OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG
INF FAKULTÄT FÜR INFORMATIK

Institut für Simulation und Graphik

Abschlussarbeit

zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

Workflow zur Erstellung von 3D-Modellen für Hands-on Training in der Radiochemie

Vorgelegt von	Florian Waldschik
Matrikelnummer	200258
am	01. April 2022
Erstgutachter	Prof. Dr.-Ing. habil. Bernhard Preim
Zweitgutachter	Prof. Dr. Christian Hansen
Betreuer	Sebastian Wagner (M. Sc.)
	Dr.-Ing. Patrick Saalfeld

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Stellen sind als solche kenntlich gemacht. Alle Bilder und Skizzen wurden selbst erstellt und nicht aus anderen Quellen entnommen.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Magdeburg, den 01. April 2022

_____

(Vorname Nachname)

Kurzfassung

Diese Arbeit stellt einen angepassten 3D-Modellierungs-Workflow vor, der sich an der Zielplattform *WebGL* orientiert. Ziel ist es, die Entwicklung eines virtuellen Radiochemielabors zu unterstützen. Zu diesem Zweck werden verschiedene Optimierungsmöglichkeiten beim Erstellen von Oberflächenmodellen erläutert. Die Organisation der zu modellierenden Objekte wird nach dem *Kanban*-Prinzip vorgenommen. Anschließend wird eine Performanzanalyse durchgeführt, die den Einfluss von hoch- und niedrigaufgelösten 3D-Modellen auf die *WebGL*-Anwendung gegenüberstellt und auswertet.

Abstract

This thesis presents a customized 3D modeling workflow that aims for *WebGL* as its target platform in order to support the implementation of a virtual radiochemistry laboratory. For this purpose it elaborates on different optimization strategies for creating surface meshes. The organizational aspect will be driven by the *Kanban* principles for project management. Following the creation of the required meshes it will be compared and analyzed how much of an impact high and low poly models have on the performance of a *WebGL* application.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AO Ambient Occlusion

API Application Programming Interface

AR Augmented Reality

CAD Computer-Aided Design

CFD Computational Fluid Dynamics

CINCH Cooperation in Education and Training in Nuclear and Radiochemistry

CPU Central Processing Unit

DIN Deutsches Institut für Normung e. V.

FPS Frames Per Second

GPU Graphics Processing Unit

HoT Hands-on Training

OJT On-the-Job Training

OS Operating System

PBR Physically Based Rendering

RAM Random Access Memory

VR Virtual Reality

WBT Web Based Training

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Ausbildung von Studenten und Auszubildenden des Fachgebiets der Radiochemie beinhaltet einen wichtigen Anteil praktischer Experimente. Da die zugrundeliegenden Versuchsaufbauten kompliziert sind und eine Vielzahl an Ausrüstung benötigen, können diese Fertigkeiten normalerweise nur in eigens dafür ausgestatteten Laboratorien vermittelt werden. Um solche Lehrinhalte dennoch einer größeren Studentenschaft nahebringen zu können, hat es sich das *A-CINCH* Projekt (engl. „*Augmented Cooperation in Education and Training in Nuclear and Radiochemistry*“) zur Aufgabe gemacht, die zuvor entwickelten Lehrszenarien digital zu erweitern. Dazu werden ausgewählte praktische Trainingseinheiten (engl. „*hands-on training*“, kurz „HoT“) mit 3D-Inhalten erweitert (engl. „*augmented reality*“, kurz „AR“) oder gänzlich in einer 3D-Anwendung nachgebildet. Die so geschaffene 3D-Umgebung bietet dabei auch einen gewissen Spielraum für Fehler, die in einem echten Labor nicht tragbar und potentiell gefährlich wären. So fördert sie Erkenntnisgewinne, die auf herkömmlichen Wege nicht realisierbar sind [DSI10]. Durch diese Ergänzungen zur physischen Laborausbildung sollen die Lehrinhalte zukünftig effektiver und effizienter vermittelt werden.

1.1 Motivation

Die Implementierung einer solchen Lehranwendung ist aufgrund der Anzahl und Komplexität der benötigten 3D-Assets mit großem Modellieraufwand verbunden. Eine gut strukturierte und effiziente Arbeitsweise (engl. „*workflow*“) für die Erstellung dieser 3D-Modelle ist daher unabdingbar, um zu einem reibungslosen Ablauf des Projekts beitragen zu können [DSI10].

Da das ausschlaggebende *A-CINCH* Projekt über einen Zeitraum von 3 Jahren (Oktober 2020 – September 2023)[1] anberaumt wurde, besteht die Möglichkeit,

dass die involvierten 3D-Modellierer und Anwendungsentwickler fluktuieren. Aus diesem Grund ist die Aufbereitung und Organisation der 3D-Assets ein weiterer wichtiger Aspekt. Zukünftige Kollegen sollen die bereits erstellten 3D-Modelle ohne große Vorkenntnisse ändern und neue Assets in gleicher Manier anfertigen können. Dafür bedarf es u. a. einer einheitlichen Objekthierarchie, die in enger Absprache mit den Entwicklern erarbeitet wird. Ansonsten sollen die Überschneidungen mit den Entwicklern allerdings so gering wie möglich gehalten werden, um eine parallelisierte Arbeitsweise zu ermöglichen, bei der sich Modellierer und Entwickler auf die ihnen zugeschriebenen Aufgaben konzentrieren können. Etwaige Änderungen an 3D-Modellen oder dem Quellcode sollen unabhängig voneinander vorgenommen werden können.

1.2 Zielsetzung

Im Zuge dieser Arbeit gilt es deshalb herauszufinden, wie ein Workflow aufgebaut ist, der den Anforderungen solcher Projekte gerecht wird. Wiederholt auftretende Elemente bei der 3D-Modellierung sollen geschickt wiederverwendet werden können (engl. „*reusability*“), während der Detailgrad der Objekte im Bezug auf die Zielplattform der Anwendung (*WebGL*) verhältnismäßig ist. Dabei soll der Mehraufwand für im Nachhinein anfallende Änderungen minimiert werden, indem ein übergreifender Modularitätsansatz verfolgt wird.

Anschließend sollen die 3D-Modelle so exportiert werden, dass sie ohne Weitere Anpassungen in die Game Engine (*Unity*) eingebunden werden können. Der Fokus liegt hier auf einer einfachen Handhabung seitens der Entwickler. Mithilfe einer durchdachten Objekt-Hierarchie und bewährten 3D-Modellierungstechniken (engl. „*best practices*“) sollen auch bestimmte Arbeitsschritte bei der Implementierung optimiert werden. Dazu zählen die Platzierung der Objekte im Raum, das Erstellen von Animationen für die 3D-Assets (z. B. die Bewegung von Einzelteilen bei Interaktionen oder der Umgang mit Flüssigkeiten) sowie der Aufbau eines übersichtlichen Objektkatalogs innerhalb der Entwicklungsumgebung.

[1] `https://www.cinch-project.eu/` (abgerufen am 16.12.2021)

1.3 Struktur der Arbeit

Die weitere Arbeit ist in folgende Kapitel unterteilt:

Kapitel 2 führt in den projektbezogenen Hintergrund ein und beleuchtet die angestrebte Zielplattform der 3D-Anwendung (*WebGL*) genauer.

Kapitel 3 setzt sich mit verwandten Arbeiten auf dem Gebiet der 3D-Modellierung und der Entwicklung von Lehranwendungen auseinander.

Kapitel 4 beschreibt das Konzept der 3D-Modellierungspipeline im vollen Umfang und geht dabei im Detail auf die angewendeten und im Zuge des Projekts entwickelten Modellierungstechniken ein.

Kapitel 5 erläutert die Umsetzung des zuvor konzeptionierten Workflows im Bezug auf die verwendete Software. Außerdem wird auf organisatorische Aspekte des Modellierungsprozesses eingegangen.

Kapitel 6 analysiert die Performanz der Lehranwendung unter Verwendung der erstellten 3D-Modelle. Dazu wird ein *Benchmark* erstellt, das sich an den zur Verfügung stehenden Endgeräten orientiert.

Kapitel 7 fasst die Ergebnisse der Arbeit zusammen, diskutiert mögliche Verbesserungsansätze und bietet einen Ausblick auf eventuell weiterführende Arbeiten im selben Themenbereich.

2 Hintergrund

Um einen konkreten Projektbezug herzustellen, werden im Folgenden die grundlegenden Elemente des *A-CINCH* Projekts dargelegt und der Anteil dieser Arbeit daran eingeordnet. Der Begriff des Hands-on Trainings wird erläutert und vom klassischen On-the-Job Training abgegrenzt. Anschließend wird die Zielplattform der 3D-Anwendung (*WebGL*) näher beleuchtet und es werden die daraus resultierenden Folgen für Detailgrad und Funktionalität der 3D-Modelle aufgegriffen.

