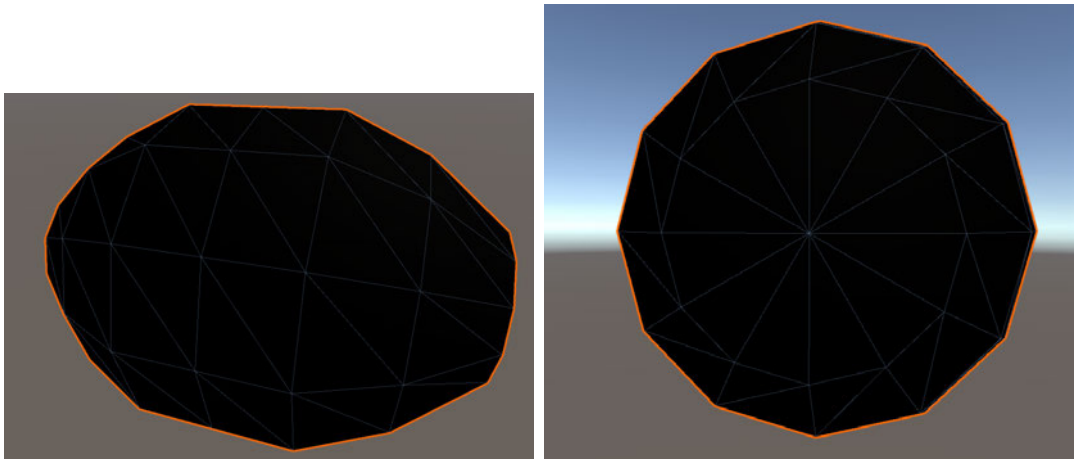

Abbildung 3.2: Ein generiertes Mesh eines Ödems, bestehend aus sieben verbundenen Kreisen. Seitlich (Links) und frontal(Rechts) gezeigt.

Wie in den Anforderungen beschrieben, sollen die Ödeme und Drusen für diese Arbeit in ihrer Generierung einer OCT-Scan basierten Form gleichen. Das Vorgehen zum Generieren dieser OCT-Scan ähnlichen Formen benötigt zuerst die Polygone, welche den Querschnitt des jeweiligen Objektes an der Position des Bilds des Scans zeigt. Diese Polygone weisen in den meisten Fällen eine annähernd runde Form auf. Ein Kreis aus Punkten soll deshalb die Ausgangsform für die Generierung dieser Formen darstellen. Diese Kreise werden mit einem vorgegebenen Radius, Anzahl von Punkten und Abstand voneinander platziert. Anschließend werden diese Kreise sequenziell mit Dreiecken verbunden, wobei zwei Dreiecke jeweils ein verbindendes Rechteck ergeben (siehe Abbildung 3.2). Um die Form zu vervollständigen, benötigt diese einen weiteren Punkt im Mittelpunkt des ersten sowie des letzten Kreises der Sequenz. Diese Kreise werden durch Dreiecke zwischen dem Mittelpunkt und jeweils zwei benachbarten Punkten des Kreises mit einer Fläche gefüllt.

3.2.2 Szenendarstellung von Ödem- & Drusen-Objekten

Um Ödeme und Drusen deutlich nach der Generierung mit randomisierten Parametern unterscheiden zu können, wird die Grundform dieser unterschiedlich angepasst.

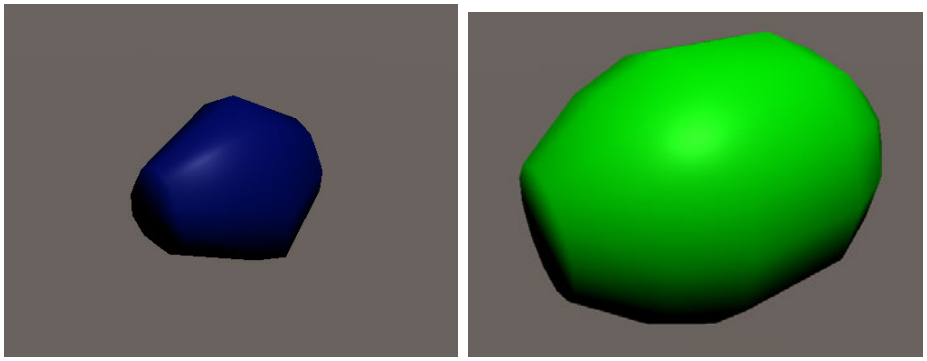

Abbildung 3.3: Das linke Bild zeigt die generierte Form einer Druse und die rechte die eines Ödems. Sie unterscheiden sich nicht nur in ihrer Größe und Färbung, sondern weisen auch unterschiedliche Formen auf.

Die Kreise der Ödeme werden so angepasst, dass ihre Radien kleiner werden, abhängig davon, wie viele Kreise zwischen ihnen und dem Kreis in der Mitte der Sequenz liegen. Hierfür wird eine quadratische Reduktion angewendet, um die Form gerundet erscheinen zu lassen. Ausgehend von den Ödemen ist das Ziel der Grundform der Drusen, klein und scharfkantig zu erscheinen. Die Kreise erhalten eine Anpassung, welche sie zu einer Dreieck-ähnlichen Form mit gerundeten Ecken annähert. Diese Form soll im Gegensatz zur Pyramiden-Form der vorhergegangenen Arbeit eine organische Erscheinung behalten und sich gleichzeitig von den Ödemen abheben. Die Kreise der Drusen werden ebenfalls in Nähe der Enden verkleinert, allerdings mit einer linearen Reduktion. Die Anzahl der Polygone sowie die gesamte Skalierung der Drusen werden im Gegensatz zu den Ödemen verringert, um den Größenunterschied zu diesen zu verdeutlichen.

3.2.3 Randomisierung der Objektform

Für die prozedurale Generierung der Objekte sollen Variationen von Ödemen und Drusen erstellt werden, welche deutlich erkennbare Unterschiede aufweisen, aber immer noch als das jeweilige Objekt erkennbar sind. Hierfür werden ausgehend von der Grundform der Objekte diverse Parameter angepasst, welche im Folgenden erläutert werden. Die Parameter der Kreise werden randomisiert, um einerseits den Radius der einzelnen Kreise anzupassen sowie die Position der Kreise parallel zu den anderen Kreisen innerhalb der Sequenz. Die Beschränkung auf parallele Positionsveränderung basiert auf dem Ursprungsgedanken, die Objekte ähnlich wie innerhalb

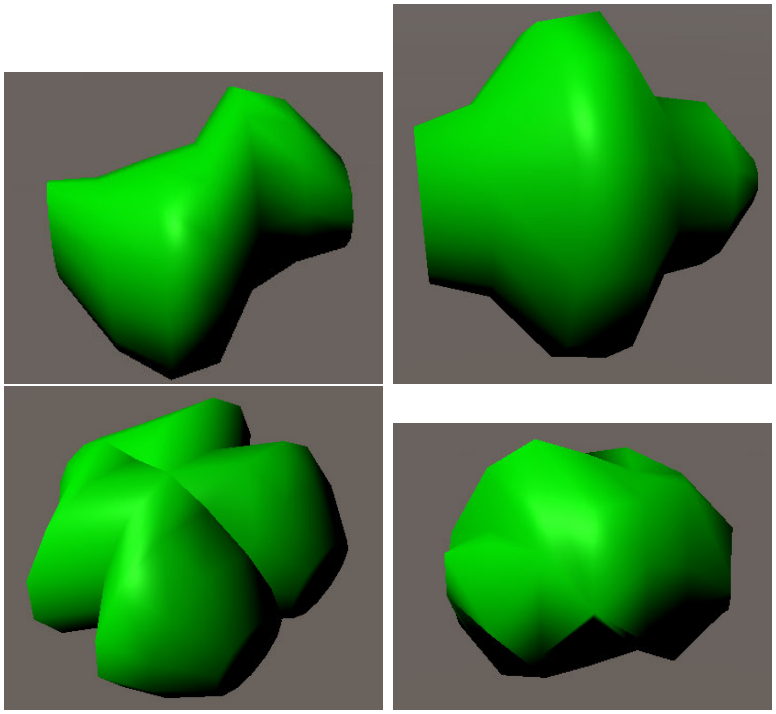

Abbildung 3.4: Die durch die Randomisierung der Parameter Meshgenerierung entstehenden Formen. Angefangen von Oben Links sind diese: die Position der verbundenen Kreise der Form, die Radien dieser Kreise, die Anzahl der parallelen Kreise und die Position der Vertices innerhalb der Kreise

eines OCT-Scans zu generieren, in welchem Fall die Polygone sich innerhalb der Bildfläche der einzelnen Aufnahmen des OCT-Scans befinden. Um die uniformen Kreise zu Polygonen basierend auf einer Kreisform zu ändern, wird die Position der einzelnen Punkte des Kreises ebenfalls zufällig innerhalb der Ebene des Kreises bestimmt. Diese Parameter sollen unabhängig voneinander so anpassbar sein, dass der Grad der Randomisierung durch eine Prozentzahl angegeben werden kann, welche die maximale Veränderung des Parameters festlegt, begrenzt auf hundert Prozent. Ein Wert von fünfzig Prozent würde in diesem Fall bedeuten, dass der Wert des Parameters bis zu fünfzig Prozent in positiver und negativer Richtung schwanken kann. Als letzter Parameter wird dem Objekt die Möglichkeit gegeben, mehrere Kreise innerhalb einer Bildebene zu platzieren. Diese Kreise werden mit allen Kreisen der nächsten und vorhergehenden Ebene verbunden. Auch für diesen Parameter kann eine Prozentzahl vorgegeben werden, in diesem Fall stehlt sie jedoch

die Wahrscheinlichkeit für zusätzliche Kreise innerhalb einer Ebene dar. Die hier aufgeführten Parameter sollen den Objekten nicht nur variierte Darstellungen geben, sondern diesen auch ein Aussehen geben, welches einer organischen Form näher liegt, da eine Form bestehend aus exakten Kreisen dem Objekt ein künstliches Aussehen verleihen würde.

3.2.4 Scenebuilder-Tool: Grundlegende Funktionen

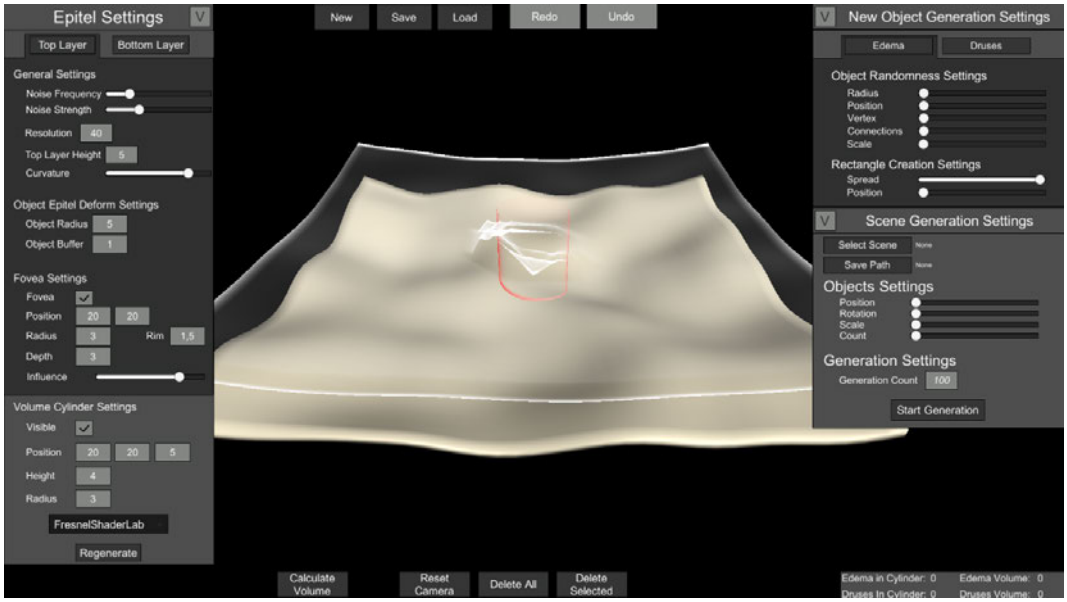

Abbildung 3.5: Die Bedienung des Scenebuilders erfolgt durch eine funktionale UI. Diese hat lediglich ein klares Labeling der Funktionen zum Ziel, nicht einen ästhetischen Mehrwert. Außerdem soll sie im Fall der Epithelschichten verdeutlichen, welche Schicht momentan zur Anpassung ausgewählt ist und welche Parameter für die jeweilige Schicht verändert werden können.

Die Objekt-Generierung soll innerhalb des Tools zur Erstellung von Szenen, (Scenebuilder) Anwendung finden. Die Level der Randomisierung der beschriebene Parameter sollen durch UI-Elemente anpassbar sein. Wenn ein neues Objekt platziert werden soll, wird ein Strahl mit der Position des Mauszeigers als Ursprung in Richtung der Kameraperspektive geschickt. Trifft dieser Strahl auf die untere Epithelschicht, wird ein Objekt an der Position der Kollision, innerhalb der gewählten Parameter,

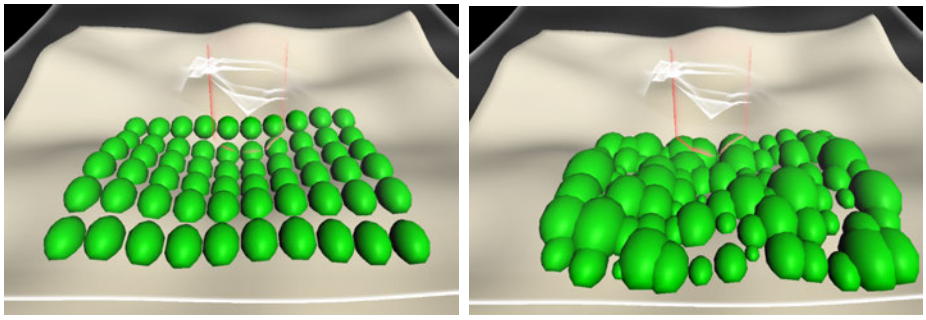

Abbildung 3.6: Mehrere Ödeme, welche durch die Rechteck-Generierung in einem Zug erstellt wurden. Links zeigt diese ohne Randomisierung und rechts mit Randomisierung der Position und Skalierung.

generiert. Da es in manchen Fällen wünschenswert sein könnte, die Objekte als eine Schicht zu generieren, ist es möglich, mehrere Objekte innerhalb eines Rechtecks zu generieren (siehe Abbildung 3.6). Um die Uniformheit Uniformität der Objekte innerhalb dieses Rechtecks zu reduzieren, gibt es auch für diese anpassbare Zufallsparameter. Diese Zufallsparameter beinhalten die Skalierung der Objekte zueinander sowie die Positionierung dieser, welche bestimmt, wie sehr die Anordnung der Objekte der Form des Rechtecks folgt. Die Objekte können auch nach ihrer Generierung angepasst werden, unabhängig davon, ob sie einzeln oder über die Rechteck-Generierung hinzugefügt wurden. Die gewünschten Objekte müssen zu diesem Zweck erst selektiert werden. Es können hierbei mehrere Objekte zur simultanen Anpassung selektiert werden. Die selektieren Objekte können in ihrer Translation, Rotation und Skalierung angepasst werden, mit jeweils korrespondierenden UI-Elementen. Zum Manipulieren der Objekte durch das Drücken und Halten der Maustaste muss die zweidimensionale Bewegung des Mauszeigers zu einem dreidimensionalen Vektor konvertiert werden. Diese Konvertierung ist trivial für Fälle, bei welchen die Kamera auf das Objekt gerichtet ist und so rotiert ist, dass ihr lokales Koordinatensystem dem globalen Koordinatensystem gleich ist. In einem solchen Fall kann für eine horizontale Bewegung des Objektes die horizontale Komponente der Mausbewegung verwendet werden. Eine Bewegung des Objektes in die Tiefe des Kamerabildes wird ebenfalls durch die horizontale Komponente der Mausbewegung gesteuert. Gleicht das Koordinaten-System der Kamera jedoch nicht dem globalen Koordinatensystem muss die Rotation der Kamera zum Objekt berücksichtigt werden. Abhängig von dieser Rotation nehmen die Komponenten der Mausbewegung Einfluss auf andere Bewegungsrichtungen. Außerdem müssen diese Bewegungen invertiert werden, sollte

die Kamera sich hinter oder unter dem Objekt befinden, und gleich bleiben, sollte sie sich hinter und unter dem Objekt befinden.

3.2.5 Scenebuilder-Tool: Epithelschicht-Generierung

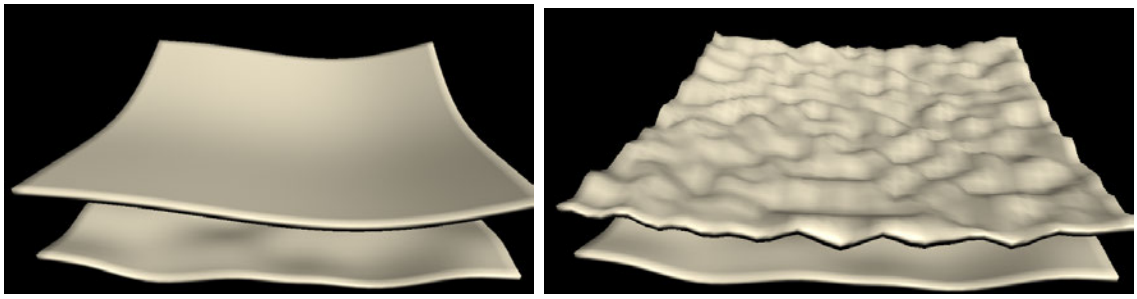

Abbildung 3.7: Links wurde die obere Epithelschicht mit Noise, einer geringen Frequenz aber große Stärke generiert. Rechts mit einer hohen Frequenz aber geringen Stärke

Die Meshgenerierung, welche die Ödeme und Drusen erstellt, wird in angepasster Form für die Generierung der Epithelschichten des Auges verwendet. Es werden zwei Schichten erstellt (Top- und Bottom-Schicht), welche sich in ihrer vertikalen Position unterscheiden. Im ersten Schritt wird hier eine quadratische Anordnung von Punkten generiert. Es wird eine Auflösung angegeben, um die Anzahl der zu generierenden Punkte pro Dimension des Quadrats anzugeben. Es wird außerdem die Länge der Seiten festgelegt, welche zusammen mit der Auflösung die Positionen der Punkte innerhalb der Quadratsfläche festlegt. Die Auflösung sowie die Länge sollten nicht verändert werden, sobald die ersten Szenen erstellt wurden, um konsistente Ergebnisse zu gewährleisten. Die Y-Koordinate der Punkte, welche in Unity für die vertikale Position eines Objektes verantwortlich ist, wird angepasst, um verschiedene Effekte auf dem Epithel zu erzeugen. Der erste dieser Effekte ist das Verwenden einer Noise-Funktion, um die Fläche des Epithels nicht durchgehend eben dazustellen, sondern Variationen in der Höhe der Fläche, wie sie auch in der Natur vorkommen würden, dazustellen. Dieser Noise soll abhängig von gegebenen X- und Y-Koordinaten die Höhe eines Punktes der Epithelschicht festlegen. Die als Input verwendeten Koordinaten kommen von der Position des Punktes. Um den Noise kontrollieren zu können, gibt es zwei Parameter: Die Noise-Frequenz und die Noise-Stärke. Ersteres sorgt bei großen Werten für rundere, große Wellen-ähnliche

Formen und scharfkantige, kleine Wellen-Formen bei geringeren Werten. Die Noise-Stärke ist ein Multiplikator, welcher auf den Rückgabewert der Noise-Funktion angewendet wird und die Amplitude der Wellen erhöht. Diese Parameter können für die Top- und Bottom-Schicht separat verändert werden. Die Höhe der Punkte wird außerdem von der Rundung des Auges beeinflusst, welche als Krümmungswert in die Epithelschichten eingebunden wird und deren tiefster Punkt die foveale Senke ist.

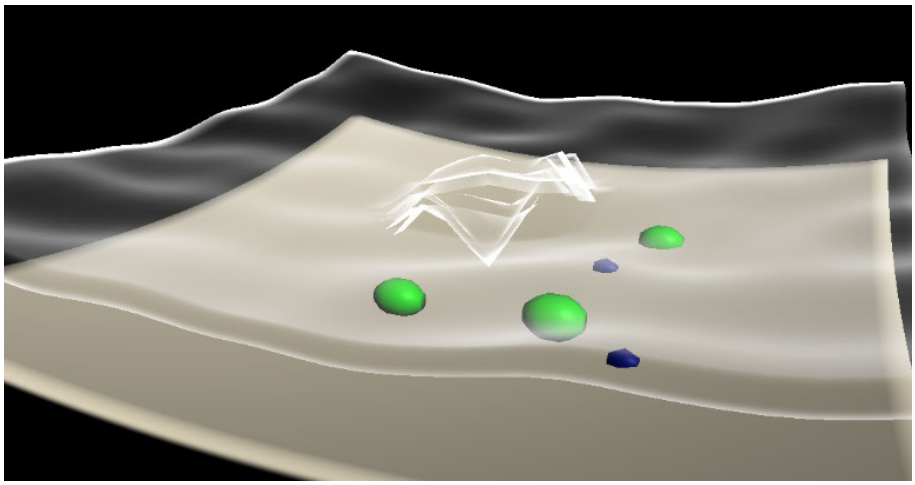

Abbildung 3.8: Die obere Epithelschicht wird mit einem Fresnel-Shader gerendert, welcher die Objekte unter dieser sichtbar belässt, aber gleichzeitig die Form der Schicht erkennen lässt.