2.1 A-CINCH Virtual Radiochemistry Lab

Das *A-CINCH* Projekt ist ein Folgeprojekt, das auf den drei Forschungsprojekten *CINCH*, *CINCH II* und *MEET-CINCH* aufbaut. In diesen vorangegangenen Projekten wurde eine Vielzahl an Lehrmethoden und -werkzeugen entwickelt, welche die Ausbildung auf dem Gebiet der Radiochemie unterstützen sollen[1]. Diese didaktischen Möglichkeiten sollen nun durch Augmented und Virtual Reality (kurz „VR“) erweitert werden, um elektronisches Lernen (engl. „*e-learning*“) zu ermöglichen[2]. Der Begriff des E-Learnings steht dabei für eine Kombination aus Distanzunterricht, virtuellen Klassenräumen und Lehrmaterialien auf Abruf (engl. „*on demand*“). Der Fokus liegt auf der Integration neuer Technologien in den herkömmlichen Lehrbetrieb [GS15]. Die Plattform, welche die zuvor gewonnenen Erkenntnisse vereinen und im Rahmen des *A-CINCH* Projekts etabliert werden soll, ist das sogenannte *CINCH Hub*.

Wie in Abbildung 2.1 zu erkennen ist, bildet das VR-Labor das Kernstück des *CINCH Hubs*. Es steht in enger Verbindung mit sämtlichen anderen Aspekten

[1] `https://www.cinch-project.eu/` (abgerufen am 27.12.2021)

[2] `https://www.cinch-project.eu/default-title-1/aims-and-objectives` (abgerufen am 27.12.2021)

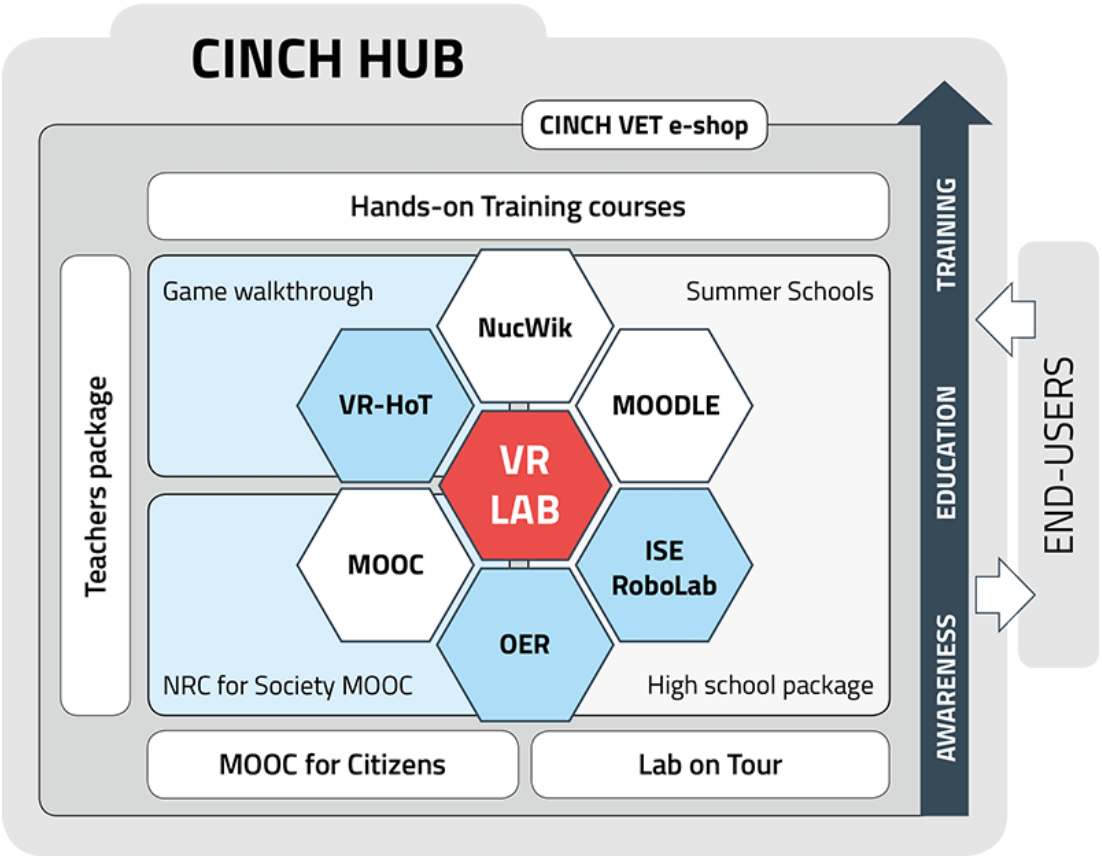


Abbildung 2.1: Schematische Übersicht des *CINCH Hubs*[3]

und bildet im Verbund mit diesen einen reichen Pool an Lehrressourcen. Dieser bietet von der Aufmerksamkeitsgenerierung zum Werben neuer Schüler über den theoretischen Lehrstoff bis hin zum praktischen Training alles an, was zur Ausbildung in der Radiochemie benötigt wird. Dabei wird auch berücksichtigt, wer mit dem Hub interagiert. So werden z. B. je nach Rolle des Nutzers verschiedene Pakete für Lehrer, Schüler oder anderweitige Interessenten angeboten.

Im Zuge dieser Arbeit wird der Aspekt des virtuellen Hands-on Trainings und die Erstellung der dafür benötigten 3D-Modelle thematisiert. Weiterhin ist anzumerken, dass die Umsetzung zurzeit nicht mehr wie in Abbildung 2.1 angegeben als VR- sondern als *WebGL*-Anwendung angestrebt wird (siehe Abschnitt 2.3).

[3] `https://www.cinch-project.eu/default-title-1/cinch-tools-and-teaching-techniques` (abgerufen am 27.12.2021)

2.2 On-the-Job und Hands-on Training

Getreu der Maxime „Lernen durch Handeln“, welche durch den griechischen Philosophen *Aristoteles* geprägt und durch Pädagogen wie *John Dewey* popularisiert wurde[4], ist das Training durch Ausführung der Tätigkeit selbst (engl. „*on-the-job training*“, kurz „OJT“) heutzutage die weitverbreitetste Art der Berufsausbildung. Gleichzeitig ist sie in ihrer traditionellen Form aber auch ineffizient, inkonsistent und zumeist schlecht strukturiert [Sis01]. Diesen Kritikpunkten kann man jedoch mit einer systematischen Herangehensweise begegnen und das OJT dahingehend verbessern. Diese optimierte Variante des OJT wird als *Hands-on Training* (kurz „HoT“) bezeichnet [Sis01] und stellt einen Kernaspekt des *CINCH Hubs* dar (siehe Abbildung 2.1).

Neben den Optimierungsmöglichkeiten gibt es auch andere Argumente, die für eine Verwendung von HoTs gegenüber OJTs sprechen. In Berufsfeldern mit niedriger Fehlertoleranz, d. h. bei Tätigkeiten, während denen selbst kleine Fehler oder Unaufmerksamkeiten zu schwerwiegenden Schäden führen können, ist ein OJT nicht erstrebenswert, solange eine gewisse Grundkompetenz fehlt. Die praktische Anwendung des Lernstoffs wird deshalb zunächst durch theoretischen Unterricht und Simulationen vorbereitet. Zu diesen Berufsfeldern zählt auch die Radiochemie, welche den Umgang mit giftigen Chemikalien und gesundheitsschädlicher Strahlung beinhaltet. Aus diesem Grund hat es sich das *A-CINCH* Projekt zum Ziel gesetzt, ausgewählte Trainingsszenarien in einer sicheren, virtuellen 3D-Lehrumgebung abzubilden. Potentielle Fehlerquellen sollen so bereits im Vorfeld durch die anschauliche und interaktive Vermittlung von wichtigen theoretischen Grundlagen reduziert werden. Dafür sollen u. a. festgelegte Abläufe, Protokolle und Routinen im Labor verinnerlicht werden. Um die überwiegend jüngere Zielgruppe der Auszubildenden zu motivieren, sich mit den HoTs auseinanderzusetzen, werden auch Elemente der Gamification als zielführend betrachtet und entsprechend implementiert[2]. Unter Gamification wird hierbei der Einsatz von Spiel-Design-Elementen im spielefremden Kontext des Labors verstanden [DDKN11].

[4] `http://www.mi-knoll.de/128401.html` (abgerufen am 22.01.2022)

2.3 Zielplattform (WebGL)

Wie in Abbildung 2.1 zu erkennen ist, wurde das virtuelle Labor inklusive der dazugehörigen HoTs zunächst als VR-Anwendung konzipiert. Im Laufe der Bearbeitung des Projekts hat sich jedoch herausgestellt, dass diese Zielplattform verschiedene Nachteile mit sich bringt:

- **Verbreitung & Kosten**
  Da die meisten Lehreinrichtungen noch nicht oder nur mit wenigen VR-Headsets ausgestattet sind, kann ein flächendeckender Zugang zu einer VR-Anwendung nicht gewährleistet werden und würde somit wieder nur limitiert stattfinden können. Neben der Voraussetzung eines VR-Headsets besteht außerdem die Notwendigkeit eines leistungsstarken PC-Systems, auf dem die VR-Anwendung ausgeführt wird[5,6], um eine bestimmte Anzahl von Bildern pro Sekunde nicht zu unterschreiten.

- **Motion Sickness**
  Selbst mit geeigneter Hardware sind körperliche Begleiterscheinungen nicht auszuschließen. Ein signifikanter Anteil von Nutzern (22 % bis 56 %, in Abhängigkeit zur bedienten Anwendung) mit einem Durchschnittsalter von rund 21 Jahren berichten, dass sie bereits nach wenigen Minuten Symptome der VR-Krankheit (engl. „*motion sickness*“) zeigen [MDS17]. Zu den häufigen Beschwerden zählen dabei Kopfschmerzen, Übelkeit und Müdigkeit [LJ00].

Aufgrund dieser Umstände wurde sich dafür entschieden, das virtuelle Labor als webbasierte Trainingsanwendung (engl. „*web based training*“, kurz „WBT“) umzusetzen [Sto13]. Als Zielplattform dient dafür *WebGL*, welches auf der bewährten Programmierschnittstelle (engl. „*application programming interface*“, kurz „API“) *OpenGL* basiert und von nahezu jedem Desktop- und Mobile-Browser unterstützt wird. Dies ermöglicht die hardwarebeschleunigte Darstellung von 3D-Grafiken im Webbrowser, ohne dass zusätzliche Software installiert werden muss [Par14].

[5] `https://support.oculus.com/articles/getting-started/getting-started-with-rift/rift-s-minimum-requirements/` (abgerufen am 26.01.2022)

[6] `https://www.vive.com/us/support/vive/category_howto/what-are-the-system-requirements.html` (abgerufen am 26.01.2022)

Da bei einer WBT-Anwendung im Vergleich zu einer rechnergestützten Anwendung (engl. „*computer based training*“, kurz „CBT“) die Hardware des Nutzers nicht als gegeben sondern nur als unterstützend angesehen werden kann [Sto13], sollte dies auch bei der Implementierung beachtet werden. Die grafische Qualität der 3D-Anwendung muss sich daher der Performanz unterordnen. Trotzdem soll das Labor alle prägnanten Merkmale aufweisen, die für die HoTs benötigt werden.

2.4 Kompromiss zwischen Detailgrad und Funktionalität

Um das virtuelle Labor so realitätsnah wie möglich aber gleichzeitig effizient im Hinblick auf die Implementierung mit *WebGL* auszustatten, ist es unerlässlich, Abzüge im Detailgrad der verwendeten 3D-Modelle zu machen. Diese Abstraktionen sollen im besten Fall aber nur im direkten Vergleich mit dem Objekt auffallen, das als Vorlage für das jeweilige 3D-Modell gedient hat. Dazu ist es nötig, den grundlegenden Aufbau des Objekts zu analysieren und es auf die für das HoT benötigten Elemente zu reduzieren [PP10]. Diese Optimierungen können dabei in verschiedenen Aspekten eines 3D-Modells vorgenommen werden.

Oberflächenmodell

Der wichtigste Teil der Abstraktion findet beim Aufbau des 3D-Modells selbst statt. Dieses besteht aus einer Menge an Punkten im Raum, einer Menge an Kanten, die diese Punkte verbinden und einer Menge an Flächen, die sich zwischen diesen Kanten erstrecken. Das so gebildete Polygonnetz (engl. „*mesh*“) approximiert die Oberfläche des darzustellenden Objekts [Mar19]. Da dieser Darstellung nur diskrete Mengen geometrischer Primitive zugrunde liegen und sie nicht durch Freiformkurven und -flächen beschrieben wird, spielt die Anzahl der verwendeten Punkte eine wichtige Rolle im Bezug auf den Detailgrad des 3D-Modells. Je größer die Menge der Punkte ist, desto genauer kann das 3D-Modell die Form des Objekts beschreiben. Ähnlich der Abtastrate eines Scanners entscheidet dabei die Dichte der Punkte im Raum darüber, wie detailliert die Merkmale des Objekts dargestellt werden können. Unter einem Merkmal (engl. „*feature*“) versteht man im Bezug auf ein 3D-Modell einen hervortretenden Punkt bzw. eine markante

Region, die vom menschlichen Auge direkt von ihrer Umgebung abgegrenzt werden kann [Mar19]. Die optische Qualität eines Merkmals steht allerdings nicht im direkten Verhältnis zur Anzahl der verwendeten Punkte. Je nach Art des Merkmals profitiert dieses von einer höheren Dichte der Punkte (z. B. Rundungen) oder zieht keinen sichtbaren Mehrwert daraus (z. B. klare Ecken und Kanten oder ebene Flächen).

Im Kontext der Optimierung spielt die Beschaffenheit des Polygonnetzes eine wichtige Rolle im Hinblick auf den Ressourcenverbrauch der *WebGL*-Anwendung. 3D-Modelle mit einer großen Menge an Polygonen benötigen mehr Zeit zum Laden und beanspruchen mehr Speicherplatz [BAS13]. Da dieser Umstand unabhängig von der optischen Qualität des 3D-Modells zum Tragen kommt, ist es umso wichtiger, die Menge der Polygone klein zu halten und die verwendeten Punkte so effizient wie möglich einzusetzen. Diese Thematik wird in Abschnitt 4.2.5 weiter ausgeführt.

Neben der Anzahl der Polygone ist auch ihre Verteilung auf verschiedene Polygonnetze innerhalb des 3D-Modells ein Faktor, der die Performanz der Anwendung beeinflusst. Je weniger Instanzen darzustellen sind, desto effizienter können sie gehandhabt werden, da verbundene Polygonnetze bei der Berechnung des Ausgabebildes (engl. „*rendering*“) auch im Verbund abgearbeitet werden können[7]. Aus diesem Grund ist es im Sinne der Optimierung, die beweglichen Teile des 3D-Modells bereits im Vorfeld festzulegen. Die statischen Teile sollten hingegen zu einem einzelnen Polygonnetz kombiniert werden.

Materialien

Das endgültige Erscheinungsbild eines 3D-Modells in der *WebGL*-Anwendung wird maßgeblich von den Materialien beeinflusst, die dem Oberflächenmodell zugewiesen sind. Unter einem Material versteht man den Sammelbegriff für Farben, Oberflächenstrukturen und Texturen. Mithilfe dieser Eingabeparameter und einem eigenständigen Programm (engl. „*shader*“) wird die letztendliche Pixelfarbe im Ausgabebild mitbestimmt. Das Material kann außerdem die Oberflächenbeschaffenheit des 3D-Modells verändern und weiterführende Effekte erzeugen (z. B. Darstellung von Flüssigkeiten oder Glas) [Büh21]. Da die verwendeten Texturen ebenfalls Speicherkapazitäten in Anspruch nehmen, bieten auch diese Raum für

[7] `https://docs.unity.cn/Manual/combining-meshes.html` (abgerufen am 30.01.2022)

Optimierungen (siehe Abschnitt 4.3). Desweiteren ist zu beachten, dass das Wechseln zwischen verschiedenen Materialien während des Renderings die ressourcenintensivste Operation ist, die die Grafik-API ausführt[8]. Es ist daher ratsam, die Zahl der verwendeten Materialien so klein wie möglich zu halten.

Optimierungen der Shader-Programme selbst werden in dieser Arbeit nicht behandelt, da diese unabhängig vom 3D-Modell vorgenommen werden und ein eigenes Fachgebiet darstellen. Sie gehören somit nicht zu dieser Modellierungspipeline. Es wird jedoch am Beispiel der Flüssigkeitsdarstellung gezeigt, wie der Aufbau eines Polygonnetzes die Shader-Entwicklung unterstützen kann (siehe Abschnitt 4.2.6).