Die foveale Senke stellt für das menschliche Auge den “Punkt des schärfsten Sehens“[RR10] dar und ist in einem OCT-Scan erkennbar durch eine nach unten zeigende ”Trichter”-Form. Diese Form wird in der Top-Schicht durch einen zur Bottom-Schicht zeigenden Kegel repräsentiert. Die Tiefe und Radius des Kegels sowie die Breite des den Kegel umgebenden Randes können angepasst werden. Die Form des Randes wird ebenfalls durch die Noise-Funktion beeinflusst, der Höhenunterschied, welcher den Rand von der umliegenden Schicht abhebt, wird durch einen “Einfluss“-Parameter bestimmt. Je größer dieser Wert ist, desto ausschlaggebender wird der Stärke-Parameter des Noises für die Bereiche der Schicht außerhalb des Randes verringert.

Alle hier genannten Parameter lösen mit ihrer Änderung ein Löschen der aktuellen Epithel-Instanz aus, gefolgt von einem erneuten Generieren des Epithels mit den geänderten Parametern. Der Ausschnitt der Noise-Funktion wird hierbei nicht erneut randomisiert. Die Darstellung der Top-Schicht kann ebenfalls verändert werden, um

die Objekte darunter sichtbar zu machen und das Anpassen dieser zu erleichtern. Hierbei wird ein sogenannter Fresnel-Effekt verwendet, dessen Funktionsweise im Implementierungskapitel ausgeführt wird. Dieser erlaubt den in Abbildung 3.8 zu sehenden Effekt, welcher es erlaubt ein Objekt unter der Schicht unbeeinträchtigt zu erkennen und gleichzeitig eine Approximation der Form der Schicht zu erhalten. Die Bottom-Schicht verwendet eine undurchsichtige Erscheinung, welche die Form der Schicht selbst klar erkennen lässt. Diese Erscheinung kann ebenfalls für die Top-Schicht ausgewählt werden, zum Beispiel wenn Parameter wie die Frequenz des Noise angepasst werden. Um unterschiedliche Konfigurationen des Noise auszuprobieren können, ist es möglich, die Schichten voneinander unabhängig, ohne Anpassen der Einstellungen, erneut generieren zu lassen. Hierbei wird ein anderer Abschnitt der Noise-Funktion als Basis der Unebenheiten verwendet.

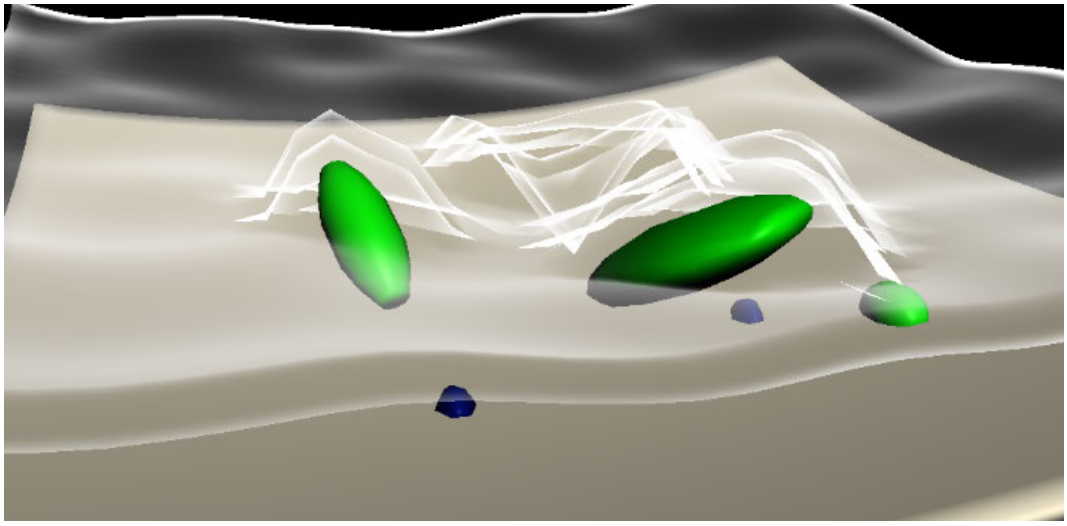

Abbildung 3.9: Die obere und untere Epithelschicht zusammen mit mehreren Ödemen, welche die obere Schicht durch ihre Nähe deformieren.

Einer der deutlichsten Effekte der Objekte ist das Wölben der Epithelschicht, welche sich über diesen befindet. Besonders im Bereich der Fovea kann dies zu, in einem OCT-Scan deutlich erkennbaren, Verformungen führen. Um diesen Effekt im SceneBuilder zu realisieren, wird, nachdem die Auswirkung von Krümmung und Noise auf die Punkte der Top-Schicht festgelegt sind, für jeden Punkt der Schicht ein Strahl geschickt. Trifft dieser Strahl auf ein Objekt, mit welchem Kollisionen erlaubt sind, wird das Programm über das Vorkommen und die Position der Kollision informiert. Im Falle der von der Top-Schicht ausgehenden Strahlen sind die Punkte der Schicht die Ursprungspunkte der Strahlen und ein nach unten gerichteter Vektor deren

Richtung. Kollisionen sind in diesem Fall nur mit Ödemen und Drusen erlaubt, nicht aber mit der Bottom-Schicht. Es wird außerdem ein Buffer-Wert festgelegt, welcher den Abstand zwischen der Schicht und dem Objekt festlegt, welches für die Wölbung dieser verantwortlich ist. Sollte der Punkt der Kollision eines Raycast zusammen mit dem festgelegten Buffer nun über dem Punkt der Schicht liegen, wird dieser um die Differenz erhöht. Um den Übergang von erhöhten und nicht erhöhten Punkten der Schicht natürlicher erscheinen zu lassen, wird ein Radius festgelegt, in welchem Punkte um einen erhöhten Punkt ebenfalls erhöht werden. Diese zusätzliche Erhöhung nimmt mit zunehmender Distanz zum erhöhten Punkt ab.

3.2.6 Scenebuilder-Tool: Volumenberechnung und Fovealer Zylinder

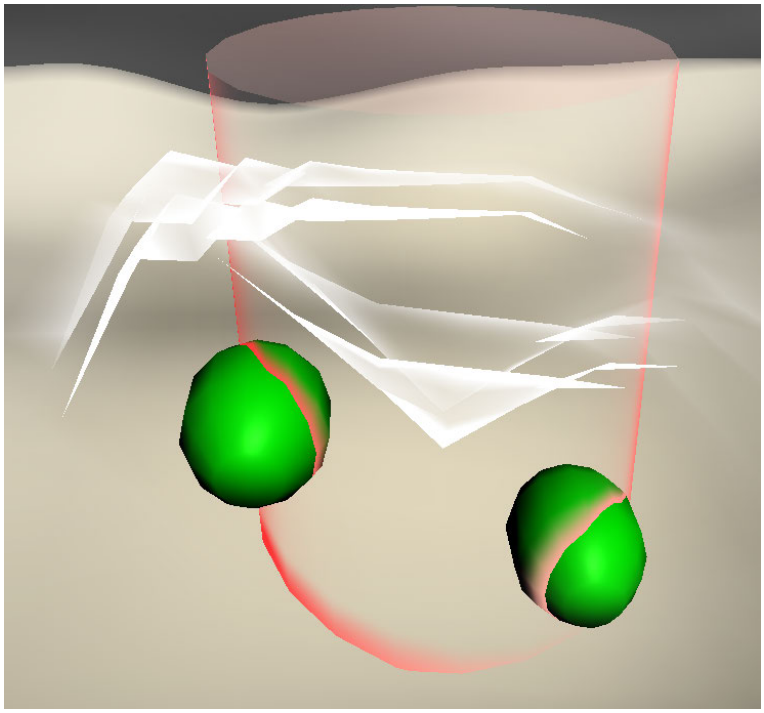

Abbildung 3.10: Der foveale Zylinder, welcher von der verborgenen Domänenregel des Klassifikationsproblems verwendet wird. Die Schnittstellen mit den Ödemen werden hervorgehoben

Wie im Unterkapitel Anforderungen ausgelegt, sollen die Szenen mit einer verborgenen Domänenregel getestet werden, hierfür wird ein Zylinder zentriert auf der

fovealen Senke platziert. Die Domänenregel benutzt die Summe der Volumen von Objekten innerhalb des Zylinders, um eine gute oder beeinträchtigte Sehfähigkeit nachzustellen. Der Zylinder bleibt dem MVCNN dabei unbekannt. Der Zylinder ist an der Position der Fovea festgemacht. Die Dimensionen des Zylinders können angepasst werden für unterschiedliche Fovea-Konfigurationen. Wie das gesamte Volumen der Objekte und der Anteil dieser innerhalb des Zylinders berechnet wird, erläutert der folgende Abschnitt.

Da die verborgene Domänenregel die Volumen der Objekte verwenden soll, ist es notwendig, diese zuverlässig zu bestimmen. In herkömmlichen Fällen kann dies durch die Methode nach Zhang und Chen [ZC01] bestimmt werden. Eine Herausforderung stellen Fälle da, in welchen die Objekte sich überschneiden oder den fovealen Zylinder schneiden. In Fall von überschneidenden Objekten ist es zwar möglich, die Objekte zu verschieben, sodass diese sich nicht mehr scheiden, allerdings führt dies dazu, dass die möglichen Objektpositionen stark eingeschränkt werden. Außerdem sind diese überschneiden Formen ebenfalls in der Realität zu finden und erhöhen damit die Belastbarkeit der Ergebnisse der Tests. Noch stärker wäre die Positionseinschränkung bei den Schnittstellen zwischen Objekten und dem fovealen Zylinder, da eine Vermeidung solcher den Nutzer zwingen würde, Objekte vollständig innerhalb oder außerhalb des Zylinders zu platzieren. Dies würde nicht nur die Möglichkeit des Nutzers einschränken, Objekte so zu platzieren, wie sie innerhalb eines OCT-Scans erscheinen und den fovealen Zylinder anschließend frei zu skalieren. Es würde ebenfalls das Klassifikationsproblem ungewollt simplifizieren, da die Gruppierung der Objekte einfacher erkennbar erscheinen würde.

Diese Limitierung stammen von Zhang und Chens Methode wie in 2.2 angesprochen. In dieser Arbeit wird ein Algorithmus vorgeschlagen, diese Limitierung zu umgehen. Es ist dem Autor des Papers zum Zeitpunkt des Schreibens von keiner existierenden Methode hierfür bekannt. Deshalb wird sie für diese Arbeit eigens entwickelt. Die Grundidee des vorgeschlagenen Algorithmus besteht darin die sich schneidenden Meshes zu einem einzigen Mesh zu kombinieren, bevor Zhang und Chens Methode angewendet wird. Zu diesem Zweck werden sogenannte Mesh-Boolean-Operationen verwendet, um erst die überschneiden Objekte zu einzelnen Objekten zu kombinieren und diese kombinierten Objekte mit dem Zylinder zu schneiden. Während bereits Implementierung dieser Operationen für Unity existieren sind diese entweder nicht ausreichend performant oder haben eine zu hohe Fehlschlagsquote. Deshalb wird die in dieser Arbeit verwendete Implementierung der Mesh-Boolean-Operationen

basierend auf Jiang et al.s Methode erstellt[JPC+16], welche eine der aktuellesten Arbeiten für Mesh-Boolean-Operationen darstellt.

Der Algorithmus zur Volumenbestimmung geht wie folgt vor: Um zu bestimmen, welche Objekte sich überschneiden, werden beim Erstellen und Manipulieren dieser Objekte Kollisionen mit anderen Objekten verfolgt. Um die überschneidenden Objekte zu finden, werden die Objekte rekursiv durchlaufen und die miteinander kollidierenden Objekte in Listen gruppiert. Die ersten zwei Elemente dieser Listen werden zu einem neuen Objekt vereinigt, mit einer Union Mesh-Boolean-Operation. Diese Operation verbindet die Meshes der Objekte zu einem einzigen Mesh ohne die sich überschneidenden Elemente der Originale. Anschließend wird über jede dieser Listen iteriert und jedes Element dem kombinierten Objekt mit einer weiteren Union-Operation hinzugefügt. Die Volumen dieser kombinierten Objekte können anschließend mit der oben beschriebenen Methode bestimmt werden und ergeben zusammen jeweils das Gesamtvolumen für Drusen und Ödeme. Um den Teil des Volumens der Objekte, welcher sich innerhalb des Zylinders befindet, zu bestimmen, wird für jedes kombinierte Objekt überprüft, ob eines der originalen Objekte mit dem fovealen Zylinder kollidiert. Sollte dies nicht der Fall sein, wird das Volumen des kombinierten Objekts als außerhalb des Zylinders angenommen. Sollte eines der Objekte kollidieren, wird eine Mesh-Boolean-Intersection-Operation auf das Objekt und den Zylinder angewendet. Diese Mesh-Boolean Operation kombiniert im Gegensatz zu der Union-Operation ausschließlich die überschneidenden Bereiche der Meshes zu einem neuen Mesh. Das Volumen dieses neuen Meshes ist der innerhalb des Zylinders befindliche Anteil des Volumens des kombinierten Objekts. Sollte die Intersection-Operation fehlschlagen, liegt das kombinierte Objekt vollständig innerhalb des Zylinders und das gesamte Volumen des Objekts wird zum im Zylinder befindlichen Anteil gezählt.

3.2.7 Scenebuilder-Tool: Daten-Persistenz

Entscheidend für das Verwenden der Daten ist das Speichern und Laden der Ergebnisse der Szenen-Erstellung. Zu Anfang des Speicherns werden die Volumen der Objekte und der Anteil des Volumens innerhalb des Zylinders berechnet, um zusammen mit dem Rest der Daten festgehalten zu werden. Für das Speichern werden für jedes Objekt die Daten seiner Generierung gespeichert. Dies beinhaltet grundlegende Informationen wie die Position und Rotation eines Objektes und ob es sich um

eine Druse oder ein Ödem handelt, aber auch spezifische Daten der prozeduralen Generierung wie der Radius der einzelnen Kreise des Meshes werden festgehalten. Die Parameter der Epithelschichten werden ebenfalls gespeichert sowie die Daten des fovealen Zylinders. Bevor die Daten geladen werden, werden alle bereits in der Szene vorhandenen Objekte gelöscht sowie die Epithelschichten. Um die Daten zu laden, werden die Objekte wie beim erstmaligen Erstellen generiert, mit dem Unterschied, dass in diesem Fall die Parameter der Generierung konkret vorgegeben sind, anstatt diese innerhalb vorgegebener Zufallsbereiche zu bestimmen.

3.2.8 Scenebuilder-Tool: Undo-/Redo-System

Um die Benutzbarkeit des Programms zu verbessern, soll ein in den meisten Programmen zum Erstellen von Inhalten übliches Undo/Redo-System hinzugefügt werden (siehe Anwendungen: Word, Photoshop, Unity). Dieses System soll es erlauben, die letzte signifikante Anpassung rückgängig zu machen bzw. rückgängig gemachte Anpassungen zu wiederholen. Als signifikante Anpassungen werden hier solche angesehen, die den Inhalt der Szene verändern. Dazu gehören: Das Hinzufügen und Entfernen von Objekten, das Manipulieren dieser Objekte, das Ändern aller Settings der Epithel-Generierung unter Ausnahme des Materials der Top-Schicht sowie das ändern der Zylinder-Settings unter Ausnahme der Sichtbarkeit des Zylinders. Ebenfalls nicht Teil des Systems ist das Ändern der Einstellung der Objektgenerierung sowie das Selektieren von Objekten. Diese Änderungen haben keinen Effekt auf den aktuellen Zustand der Szene, sondern sind lediglich die Grundlage von späteren signifikanten Änderungen.

3.2.9 Scenebuilder-Tool: Prozedurale Szenengenerierung

Nachdem das intellektuelle Erstellen einer Szene abgeschlossen ist und diese gespeichert wurde, soll dieser Speicherstand verwendet werden, um weitere Szenen zu generieren. Hierfür muss zuerst der Speicherstand einer Szene ausgewählt werden. Die Ausgangsszene wird abhängig von der gewünschten Anzahl an generierten Szenen wiederholt geladen. Die Position, Rotation und Skalierung der Objekte der Szene werden basierend auf einer Prozentzahl randomisiert. Null Prozent bedeuten hier, dass der jeweilige Wert der generierten Szene genau dem Original entspricht und hundert, dass der Wert gänzlich unabhängig von der Ausgangsszene zufällig

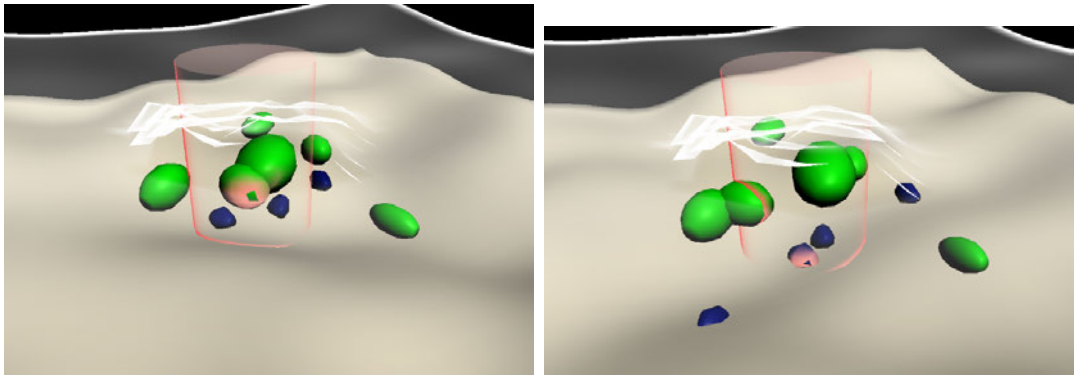

Abbildung 3.11: Links ist eine intellektuell erstellte Szene zu sehen. Basierend auf dieser Szene wurde die rechts zu sehende Szene prozedural generiert, mit Randomisierung der Position, Rotation, Skalierung und Anzahl der Objekte.

bestimmt wird. Zusätzlich gibt es einen Parameter für die Anzahl der Objekte. Wenn dieser auf null gesetzt wird, wird die Szene mit der Anzahl der Objekte innerhalb der Ausgangsszene generiert. Bei Werten über null werden für jedes Objekt der Ausgangsszene zwischen null und dem gesetzten Wert an Objekten generiert. Die zusätzlich erstellten Objekte haben dieselben Ausgangswerte in Position, Rotation und Skalierung wie ihre Originale kombiniert mit einem für jedes Objekt generierten Zufallswert innerhalb der gesetzten Parameter. Für die Epithelschichten wird mit dem Laden lediglich deren Noise neu generiert, indem ein zufälliger Abschnitt des Perlinnoise gewählt wird.

Die beschriebene Szenengenerierung hat den Nachteil, dass die Positionsbeziehungen innerhalb von Objekt-Gruppen (zum Beispiel einer Kette von Ödemen) durch die Randomisierung der Objekt-Positionen unerkenntlich werden. Aus diesem Grund wird ein Feature erstellt, welches solche Beziehungen selbst generiert. Dieses Feature bietet die Möglichkeit, mit der Szenengenerierung nicht nur die Objekte der Ausgangsszene mit neuen Parametern zu generieren, sondern diese mit vorgegebenen Mustern von Objekten zu ersetzten. Diese Muster sind:

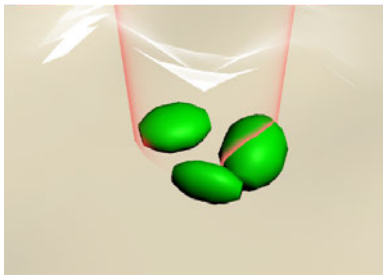

Abbildung 3.12: Die Abbildung zeigt eine prozedural generierte OCT-Scene, in welcher ein Ödem-Objekt mit dem Spread Muster generiert wurde und dadurch in drei dicht beieinanderliegenden Ödem-Objekten resultierte.

Spread Dies entspricht der originalen Generierung, für jedes Objekt werden eine oder mehrere Kopien dieses Objektes mit randomisierter Position, Rotation und Skalierung generiert.

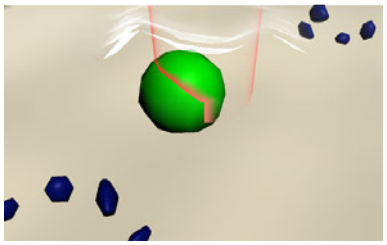

Abbildung 3.13: Die Abbildung zeigt eine prozedural generierte OCT-Scene, in welcher ein Ödem-Objekt mit dem Single Muster generiert wurde und dadurch in dem zentralen großen Ödem-Objekt resultiert.

Single Ein einziges, überdurchschnittlich großes Objekt wird generiert.

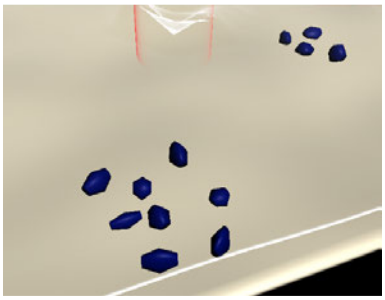

Abbildung 3.14: Die Abbildung zeigt eine prozedural generierte OCT-Scene, in welcher zwei Drusen-Objekte mit dem Field Muster generiert wurden und dadurch in 9 und 4 Drusen-Objekten resultierten. Dies Objekte sind in einem losen Gitter angeordnet

Field In diesem Fall werden die Objekte innerhalb eines Quadrates generiert, ähnlich der Rechteckgenerierung des Scenebuilders. Die Position der Objekte innerhalb des Quadrates wird ebenfalls randomisiert, um eine zu gleichmäßige und künstlich erscheinende Anordnung zu vermeiden.