[8] `https://docs.unity.cn/Manual/optimizing-draw-calls.html` (abgerufen am 30.01.2022)

3 Verwandte Arbeiten

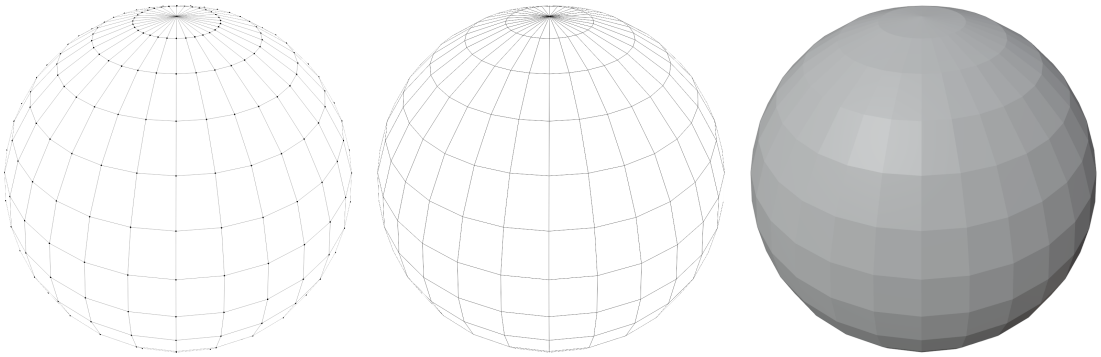
Dieses Kapitel untersucht Literatur, die sich mit dem Erstellen von 3D-Modellen und dem Entstehungsprozess dahinter beschäftigt. Speziell werden Arbeiten beleuchtet, welche die dabei verwendeten Praktiken erläutern und konkreten Bezug zur 3D-Modellierungssoftware herstellen. Es werden Arbeiten präsentiert, die sich mit der Umsetzung von virtuellen Lehrumgebungen auseinandersetzen und Schlüsse auf das zugrundeliegende *A-CINCH* Projekt zulassen. Desweiteren wird der organisatorische Aspekt der Arbeit aufgegriffen.

3.1 Bewährte Praktiken in der 3D-Modellierung

Der grundlegende Begriff des Oberflächenmodells (engl. „*surface mesh*“) wird dem Buch [Mar19] entnommen. *Mari* definiert ihn dort wie folgt:

> „A surface *mesh* is formally defined as a triplet of sets *M = (V, E, F)*, where V is the set of *vertices*, which are points in the 3D Euclidean space $\mathbb{R}^3$ sampling the surface at stake; E is the set of *edges*, which are segments whose endpoints are vertices of V : $E \subset V \times V$; and F is the set of *faces*, which are polygons whose vertices and edges are vertices and edges of V and E respectively.“ [Mar19]

Anhand dieser Definition werden die für das HoT erstellten 3D-Repräsentationen der Objekte charakterisiert. Die Menge V der Punkte im Raum dient als Kriterium für die Komplexität eines 3D-Modells. Je größer V ist, desto höher ist das Polygonnetz aufgelöst und desto mehr Ressourcen verbraucht es in der späteren Anwendung (siehe Abschnitt 3.2). Die Menge E der Kanten, welche durch die Verbindung der Punkte entsteht, formt die Oberfläche des Modells. Der Verlauf der Kanten spielt eine wichtige Rolle beim Approximieren der Form des Objekts.



(a) *Vertices* **(b)** *Edges* **(c)** *Faces*

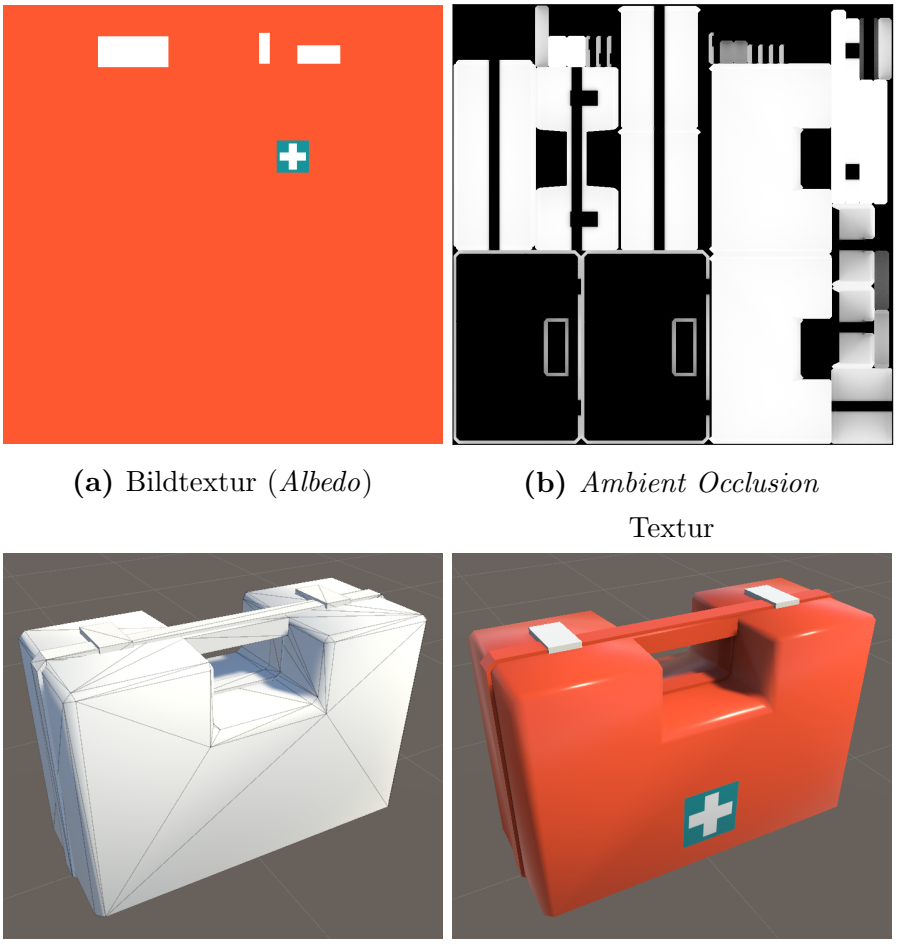
Abbildung 3.1: Bestandteile eines Oberflächenmodells (*Mesh*)

Deswegen dienen sie als Nähte für das Aufspannen des Polygonnetzes während dem Vorgang des *UV-Mappings* (siehe Abschnitt 4.3.2). Die Menge F der Flächen ist maßgeblich für das Erscheinungsbild des 3D-Modells verantwortlich, da sie als Träger der Materialien dienen, die Farbe und Oberflächenbeschaffenheit des Modells beeinflussen (siehe Abschnitt 4.3.1).

Nach der so geschaffenen mathematischen Grundlage wird im Buch [NFHS19] von *Nischwitz* et al. der weitere Weg vom geometrischen Modell bis zur Ausgabe von in Echtzeit berechneten Bildern erklärt. Im Zuge dessen wird auf die Zusammenhänge zu anderen Fachgebieten wie der Programmierung in *OpenGL* eingegangen, welche die Basis der Zielplattform *WebGL* bildet (siehe Abschnitt 2.3). Dabei werden gängige Einsatzmöglichkeiten für Bildtexturen erläutert und damit verbundene Möglichkeiten der Optimierung für Echtzeitanwendungen vorgestellt. Dazu zählen das gezielte Entfernen unnötiger Flächen aus der Menge F eines Oberflächenmodells (siehe Abschnitt 4.3.2), das Verwenden von Bilddateien als Speichermedium für Farbinformationen, die auf den Flächen des Modells dargestellt werden sollen (siehe Abbildung 3.2a) und der Einsatz von *Ambient Occlusion* Schatten, welche ebenfalls effizient in einer Textur gespeichert werden können (siehe Abbildung 3.2b). *Nischwitz* et al. bezeichnen sie als eine spezielle

> „[...] Art von Schatten, die durch Verdeckung („*Occlusion*“) eines gewissen Anteils der ambienten Hintergrundstrahlung durch umgebende Objekte verursacht werden.“ [NFHS19]

Für einen systematischen Aufbau des zuvor definierten Polygonnetzes wird das



(a) Bildtextur (*Albedo*)

(b) *Ambient Occlusion* Textur

(c) 3D-Modell

(d) Projektion auf das 3D-Modell

Abbildung 3.2: 3D-Texturierung

Buch [Sim13] zurate gezogen. *Simonds* beschreibt darin einen ausführlichen Leitfaden, der sich Schritt für Schritt im Detail mit jeder Phase der Erstellung eines 3D-Modells auseinandersetzt. Dabei erläutert er von der Konzeptphase bis zum finalen Modell jeden Abschnitt des Modellierungsprozesses und geht auf deren Umsetzung in der Open-Source-Software *Blender* ein. Durch den direkten Bezug zum verwendeten Modellierungswerkzeug können spezialisierte Optimierungen vorgenommen werden, wie z. B. bei der internen Speicherverwaltung durch *Daten-Blöcke* (siehe Abschnitt 4.2.4), der Nutzung des *Ursprungspunkts* (siehe Abschnitt 4.2.6) und beim Prozess des *UV-Mappings* (siehe Abschnitt 4.3.2). Auch wenn das Buch bereits 9 Jahre alt ist, hat es nichts von seiner Aktualität eingebüßt, da die beschriebenen theoretischen Hintergründe immer noch Anwendung finden. Lediglich die grafische Oberfläche der Software und die zur

Verfügung stehenden Werkzeuge haben sich mit der Zeit weiterentwickelt[1] und haben u. a. mit dem Anbieten einer neuen Palette von Tastaturkürzeln einen Schritt auf den Industriestandard zu gemacht[2]. Allerdings beschränkt sich dieses Werk auf das Erstellen von 3D-Modellen und das in Szene setzen zur Bilderzeugung. Die weitere Verwendung in einer Game Engine wird nicht thematisiert.

Einen holistischen und praxisorientierten Ansatz gibt hingegen *Villanueva* in ihrem Buch [Vil22]. Nachdem sie auf die Grundlagen der 3D-Modellierung und der Spieleentwicklung eingegangen ist, beschreibt sie den Werdegang eines 3D-Modells von der Sammlung von Referenzen bis zur Animation in der Game Engine (*Unity*). Ihre verwendete Modellierungssoftware (*Maya*) unterscheidet sich zwar von der in dieser Arbeit genutzten Software (*Blender*), aber die grundlegenden Vorgehensweisen und Konventionen können in diese übernommen werden, da die Prinzipien universeller Natur sind. Gerade im Bereich der *3D-Texturierung* (siehe Abbildung 3.2) kommt dies zum Tragen, was sie folgendermaßen beschreibt:

> „3D texturing is the process of applying 2D images to objects in 3D space. [...] To achieve this, you utilize a texturing program. Regardless of which texturing program you use (such as Substance Painter, 3D Coat, or Quixel), the principles are similar.“ [Vil22]

Desweiteren beschäftigt sich das letzte Kapitel des Buches explizit mit der Integration des 3D-Modells in die Game Engine (*Unity*) und besitzt somit Relevanz für Abschnitt 5.3 dieser Arbeit.

3.2 Virtuelle Lehrumgebungen (VLEs)

Wenn eine virtuelle Lehrumgebung (engl. „*virtual learning environment*“, kurz „VLE“) ausreichend Interaktivität vorweist und dem Nutzer genügend Kontrolle gegeben wird, kann solch eine Simulation der Ausführung des Experiments im echten Labor sehr nahe kommen [GKB07]. Da die Implementierung einer VLE jedoch sehr zeitintensiv und somit kostspielig ist, haben *Dere* et al. in ihrem Paper [DSI10] eine Methodologie vorgestellt, mit der ein Repositorium aus

[1] `https://www.blender.org/download/releases/2-90/` (abegrufen am 11.03.2022)

[2] `https://docs.blender.org/manual/en/latest/interface/keymap/industry_compatible.html` (abgerufen am 11.03.2022)

3D-Modellen inkl. dazugehöriger Animationen für zukünftige VLEs geschaffen werden kann. Durch eine parallelisierte Arbeitsweise und die Verwendung eines solchen Repositoriums konnte der Arbeitsaufwand um 66 % gesenkt werden. Der Kernaspekt dieser Methodologie sind die aufgestellten Designkriterien, welche für all ihre erstellten 3D-Modelle bindend sind und auch in dieser Arbeit Anwendung finden.

Entsprechend dieser Kriterien sollen die 3D-Modelle:

(A) ihre korrekte Funktionalität besitzen.

(B) so realistisch wie möglich sein.

(C) akkurate Maße besitzen.

(D) schnell darstellbar sein.

(E) wiederverwendbar sein.

(F) erweiterbar sein.

(G) mit geringen Produktionskosten erstellt werden.

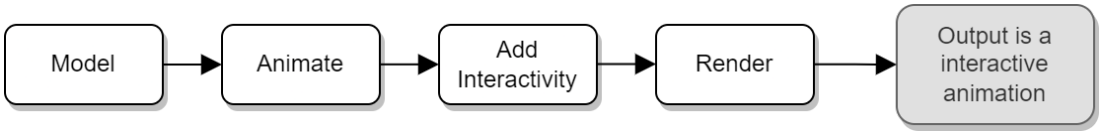


Abbildung 3.3: 3D-Workflow für Lehranwendungen (nach *Dere* et al. [DSI10])

Abgeleitet vom allgemeinen Prozess der Erstellung von 3D-Inhalten in der Unterhaltungsindustrie stellen sie außerdem einen angepassten Prozessablauf für 3D-Inhalte in VLEs vor (siehe Abbildung 3.3). Dieser umfasst mehrere komplexe Entwicklungsphasen, auf die im Paper jeweils kurz eingegangen wird. Nach diesem Ablauf wird auch bei der Entwicklung des HoT vorgegangen. Im Gegensatz zu *Dere* et al. beschäftigt sich diese Arbeit jedoch in detaillierterer Form mit den einzelnen Phasen des Workflows.

Eine weitere Vorgehensweise zur Erstellung von 3D-Inhalten stellen *Ortiz* et al. in ihrem Paper [OGE+20] vor (siehe Abbildung 3.4). Sie legen den Fokus auf Augmented bzw. Virtual Reality und betonen den positiven Effekt, den VLEs auf

den Lernerfolg ihrer Nutzer haben. Dabei wird die Wichtigkeit aktueller Techniken z. B. in der Materialerstellung hervorgehoben. Zu diesem Zweck werden PBR-Materialien (siehe Abschnitt 4.3.1) beleuchtet und ihr Schaffungsprozess an einem Beispiel erläutert. Diese Art von Materialien wird u. A. auch in dieser Arbeit verwendet, um bestimmte Oberflächen ohne großen Mehraufwand realistisch darzustellen. Desweiteren deckt sich der vorgestellte Prozessablauf für die Modellierungsphase eines Objekts mit dem in dieser Arbeit verfolgten Ablauf. Es werden jedoch Änderungen an bestimmten Stellen vorgenommen, um eine effizientere Arbeitsweise zu ermöglichen, da sich die im HoT benötigten anorganischen 3D-Modelle von den im Paper beschriebenen Tieren unterscheiden.

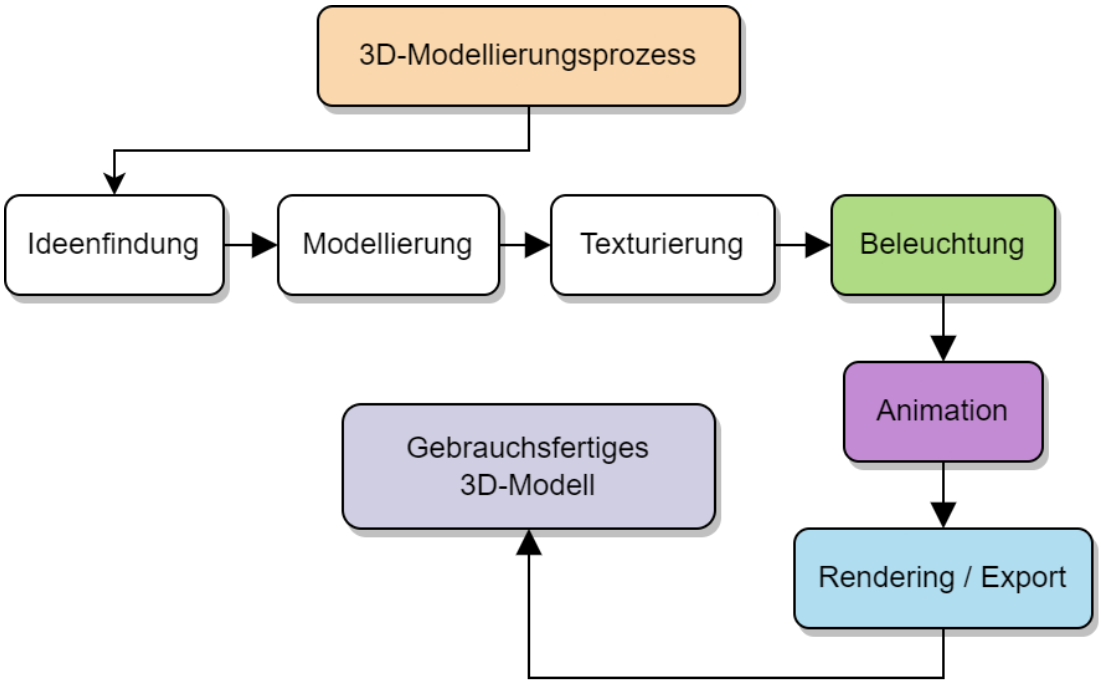


Abbildung 3.4: 3D-Modellierungsprozess (nach *Ortiz* et al. [OGE+20])

3.3 Organisatorischer Aspekt

Da die Modellierung einer großen Anzahl von Objekten für das HoT eine langwierige Aufgabe ist, müssen Vorkehrungen für eine organisierte Arbeitsweise getroffen werden, um einen konstanten Fortschritt gewährleisten zu können. Damit alle Beteiligten den momentanen Status der einzelnen Aufgaben verfolgen können,

muss dieser angemessen visualisiert werden. Zum Erfüllen dieser Anforderungen wird in dieser Arbeit das in der Softwareentwicklung bewährte *Kanban*-System verwendet [EG11]. Dabei wird die visuelle Darstellung der Arbeitsziele, die Limitierung der gleichzeitig bearbeiteten Aufgaben und die Maximierung der Effizienz priorisiert. Außerdem wird der Fokus auf die Reduzierung der Projektdauer gelegt. Für den reibungslosen Ablauf wird dafür eine sogenannte *Kanban*-Tafel eingesetzt, welche alle relevanten Informationen beinhaltet[3]. Diese allgemeine Methode der Prozesssteuerung adaptiert *Epping* in seinem Buch [Epp11]. Er nimmt darin konkretem Bezug auf den Anwendungsfall, den ein Softwareprojekt darstellt und betont die Individualisierungsmöglichkeiten der Technik, welche sie für eine Vielzahl an Projekten einsetzbar macht. Mit Hilfe dieses Leitfadens wurde eine entsprechende Modellübersicht konzeptioniert und ein daran angepasstes Feedbacksystem erstellt (siehe Abschnitt 5.1).