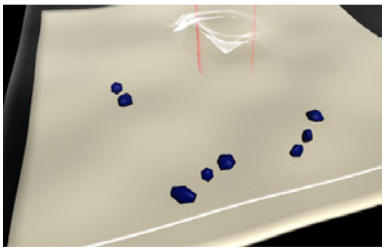

Abbildung 3.15: Die Abbildung zeigt eine prozedural generierte OCT-Scene, in welcher drei Drusen-Objekte mit dem Chain Muster generiert wurden und dadurch in 2 und 3 Drusen-Objekten resultierten. Diese Objekte sind in Ketten angeordnet

Chain Das erste Objekt wird zufällig platziert, wie auch mit der Spread-Methode. Anschließend wird eine zufällige Anzahl von weiteren Objekten hinzugefügt. Jedes dieser Objekte muss innerhalb eines maximalen Radius zum vorherigen Objekt generiert werden.

Ziel dieser Muster ist, Objekt-Konfigurationen und -Mengen, welche in OCT-Scans mit erhöhter Wahrscheinlichkeit vorkommen, zu verwenden. Außerdem sollen sie die Option geben, eine Szenengenerierung weniger abhängig von den Ausgangsszenen und stattdessen abhängig von kontrollierter Zufälligkeit zu verwenden.

3.2.10 Mesh-Boolean-Operationen

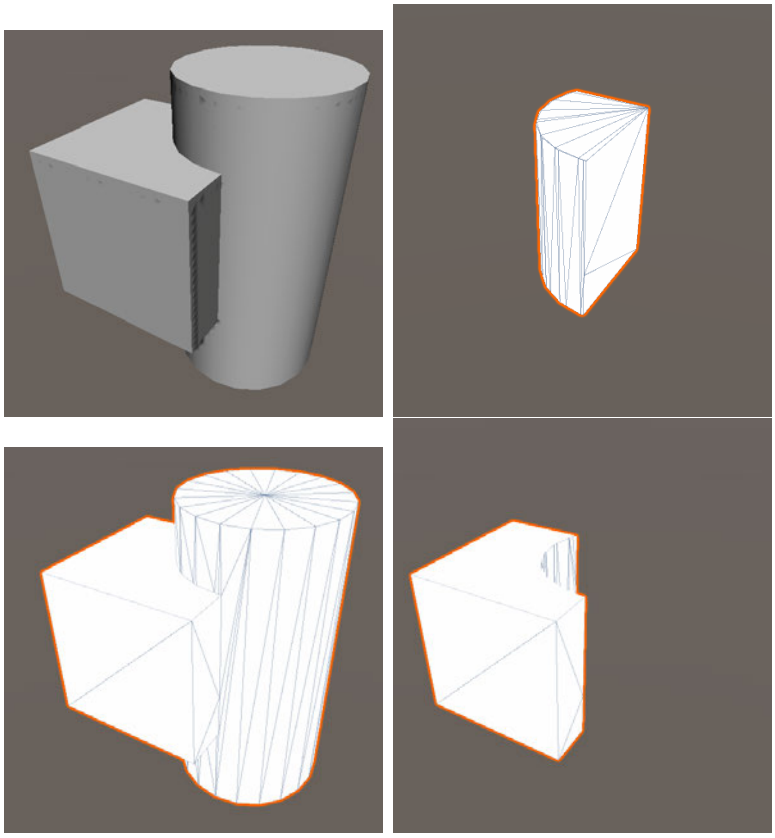

Abbildung 3.16: Links Oben sind die zwei Ausgangsformen einer Mesh-Boolean-Operation zu sehen. Es werden folgende Operationen auf diese angewandt: Rechts Oben: Intersection, Links Unten: Union, rechts Unten: Difference

Mesh-Boolean-Operationen dienen dazu, aus bereits vorhandenen Meshes durch Kombination ein neues Mesh zu erstellen, dieser Prozess nennt sich ”Constructive Solid Geometry”. Die in dieser Arbeit beschriebene Methode für Mesh-Boolean-Operationen basiert vor allem auf Jiang et al. [JPC+16] und stellt eine der aktuellsten Methoden dar. Laidlaw et al.[LTH86] sowie Mei und Tipper[MT13] werden zusätzlich verwendet, für stellen, an welchen die Implementierung von Jiang et al. nicht all umfassend beschrieben ist. Um zwei Objekte zu kombinieren, müssen die Dreiecke der Meshes sich an mindestens einer Position überschneiden. Das Kombinieren der Meshes kann in drei Formen angewendet werden: Union, Difference und Intersection. Union bedeutet, dass die sich überschneidenden Bereiche der Meshes entfernt

werden und sie entlang der Kanten der geschnittene Dreiecke verbunden werden. Difference bedeutet, dass der nicht überschneidende Bereich des ersten Meshes und der überschneidende Bereich des zweiten Meshes kombiniert werden, um einen Effekt zu erzielen ähnlich einem "Ausschneiden" des zweiten Meshes aus der Form des Ersten. Die Intersection Operation hat zum Effekt, dass nur die überschneidenden Bereiche beider Meshes kombiniert werden, um die Schnittstelle selbst darzustellen. Die Schritte bis zum Kombinieren der gewünschten Mesh-Bereiche selbst sind für alle Operationen dieselben, weshalb das Implementieren einer Operation das Implementieren aller drei trivial gestaltet. Der Prozess der Operationen besteht aus folgenden Schritten, welche innerhalb dieses Kapitels erläutert werden: Die Vorauswahl der Dreiecke, welche sich schneiden könnten, der Test aller ausgewählten Dreiecke auf einen Schnitt zwischen diesen, das Trennen der geschnittenen Dreiecke in Polygone, das Triangulieren dieser Polygone, das Einteilen der neuen und nicht geschnittenen Dreiecke in außerhalb und innerhalb des Meshes.

3.2.11 Mesh-Boolean-Operationen: Schnittstellen-Vorauswahl

Die Mesh-Boolean Operation muss die Schnittlinien der beiden Meshes finden. In einem naiven Ansatz würde dies das Überprüfen jedes Dreiecks des ersten Meshes mit jedem Dreieck des zweiten benötigen. Dies führt zu einer quadratischen Laufzeit und sollte nach Möglichkeit reduziert werden. Einer der Ansätze hierfür ist ein Shared-Octtree [JPC+16]. Ein Octtree teilt die Vertices eines Meshes in kleiner werdende Würfel ein, welche einer Baum-Hierarchie unterliegen. Die Größe des Würfels, welcher die "Wurzel" der Baum-Hierarchie darstellt, wird anhand der geringsten und der maximalen Vertex-Koordinaten aller Achsen bestimmt. Dieser Würfel wird in acht Würfel gleicher Größe unterteilt. Jeder dieser Würfel stellt einen der "Äste" innerhalb der Baum-Hierarchie unter dem ersten Würfel dar. Für jeden dieser Würfel wird überprüft, welche Vertices der Meshes sich innerhalb des Würfels befinden. Ob ein Punkt sich innerhalb eines Quaders befindet, kann überprüft werden, indem die einzelnen Koordinaten des Punktes mit denen der kleinsten und größten Ecke des Quaders verglichen werden. Sollten alle Koordinaten der kleinsten Ecke geringer als die des Punktes und alle der größten Ecke größer als die Koordinaten des Punktes sein, liegt dieser innerhalb des Würfels. Durch die Vertices kann bestimmt werden, welche Dreiecke des Meshes sich in den Würfeln befinden. Sollte die Anzahl der Dreiecke innerhalb eines Würfels über einem zu Beginn des Prozesses festgelegten

Schwellwert liegen, wird dieser Würfel ebenfalls in acht kleinere Würfel unterteilt. Wie oft diese Würfel unterteilt werden, hängt von der Größe der maximalen Anzahl von Dreiecken ab, sollte aber zusätzliche von einem weiteren Schwellwert beschränkt werden, welcher eine maximale Anzahl von Unterteilungen festlegt. Die letzten Würfel des Octtrees, welche nicht unterteilt werden und die "Blätter" der Baum-Hierarchie darstellen, enthalten die Informationen über die in ihnen befindlichen Dreiecke. Für einen Shared-Octtree wird für beide Meshes der Operation ein Octtree generiert. Das Vergleichen aller Dreiecke des ersten Octtrees mit denen des Zweiten wird ersetzt durch ein Suchen nach Schnittstellen zwischen den Blättern der jeweiligen Octtrees. Anschließend müssen lediglich die Dreiecke innerhalb der sich überschneidenden Würfel auf ein gegenseitiges Schneiden untersucht werden. Der erhöhte anfängliche Berechnungsaufwand, welcher durch das Generieren der Octtrees und das Suchen nach Würfelüberschneidungen entsteht, wird durch die Reduktion von zu überprüfenden Dreiecksschnittstellen in den meisten Fällen überwogen.

3.2.12 Mesh-Boolean-Operationen: Schnittstellen-Test

Nachdem festgestellt wurde, welche Paare von Dreiecken auf eine Schnittstelle untersucht werden sollen, beginnen die Schnittstellen-Tests. In dieser Arbeit wird die Methode nach Held [Hel97, S.3ff] verwendet. Das Feststellen einer Schnittstelle und Bestimmen deren Position und Typ geschieht in mehreren Schritten, welche das Ziel haben, simple Ausschlussfälle vor der eigentlichen Schnittstellen-Suche aussortieren zu können (siehe Abbildung 3.18). Der Erste definiert die Ebene, in welcher sich die Punkte des ersten Dreiecks befinden, und überprüft die Position der Punkte des zweiten Dreiecks relativ zu dieser. Im ersten und häufigsten Ausschlussfall liegen alle Punkte des zweiten Dreiecks auf einer Seite der Ebene des ersten Dreiecks. Dies macht eine Schnittstelle zwischen den Dreiecken unmöglich und führt zum Abbruch der Tests. Die nächste Option ist, dass einer der Punkte auf einer Seite der Ebene ist und die verbleibenden zwei Punkte auf der anderen Seite. Dies bedeutet, dass eine Schnittstelle möglicherweise vorliegt. Es ist allerdings trotzdem möglich, dass das zweite Dreieck nur neben dem Ersten liegt und damit seine Fläche schneidet. Die verbleibenden möglichen Ergebnisse des Tests stellen Grenzfälle der Schnittstellensuche dar und treten auf wenn einer oder mehrere der Punkte innerhalb der Ebene liegen. Sollte dies für nur einen Punkt des Dreiecks der Fall sein, ist es möglich, dass nur eine der Spitzen des zweiten Dreiecks das erste schneidet. Sollten zwei Punkte in

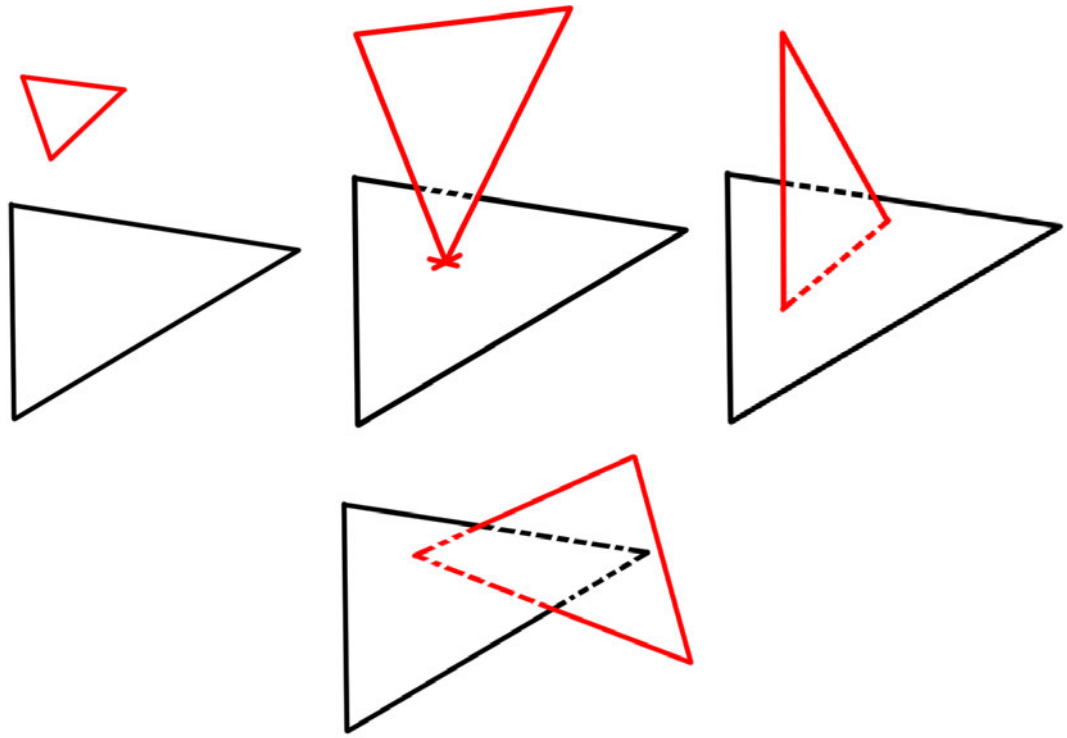

Abbildung 3.17: Die vier Fälle, welche vor dem eigentlichen Schnittstellen-Test aussortiert werden. Von oben links: Alle Punkte des Dreiecks befinden sich auf einer Seite der Ebene des anderen Dreiecks, nur ein Punkt befindet sich in der Ebene des anderen Dreiecks, nur eine Kante befindet sich in der Ebene des Dreiecks, alle Punkte liegen in derselben Ebenen und die Dreiecke sind somit koplanar.

der Fläche liegen ist es möglich das eine der Kanten des zweiten Dreiecks das erste schneidet. Falls alle Punkte des zweiten Dreiecks in der Fläche des ersten liegen, sind die Dreiecke koplanar, also liegen in derselben Ebene. Diese Grenzfälle sind, da die Meshes eigens generiert werden, innerhalb dieser Arbeit, absehbar und aufgrund der randomisierten Form aller Meshes extrem unwahrscheinlich. Sollte einer dieser Grenzfälle auftreten, wird dies durch das System erfasst und deren tatsächliche Häufigkeit kann überprüft werden. Dieser Test wird wiederholt mit dem ersten Dreieck als Ebene und dem zweiten als zu überprüfendes Element. Der einzige Test, welcher nicht wiederholt werden muss, ist der Test auf Koplanarität, da dessen Ergebnis in beiden Fällen äquivalent sein muss. Sollten beide Tests das Ergebnis haben, dass

```
SET Dreieck1 = Liste mit Vertices des ersten Dreiecks
SET Dreieck2 = Liste mit Vertices des zweiten Dreiecks
SET Plane1Normal = Normalenvektor von Dreieck1
SET Plane2Normal = Normalenvektor von Dreieck2

FOREACH Vertex in Dreieck2
    GET Distanz von Vertex zu Ebene von Dreieck1 mit Plane1Normal Ebenenformel

IF Alle Distanzen Null
    Dreiecke sind Coplanar RETURN
ELSE IF Zwei Distanzen Null
    Kannte von Dreieck2 liegt in Ebene von Dreieck1 RETURN
ELSE IF Eine Distanz Null
    Ecke von Dreieck2 liegt in Ebene von Dreieck1 RETURN
IF Alle Distanzen größer Null oder alle Distanzen kleiner Null
    Vertices liegen auf derselben Seite von Dreieck2 RETURN

Test wird wiederholt mir Dreieck1 und Plane2Normal
```

Abbildung 3.18: Der Pseudocode für das Ausschließen von Dreiecken für eine mögliche Schnittstelle.

die Dreiecke die Ebene des jeweils anderen schneiden, werden die Schnitt-Linien der Dreiecke mit den jeweils anderen Ebenen berechnet.

Der Ablauf zum Berechnen der Schnittlinien ist in diesem Fall gleich für beide Dreiecke. Es wird zuerst bestimmt, welcher der Punkte des Dreiecks auf der anderen Seite der Ebene des zweiten Dreiecks liegt. Anschließend werden die Linien von diesem Punkt zu den verbleibenden Punkten des Dreiecks bestimmt. Die Schnittpunkte dieser Linien mit der Ebene des zweiten Dreiecks definieren die Linie, in welcher das Dreieck die Ebene schneidet. Dieser Prozess wird für das andere Dreieck und die erste Ebene wiederholt. Die entstehenden Linien liegen auf derselben Gerade. Die Beziehung der Linien zueinander gibt das Ergebnis der Schnittstellensuche an. Die möglichen Ergebnisse lauten wie folgt: Fall: ”die Linien überschneiden sich nicht und es besteht Abstand zwischen ihnen”. Dies bedeutet, dass die Dreiecke sich nicht schneiden. Fall: ”Die Linien überschneiden sich in nur einem Punkt”. Die Kanten der Dreiecke liegen aufeinander. Ob dies als eine Schnittstelle gezählt wird, ist implementierungsabhängig. Innerhalb dieser Arbeit wird ein solches Ergebnis als keine Schnittstelle gezählt. Fall: ”Die Linien überschneiden sich teilweise”. In diesem Fall schneiden sich die Dreiecke an einer ihrer Kanten. Fall: ”Eine Linie ist vollständig von der anderen Linie umfasst”. In diesem Fall schneidet eines der Dreiecke nur die Fläche des anderen Dreiecks, ohne dessen Kanten zu berühren. Fall: ”Die Linien sind äquivalent”. Dies bedeutet, dass die Dreiecke sich so schneiden, dass ihre Kanten sich auf zwei Seiten schneiden. Sollte dieser Test einen Schnitt feststellen,

wird dieser mit dem jeweiligen Dreieck festgehalten, zusammen mit der Information, ob dieser eine der Kanten des Dreiecks schneidet. Sollte dies der Fall sein, werden die Indizes der Punkte, zwischen welchen die Kante liegt, ebenfalls festgehalten. Ein zu berücksichtigender Fall ist, wenn der Schnitt eines Dreiecks den des anderen umschließt, da in einem solchen Fall der Schnitt des umfassten Dreiecks keine Kante schneidet und der des umfassenden zwei.

3.2.13 Mesh-Boolean-Operationen: Verarbeitung von Schnittstellen

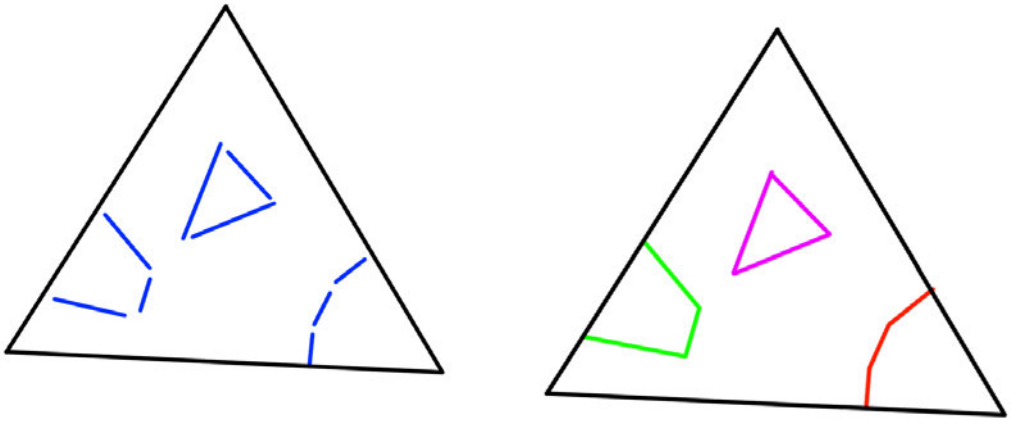

Abbildung 3.19: Links zeigt ein Dreieck, welches mehrere Male von anderen Dreiecken geschnitten wurde, wobei jedes blaue Linien-Segment einen Schnitt darstellt. Rechts zeigt, wie diese Schnitte zu Ketten verbunden werden, was abhängig davon ist, ob sie zwei, eine oder keine Kante berühren und unterschiedlich gelabelt werden.

Nachdem die Schnitte für jedes Dreieck erfasst wurden, wird jedes der Dreiecke basierend auf den Schnitten durch dieses in Polygone unterteilt. Der erste Schritt, um die Polygone zu unterteilen, ist, die einzelnen Schnitte zu vollständigen Ketten zusammenzufügen (Abbildung 3.19), um die Trennungen deutlich zu machen. Hierzu wird die Liste aller Schnitte iterativ durchlaufen (siehe Abbildung 3.20). Der erste vorhandene Schnitt, mit einem Endpunkt, welcher auf einer Kante liegt, wird ausgewählt. Der Endpunkt des Schnitts, welcher nicht auf einer Kante liegt, wird

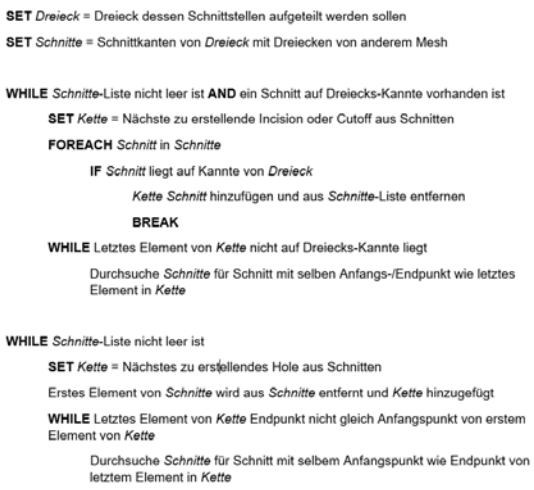

Abbildung 3.20: Der Pseudocode für das Aufteilen der Schnitte in Incisions, Cutoffs und Holes.

mit den Endpunkten der anderen Schnitte verglichen, und überprüft, ob diese auf derselben Position liegen. Wird ein solcher Endpunkt gefunden, wird der Schnitt der aktuellen Kette hinzugefügt und der nicht verbundene Endpunkt des neuen Schnittes wird weiter verglichen. Dieser Prozess wird wiederholt bis ein Schnitt gefunden wird dessen anderer Endpunkt ebenfalls auf einer Kante des Dreiecks liegt und damit die Kette vollständig ist. Die Schnitte dieser Kette werden aus der Liste entfernt und der Prozess wird wiederholt, solange ein Schnitt mit einem Endpunkt auf einer Kante vorhanden ist. Diese Ketten können zu zwei Arten von Trennungen führen: Cutoffs und Incisions. Cutoffs entstehen durch Ketten, deren Schnitte mit einem Endpunkt auf einer der Seiten beginnen und mit dem Endpunkt eines Schnittes auf einer anderen Kante enden. Diese Art von Kette kann auch durch einen einzelnen Schnitt entstehen. Incisions entstehen, wenn die Kette auf derselben Kante endet auf welcher sie begonnen hat. Sollten Schnitte vorhanden sein, welche keiner dieser Kategorien entsprechen, handelt es sich um Holes. Um diese Ketten zu finden, wird das erste noch verbleibende Element innerhalb der Liste ausgewählt. Einer der Endpunkte wird gespeichert und als Erfolgsbedingung gesetzt. Der andere Punkt wird wie zuvor mit den Endpunkten der noch vorhandenen Schnitte verglichen und Schnitte mit gleichen Endpunkten werden der Kette hinzugefügt. Sollte der andere Punkt einer Kette dem Punkt der Erfolgsbedingung gleichen, wird das Hole vervollständigt und die Schnitte aus der Liste entfernt.

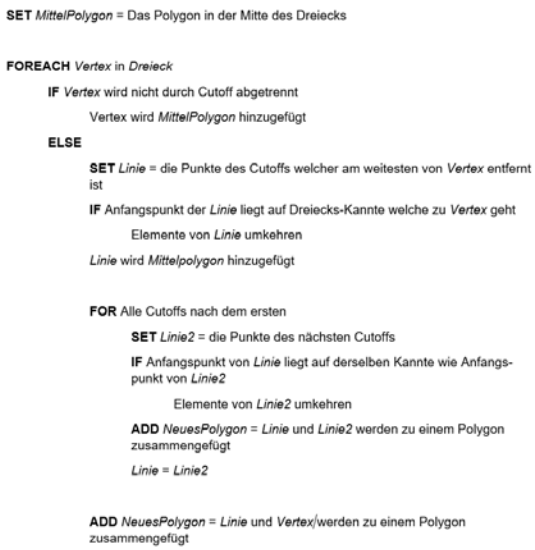

Abbildung 3.21: Der Pseudocode für das Einteilen der Cutoffs in Polygone

Die erstellten Ketten werden ihren Kategorien nach in Listen eingeteilt: Cutoff, Incision und Holes. Die Cutoffs werden innerhalb der Liste geordnet, abhängig davon, wie weit die Endpunkte der Kette auf den Kanten von der Ecke des Dreiecks, welche sie abtrennen, entfernt sind. Die Incisions werden danach geordnet, welche Endpunkte auf derselben Kante innerhalb von anderen Endpunkten liegen. Relevant für das Ordnen der Cutoffs und Incisions ist, dass die Ketten sich in keinem Fall kreuzen können. Dies wäre nur im Falle von selbst-schneidenden Meshes möglich. Solche Meshes sind für die Zwecke dieser Arbeit nicht von Nutzen und werden deshalb nicht innerhalb der Mesh-Generierung erlaubt. Aus diesem Grund und um Ressourcen zu sparen, werden die durch solche Meshes entstehenden Sonderfälle nicht innerhalb dieses Prozesses behandelt.

Das Einteilen in Polygone beginnt mit dem Polygon in der Mitte des Dreiecks, dessen Ecken von den jeweiligen Cutoffs abgeschnitten werden (siehe Abbildung 3.21). Sollten mehrere Cutoffs für dieselbe Ecke existieren, werden diese iterativ zu neuen Polygonen, bestehend aus den Schnitten dieses Cutoffs und dem folgendem Cutoff zusammengefügt. Für diese Polygone müssen die Punkte der Schnitte des zweiten Cutoff in umgekehrte Reihenfolge hinzugefügt werden. Dieser Schritt ist notwendig, um die Polarität der neuen Polygone, also ob die Punkte des Polygons mit oder gegen den Uhrzeigersinn angeordnet sind, gleich der des originalen Dreiecks

zu halten. Der letzte Cutoff einer Ecke bildet ein Polygon mit seinen Schnitten und der Ecke, welcher er abtrennt.

Nachdem die Cutoffs des Dreiecks zu Polygonen zusammengefügt wurden, folgt das Hinzufügen der Incisions innerhalb der neuen Polygone. Die den Kanten des Dreiecks zugeordneten Listen der Incisions werden gegen die neu erstellen Polygone überprüft. Sollte ein Polygon einen Teil der Kante verwenden, auf welchem eine der Incisions liegt, werden die auf der Kante befindlichen Punkte des Polygons mit einem der Endpunkte der Incision verglichen. Sollte dieser Endpunkt auf derselben Seite beider Polygon-Punkte liegen, wird das nächste Polygon überprüft. Sollte der Endpunkt auf der jeweils anderen Seite der Polygon-Punkte liegen, befindet die Incision sich innerhalb der Kannte des Polygons. Es muss lediglich einer der Endpunkte der Incision überprüft werden, da sich überschneidende Ketten durch die bereits erwähnten Vorgaben ausgeschlossen werden. Die Punkte der Incision werden nun in die Liste von Punkten, welche das Polygon bilden, eingefügt. Für das Einfügen muss sichergestellt werden, dass die Punkte der Incision zwischen den auf der Dreieckskante befindlichen Punkten des Polygons der Liste hinzugefügt werden. Außerdem muss die Distanz des Polygon-Punktes, nach welchem die Punkte der Incision eingefügt wurden, geringer zum ersten Punkt der Incision-Punkte sein, welcher einen der Endpunkte darstellt, als zum letzen sein, welcher den anderen Endpunkt der Incision darstellt. Sollte dies nicht der Fall sein, muss die Reihenfolge der Punkte der Incision umgekehrt werden, bevor diese der Liste des Polygones hinzugefügt werden. Der Bereich des Polygons, welcher durch die Incision ausgeschnitten wird, muss durch ein weiteres Polygon gefüllt werden. Dieses Polygon besteht ausschließlich aus der Liste der Punkte der Incision, welche umgekehrt werden muss, damit das resultierende Polygon dieselbe Polarität wie das Ausgangsdreieck besitzt. Sollten sich innerhalb dieser Incision weitere Incisions befinden, müssen diese ebenfalls aus dem neuen Incisions-Polygon geschnitten und ein weiteres Polygon für diese Incisions erstellt werden.

Sobald alle Polygone basierend auf Cutoffs und Incisions erstellt wurden, müssen in diese Polygone die Holes eingefügt werden. Hierfür wird die Liste der Holes iterativ durchlaufen. Für jedes Hole wird einer der Punkte der Schnitte des Holes gewählt. Dieser Schnitt wird gegen alle Polygone getestet, mit einem Winding-Nummbers Test [KB18]. Dieser Test hat zum Ziel, zu bestimmen, ob ein Punkt innerhalb oder außerhalb eines Polygones liegt. Sollte dies der Fall sein, wird das Hole dem jeweiligen Polygon zugewiesen und das nächste Hole überprüft.

3.2.14 Mesh-Boolean-Operationen: Triangulierung

Für die Triangulierung soll eine Library verwendet werden, welche einen Triangulierungsalgorithmus implementiert. Eine solche Library legt mehrere Vorgaben vor, welche die Eingabe-Daten erfüllen müssen, um von der Library fehlerfrei verarbeitet zu werden. Deshalb müssen die getrennten Polygone erst in ein für die Library verwendbares Format überführt werden, bevor sie dieser übergeben werden. Anschließend kann die Library diese Polygone triangulieren und die resultierenden Dreiecke zurückgeben. Diese müssen aus dem Format der Library zurück geändert werden. Die neuen Dreiecke müssen anschließend in die originalen Meshes eingefügt werden. Hierfür müssen zuerst die durch die Schnitte entstanden neuen Vertices hinzugefügt werden. Die Vertices innerhalb der Dreiecke können hierbei direkt der Liste als neue Einträge angehängt werden. Vertices, welche durch Schnitte mit der Kante entstanden, sind bei einem geschlossenen Mesh in jedem Fall auf zwei Ausgangsdreiecken zu finden. Um doppelte Einträge dieser Vertices und damit einhergehende Fehler beim Verbinden der Dreiecke zu vermeiden, müssen diese Vertices zu einem einzigen Eintrag reduziert werden. Hierfür wird beim Hinzufügen der neuen Vertices überprüft, ob dieses bereits innerhalb der Vertex-Liste vorhanden ist. Sollte dies der Fall sein, wird das Vertex nicht hinzugefügt und alle Dreiecke, welche auf dieses Vertex verweisen, müssen auf den Index des ersten Vorkommens innerhalb der Dreiecksliste zeigen.

3.2.15 Mesh-Boolean-Operationen: Verbinden von Schnitt-Meshes

Alle Dreiecke, unabhängig davon, ob sie bereits existierten oder neu hinzugefügt wurden, müssen in die Kategorien von außerhalb oder innerhalb des jeweils anderen Meshes eingeteilt werden. Hier wird sich dem Ansatz des Papers von Jiang et al. bedient[JPC+16]. Dieser Ansatz stellt die Zugehörigkeit aller Vertices der Meshes durch die Vertices der sich schneidenden Dreiecke fest. Hierbei wird die Zugehörigkeit der Vertices eines Dreiecks von Mesh A festgestellt, durch die Ebenenformel des von diesem geschnitten Dreiecks von Mesh B. Diese Formel kann feststellen, ob ein Punkt vor oder hinter der Ebene liegt. Da von einer konsistenten Polarität aller Dreiecke der Ausgangsmeshes ausgegangen werden kann, reicht diese Information, um die Vertices des anderen Meshes in innerhalb oder außerhalb einzuteilen. Die

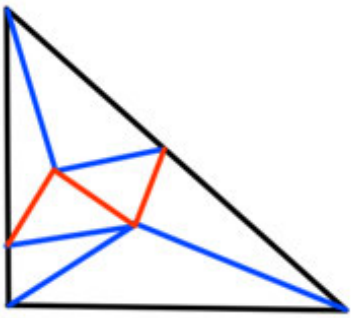

Abbildung 3.22: Das Bild zeigt ein Dreieck (schwarze Linien), welches mehrmals geschnitten wurde (rote Linien). Mit diesen Schnitten wird das Dreieck zu neuen Dreiecken trianguliert (blaue Linien). Es ist ebenfalls zu erkennen, dass neue Dreiecke entstehen, welche aus zwei roten und keinen schwarzen Linien bestehen.

nach diesem Kategorien eingeteilten Vertices können jeweils als die Grenze der Zugehörigkeiten angesehen werden. Die restlichen Vertices der Meshes, welche nicht zu geschnittenen Dreiecken gehören, können durch diese Grenzen den Kategorien zugewiesen werden. Hierfür werden die Vertices einer Kategorie iterativ durchlaufen. Durch die Dreiecksliste kann überprüft werden, mit welchen Vertices das jeweilige Ausgangsvertex verbunden ist. Sollten diese Vertices noch nicht Teil der Grenze der anderen Kategorie sein, werden diese rekursiv derselben Kategorie wie die des Ausgangsvertex hinzugefügt. Alle Vertices welche nach dieser Operation noch keiner Kategorie zugeordnet sind, können der verbleibenden Kategorie hinzugefügt werden, da in dieser Arbeit nur vollständig verbundenen Meshes verwendet werden und jedes noch nicht kategorisierte Vertex, damit auf der anderen Seite der Grenze liegen muss. Dieser Prozess ist in der Lage, die Vertices, welche ursprünglich vorhanden sind, eindeutig einzuteilen. Er stößt jedoch auf Probleme bei den durch die Schnittstellen entstandenen Vertices und Dreiecken. Aus der Arbeit von Jiang et al. geht nicht hervor, ob die möglichen Probleme behandelt sind, welche durch das unangepasste Anwenden dieses Ansatzes auf die durch die Schnittstellen entstandenen Dreiecke entstehen. Diese Probleme entstehen durch drei verschiedene Grenzfälle. Der erste Fall entsteht, wenn ein Dreieck von zwei oder mehr Dreiecken des anderen Meshes geschnitten wird, deren normalen Vektoren in entgegengesetzte Richtungen zeigen. In einem solchen Fall würden die Vertices basierend auf dem ersten Dreieck unterschiedlich kategorisiert werden als durch das zweite. Fall zwei tritt ein, wenn es sich bei dem zur Kategorisierung verwendeten Dreieck um ein Dreieck handelt,

dessen Schnitt Teil einer Incision ist. In einem solchen Fall würden Punkte inkorrekt kategorisiert werden, welche nicht durch diese Incision kategorisiert werden sollten. Ein letzter zu berücksichtigender Fall tritt ein, wenn die durch die Triangulierung erstellten Dreiecke kategorisiert werden sollen. In Bild 3.22 ist zu erkennen, dass es möglich ist, für Dreiecke nur aus Vertices der Schnittlinie zu bestehen, welche nicht kategorisiert werden und damit eine Festlegung der Kategorie des Dreiecks verhindern.

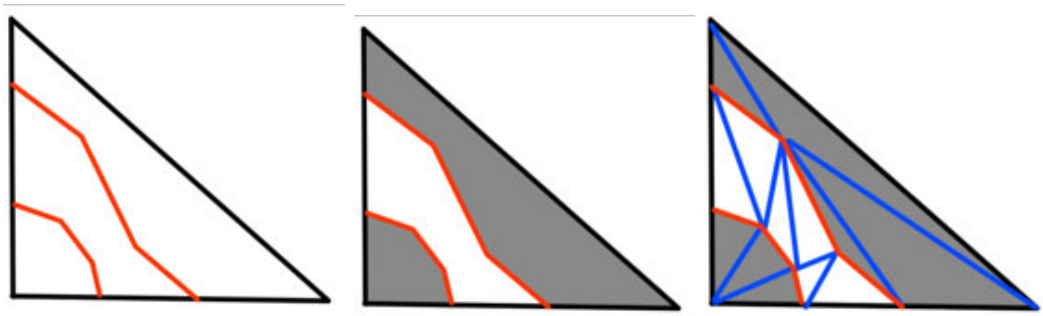

Abbildung 3.23: Die Bilder zeigen den vorgeschlagenen Prozess zum Kategorisieren von durch Triangulierung entstandenen Dreiecken. Das Dreieck wird zuerst anhand der Schnitte in Polygone eingeteilt (rechts). Diese Polygone werden kategorisiert (mitte). Die Dreiecke der Triangulierung übernehmen die Kategorie ihrer Ursprungs-Polygone (links).

Aus diesen Gründen wird stattdessen ein anderes Vorgehen für die Kategorisierung dieser Vertices und Dreiecke vorgeschlagen. Ziel des Ansatzes ist hierbei, die Dreiecke und Vertices bereits in der Phase des Aufteilens der geschnittenen Dreiecke in Polygone zu kategorisieren (siehe Bild 3.23). Die Kategorisierung der Vertices des Ausgangdreiecks würden hier nach dem Aufteilen des Dreiecks in die Cutoff-Polygone vorgenommen werden. Die Tests, um diese Vertices zu kategorisieren, werden immer mit einem Dreieck der Cutoff-Kette, welche am nächsten zu dem jeweiligen Vertex ist. Dies wird nur für die Ecken des Dreiecks, welche durch einen Cutoff abgetrennt sind, durchgeführt, da die nicht abgetrennten Ecken Teil des ”Mittelpolygons” sind und damit dessen Kategorie teilen. Die Polygone der abgetrennten Vertices haben die gleiche Kategorisierung wie diese. Der Rest der Cutoff-Polygone kann von den äußersten Polygonen angefangen abwechselnd kategorisiert werden. Polygone, welche durch Incisions oder Holes entstehen, werden der gegenteiligen Kategorie der Polygone, aus welchen sie einen Teil herausnehmen, zugewiesen. Die Dreiecke, welche aus

der Triangulierung dieser Polygone entstehen, sind Teil derselben Kategorie wie das Polygon, dessen Teil sie ursprünglich waren. Aus diesem Grund müssen die durch die Schnitte entstanden Vertices nicht kategorisiert werden, da dies lediglich eine Vorstufe der Kategorisierung der Dreiecke eines Meshes ist. Die zugewiesenen Vertices des Ausgangsdreiecks werden verwendet, um die nicht geschnittenen Dreiecke wie beschrieben einzuteilen.

Im letzten Schritt der Operation werden die eingeteilten Dreiecke des Meshes abhängig von der gewünschten Operation verschiedenen Kategorien zugeordnet. Wie im Abschnitt der Volumenberechnung bereits beschrieben, gibt es hier drei Optionen. Die erste Option ist die Union-Operation. In diesem Fall werden die als außerhalb kategorisierten Dreiecke beider Meshes verwendet. Option zwei ist die Difference Operation, welche die äußeren Dreiecke des ersten Meshes und die inneren Dreiecke des zweiten Meshes verwendet. Diese inneren Dreiecke müssen in ihrer Polarität umgekehrt werden, um mit ihren normalen Vektoren aus dem Mesh hinauszuzeigen wie die Dreiecke des ersten Meshes. Die letzte Option ist die Intersection Operation, für welche nur die inneren Dreiecke beider Meshes verwendet werden. In diesem Fall müssen alle Dreiecke umgekehrt werden. Diese Dreiecke zusammen mit ihren zugehörigen Vertices werden zu einem einzigen Mesh zusammengefügt und überschreiben das erste der beiden als Input gegebenen Meshes.

3.2.16 Tools für Evaluation der Tools

Da die in dieser Arbeit erstellten Tools mit großen Datenmengen arbeiten, ist das intellektuelle Überprüfen jedes Zwischenschritts und Ergebnisses stark zeitaufwendig. Aus diesem Grund werden mehrere Tools erstellt, um die Ergebnisse zusammenzufassen und Metadaten von Prozessen wie der Szenengenerierung zu sammeln.

Heatmaps sind ein Tool, welches es erlaubt, 2D-Verteilungen von Datenpunkten qualitativ zu veranschaulichen. In dieser Arbeit werden die Heatmaps verwendet, um die Verteilung der Objekte der Szenen der prozeduralen Szenengenerierung (siehe 3.2.9) zu veranschaulichen (Siehe Abbildung 3.24). Die Heatmaps werden dabei nicht verwendet, um festzulegen welche Szenen verwendet werden und welche nicht, sondern haben nur das Ziel, zu visualisieren, wie die gewählten Szenen verteilt sind. Durch diese Visualisierungen sollen Fehler vermieden werden, welche durch eine inkorrekte Verteilung der Objekte entstehen. Die Heatmaps werden beim Einteilen

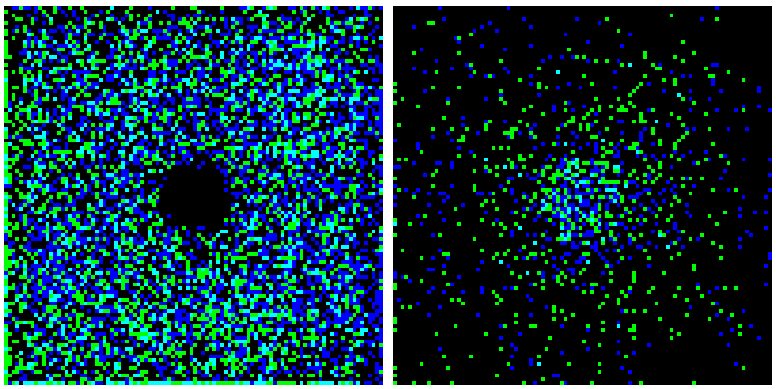

Abbildung 3.24: Zwei Heatmaps für eine Szenengenerierung mit starker Randomisierung der Objektpositionen. Die grünen Punkte zeigen Positionen, an welchen Ödeme generiert wurden, die blauen sind die von Drusen. Links zeigt die Heatmap der Szenen mit einem geringen Volumen innerhalb des fovealen Zylinders, Rechts die Heatmap der Szenen mit hohem Volumen. Das abgebildete Beispiel zeigt einen Fall, in welchem der Schwellwert für hohes Volumen sehr gering ist, wodurch die Heatmap für geringes Volumen keine Objekte im fovealen Zylinder aufweist.

der Szenen in hohes und niedriges Zylinder-Volumen generiert. Es wird jeweils eine Heatmap für hohes Volumen und eine für niedriges Volumen generiert.

3.3 Praktische Anwendung des Tools

Dieses Unterkapitel soll den Workflow innerhalb des Scenebuilders erläutern, angefangen mit dem Anpassen der Epitheleinstellungen. Dies wird gefolgt von dem Hinzufügen von Objekten. Abschließend wird die erstellte Szene gespeichert und als Basis für das Generieren weiterer Szene verwendet.