3.4 Weitere Arbeiten

Das Optimieren der Topologie von 3D-Modellen spielt eine wichtige Rolle für ihre Dateigröße und die Ladezeiten der Oberflächenmodelle auf Endgeräten mit niedrigen Hardwarespezifikationen. Diesen Zusammenhang belegen *Bhawar* et al. in ihrem Paper [BAS13], indem sie jeweils zwei Versionen (hoch und niedrig aufgelöstes Polygonnetz) verschiedener 3D-Modelle in den beiden Kriterien auf zwei Endgeräten (Desktop PC und Tablet) gegenüberstellen. Mit diesem Vergleich konnten sie zeigen, dass die Dateigröße um bis zu 38 % und die Ladezeit um bis zu 66 % reduziert werden kann. Gleichzeitig gaben 89 % der Befragten an, dass die Funktionalität und das Aussehen der niedrig aufgelösten 3D-Modelle vergleichbar mit ihren hoch aufgelösten Versionen ist. Dieses Ergebnis wird durch die im Paper vorgestellte Methodologie erreicht, mit der die Polygonnetzdichte der 3D-Modelle um bis zu 91 % reduziert wird. Die Optimierungen werden dabei manuell vorgenommen, da die getesteten automatisierten Lösungen keine zufriedenstellenden Resultate geliefert haben.
Aufgrund dieser Ergebnisse und der Tatsache, dass sich ihr Vorgehen auf den selben Prozessablauf zur Erstellung von 3D-Inhalten in VLEs stützt, der im Paper [DSI10] vorgestellt wurde (siehe Abschnitt 3.2), findet ihre Methodologie in

[3] `https://www.atlassian.com/agile/kanban/kanban-vs-scrum` (abgerufen am 18.03.2022)

dieser Arbeit Anwendung. Es ist jedoch anzumerken, dass es sich dabei um eine Methodologie zur Optimierung von bereits vorhandenen 3D-Modellen handelt (siehe Abschnitt 4.2.5). Dementsprechend kann sie nicht verwendet werden, wenn ein Objekt von Grund auf neu modelliert werden muss.

Für tiefere Einblicke in die verwendeten PBR-Materialien werden dedizierte Leitfäden herangezogen, die sich mit diesem Themengebiet im Detail auseinandersetzen. *McDermott* geht in seinem Buch [McD18] auf die grundlegenden theoretischen Hintergründe ein und gibt anschließend praktische Richtlinien für die Erstellung der dafür benötigten Texturen vor. Allerdings wird das Thema hier gesondert behandelt und stellt somit keinen Bezug zu anderen Aspekten eines Projekts her. Zu diesem Zweck wird das Buch [Kum20] von *Kumar* referenziert. Dieser beschreibt einen kompletten Texturierungsworkflow im Bereich der Spieleindustrie und geht dabei auch auf Themen wie dem UV-Mapping (siehe Abschnitt 4.3.2) ein. Er beschränkt sich dabei jedoch nicht nur auf PBR-Materialien, sondern vergleicht diese mit klassischen Materialien in der 3D-Modellierung. Mit Hilfe dieser Arbeiten kann genau abgewogen werden, welche Materialart für die verschiedenen Anwendungsfälle im HoT vorteilhafter ist. Auf die Unterschiede und die daraus resultierende Verwendung wird in Abschnitt 4.3.1 eingegangen.

4 Konzeptionierung der 3D-Modellierungspipeline

Dieses Kapitel beschreibt den generellen Vorgang beim Erstellen eines 3D-Modells, beginnend bei der initialen Informationslage über das Ausführen der Modellierung bis hin zum Erstellen der nötigen Materialien und der dazugehörigen Texturen.

4.1 Vorbereitung

Im Zuge des Projekts gibt es eine Vielzahl an Objekten, die als 3D-Modell umgesetzt werden sollen. Jedes Experiment im HoT wurde in Form eines Storyboards beschrieben, das die zu implementierenden Abläufe protokolliert, die jeweils benötigten Objekte aufzählt und die einzelnen Schritte teilweise bebildert. Für einige Abläufe steht zudem Videomaterial zur Verfügung. Die gegebenen Referenzen reichen allerdings nicht immer aus, um das jeweilige Objekt eindeutig in ein 3D-Modell überführen zu können. Anhand des Grades an Vorabinformationen lassen sich die Objekte in drei Kategorien unterteilen und a priori grob nach Aufwand sortieren, der unternommen werden muss, um ein Objekt mit den zur Verfügung stehenden Referenzen zu modellieren.

4.1.1 Genormte Objekte

Diese Kategorie stellt die am leichtesten umsetzbare Form der Objekte dar. Hierzu zählen Objekte, die aufgrund ihrer Häufigkeit und außer Frage stehenden Effizienz bereits eine anerkannte Normierung erfahren haben. Die wohl bekannteste Art der Normierung ist dabei die exakte Bemaßung durch das *Deutsche Institut*

für Normung e. V. (kurz „DIN“). Durch diesen international anerkannten Konsens ist das Objekt eindeutig beschrieben und kann entsprechend der einsehbaren Maßangaben in ein 3D-Modell überführt werden.

4.1.2 Bemaßte Objekte

Die zweite Gruppe beschreibt Objekte, die zwar bereits durch Maßangaben charakterisiert sind, aber nicht unbedingt eindeutigen Standards unterliegen. Dazu zählen vor allem Objekte, die z. B. in Produktkatalogen des jeweiligen Herstellers zu finden sind oder durch einen Labormitarbeiter selbst ausgemessen und mit genügend Referenzmaterial versehen wurden. Auch hier kann die Modellierung relativ leicht umgesetzt werden. Eine Allgemeingültigkeit kann jedoch nicht erwartet werden. Genauso kann es zu etwaigen Unterschieden in der 3D-Repräsentation des Objekts kommen, wenn bestimmte Aspekte des Objekts nicht genau bemaßt wurden, weil sie zur Zeit der Messung als nicht relevant eingestuft oder schlichtweg übersehen wurden sind.

4.1.3 Objekte ohne Maßangaben

In die letzte Klasse fallen die am schwersten umsetzbaren Objekte. Hierunter fallen alle Gegenstände, für die es keine allgemeinen Richtlinien, aber auch keine Bemaßung seitens der Hersteller oder Auftraggeber gibt. Meistens liegen diese nur als Bildreferenz vor und unterliegen deshalb den größten Schwankungen bei der Überführung in ein 3D-Modell, d. h. dass es bei der Umsetzung durch verschiedene Modellierer mit großer Wahrscheinlichkeit zu unterschiedlichen Ergebnissen kommt. Eine Allgemeingültigkeit kann deshalb auch hier nicht erwartet werden.

4.1.4 Festlegung der Interaktionsmöglichkeiten

Unabhängig von der Bemaßung muss das Objekt im Vorfeld der 3D-Modellierung genau durchdacht werden, um für den Einsatz in der Zielplattform optimiert werden zu können (siehe Abschnitt 2.4) [PP10]. Zu diesem Zweck werden die Merkmale des zu modellierenden Objekts zunächst in zwei Kategorien unterteilt:

- **Statisch**
  Diese Bereiche des Objekts werden während des Einsatzes im HoT nicht bedient und erfahren keine Interaktion seitens der Nutzer. Eine Animation dieser Bereiche ist nicht vorgesehen. Aufgrund dieser Eigenschaften können sie im Oberflächenmodell zu einem einzelnen Polygonnetz zusammengefasst werden. Darüber hinaus sind diese Bereiche durch ihre unbewegliche Natur prädestiniert für die Berechnung der Textur für die selbst induzierte Umgebungsverdeckung (engl. „*ambient occlusion*“, kurz „AO“), welche in Abschnitt 4.3.4 aufgegriffen wird.