3.3.1 Grundlegende Szenen-Einstellungen

Wenn das Scene-Builder Tool gestartet wird, wird eine neue leere Szene geöffnet. Eine neue Szene innerhalb des Tools enthält die zwei Epithelschichten mit einem

zufällig generierten Noise und keine Objekte. Die grundlegenden Inputs zum Interagieren mit der Szene sind in Tabelle 3.1 aufgeführt. Mit den Inputs zum Bewegen der Kamera kann die Szene examiniert werden. Im ersten Schritt sollte die Auflösung der Schichten festgelegt werden, welche bestimmt, wie detailliert die Form der Schichten und vor allem die Form ihres Noise erkennbar ist. Anschließend sollte die Höhe der Top-Schicht und die Krümmung beider Schichten eingestellt werden, um dem Erscheinungsbild des OCT-Scans, welcher als Vorlage dient, zu gleichen. Für das Bestimmen der Höhe ist innerhalb eines der OCT-Scans-Bilder das Verhältnis der Distanz der beiden Schichten zur Breite des Bildes relevant. Anschließend sollte die Position der Fovea festgelegt werden. Die Y-Koordinate kann abhängig davon, auf dem wievielten Bild sie innerhalb des Scans zu finden ist, geschätzt werden und die X-Koordinate abhängig davon, welche Distanz sie auf dem Bild zum linken Bildrand hat. Nachdem die Position der Fovea bestimmt wurde, kann sie mithilfe der weiteren Fovea-Parameter in die gewünschte Form gebracht werden. Diese Schritte legen die grundlegende Form der Fovea fest. Mithilfe der Noise-Einstellungen soll diese Form verfeinert werden. Die Einstellung zur Frequenz gibt an, ob eine große Anzahl von kleinen scharfkantigen Wellen oder eine kleine Anzahl von großen weichen Wellen generiert wird. Die Stärke-Einstellung legt den Höhenunterschied zwischen dem tiefsten und höchsten Punkten dieser Wellen fest. Sobald diese Einstellung das gewünschte Erscheinungsbild geben, kann durch wiederholtes Klicken des ”Regenerate”-Buttons ein Muster gefunden werden, welches der Vorlage am besten entspricht. Die Einstellungen zur Objekt-Verformung können am leichtesten angepasst werden, sobald die ersten Objekte hinzugefügt wurden, welche die Top-Schicht verformen. Sollten die Schichten zufriedenstellend generiert sein, kann der foveale Zylinder platziert werden. In den meisten Fällen ist es wünschenswert, diesen an dieselbe Position wie die Fovea zu platzieren und anschließend dessen Dimensionen anzupassen.

3.3.2 Hinzufügen von Objekten

Nachdem die Epithelschichten gesetzt sind, kann mit dem Hinzufügen der Drusen und Ödeme begonnen werden. Optional kann vor diesem Schritt der aktuelle Stand der Szene gespeichert werden. Hierfür muss der ”Save”-Button gedrückt werden und ein Ordner ausgewählt werden, in welchem die Datei gespeichert werden soll. Bevor ein Objekt hinzugefügt wird, sollten dessen Eigenschaften eingestellt werden. Angefangen wird dies mit dem ”Edema”- und ”Druses”-Buttons, um den Typ des Objekts

Input	Funktion
WASD/Pfeiltasten	Kamera bewegen: Vorwärts, Links, Zurück, Rechts
Leertaste	Kamera bewegen: Oben
Shift halten & Leertaste	Kamera bewegen: Unten
Rechtsclick halten	Kamera rotieren
Mausrad	Kamera Zoom
Strg halten & Linksklick	Objekt generieren
Strg halten & Linksklick halten	Objekte in Rechteck generieren
Linksklick auf Ödem/Druse	Ödem/Druse selektieren
Shift halten & Linksklick auf Ödem/Druse	Weitere Ödeme/Drusen selektieren
Linksklick halten	In Rechteck selektieren
Shift halten & Linksklick halten	Weitere in Rechteck selektieren
Strg halten & z	Letzte Änderung Rückgängig machen
Strg halten & Shift halten & z	Letzte Änderung Wiederherstellen
Entf	Selektierte Objekte löschen

Tabelle 3.1: Die grundlegenden Inputs für das Scenebuilder-Tool.

festzulegen. Anschließend kann die Randomisierung der Form durch die Slider unter diesen Buttons entschieden werden. Sobald diese Parameter wie gewünscht eingestellt sind, können Objekte hinzugefügt werden. Der Input für das Hinzufügen eines Objekts kann in Tabelle 3.1 gefunden werden. Wenn dieser Input gegeben wird, wird ein Objekt auf der unteren Epithel-Schicht an der Position des Mauszeigers generiert. Sollte dieses Objekt nun weiter angepasst werden, muss es selektiert werden (siehe Tabelle 3.1). Die Art der Manipulation wird über die mit dem Selektieren erschienen Buttons gewählt: Position, Rotation und Skalierung. Das Klicken auf diese Buttons lässt die jeweiligen "Handles" um das Objekt erscheinen, das Klicken und Ziehen, welcher die jeweilige Eigenschaft des Objekts in einer Koordinate verändert. Dieser Prozess wird wiederholt, bis alle gewünschten Objekte sich innerhalb der Szene befinden. Sollte es wünschenswert sein, mehrere Objekte als eine Schicht zu erstellen, kann dies durch ein Halten und Ziehen des Inputs zum Hinzufügen von Objekten erreicht werden. Die Einstellungen für dieses Generieren können ebenfalls mit Slidern angepasst werden. Sollte während des Hinzufügens oder Manipulierens der Objekte oder während des Veränderns der Epithel-Einstellungen ein Fehler unterlaufen, kann dieser durch Drücken des "Undo"-Buttons rückgängig gemacht werden. Wenn alle Objekte platziert sind und alle Anpassungen der Epithelschichten abgeschlossen sind, kann die Szene gespeichert werden.

3.3.3 Prozedurale Szenengenerierung

Sollen die Szenen als Basis für ein maschinelles Lernverfahren verwendet werden, können weitere Szenen prozedural generiert werden, auf Basis der bereits erstellten Szenen. Hierfür wird eine Szene mit dem "Select Scene"-Button ausgewählt und der Ordner, in welchem die generierten Szenen gespeichert werden sollen, mit dem "Save Path"-Button. Der Grad der Randomisierung der originalen Elemente kann durch das Verstellen, der Slider bestimmt werden. Soll die originale Szene noch erkennbar bleiben nach der Generierung, dann sollte der Positions-Slider gering gehalten werden. Nachdem die Anzahl der zu generierenden Szenen in das Textfeld eingegeben wurde, kann das Generieren mit dem "Start Generation"-Button begonnen werden. Wie lange das Generieren dauert, hängt nicht nur von der Anzahl der zu generierenden Szenen, sondern auch von der Komplexität der Ausgangsszene, ab.

4 Implementierung

In diesem Kapitel werden die Implementierungsdetails der im Unterkapitel Umsetzung erläuterten Features ausgeführt. Es wird sich vor allem damit befasst, wie diese Konzepte innerhalb der Unity-Engine realisiert werden und welche Einschränkungen sich aus dem Verwenden dieser ergeben. Der Aufbau dieses Kapitels folgt derselben Struktur wie das Umsetzungs-Unterkapitel, angefangen mit der Meshgenerierung, gefolgt von den Features des Scenebuilders, abgeschlossen mit den Mesh-Boolean-Operationen.

4.1 Mesh-Generierung

Um ein Mesh in Unity zu generieren, muss zuerst ein neues Objekt erstellt werden oder das Mesh eines existierend Objektes verändert werden. Um die neue Form des Meshes zu definieren, müssen dessen Vertices überschrieben werden, welche aus einem Array von Vektoren bestehen sowie die Dreiecke, welche aus einem Array von Integern bestehen. Wie in der Konzeption beschrieben, sollen die Drusen und Ödeme auf eine OCT-Scan basierte Art generiert werden. Hierfür werden mehrere Polygone in festen Abständen auf einer Achse generiert und miteinander verbunden. Wenn die Dreiecke und Vertices gesetzt wurden, sollten in den meisten Fällen die Normals des Meshs neu berechnet werden, um sicherzustellen, dass sie auf die neue Form des Meshes passen.

Um die Epithelschichten zu generieren, wird zuerst ein quadratisches Feld von Vertices generiert. Die Anzahl der Vertices ist abhängig von der gewünschten Auflösung. Diese Vertices werden mit Dreiecken verbunden. Die wichtigste Eigenschaft der Vertices ist ihre Y-Koordinate, auf welche mehrere Faktoren einwirken. Zuerst wird der Wert des Noises an den X- und Z-Koordinaten des Vertices bestimmt. Als Noise-Funktion wird hier der in Unity integrierte sogenannte Perlinnoise [Per02] verwendet.

Die übergebenen Koordinaten werden mit der Frequenz multipliziert, um den Perlinnoise in kleineren oder größeren Abständen zu durchlaufen. Nach dem Noise wird die Krümmung des Auges auf die Y-Koordinate angewendet. Die Krümmung wird durch den Abstand zu einem Punkt erzielt, welcher für alle Vertices gleich sein muss. Dieser Punkt liegt immer über der Fovea, wobei größere Abstände von dem Punkt zur Fovea zu einer geringeren Krümmung führen. Da es für einen Nutzer intuitiver erscheint, wenn diese Krümmung durch das Erhöhen eines Wertes stärker wird, wird eine maximale Distanz festgelegt, von welcher der Wert des Krümmungs-Slider subtrahiert wird. Die durch die Krümmung notwendige Erhöhung der Y-Koordinate wird zusätzlich zum Wert des Noises addiert. Sollte es sich bei der zu generieren Epithelschicht um die Top-Schicht handeln, wird zusätzlich überprüft, ob sich das jeweilige Vertex innerhalb des Radius der Fovea befindet. Sollte dies der Fall sein, wird überprüft, ob es sich im inneren Teil der Fovea befindet, dessen Vertices sich zu einer Trichter-Form senken und die Krümmung sowie den Noise ignorieren, oder es im äußeren Bereich der Fovea liegt, welcher den Noise behält, aber erhöht wird. Der letzte Einfluss auf die Y-Koordinate kommt durch die Objekte unter der Top-Schicht, welche sich in einer ausreichend geringen Distanz von dieser befinden, um Wölbungen zu verursachen. Um festzustellen, ob ein Objekt sich nahe genug an der Position eines Vertex befindet, um die Höhe dieses zu beeinflussen, wird ein Raycast verwendet. Ein Raycast in Unity ist ein Strahl, welcher von einer Ursprungsposition ausgehend detektiert, ob sich ein Objekt innerhalb einer spezifizierten Richtung befindet [Uni20]. Sollte dies der Fall sein, wird der Punkt der Kollision zurückgegeben. Außerdem ist es möglich zu spezifizieren, mit welchen Objekten dieser Raycast kollidieren kann und mit welchen nicht. Auf diese Art können Kollisionen mit einer bereits existierenden Top-Schicht verhindert werden, sollte diese neu generiert werden. Für das Testen der Vertices wird jeweils ein Raycast mit Ursprung bei deren X- und Z-Koordinaten und einer stark erhöhten Y-Koordinate in Richtung der Bottom-Schicht durchgeführt. Die erhöhte Y-Koordinate sorgt dafür, dass auch Objekte, welche nicht nur nahe an, sondern über der Top-Schicht liegen, korrekt abgefangen werden. Sollte eine Kollision gefunden werden, wird überprüft, ob die Y-Koordinate des Kollisionspunktes zusammen mit einem festgelegtem Buffer-Wert über der Y-Koordinate des jeweiligen Vertices liegt. Sollte diese der Fall sein, wird die Höhe des Vertices überschrieben mit dem höheren Wert. Die Wölbungen durch Objekte machen es notwendig, dass signifikante Veränderung bezogen auf die Objekte ein erneutes Generieren der Top-Schicht auslösen. Signifikante Veränderungen beinhalten das Hinzufügen und Löschen von Objekten sowie das Manipulieren bereits erstellter Objekte in Translation, Rotation

oder Skalierung. Im letzten Schritt der Generierung wird der Epithelschicht eine Dicke gegeben. Hierfür werden alle bereits erstellen Vertices kopiert und abhängig von der gewünschten Dicke der Epithelschicht negativ in ihrer Y-Koordinate bewegt. Anschließend werden auch diese Vertices mit Dreiecken verbunden, allerdings muss die Reihenfolge der Vertex-Indeces für diese umgekehrt werden, damit die Normalen-Vektoren dieser Dreiecke nach außen zeigen.

4.2 Materials und Shading in Unity

Für die Epithelschichten sowie für den fovealen Zylinder ist nicht nur die Form, sondern auch das Material von Relevanz. Ein Material in 3D-Programmen legt die Erscheinung eines Objektes abgesehen von seiner Form durch Parameter wie Farbe und Durchsichtigkeit fest. Jedes Material hat einen oder mehrere Shader, welche vorgeben, wie das Objekt mit Licht interagiert, z. B. ein Glass-Shader, welcher größtenteils durchsichtig ist, oder ein Emissions-Shader, welcher Licht ausstrahlt. Für die Bottom-Schicht, Drusen und Ödeme kann hier ein Diffuse-Material verwendet werden, welches das Standard-Material für feste, nicht metallische Objekte ist. Der einzige Parameter, welches für diese angepasst werden muss, ist die Farbe des Objekts, um die Drusen und Ödeme deutlich voneinander und von der Bottom-Schicht zu unterscheiden. Für die Top-Schicht wird ein ”Fresnel”-Shader verwendet. Dieser sorgt dafür, dass alle Flächen eines Objektes, welche senkrecht zur Perspektive des Betrachters stehen, transparent erscheinen und diese Transparenz stärker verlieren, je größer der Winkel zwischen dem Normalen-Vektor der Fläche und dem Vektor der Kameraperspektive wird. Dies erlaubt es, die Objekte, welche sich unter der Schicht befinden, deutlich zu erkennen, aber lässt trotzdem die Form der Unebenheiten der Top-Schicht erkennen. Das Material der Top-Schicht kann durch ein Dropdown-Menü zu einem Diffuse-Material geändert werden, welches nicht durchsichtig ist, aber die Form der Schicht leichter erkennbar macht. Der foveale Zylinder verwendet eine abgewandelte Form dieses Shaders. Der Shader des Zylinders unterscheidet sich nicht nur in seiner Farbe, sondern auch dadurch, dass Schnittstellen mit Objekten hervorgehoben werden. Dieser Effekt soll es dem Nutzer erleichtern, den Teil des Volumens eines Objektes, welcher sich innerhalb des Zylinders befindet, zu einem nutzvollen Grad einzuschätzen

4.3 Weitere Scenebuilder-Details

Die Umsetzung der verbleibenden Features des Scenebuilders stellt eine Vielzahl von Herausforderungen an die Implementierung. Um festzustellen, welche Objekte für die Volumenberechnung gegeneinander getestet werden müssen, werden Unitys Collider verwendet. Solche Collider werden in Game-Engines zum Beispiel für Physik-Simulationen verwendet. Sie erlauben es außerdem, zu überprüfen, ob ein Collider sich in einen anderen Collider bewegt oder dieser von einem Collider verlassen wird. Um sicherzustellen, dass die Form der Collider das Mesh korrekt repräsentiert werden, werden Unitys Mesh-Kollider benutzt. Diese verwenden die tatsächliche Form eines Objektes für Kollisionen anstatt einer Approximation wie einer Bounding-Box. Dies ist durch die geringe Anzahl von Objekten innerhalb einer Szene und der geringen Anzahl von Dreiecken in den Epithelschichten und Ödemen möglich. Da es nicht möglich ist, alle sich momentan in einem Collider befindenden anderen Collider abzufragen, muss jedes eintreten und verlassen eines Colliders von allen Ödemen und Drusen sowie dem fovealen Zylinder in einer Liste gespeichert werden. Wenn für die Volumenberechnung alle überschneidenden Objekte gefunden wurden und in Gruppen eingeteilt wurden, werden die Union-Operationen durchgeführt, um diese Gruppen zu jeweils einem Objekt zu kombinieren. Die Reihenfolge der Elemente innerhalb der Liste ist dabei zu berücksichtigen, da es nicht vorkommen darf, dass zwei Elemente, welche zwar Teil derselben Gruppe sind, aber sich nicht gegenseitig überschneiden, versucht werden zu kombinieren. Deshalb muss bei der Reihenfolge darauf geachtet werden, dass der Zwischenstand von kombinierten Objekten das Objekt enthält, welches das nächste hinzuzufügende Objekt überschneidet.

4.4 Datenpersistenz

Für das Speichern und Laden muss ein Datenformat gewählt werden. Die Wahl fällt hierbei auf JSON, da dieses Format ein schnelles Serialisieren und Deserialisieren erlaubt[ECM99] und bereits in Unity integriert ist. Wenn eine Szene geladen wird, werden zuerst alle vorhandenen Objekte gelöscht. Anschließend werden die Objekte mit den Einstellungen der JSON-Datei geladen. Das Laden wird außerdem verwendet, wenn mit dem ”New”-Button eine neue Szene erstellt werden soll. Anstatt eine

Liste von Standardwerten für die verschiedenen Parameter einer Szene zu verwenden, wird eine Szene geladen, welche die Standardwerte besitzt und keine Objekte aufweist.

4.5 Prozeduralle Szenengenerierung

Für die Implementierung der Szenengenerierung werden mehrere der bereits bestehenden Features kombiniert. Das Laden von Szenen wird verwendet, um eine Szene in die Anwendung zu laden, das Generieren von Objekten wird verwendet, um weitere Objekte basierend auf den existierenden zu generieren und das Neu-Generieren der Epithelschicht wird verwendet, um Instanzen der Epithelschichten mit gleichen Parametern wie die Ausgangschicht, aber anderen Noise-Muster zu generieren. Entscheidend ist hier, dass die Epithelschicht nicht im selben Frame wie die Objekte geladen wird. Grund ist, dass die Raycasts, welche die Verformung der Epithelschicht durch die unter dieser befindlichen Objekte bestimmen, im "Physics"-Abschnitt des Unity Scriptablaufs liegen. Dies bedeutet, dass Raycasts vor der Update-Methode, in welcher die Szene generiert wird, durchgeführt werden [Uni19]. Aus diesem Grund ist es notwendig, die Generierung der Szenen innerhalb einer Coroutine durchzuführen, welche nach dem Generieren der Objekte bis zum Anfang des nächsten Frames wartet, um mit dem Generieren der Epithelschicht zu beginnen. Anschließend wird das Volumen der Objekte bestimmt. Sollte es hier zu Fehlern bei einer der Mesh-Boolean-Operationen kommen, wird dies in den Werten der gesamten oder der im Zylinder befindlichen Volumen deutlich. Die Symptome dieser Fehler sind zum Beispiel: Dass alle Gesamt-Volumen null sind, dass einer der Volumen-Werte negativ ist oder dass einer der Zylinder-Volumen-Werte größer ist als der jeweilige gesamte Volumen-Wert. In einem solchen Fall wird die Generierung verworfen und nicht zur Zahl der gesamten generierten Szenen gezählt. Die Anzahl der insgesamt erlaubten Versuche ist auf das Doppelte der zu generierenden Szenen Anzahl beschränkt, um Endlosschleifen-Szenarios bei grundlegenden Fehlern zu vermeiden. Nachdem die Volumen bestimmt sind, wird die generierte Szene gespeichert.

4.6 Undo-/Redosystem

Das Undo-System stellt eine simple Addition zum Scenebuilder dar, welche die User-Experience des Erstellens einer Szene stark verbessert. Diese Art von System ist in den meisten Programmen und Tools, zum Erstellen von Inhalten, üblich. In der Ausführung des Systems für diese Arbeit werden für jede mögliche signifikante Änderung Aktionen erstellt. Diese Aktionen enthalten die Ausführung und Umkehrung der jeweils gewünschten Änderung sowie alle hierfür benötigten Daten. Der Inhalt dieser Daten kann sich abhängig von der gewünschten Änderung unterschiedlich gestalten. Beispielsweise wird für das Bewegen eines Objektes eine Referenz auf das Objekt sowie der Vektor der Bewegung benötigt, welcher für die Ausführung auf die Position des Objektes addiert und für die Umkehrung subtrahiert wird. Die Anzahl der möglichen umgekehrten Anpassungen ist beschränkt, um bei langer kontinuierlicher Verwendung des Tools den Arbeitsspeicher nicht unnötig zu belasten, da ab einer bestimmten Anzahl von Anpassungen davon ausgegangen werden kann, dass der Nutzer Fehler in der Szene eher beheben würde, anstatt eine große Menge von nicht fehlerhaften Anpassungen rückgängig zu machen, um eine weit zurückliegende fehlerhafte Änderung zu beheben.

4.7 Mesh-Boolean-Operationen

Die Mesh-Boolean-Operationen werden durch das Folgen der im Umsetzungs Unterkapitel beschriebenen Quellen erleichtert. Die größte Schwierigkeit stellt das Triangulieren von Polygonen dar. Es gibt verschiedene Algorithmen für dieses Problem, wobei eine Balance zwischen der Effizienz des Algorithmus und der Schwierigkeit, diesen zu implementieren, getroffen werden muss. Um dieses Problem zu umgehen, soll eine Library verwendet werden, welche einen der effizienteren Algorithmen implementiert und idealerweise bereits in C# geschrieben ist. Die Wahl ist hier auf die Poly2Tri Library gefallen, welche bereits in C# geschrieben ist und damit das Verwenden in Unity stark erleichtert. Diese Library kann simple Polygone, also Polygone, welche sich nicht selbst schneiden und keine Löcher aufweisen, sowie Polygone, welche sich nicht selbst schneiden und Löcher aufweisen, triangulieren. Ein Nachteil dieser Library ist, dass sie nicht in der Lage ist, die Dreiecke mit den Schnitten der anderen Dreiecke zu triangulieren, da dies von der Library als

ein sich selbst schneidendes Polygon angesehen werden würde. Deshalb müssen die durch die Schnitte entstehenden Polygone erst getrennt und dem Algorithmus einzeln übergeben werden.