- **Interaktiv**
  Diese Bereiche des Objekts spielen im HoT eine aktive Rolle und werden vom Nutzer direkt oder indirekt angesteuert. Aus diesem Grund müssen sie so separiert werden, dass Animationen des Oberflächenmodells später in der Game Engine (*Unity*) leicht umzusetzen sind. Hierzu zählen grundsätzlich alle Polygonnetze, die sich unabhängig vom statischen Polygonnetz bewegen sollen oder anderweitig in Szene gesetzt werden (z. B. durch Highlighting).

4.2 Modellierung

Nachdem alle nötigen Informationen zum Aufbau des 3D-Modells eingeholt sind und die relevanten Vorüberlegungen getätigt wurden, kann das Objekt nun in eine virtuelle Repräsentation überführt werden. Dieser Prozess beginnt mit der Modellierung des Oberflächenmodells.

4.2.1 Modellhierarchie

Der grundlegende Aufbau eines 3D-Modells ist hierarchischer Natur. Wie viele andere Datenstrukturen in der Informatik wird er deshalb mithilfe eines Baums dargestellt. Unter einem Baum versteht man dabei eine Menge von Knoten und Kanten, die in einer besonderen Beziehung zueinander stehen. Jeder Baum besitzt dabei genau einen Knoten, der als Wurzel bezeichnet wird und keinem anderen Knoten untersteht. Alle anderen Knoten sind jeweils durch genau eine Kante mit ihrem Elternknoten verbunden und werden als Kinder bezeichnet [SS20].

Im Falle eines 3D-Modells steht der Baum selbst für das gesamte Oberflächenmodell und jeder Knoten darin speichert mindestens die drei Transformationsvektoren für Position, Rotation und Skalierung. Außerdem können diese Konten jeweils ein eigenständiges Polygonnetz und alle dazugehörigen Informationen (z. B. Materialzuweisungen oder Modifikatoren) speichern. Die Transformationswerte werden dabei immer im Bezug auf den Elternknoten gespeichert. Bei der Transformation eines Knotens werden alle Kinder entsprechend mittransformiert, ohne deren lokale Transformationswerte zu ändern. Prinzipiell kann jeder Knoten in der Hierarchie die Rolle des Eltern- bzw. des Kinderknotens übernehmen, allerdings sind bestimmte Arten besser für die Aufgabe des Elternknotens geeignet als andere (siehe Abschnitt 4.2.2).

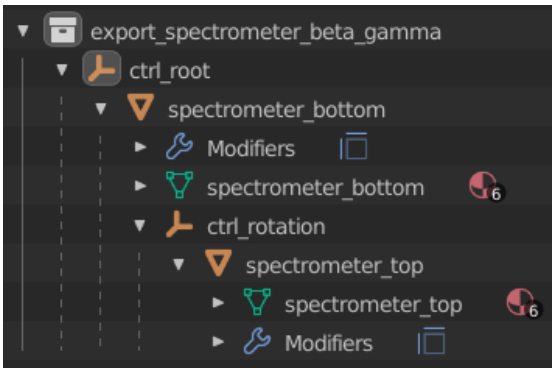


Abbildung 4.1: Beispielhierarchie (*Beta-Gamma-Spectrometer*)

Durch diesen klaren hierarchischen Aufbau lassen sich Oberflächenmodelle in sinnvolle Abschnitte unterteilen, die die Beziehungen der realen Komponenten des Objekts abbilden. Die in Abschnitt 2.4 und 4.1.4 beschriebenen Aspekte zur Einteilung nach Interaktionsmöglichkeiten finden hier ihre Anwendung. Polygonnetze, die in die Kategorie „interaktiv“ fallen, erhalten ihre eigenen Knoten, während die statischen Bereiche des 3D-Modells nach Möglichkeit in einem Knoten zusammengefasst werden.

4.2.2 Verwendung von leeren Knoten

Neben den Knoten, die ein Polygonnetz speichern, können im Baum auch leere Knoten (engl. „*empties*“) existieren, die nur ihre Transformationswerte mit sich

führen. Diese Art von Knoten verbraucht nahezu keinen Speicherplatz und erfüllt verschiedene Aufgaben innerhalb des Oberflächenmodells.

- **Strukturierung**
  Ähnlich einer Ordnerstruktur eines Betriebssystems dienen leere Knoten mit einer geeigneten Beschriftung der Übersichtlichkeit im Baum. So können zurzeit nicht relevante Unterbäume bei Bedarf ausgeblendet werden. Außerdem wird das 3D-Modell so entsprechend der zu erfüllenden Funktionen in Bauteilgruppen gegliedert. Da leere Knoten den Namen der Gruppe erhalten, muss kein Polygonnetz diese Aufgabe übernehmen. Dies könnte zu Verwirrungen und Doppeldeutigkeiten führen, da die Benennung eines Polygonnetzes sich auf den mit diesem Netz dargestellten Bereich des Objekts und nicht auf andere Netze beziehen sollte.

- **Markierung von Hilfspunkten**
  Um die Arbeitsschritte in der späteren Entwicklungsumgebung zu unterstützen, sollten relevante Positionen im Raum des Oberflächenmodells markiert werden. Da die spätere Entwicklungsumgebung nicht drauf ausgelegt ist, Polygonnetze eines 3D-Modells bearbeiten zu können, sollte dieser Arbeitsschritt direkt in der Modellierungssoftware vorgenommen werden. Dazu platziert man einen leeren Knoten an einer Stelle des 3D-Modells, die bei der Interaktion im HoT Verwendung finden wird. Dies können beispielsweise Punkte sein, an denen andere Oberflächenmodelle während des HoTs platziert werden (z. B. beim Stecken von Zentrifugenröhrchen in eine Zentrifuge). Der Wurzelknoten des sich bewegenden 3D-Modells wird dafür auf den leeren Markierungsknoten ausgerichtet. Die beiden Knoten sind nun deckungsgleich im Raum positioniert. Durch ein einheitliches Schema bei der Benennung der Knoten können diese auch im Kontext eines Scripts angesteuert werden, um weitere Interaktionslogiken zu ermöglichen.

- **Animation**
  Das Animieren von Szenen in der Computergrafik basiert auf Interpolationen, die zwischen festen Werten von Eigenschaften eines 3D-Modells wie z. B. Position oder Rotation vorgenommen werden. Die Werte, die das Modell zu bestimmten Zeitpunkten annehmen soll, werden auf einem Zeitstrahl (engl. „*timeline*“) gespeichert und dienen als Start- bzw. Endpunkt der Interpolation. Die Zwischenwerte werden anhand dessen automatisch

berechnet, was je nach Interpolationsart (linear, kurvig oder konstant) zu unterschiedlichen Bewegungsformen (gleichförmig, beschleunigt oder teleportierend) führt. Der Zeitstrahl wird standardgemäß jedoch nicht in Sekunden, sondern in Bilder (engl. „*frames*“) eingeteilt [Bri18]. Da sich die aufgenommenen Schlüsselwerte (engl. „*keyframes*“) einer Eigenschaft des 3D-Modells immer nur auf den jeweiligen Elternknoten beziehen, kann diese Beziehung für das Ausrichten von Knoten zu bestimmten Zeitpunkten genutzt werden. Anstatt die Transformationswerte als Weltkoordinaten zu speichern, wird nur die Differenz zu den Ausgangswerten im lokalen Koordinatensystem gespeichert. Ein leerer Knoten kann dabei als zentraler Referenzpunkt dienen, der in einer günstigeren Position liegt als der Wurzelknoten des gesamten 3D-Modells. Während des Setzens der Keyframes kann sich so auf die darzustellende lokale Bewegung konzentriert werden, da nur die Transformationen in Relation zum nächsten leeren Knoten mit einbezogen werden müssen.

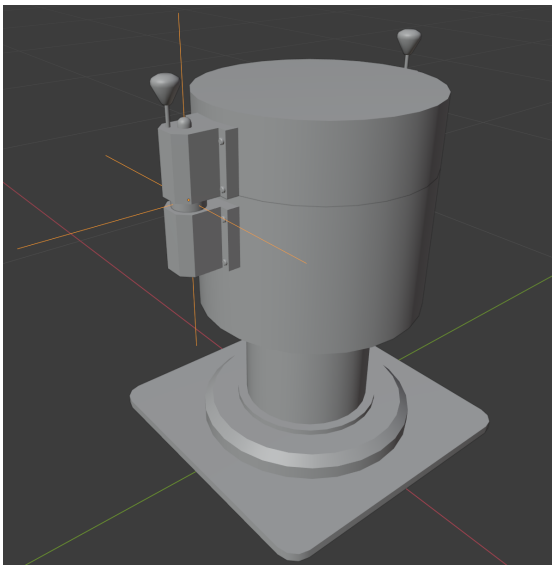


Abbildung 4.2: Markierung des Rotationspunkts (*Beta-Gamma-Spectrometer*)

Damit ein leerer Knoten diese Aufgaben zufriedenstellend erfüllen kann, muss sein Verhalten konsistent mit dem Verhalten anderer leerer Knoten in der Hierarchie und in anderen Oberflächenmodellen sein. Für seine Kinderknoten sollte er deshalb ein neutrales Element darstellen. In der linearen Algebra bezeichnet man damit ein Element, das verknüpft mit einem anderen Element wieder das verknüpfte Element selbst ausgibt [LM21]. Im Kontext eines Oberflächenmodells

bedeutet es, dass der leere Knoten selbst keine verändernden Transformationswerte speichert.

$$Rotation = \begin{bmatrix} 0° \\ 0° \\ 0° \end{bmatrix} \; Skalierung = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \tag{4.1}$$

Die Position des Knotens ist definitionsgemäß variabel und richtet sich nach seinem Einsatzzweck.