Die Triangulierung ist ebenfalls nicht in der Lage, mit Polygonen von sehr geringer Größe umzugehen. Da solche kleinen Polygone einen vernachlässigbaren Effekt auf die Volumenberechnung haben, werden sie für die Triangulierung übersprungen. Bevor die Polygone übergeben werden, müssen sie in die korrekte Form für die Library gebracht werden. Dies bedeutet, dass sie in die Klassen der Library umgewandelt werden müssen. Die größte Herausforderung liegt hier dabei, dass die Library nur zweidimensionale Daten akzeptiert. Deshalb müssen alle Polygone sowie alle deren Holes auf eine der Achsen-Ebenen projiziert werden. Die aus der Triangulierung resultierenden Dreiecke müssen anschließend wieder zu ihrer drei dimensionalen Form projiziert werden. Durch die zweidimensionale Projektion ist die Information über die Polarität der Polygone möglicherweise verloren gegangen. Aus diesem Grund werden die Normalenvektoren der neuen Dreiecke mit dem des Ausgangsdreiecks verglichen. Sollte der Winkel zwischen diesen einen signifikanten Betrag erreichen, welcher über mögliche Rundungsfehler der Triangulierung hinausgeht, wird das neue Dreieck umgekehrt. Das Umkehren wird durch ein Vertauschen der ersten beiden Indizes der Dreieckspunkte innerhalb der Dreiecksliste vorgenommen. Die Polarität der Dreiecke ist entscheidend, um deren Volumen korrekt zu berechnen. Außerdem ist es notwendig, um die sich ergebenden Meshes korrekt visuell darzustellen, wegen Unitys Backfaceculling, welches die Dreiecke, deren Rückseite der Kamera zugewendet ist, nicht rendert.

5 Evaluation

In diesem Kapitel wird die Evaluierung des MVCNNs mit den durch die Szenengenerierung entstehenden Daten durchgeführt. Zu diesem Zweck wird zuerst das Ziel der Evaluation erläutert, gefolgt von dem Ablauf der Testreihen. Anschließend werden die Details der Testreihen zusammen mit ihren Ergebnissen dargestellt. Das Kapitel wird abgeschlossen mit einer Auswertung und Deutung der Ergebnisse.

5.1 Ziel

Diese Evaluation hat das Ziel, zu testen, welche Beschaffenheit ein Datensatz von OCT-Szenen aufweisen muss, um von einem MVCNN mit einer ausreichenden Genauigkeit klassifiziert werden zu können. Dies soll darüber informieren, welche Eigenschaften ein auf Realwelt Daten basierender Datensatz sein sollte und worauf bei dessen Zusammenstellen geachtet werden sollte. Außerdem soll es die Grenzen der Klassifikationsfähigkeit des MVCNNs aufzeigen und in welcher Richtung dieses in zukünftigen Arbeiten weiter getestet werden kann.

5.2 Ablauf

Der Ablauf aller Testreihen beinhaltet dieselben Schritte, wobei die Parameter mancher dieser Schritte angepasst werden, basierend auf den Ergebnissen der vorhergehenden Testreihe. Diese Schritte umfassen:

1. Zuerst muss ein Ursprung für die zu generierenden Szenen gewählt werden. Dies kann ein OCT-Scan sein, welcher als Vorlage für die zu generierende Szene verwendet wird. Für die Fälle diese Arbeit werden oft Szenen lediglich nach Anforderungen erstellt, wie ”Eine Szene, welche nur aus Ödemen besteht”.

2. Die Parameter der Randomisierung der Szenengenerierung werden festgelegt. Diese ändern sich nicht innerhalb einer Testreihe.

3. Der Scenebuilder wird verwendet, um eine Szene zu erstellen. Der Ablauf des Erstellens besteht aus dem Justieren der grundlegenden Einstellungen wie den Eigenschaften der Epithelschichten, dem Erstellen der Objekte und dem Manipulieren dieser, bis die Szene die gewünschte Erscheinung hat. Für die hier erstellten Testreihen wurde der foveale Zylinder dieser Szenen immer an derselben Positionen mit derselben Skalierung belassen. (siehe Kapitel 3.3)

4. Nachdem eine oder mehrere erstellte Szenen gespeichert wurden, werden weitere Szenen prozedural generiert. Hierfür wird eine der intellektuell erstellten Szenen ausgewählt und die Parameter für die Randomisierung der Elemente der Szenen aus Schritt 2. werden verwendet. Mit diesen Einstellungen wird die benötigte Menge von Szene generiert. Die Randomisierung der Werte funktioniert ausgehend von den Ursprungswerten der Objekte. Eine 50-prozentige Randomisierung der Skalierung erlaubt Objekten mit einer Skalierung von 0,5 zwischen 0,25 und 0,75 zu schwanken. Für die Position bedeutet eine Randomisierung von 50%, dass das Objekt sich von seiner Ursprungsposition um 50% der maximalen X- und Y-Koordinate der Epithelschicht bewegen kann. Diese Randomisierung funktioniert gleichverteilt.

5. Die generierten Szenen müssen in Szenen mit einem hohen und Szenen mit einem niedrigen Volumen innerhalb des fovealen Zylinders geteilt werden. Deshalb muss ein Schwellenwert bestimmt werden, welcher die Grenze zwischen hohem und niedrigen Volumen definiert. Hierfür werden die Drusen- und Ödem-Volumen innerhalb des Zylinders jeweils für jede Szene zu einem Wert summiert und diese Werte werden in einer Liste geordnet gespeichert. Basierend auf diesen Werten wird ein Histogramm erstellt. Mit diesem Histogramm wird ein Schwellenwert bestimmt, welcher sich in der Nähe des Medians der Werte befindet. Der Schwellenwert wird für jede Testreihe individuell festgelegt.

6. Es werden Kamerastacks für jede Szene erstellt. Jeder Kamerastack besteht aus den Bildern der 20 Kamerapositionen, welche jeweils zu einem View-CNN gehören. Wie dieser Kamerastack gelabelt wird, hängt von dem gesamten Zylinder-Volumen der jeweiligen Szene ab. Dieses wird mit dem gewählten Schwellenwert verglichen und die Kamerastacks werden abhängig von diesem Vergleich in dem ”high Volume” und ”low Volume” Ordner gespeichert. Diese

Ordner dienen als die Labels für das MVCNN. Hierbei ist es wichtig, dass der foveale Zylinder nicht auf den Bildern der Kameras zu sehen ist.

7. Das MVCNN wird mit den Kamerastacks trainiert. Die Kamerastacks werden automatisch in Trainings- und Testdatensatz unterteilt. Das MVCNN trainiert erst die View-CNNs, wendet die trainierten CNNs dann erneut auf den Datensatz an und kombiniert die Zwischenergebnisse vor der letzten Convolutional Layer [SMKLM15]. Mit diesen Zwischenergebnissen wird das Final-CNN trainiert. Während des Trainings werden automatisch Tabellen generiert, welche die Ergebnisse des Trainings festhalten. Die Ergebnisse dieses Trainings informieren die Parameter der nächsten Testreihe.

5.3 Testaufbau und Ergebnisse

In diesem Unterkapitel werden die Testreihen, der Inhalt ihrer Tests und die Ergebnisse dieser beschrieben. Hierfür werden zuerst die Parameter der Szenengenerierung der Testreihe beschrieben, welche für jeden Test die gleichen sind. Anschließend wird für jeden Test innerhalb der Testreihe beschrieben, wie die Ausgangsszene beschaffen ist und was das Ziel dieses Tests ist bzw. welche Herausforderung er darstellen soll. Dies wird gefolgt von den Ergebnissen des jeweiligen Tests. Diese werden dargestellt in Form eines Diagramms, welches die Trainings- und Testaccuracy sowie den Loss des Final-CNNs pro gesehene Kamerastacks zeigt sowie einer Analyse dieser Werte.

Dieser Abschnitt führt die Parameter auf, welche über alle Tests gelten. Wenn nicht anders erwähnt, werden 400 Kamerastacks pro Test generiert. Die Bilder dieser Kamerastacks werden mit einer Größe von 100 mal 100 Pixeln erstellt. Das MVCNN teilt die Kamerastacks automatisch in ein Verhältnis von 70% Trainings- zu 30% Testdaten auf. Es wird eine Batchsize von 20 Kamerastacks verwendet, was bedeutet, dass jedes View-CNN mit einer Batchsize von 20 Bildern arbeitet.

5.3.1 Testreihe 1

Die erste Testreihe hat das Ziel, einen ersten Einblick in die von dem MVCNN erreichte Accuracy zu geben und was für dieses die größten Schwierigkeiten aufweist.

Zu diesem Zweck werden verschiedene Ausgangsszenen mit den gleichen Parametern zur Randomisierung der Szenegenerierung übergeben. Die Ausgangsszenen wurden intellektuell erstellt und die Objekte in ihnen so platziert, dass mehrere sich innerhalb des fovealen Zylinders befinden. Die ersten Ausgangsszenen sind eine Szene mit mehreren Ödemen und eine Szene mit mehreren Drusen. Diese sollen unter anderem testen, ob ein signifikanter Unterschied zwischen dem Verwenden von Drusen und Ödemen besteht. Die letzte Szene ist eine Mischung aus Ödemen und Drusen und soll prüfen, ob das Mischen der Objekte einen signifikanten Unterschied in der Klassifikationsgüte mit sich bringt. Das MVCNN wird für 4 Epochen trainiert.

Parameter	Randomisierung
Position	5 %
Rotation & Skalierung	10 %
Objekt Anzahl	1

Tabelle 5.1: Die Parameter der Szenengenerierung von Testreihe 1 und die Werte der Randomisierung dieser. Die Objektanzahl beschreibt, wie viele Objekte für jedes in der Ausgangsszene vorhandene Objekt generiert werden können.

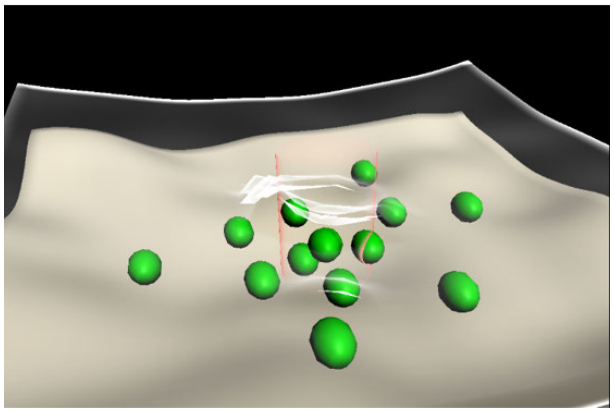

Abbildung 5.1: Eine Ausgangsszene für die Szenengenerierung mit einer hohen Anzahl von Ödem-Objekten (siehe Test 1onlyEdema).

1. Nur Ödeme (1onlyEdema) Aufbau Die Ausgangsszene besteht aus 14 Ödem-Objekten gleicher Größe und Rotation. Die Herausforderung dieser Ausgangsszene liegt in der hohen Anzahl der Objekte. Außerdem soll festgestellt werden, ob ein signifikanter Unterschied in der Klassifikationsgüte des MVCNNs zwischen dieser und der Ausgangsszene in 1onlyDruses besteht.

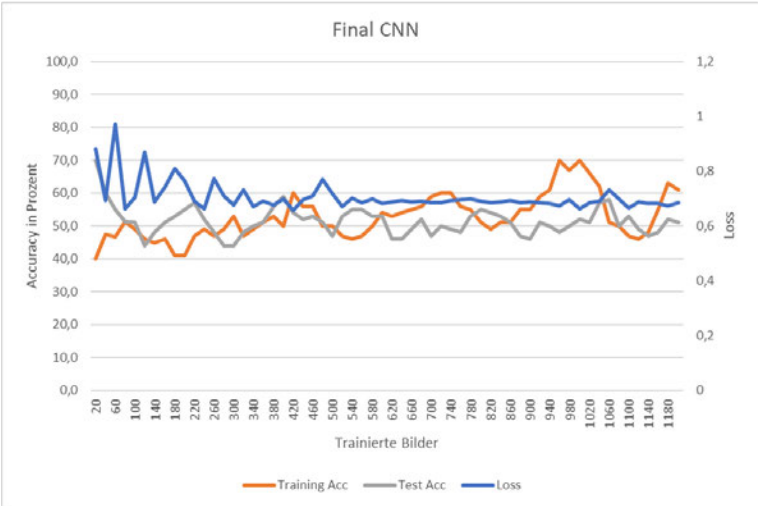

Abbildung 5.2: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 1onlyEdema. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

1. Nur Ödeme (1onlyEdema) Ergebnis Die nur aus Ödemen bestehenden Szenen führen zu einer Trainings- und Testaccuracy, welche nicht stark über 50% hinausgeht. Es ist außerdem basierend auf den Schwankungen möglich, dass es sich hierbei um ein lokales Maximum handelt und die Accuracy sich mit weiteren Epochen wieder reduzieren würde. Dieses Ergebnis ist sehr gering für ein 2-Klassenproblem, da bei einem solchen ein zufälliges Auswählen in einer ungefähr 50-prozentigen Accuracy resultieren sollte.

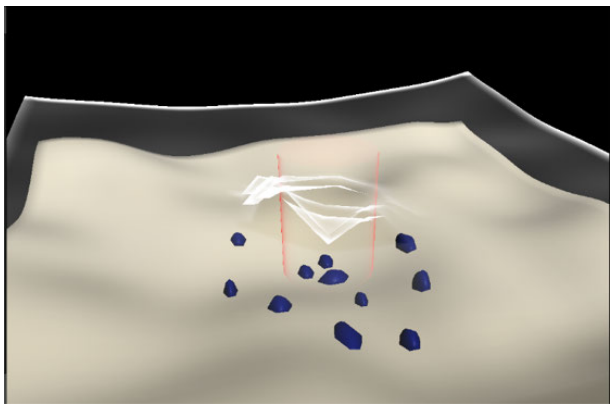

Abbildung 5.3: Eine Ausgangsszene für die Szenengenerierung mit einer hohen Anzahl von Drusen-Objekten (siehe Test 1onlyDruses).

1. Nur Drusen (1onlyDruses) Aufbau Die Ausgangsszene besteht aus 11 Drusen-Objekten gleicher Größe und Rotation. Die Herausforderung dieser Ausgangsszene liegt in der hohen Anzahl der Objekte. Außerdem soll festgestellt werden, ob ein signifikanter Unterschied in der Klassifikationsgüte des MVCNNs zwischen dieser und der Ausgangsszene in 1onlyEdema besteht.

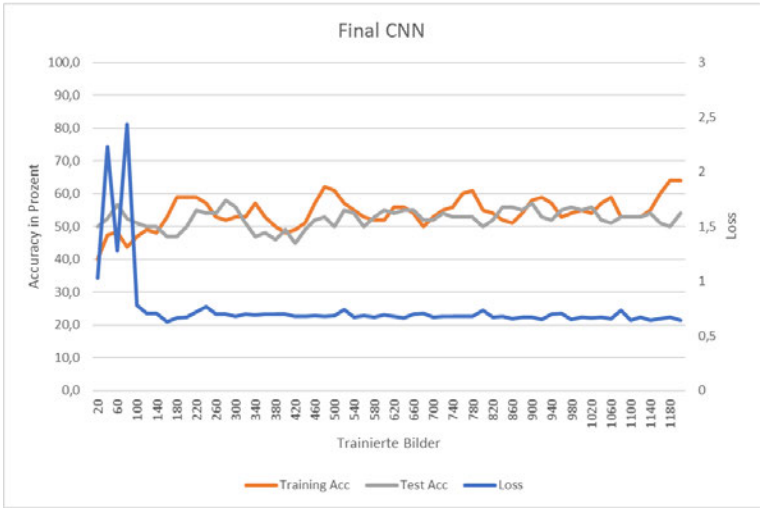

Abbildung 5.4: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 1onlyDruses. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

1. Nur Drusen (1onlyDruses) Ergebnis Die Szenen, welche nur aus einer Vielzahl von Drusen bestehen, verhalten sich in ihrer Accuracy sehr ähnlich wie die Ergebnisse von Test onlyEdema1, (siehe Ergebnis 1onlyEdema). Sie unterscheiden sich in einem anfänglich stark schwankenden Loss, welcher sich später auf dieselben Level wie in 1onlyEdema reduziert.

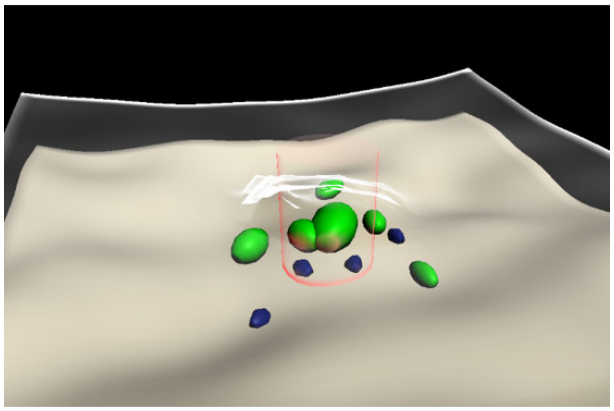

Abbildung 5.5: Eine Ausgangsszene für die Szenengenerierung mit einer hohen Anzahl von Ödem- und Drusen-Objekten (siehe Test 1mixed).

1. Gemischt (1mixed) Aufbau Die Ausgangsszene besteht aus 6 Ödem-Objekten und 4 Drusen-Objekten. Mit unterschiedlichen Rotation und Skalierungen. Diese Szene soll mit unterschiedlichen Objekten mit unterschiedlichen Parametern annähernd die Herausforderungen von auf Realwelt Daten basierenden Szenen simulieren.

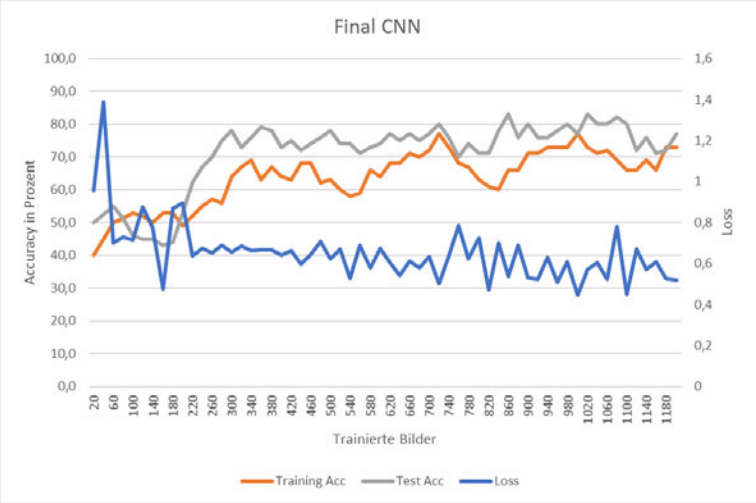

Abbildung 5.6: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 1mixed. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

1. Gemischt (1mixed) Ergebnis Die Szenen mit Drusen und Ödemen erreichen eine bessere Accuracy wie die Szenen mit nur Ödemen 1onlyEdema oder nur Drusen 1onlyDruses. Entscheidend ist hier nicht nur, dass die Accuracy höhere Werte erreicht, sondern auch, dass diese das Ergebnis eines insgesamt steigenden Trends sind und dadurch mit großer Wahrscheinlichkeit nicht nur ein lokales Maximum, sondern eine tatsächliche Verbesserung der Accuracy des MVCNN bedeuten. Dies wird ebenfalls durch den fallenden Loss bestätigt.

5.3.2 Testreihe 2

Ziel der zweiten Testreihe ist es zu testen, ob die Accuracy des MVCNN sich verbessert, sollten die Szenen weniger Objekte beinhalten. Gleichzeitig sollen die Unterschiede zwischen den generierten Szenen verstärkt werden, durch erhöhte Randomi-

sierungswerte der Szenengenerierung. Für die zweite Testreihe wird, wie in Tabelle 5.2 zu sehen, die Randomisierung der Skalierung und Rotation stark erhöht. Außerdem wird dafür gesorgt, dass jedes Objekt, welches in der jeweiligen Ausgangsszene vorhanden ist, eine 50% Wahrscheinlichkeit hat nicht in einer generierten Szene zu erscheinen. Es werden die Ausgangsszenen von Testreihe 1 verwendet, allerdings werden mehrere Objekte aus jeder Ausgangsszene gelöscht, um sie zu simplifizieren. Das MVCNN wird anschließend erneut mit denselben Einstellungen wie in Testreihe 1 trainiert.

Parameter	Randomisierung
Position	5 %
Rotation & Skalierung	50 %
Objekt Anzahl	0 - 1

Tabelle 5.2: Die Parameter der Szenengenerierung von Testreihe 2 und die Werte der Randomisierung dieser. Die Objektanzahl beschreibt, wie viele Objekte für jedes in der Ausgangsszene vorhandene Objekt generiert werden können.

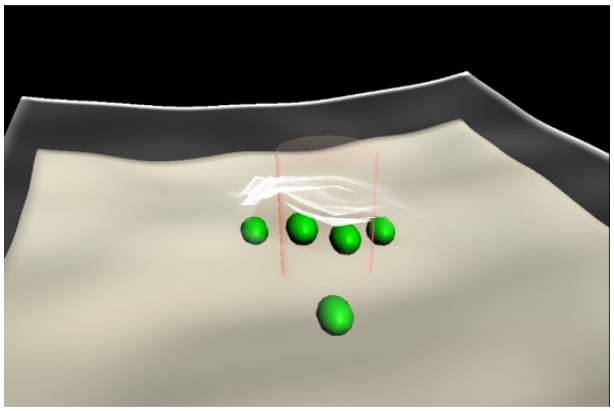

Abbildung 5.7: Eine Ausgangsszene für die Szenengenerierung mit einer geringen Anzahl von Ödem-Objekten (siehe Test 2onlyEdema).

2. Nur Ödeme (2onlyEdema) Aufbau Die Ausgangsszene besteht aus 5 Ödem-Objekten gleicher Größe und Rotation. Diese Szene soll eine simplifizierte Herausforderung von 1onlyEdema darstellen. Außerdem soll festgestellt werden, ob ein signifikanter Unterschied in der Klassifikationsgüte des MVCNNs zwischen dieser und Ausgangsszene 2onlyDruses besteht.

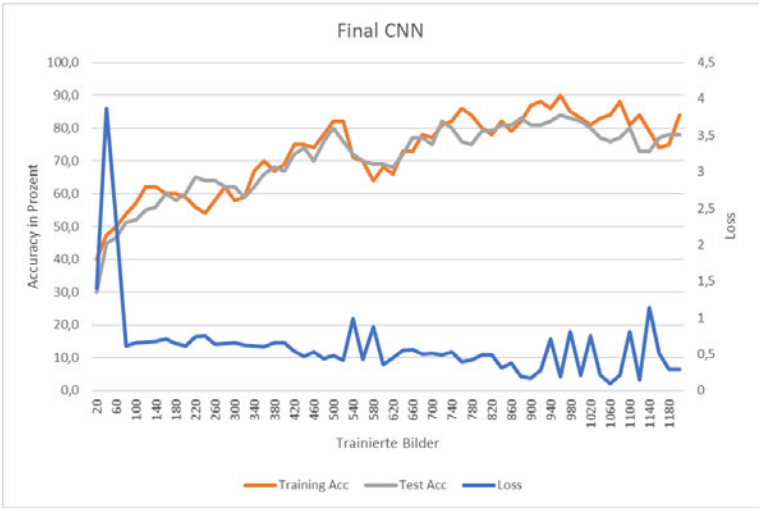

Abbildung 5.8: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 2onlyEdema. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

2. Nur Ödeme (2onlyEdema) Ergebnis Die Ergebnisse der Szenen, welche ausschließlich Ödeme beinhalten, zeigen sich stark verbessert, nachdem die Anzahl der Objekte reduziert wurde verglichen mit den Ergebnissen von Test 1onlyEdema. Die Trainings- und Testaccuracy erreichen Werte um die 80%.

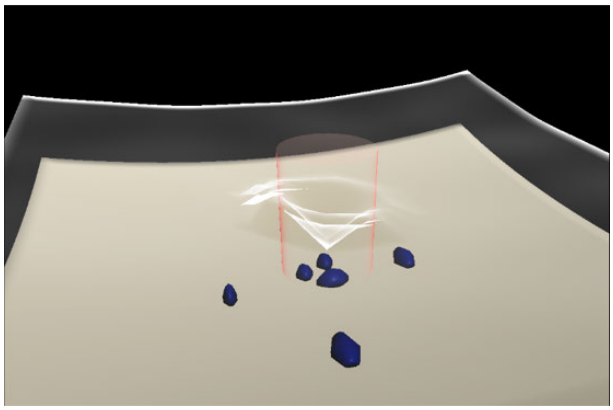

Abbildung 5.9: Eine Ausgangsszene für die Szenengenerierung mit einer geringen Anzahl von Drusen-Objekten (siehe Test 2onlyDruses).

2. Nur Drusen (2onlyDruses) Aufbau Die Ausgangsszene besteht aus 6 Drusen-Objekten gleicher Größe und Rotation. Diese Szene soll eine simplifizierte

Herausforderung von 1onlyDruses darstellen. Außerdem soll festgestellt werden, ob ein signifikanter Unterschied in der Klassifikationsgüte des MVCNNs zwischen dieser und Ausgangsszene 2onlyEdema besteht.

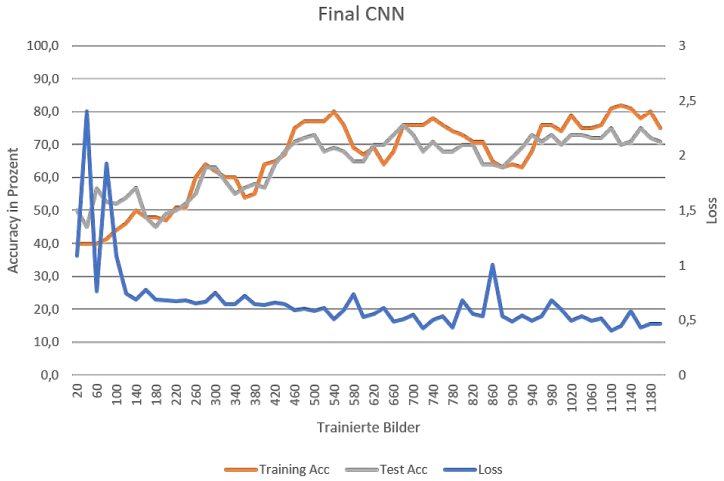

Abbildung 5.10: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 2onlyDruses. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

2. Nur Drusen (2onlyDruses) Ergebnis Die Ergebnisse der Szenen, welche ausschließlich Drusen beinhalten, zeigen sich ebenfalls stark verbessert, nachdem die Anzahl der Objekte reduziert wurde, verglichen mit den Ergebnissen von Test 1mixed. Während die Trainings- und Testaccuracy geringere Werte als 2onlyEdema erreichen, sinkt der Loss hier mit weniger Schwankungen.

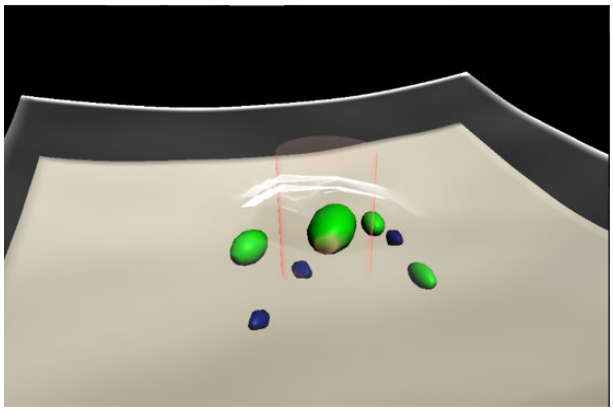

Abbildung 5.11: Eine Ausgangsszene für die Szenengenerierung mit einer geringen Anzahl von Ödem- und Drusen-Objekten (siehe Test 2mixed).

2. Gemischt (2mixed) Aufbau Die Ausgangsszene besteht aus 4 Ödem-Objekten und 3 Drusen-Objekten. Mit unterschiedlichen Rotation und Skalierungen. Diese Szene soll mit unterschiedlichen Objekten mit unterschiedlichen Parametern annähernd die Herausforderungen von auf Realwelt Daten basierenden Szenen simulieren, aber mit einem geringeren Schwierigkeitsgrad als 1mixed.

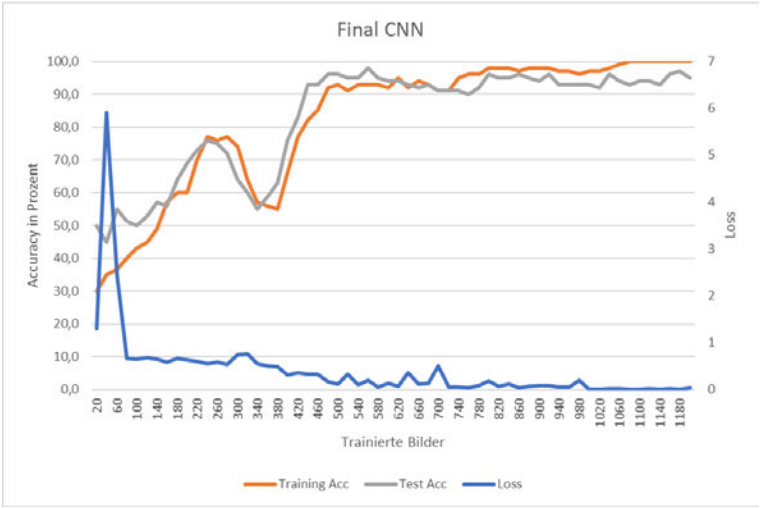

Abbildung 5.12: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 2mixed. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

2. Gemischt (2mixed) Ergebnis Die Ergebnisse der Szenen, welche Ödeme und Drusen mischen, zeigen sich genau, wie die restlichen Tests von Testreihe 2 stark verbessert, nachdem die Anzahl der Objekte reduziert wurde, verglichen mit den Ergebnissen von Test 1mixed. Die Trainings- und Testaccuracy erreichen Werte von über 90% und sehr geringen Loss, verglichen mit den vorhergegangenen Tests.

5.3.3 Testreihe 3

Ziel dieser Testreihe ist es, die generierten Szenen unabhängig von der Objekt-Konfiguration einer jeweiligen Ausgangsszene zu machen. Durch diese Änderung soll der Parameter Raum gleichverteilt exploriert werden, anstatt dass nur die Bereiche in der unmittelbaren Nähe der originalen Objekte potenziell Objekte beinhalten können. Deshalb wird für diese Testreihe ein stärkerer Fokus auf die prozedurale Szenengenerierung gelegt. Dies bedeutet, dass die Ausgangsszene aus lediglich einer Druse und einem Ödem in der Mitte der Szene besteht. Die Objekte werden durch die in 3.2.9 beschriebenen Muster generiert, welche zufällig für jede zu generierende Szene gewählt werden. Diese Testreihe wird mit geringen Leveln von Randomisierung durchgeführt (siehe Tabelle 5.3), wodurch die möglichen Positionen, Rotation und Skalierungen stärker begrenzt werden. Dies soll der folgenden Testreihe 4 gegenüber gestellt werden, welche eine sehr viel stärkere Randomisierung und damit größere Unterschiede zwischen den Szenen aufweist.

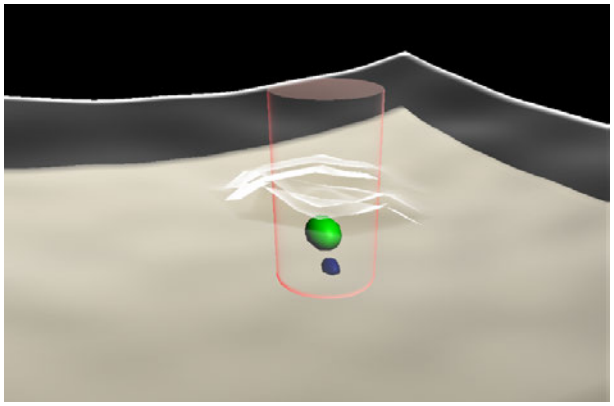

Abbildung 5.13: Eine Ausgangsszene für die Szenengenerierung bestehend aus einem einzigen Ödem- und Drusen-Objekt.

Bevor die Szenen mit den erwähnten Mustern generiert werden, wird die Generierung mit einem einzelnen Ödem getestet 3oneEdema, mit der in Tabelle 5.3 beschriebenen Randomisierung, abgesehen von der Objekt-Anzahl. Grund hierfür sind die Ergebnisse der vorherigen Testreihen, welche keine sehr hohe Accuracy erreichten. Deshalb soll der Prozess mit einem möglichst simplen Beispiel erneut getestet werden, bevor die erwähnten Muster verwendet werden. Die Anzahl der Epochen, für welche das MVCNN trainiert wird, wird auf 9 erhöht im Vergleich zu Testreihe 1 und 2. Einerseits um sicherzustellen, dass das Training nicht vor einer möglichen signifikanten Verbesserung beendet wurde, andererseits um zu vermeiden, dass ein möglicher Höhepunkt der Accuracy nicht lediglich ein lokales Maximum darstellt. Außerdem soll überprüft werden, ob durch eine höhere Anzahl von Kamerastacks die Accuracy verbessert werden kann.

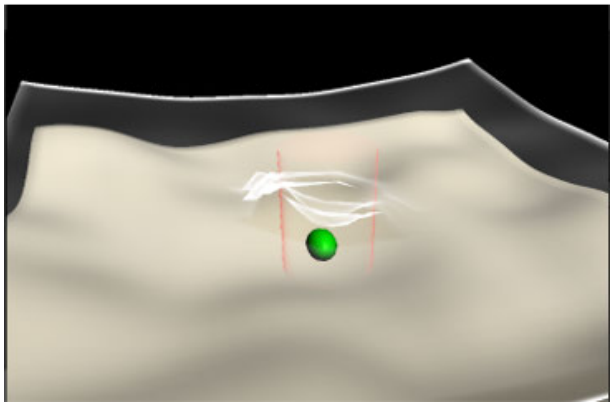

Abbildung 5.14: Eine Ausgangsszene für die Szenengenerierung mit einem einzelnen von Ödem-Objekt.

Parameter	Randomisierung
Position	25 %
Rotation & Skalierung	25 %
Objekt Anzahl	1 - 9

Tabelle 5.3: Die Parameter der Szenengenerierung von Testreihe 3 und die Werte der Randomisierung dieser. Die Objektanzahl beschreibt, wie viele Objekte für jedes in der Ausgangsszene vorhandene Objekt generiert werden können.

3. Ein Ödem-Objekt (3oneEdema) Aufbau Die Ausgangsszene besteht aus einem einzigen Ödem-Objekt, welches in der Mitte des fovealen Zylinders platziert ist (siehe Abbildung 5.14). Dies soll eine einfache Herausforderung für das MVCNN darstellen.

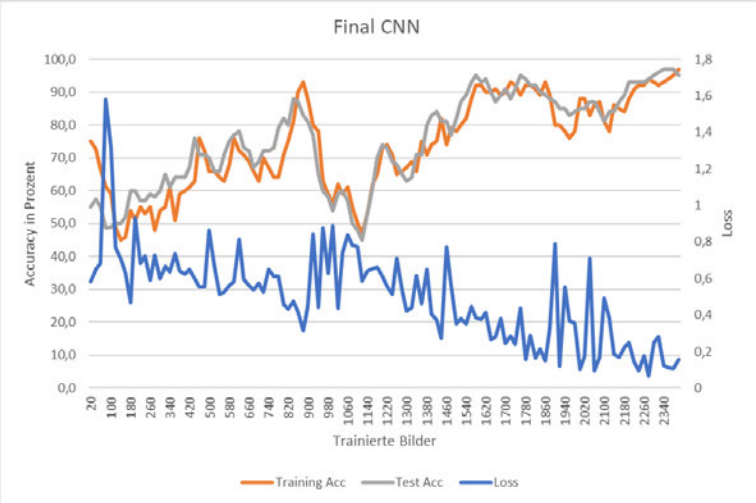

Abbildung 5.15: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 3oneEdema. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

3. Ein Ödem-Objekt (3oneEdema) Ergebnis Test 3oneEdema sollte eine der simpelsten Herausforderungen darstellen. Die Schwankungen der Accuracy sind sehr stark, aber durch ihre insgesamt steigende Tendenz kombiniert mit der fallenden Tendenz des Loss ist es trotzdem wahrscheinlich, dass das MVCNN einen tatsächlichen Lerneffekt erfährt.

3. Gemischt (3mixed) Aufbau Die Ausgangsszene besteht aus einem Ödem-Objekt und einem Drusen-Objekt, welche in der Mitte des fovealen Zylinders platziert sind (siehe Abbildung 5.13). Die Objekte der Szenen werden durch die in Unterkapitel 3.2.9 erwähnten Muster generiert. Ziel soll es hier sein, die Positionen der Objekte der generierten Szenen unabhängig von den Objekt-Positionen der Ausgangsszene zu generieren. Dies soll die Ergebnisse des MVCNN unabhängig von bestimmten Konfigurationen von Ausgangsszenen gestalten, welche dieses möglicherweise leichter lernen kann als andere.

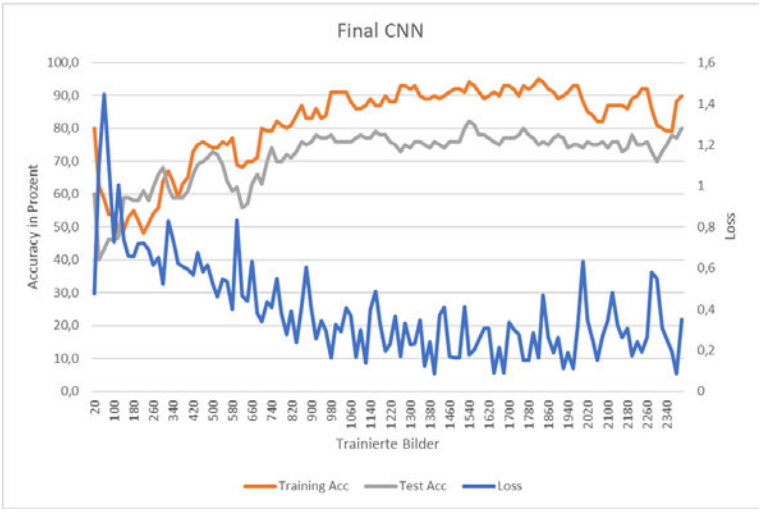

Abbildung 5.16: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 3mixed. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

3. Gemischt (3mixed) Ergebnis Test 3mixed verwendet eine geringe Randomisierung für die Szenengenerierung. Dies führt zu einer Trainings-Accuracy, welche nach ungefähr 5 Epochen ihr Maximum zu erreichen scheint. Dies spiegelt sich in der Loss-Funktion wider, welche sich nach diesem Punkt erhöht. Die Test-Accuracy scheint schon vorher einen Punkt zu erreichen, nach welchem sie sich nicht weiter verbessert (siehe Abbildung 5.16). Außerdem zeichnet sich die Testaccuracy als deutlich geringer gegenüber der Trainingsaccuracy ab.

3. Gemischt 800 Kamerastacks (3mixed800) Aufbau Die Ausgangsszene ist die selbe wie in 3mixed. Der einzige Unterschied dieses Test ist, dass basierend auf denselben Parametern, eine höhere Anzahl von Kamerastacks, namentlich 800, generiert wird. Dies soll überprüfen, ob das MVCNN mit einem besser explorierten Parameterraum eine höhere Accuracy erreicht.

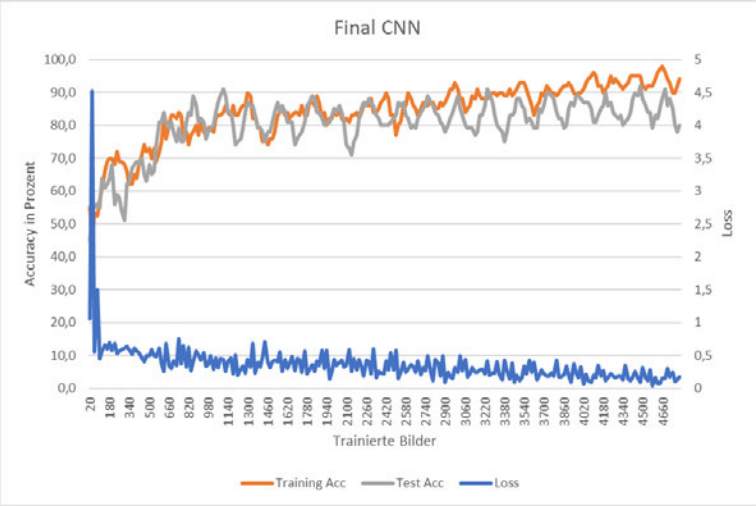

Abbildung 5.17: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 3mixed800. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

3. Gemischt 800 Kamerastacks (3mixed800) Ergebnis Test 3mixed800 ist eine Wiederholung von Test 3mixed mit dem Unterschied, dass die Szenengenerierung 800 Kamerastacks generiert. Die Trainings-Accuracy steigt in diesem Fall auch bis über 90%. Der Loss scheint sehr viel zuverlässiger zu fallen als in 3mixed.

5.3.4 Testreihe 4

Die vierte Testreihe verwendet dieselbe Ausgangsszene wie Testreihe 3. Genau wie diese soll Testreihe 4 die Szenengenerierung mit den Mustern aus Unterkapitel 3.2.9 verwenden. Das Ziel von Testreihe 4 ist es, die Tests von Testreihe 3 mit erhöhter Randomisierung zu wiederholen. Dies soll festzustellen, ob durch größere Unterschiede zwischen den einzelnen Trainingsdaten die Accuracy des MVCNN verbessert werden kann. Der Test mit einem einzigen Objekt und ohne Muster wird ebenfalls mit den erhöhten Randomisierungswerten wiederholt.

Parameter	Randomisierung
Position	50 %
Rotation & Skalierung	50 %
Objekt Anzahl	1 - 9

Tabelle 5.4: Die Parameter der Szenengenerierung von Testreihe 4 und die Werte der Randomisierung dieser. Die Objektanzahl beschreibt, wie viele Objekte für jedes in der Ausgangsszene vorhandene Objekt generiert werden können.

4. Ein Ödem-Objekt (4oneEdema) Aufbau Die Ausgangsszene ist dieselbe wie in 3oneEdema. Der Unterschied liegt lediglich in der Randomisierung der Position, Rotation und Skalierung der Szenengenerierung. Ziel ist es hier, zu überprüfen, ob eine größere Randomisierung der Werte und dadurch größere Unterschiede zwischen den Szenen die Accuracy des MVCNN verbessern.

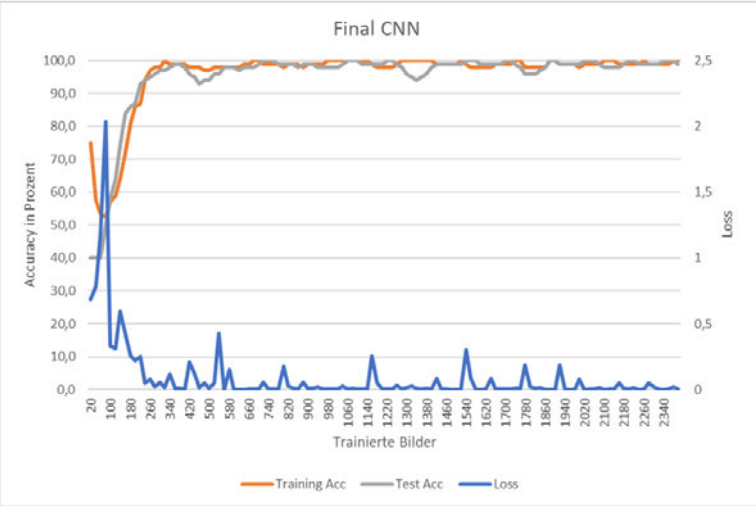

Abbildung 5.18: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 4oneEdema. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

4. Ein Ödem-Objekt (4oneEdema) Ergebnis Test 4oneEdema wiederholt Test 3oneEdema mit einer erhöhten Randomisierung von Position, Rotation und Skalierung. Die Accuracy erreicht in diesem Fall nach weniger als einer Epoche

bereits fast 100%, während der Loss sich ähnlich schnell verringert. Dies entspricht der Erwartung für ein simples Beispiel.

4. Gemischt (4mixed) Aufbau Die Ausgangsszene ist dieselbe wie in 3mixed. Wie in 4oneEdema bereits beschrieben, ist das Ziel dieses Tests, den Effekt einer größeren Randomisierung der Parameter auf die Accuracy zu testen.

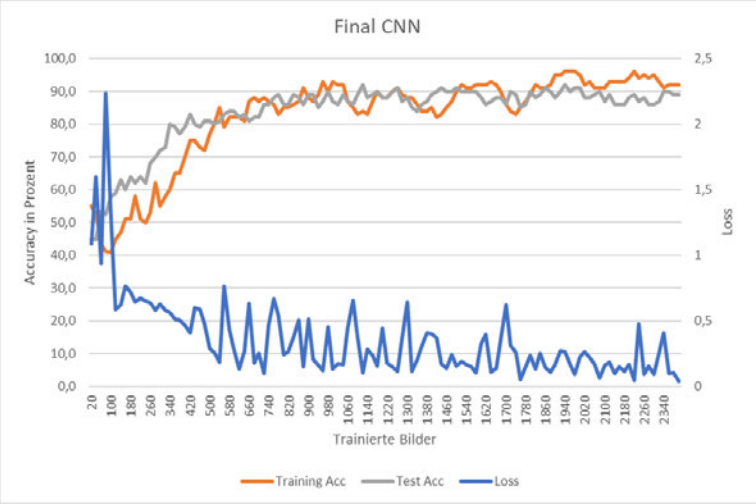

Abbildung 5.19: Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 4mixed. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

4. Gemischt (4mixed) Ergebnis Test 4mixed wiederholt Test 3mixed mit einer erhöhten Randomisierung von Position, Rotation und Skalierung. Die Ergebnisse dieses Tests verhalten sich besser als 3mixed. Vor allem die Testaccuracy, welche der Trainingsaccuracy mehr gleicht und wie diese um 90% liegt.

4. Gemischt 800 Kamerastacks (4mixed800) Aufbau Die Ausgangsszene ist dieselbe wie in 3mixed800. Dieser Test soll genau wie 3mixed überprüfen ob die erhöhte Randomisierung von 4mixed durch eine höhere Kamerastackanzahl von 800 verbessert wird.

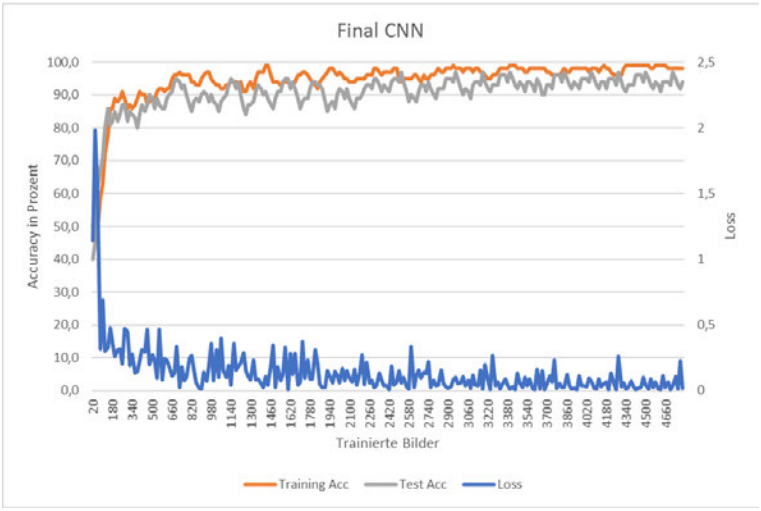

Abbildung 5.20: (fehler in x-achse)Ein Diagramm der Trainingswerte des MVCNNs mit den Daten basierend auf Test 4mixed800. Das Diagramm zeigt die Test- und Trainingsaccuracy sowie die Lossfunktion abhängig von der Anzahl der Bilder, welche für das Training verwendet wurden.

4. Gemischt 800 Kamerastacks (4mixed800) Ergebnis Test 4mixed800 wiederholt Test 4mixed mit 800 Kamerastacks. Auch hier zeigt sich eine Verbesserung der Accuracy, wobei dieser Effekt sich geringer für die Testaccuracy zeigt, welche meist unter der Trainingsaccuracy liegt.

5.4 Interpretation

Die Ergebnisse der vier Testreihen begründen mehrere Vermutungen. Die ersten zwei Testreihen hatten das grundlegende Feststellen der Machbarkeit des Szenen-Klassifizierens zum Ziel. Dies scheint sich bestätigt zu haben, was man anhand der Ergebnisse für ein einzelnes Objekt sehen kann. Außerdem deuten die Ergebnisse der ersten zwei Testreihen zusammen darauf hin, dass komplexere Szenen mit einer größeren Anzahl von Objekten schwieriger zu klassifizieren sind, was sich zum Beispiel am Unterschied zwischen Ergebnis 1onlyEdema und 2onlyEdema erkennbar macht.

Für die dritte und vierte Testreihe wurde die prozedurale Generierung verstärkt benutzt, um die Ergebnisse weniger abhängig von einer gewählten Ausgangsszene zu machen und gleichzeitig die Menge der Objekte auf sinnvolle Werte zu beschränken.

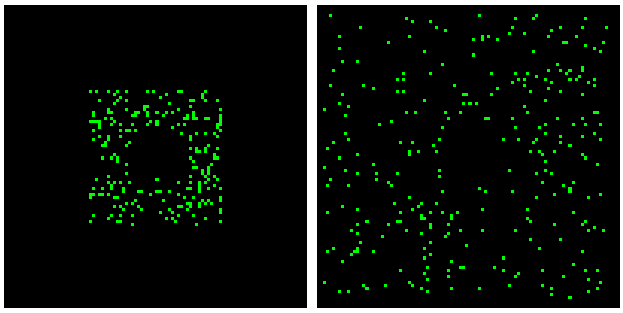

Abbildung 5.21: Links zeigt die Low-Volume-Heatmap der Szenengenerierung des Tests 3oneEdema, rechts die Low-Volume-Heatmap von 4oneEdema. 4oneEdema wurde mit einem höheren Grad von Randomisierung generiert, unter anderem in der Position der Objekte, was deutlich in der Heatmap erkenntlich wird.

Die Ergebnisse des ersten Tests mit einem einzigen Ödem-Objekt (siehe Ergebnis 3oneEdema) innerhalb der dritten Testreihe, waren überraschend darin, dass das MVCNN einige Epochen benötigt hat, um auf eine 90-prozentige Accuracy zu kommen. Vor allem wenn dieses Ergebnis mit dem Ergebnis des Tests mit einem Objekt aus Testreihe 4 verglichen wird (siehe Ergebnis 4oneEdema). Während erwartet war, dass der Test innerhalb Testreihe 4 zu einem besseren Ergebnis führt, ist der Grad dieser Verbesserung stärker als erwartet. Eine mögliche Erklärung gibt Abbildung 5.21, welche die Heatmaps von 3oneEdema und 4oneEdema zeigt und die Unterschiede, welche durch die erhöhte Randomisierung entstehen, deutlich macht.

Überraschend innerhalb der Test war die signifikante Verbesserung der Ergebnisse innerhalb von Testreihe 3 und 4 durch die erhöhte Anzahl von generierten Kamerastacks. Während eine solche Verbesserung an sich nicht unerwartet ist, da 400 Kamerastacks nicht unbedingt ausreichend sein müssen, um den Parameterraum abzudecken, ist die Stärke der Verbesserung eine nähere Untersuchung wert. Zuerst werden die Ergebnisse der Generierung untersucht, um festzustellen, ob ein Fehler in der Randomisierung der Daten die Ursache ist. Abbildung 5.22 zeigt die Heatmaps von 3mixed und 3mixed800. Die möglichen Positionen sind, wie zu erwarten, besser abgedeckt in 3mixed800. Die Verteilung von 3mixed sieht jedoch nicht so aus,

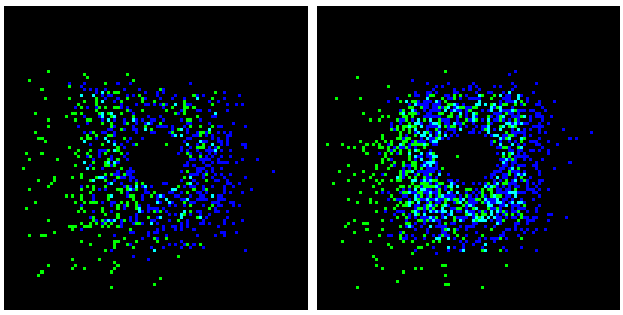

Abbildung 5.22: Links zeigt die Low-Volume-Heatmap der Szenengenerierung des Tests 3mixed, rechts die Low-Volume-Heatmap von 3mixed800. Für 3mixed800 wurde die doppelte Menge von Kamerastacks generiert, was durch die erhöhte Menge von Punkten und die verstärkte Intensität der Farbe in dichten Bereichen erkenntlich wird.

als ob ein schwerwiegender Fehler vorliegt, welcher die Ausbreitung der Verteilung reduziert. Es gilt auch zu bedenken, dass es sich hier um ein MVCNN handelt welches aus 20 CNNs pro Kamera und einem finalen CNN besteht. Es ist möglich, dass ein MVCNN eine größere Datenmenge für zuverlässige Aussagen benötigt, um dafür zu sorgen, dass eine ausreichende Menge der CNNs eine zuverlässige Aussage treffen können und die Accuracy des finalen CNNs nicht negativ beeinflussen. Diese Möglichkeit könnte in weiteren Arbeiten untersucht werden.

Trotz der unerwartet starken Verbesserungen durch erhöhte Kamerastackmengen weisen die Ergebnisse der letzten Tests insgesamt darauf hin, dass das MVCNN mit den gegebenen Klassifikationsproblemen gut zurechtkam. Vor allem wenn sich stark unterscheidende Szenen zur Verfügung gestellt werden können, ist das MVCNN in der Lage, dieses zuverlässig zu klassifizieren. Die Ergebnisse dieser Tests deuten stark darauf hin, dass mit ausreichend variierten Szenen und Kamerastack-Mengen auch komplexere Probleme mit mehr Klassen oder schwieriger zu identifizierenden Anomalien klassifiziert werden könnten.

6 Fazit

In diesem Kapitel werden die erstellten Tools und durch die Evaluation erfassten Ergebnisse dem Researchproposal gegenüber gestellt. Hierbei soll festgestellt werden, zu welchem Grad die gestellten Ziele erreicht wurden und wie diese sich im Lauf der Arbeit entwickelt haben.

Das Ziel dieser Arbeit war es, das innerhalb der Vorarbeit entstandene MVCNN auf komplexere Klassenprobleme anzuwenden. Diese Probleme sollten auf idealisierten 3D-Nachstellungen von OCT-Scans basieren. Um diese Nachstellungen zu erstellen, wurde das Scenebuilder-Tool entwickelt. Dieses erlaubt es, synthetische Szenen zu erstellen und die Komplexität dieser beliebig anzupassen, beispielsweise durch die Anzahl der Drusen- und Ödem-Objekte innerhalb einer Szene oder durch die Form der Epithelschicht. Als Teilproblem, an welchem die Klassifikationsfähigkeit des MVCNN getestet werden sollte, wurde das Volumen der Drusen und Ödeme in der Umgebung der fovealen Senke gewählt. Dieses Kriterium sollte entscheiden, ob ein OCT-Scan ein gesundes oder ein beeinträchtigtes Sehvermögen darstellt. Für das Teilproblem sollten parametrisierbare Datensätze erstellt und gelabelt werden. Das Scenebuilder-Tool ermöglicht den vollständigen Workflow angefangen von dem intellektuellen Erstellen einer Szene bis hin zu einem Datensatz aus gelabelten Bildsätzen. Für die Berechnung des Volumens der Objekte wurden ein neuer Algorithmus unter Verwendung von Mesh-Boolean-Operationen erstellt. Diese erlauben unter anderem die Berechnung des Volumens von Schnittstellen zwischen den Objekten mit einem auf der fovealen Senke zentrierten Zylinder. Dieser Schritt ist notwendig für das automatische Labeln der Daten als ”gesund” und ”beeinträchtigt”. Diese Datensätze wurden für eine Evaluation unterschiedlicher Parameter verwendet. Außerdem wurden Tools für das Überwachen der Generierung der Datensätze erstellt. Dies soll es erleichtern, Anomalien in der Klassifikation, welche auf Fehlern innerhalb der Daten-Generierung basieren, zu erkennen. Es besteht die Möglichkeit für das Testen weiterer Parameter. Außerdem kann ein komplexeres Klassifikationsproblem gewählt

werden wie zum Beispiel das Unterscheiden von hohen Drusen-, verglichen mit hohen Ödem-Volumen verglichen mit geringem Volumen beider Objekt-Arten.

Insgesamt wurden die meisten Anforderungen des Researchproposals erfüllt. Es besteht jedoch das Potenzial für Verbesserungen an den erstellten Tools sowie weitere Untersuchung mit durch diese erstellte Datensätze.

7 Ausblick

Die Ergebnisse dieser Arbeit könnten in zukünftigen Arbeiten Verwendung finden. Einerseits könnten die in dieser Arbeit erstellten Tools weiter verbessert und die bereits bestehenden Features für weitere Tests verwendet werden. Andererseits könnten folgende Arbeiten die Erkenntnisse dieser Arbeit als Basis verwenden.

7.1 Verbesserungen

Dieses Unterkapitel führt möglichen Verbesserungen von Features der erstellten Tools aus sowie die Tests, welche mit diesen durchgeführt werden könnten. Die hier ausgeführten Optionen stellen lediglich einen kleine, aus der Perspektive dieser Arbeit als sinnvoll erachtete Selektion der möglichen Verbesserungen dar.

Die Szenengenerierung könnte um weitere Parameter erweitert werden, welche wiederum weitere Tests und Vergleiche ermöglichen würden. Mögliche Parameter wären: unterschiedliches Shading, die Wahrscheinlichkeit, dass Objekte sich gruppieren oder stark verteilt sind, weitere Muster für Objekt-Beziehungen, usw.

Die Benutzerfreundlichkeit ist ein Bereich, welcher für die Tools in dieser Arbeit nur auf dem grundlegendsten Level befriedigt wurde. Features wie die Handels der Objektmanipulierung sind ein Beispiel dafür, wie das Verwenden der Tools erleichtert werden kann. Die Tools könnten in dieser Hinsicht verbessert werden durch Features wie eine moderne, sinnvoll gestaltete UI, das Einstellen von Keybindings, usw. Dies würde nicht direkt die Ergebnisse der Szenengenerierung verbessern, aber würde das längere Arbeiten mit den Tools erleichtern und somit das Erstellen einer größeren Zahl von Ausgangsszenen vereinfachen und die pro Szene benötigte Zeit reduzieren.

Das Klassifikationsproblem könnte zu einem Problem mit mehr als zwei Klassen geändert werden. Dies würde die Fähigkeiten des MVCNN weiter ausschöpfen und

es ermöglichen, das Verhalten dieses unter erschwerten Bedingungen zu beobachten. Ein weiterer Ansatz wäre, nicht die Summe des Volumens innerhalb des Zylinders als entscheidende Variable zu wählen, sondern das Verhältnis des Volumens außerhalb des Zylinders verglichen mit dem innerhalb.

7.2 Weitere Verwendung

Die Evaluation hat verschiedene Aspekte der Daten aufgezeigt, welche einen Effekt auf die Klassifikationsgüte aufweisen. Weitere Arbeiten könnten untersuchen, wie OCT-Scans, welche oft schwer in einzelne Objekte wie Drusen und Ödeme aufteilbar sind, zu 3D-Szenen bestehend aus eindeutigen Objekten umgewandelt werden könnten. Es könnten außerdem weitere Untersuchungen in die Fähigkeit des MVCNN gemacht werden und wie dieses verbessert werden kann. Dies könnten simple Änderungen wie eine kleinere oder größere Kamera-Anzahl sein oder tiefere Eingriffe in die Architektur der verwendeten CNNs.

Insgesamt soll diese Arbeit einen Schritt darstellen in Richtung eines durch maschinelle Lernverfahren gestützten Vorhersage-Tools für Sehstörungen. Ein solches Tool sollte in der Lage sein, OCT-Scans von Menschen, welche noch nicht in ihrer Sehfähigkeit beeinträchtigt sind, zu untersuchen und eine Wahrscheinlichkeit für das Erscheinen von Beeinträchtigungen zu geben. Ein Tool wie das in dieser Arbeit untersuchte MVCNN stellt eine wichtige Vorstufe für ein solches Tool dar. Die in dieser Arbeit erstellten Tools für das Generieren von Datensätzen für die Evaluation dieses MVCNNs sind deshalb ein signifikantes Zwischenergebnis für dieses Endziel.

Literaturverzeichnis

[AECM08] Eike Falk Anderson, Steffen Engel, Peter Comninos und Leigh McLoughlin: *The case for research in game engine architecture*, in *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, S. 228–231, 2008.

[ASS+18] Eman Ahmed, Alexandre Saint, Abd El Rahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamila Aouada und Bjorn Ottersten: *A survey on deep learning advances on different 3D data representations*, *arXiv preprint arXiv:1808.01462*, 2018.

[BSW+16] C Brandl, KJ Stark, M Wintergerst, M Heinemann, IM Heid und RP Finger: *Epidemiologie der altersbedingten Makuladegeneration*, *Der Ophthalmologe*, Bd. 113(9):S. 735–745, 2016.

[ECM99] ECMA: *Introducing JSON*, 1999.
URL https://www.json.org/json-en.html

[GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville: *Deep Learning*, MIT Press, 2016, http://www.deeplearningbook.org.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald und Werner Stuetzle: *Mesh optimization*, in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, S. 19–26, 1993.

[Hel97] Martin Held: *Erit—a collection of efficient and reliable intersection tests*, *journal of graphics tools*, Bd. 2(4):S. 25–44, 1997.

[HSL+91] David Huang, Eric A Swanson, Charles P Lin, Joel S Schuman, William G Stinson, Warren Chang, Michael R Hee, Thomas Flotte, Kenton Gregory, Carmen A Puliafito *et al.*: *Optical coherence tomography*, *science*, Bd. 254(5035):S. 1178–1181, 1991.

[JPC+16] Xiaotong Jiang, Qingjin Peng, Xiaosheng Cheng, Ning Dai, Cheng Cheng und Dawei Li: *Efficient booleans algorithms for triangulated meshes of geometric modeling*, *Computer-Aided Design and Applications*, Bd. 13(4):S. 419–430, 2016.

[KB18] G Naresh Kumar und Mallikarjun Bangi: *An extension to winding number and point-in-polygon algorithm*, *IFAC-PapersOnLine*, Bd. 51(1):S. 548–553, 2018.

[LTH86] David H Laidlaw, W Benjamin Trumbore und John F Hughes: *Constructive solid geometry for polyhedral objects*, in *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, S. 161–170, 1986.

[MT13] Gang Mei und John C Tipper: *Simple and robust boolean operations for triangulated surfaces*, *arXiv preprint arXiv:1308.4434*, 2013.

[ON15] Keiron O'Shea und Ryan Nash: *An introduction to convolutional neural networks*, *arXiv preprint arXiv:1511.08458*, 2015.

[Per02] Ken Perlin: *Improving noise*, in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, S. 681–682, 2002.

[RR10] Ethan A Rossi und Austin Roorda: *The relationship between visual resolution and cone spacing in the human fovea*, *Nature neuroscience*, Bd. 13(2):S. 156–157, 2010.

[Sch06] Wolfgang F Schrader: *Altersbedingte makuladegeneration*, *Der Ophthalmologe*, Bd. 103(9):S. 742–748, 2006.

[SGWM18] Jong-Chyi Su, Matheus Gadelha, Rui Wang und Subhransu Maji: *A deeper look at 3d shape classifiers*, in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, S. 0–0, 2018.

[SMKLM15] Hang Su, Subhransu Maji, Evangelos Kalogerakis und Erik Learned-Miller: *Multi-view convolutional neural networks for 3d shape recognition*, in *Proceedings of the IEEE international conference on computer vision*, S. 945–953, 2015.

[Uni19] Unity: *Unity Manual: Order of execution for event functions*, 2019. URL https://docs.unity3d.com/Manual/ExecutionOrder.html

[Uni20] Unity: *Unity Manual: Physics.Raycast*, 2020. URL https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

[ZC01] Cha Zhang und Tsuhan Chen: *Efficient feature extraction for 2D/3D objects in mesh representation*, in *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, Bd. 3, S. 935–938, IEEE, 2001.

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Mittweida, den 23. November 2021

Noah Stolz