4.2.3 Verwendung des vorwärts gerichteten Vektors

Jedes Oberflächemodell besitzt eine Ausrichtung im Raum, die durch das Koordinatensystem vorgegeben wird, in dem es sich befindet. Dieser vorwärts gerichtete Vektor (engl. „*forward vector*“) kann im Quellcode der späteren Anwendung direkt referenziert werden[1]. Da die Vorwärtsrichtung eines Objekts durch dessen Verwendungszweck im HoT definiert ist und deshalb nicht mathematisch ermittelt werden kann, muss sie manuell festgelegt werden. Aus diesem Grund wird jedes Objekt so modelliert, dass dessen Frontseite in die vordefinierte Richtung zeigt. Bei symmetrischen Objekten, die keine klare Ausrichtung erkennen lassen (z. B. Reagenzgläser, Kolben o. Ä.), kann eine beliebige Seite als Front definiert werden. Durch diese Konvention wird eine einheitliche Basis geboten, mit der Interaktionsmöglichkeiten implementiert werden können. Bei sämtlichen Interaktionen, für die das Oberflächenmodell ausgerichtet werden muss, ist die mathematische Repräsentation der Frontseite in Form eines 3D-Vektors sehr hilfreich. Dazu zählen zum Beispiel die Verwendung eines Flüssigkeitsbehälters (siehe Abschnitt 4.2.6) oder die Bedienung eines Laborgeräts mit Eingabefeld an der Vorderseite.

Die Konvention begünstigt außerdem die Verwendung von Platzhalterobjekten. Falls die Modellierung eines benötigten 3D-Modells zum Zeitpunkt der Implementierung einer Interaktionsmöglichkeit noch nicht abgeschlossen ist, kann mit einem abstrakten Platzhalterobjekt (z. B. ein gerichteter Kegel) auf der Position des zukünftigen Oberflächenmodells gearbeitet werden. Sobald das eigentliche

[1] `https://docs.unity3d.com/ScriptReference/Transform-forward.html` (abgerufen am 09.02.2022)

Abbildung 4.3: Vorwärtsvektor (*orange*)

Objekt einsatzbereit ist, kann das Platzhalterobjekt ausgetauscht werden. Gleiches gilt natürlich auch für die nachträgliche Aktualisierung eines 3D-Modells im Falle von etwaig anfallenden Änderungswünschen. Somit wird eine parallelisierte und asynchrone Arbeitsweise von Modellierer und Entwickler unterstützt.

4.2.4 Wiederkehrende Elemente

Das Wiederverwenden von vorhandenen Elementen bzw. Lösungen ist auf vielen Gebieten ein lang bewährtes Mittel zur Problemlösung. In der Softwareentwicklung wird die Wiederverwendbarkeit (engl. „*reusability*“) von bereits erstellten Komponenten als Qualitätsmerkmal angesehen, da sie viele Vorteile mit sich bringt[2]. Dazu zählen z. B. die effiziente Implementierung von Änderungen, die Einhaltung des Modularitätsansatzes und das Bilden von generalisierten Standards [PD93]. In der 3D-Modellierung bietet es ebenfalls einen großen Mehrwert, Assets aller Art (Polygonnetze, Texturen, Animationen usw.) wiederverwenden zu können. Auch oder gerade weil die Interoperabilität zwischen verschiedenen Dateiformaten und Render Engines nicht immer gegeben ist, wird die Wiederverwendbarkeit von Oberflächenmodellen ebenfalls als eine erstrebenswerte Eigenschaft angesehen [BDRA11]. 3D-Assets, die unter den selben Konventionen

erstellt wurden, besitzen dabei den Vorteil, miteinander kompatibel und aufeinander abgestimmt zu sein, den Assets aus verschiedenen Quellen oft nicht vorweisen können.

Im Zuge dieser Arbeit wird zwischen zwei verschiedenen Arten der Wiederverwendbarkeit auf unterschiedlichen Ebenen unterschieden. Zum einen die Wiederverwendbarkeit bestimmter Bestandteile eines Polygonnetzes und zum anderen das Wiederverwenden von kompletten Polygonnetzen.

Wiederkehrende Bestandteile

Symmetrien sind allgegenwärtig, sowohl in unbelebten Objekten, die fast ausschließlich symmetrische Elemente enthalten, als auch in Organismen, die zwar nicht perfekt symmetrisch sind aber vom Grundaufbau her auch Dopplungen aufweisen. Sie stellen außerdem wichtige Anhaltspunkte zur Formwahrnehmung der Objekte dar [GPF09]. Diese Gegebenheiten können in der 3D-Modellierung gezielt genutzt werden, um den Aufwand beim Konstruieren des Polygonnetzes je nach Art der Symmetrie (Rotations- bzw. Axialsymmetrie) auf einen Bruchteil zu reduzieren. Aus diesem Grund stellt gängige Modellierungssoftware speziell für diesen Zweck eigene Werkzeuge bereit [Fla10]. Je mehr wiederkehrende Elemente ein Objekt enthält, desto effizienter kann es modelliert werden, da die relevante Geometrie nur zum Teil erzeugt und sie anschließend entsprechend der Symmetrie propagiert werden muss. Bei axialsymmetrischen Objekten bedeutet das meist die Ersparnis der Hälfte des Polygonnetzes, während bei rotationssymmetrischen Objekten nur die Kontur modelliert und eine Rotationsachse definiert werden muss. Hiervon profitieren vor allem die vielen Glasgefäße wie z. B. Reagenzgläser und Kolben, die im HoT ihre Anwendung finden. Da das native Dateiformat der Modellierungssoftware die Modifizierungen auf einem Stapel speichert, können Änderungen auch im Nachhinein effizient durchgeführt werden[3].

[2] `https://web.archive.org/web/20141023013122/http://lombardhill.com/what_reuse.htm` (abgerufen am 07.03.2022)

[3] `https://docs.blender.org/manual/en/latest/modeling/modifiers/introduction.html` (abgerufen am 07.03.2022)

Wiederkehrende Polygonnetze

In Fällen, bei denen das selbe Polygonnetz mehrmals in der Hierarchie des Oberflächenmodells vorkommt (z. B. Türen, Griffe, Knöpfe usw.), wäre es nicht im Sinne des ressourcensparenden Ansatzes, das selbe Polygonnetz mehrmals abzuspeichern. Stattdessen sollte das Polygonnetz nur einmal gespeichert und die jeweiligen Transformationsvektoren für alle Vorkommnisse im Oberflächenmodell hinterlegt werden. Jeder Vektor greift dann auf die selben Informationen zurück und platziert an seiner Position eine Instanz vom Polygonnetz entsprechend seiner Rotations- und Skalierungswerte. Diese Technik wird durch das Verwenden von verbundenen *Daten-Blöcken* in der Modellierungssoftware realisiert und hat den Vorteil, dass zugewiesene Materialien, erstellte UV-Maps (siehe Abschnitt 4.3.2) und andere Eigenschaften ebenfalls übernommen werden[4]. Zudem können so auch Wiederholungen im Oberflächenmodell, die keiner Symmetrie oder anderen Logik folgen, effizient umgesetzt werden [Sim13].

4.2.5 Optimierung von hochaufgelösten Oberflächenmodellen

Im Zuge eines Projektes dieser Art kann es dazu kommen, dass bestimmte Objekte bereits als 3D-Modelle umgesetzt wurden oder von bestimmten Quellen mitgeliefert werden. Um einen reibungslosen Einsatz dieser Modelle im HoT zu ermöglichen, müssen sie aber auch sie die zuvor beschriebenen Kriterien erfüllen. In Bereichen, in denen ein 3D-Modell andere Zwecke wie z. B. den eines Simulationskörpers oder eines Designobjekts erfüllt, ist eine diskrete Repräsentation der Oberfläche durch Punkte im Raum nicht üblich. Für den nötigen Detailgrad wird es stattdessen durch Kurven und Freiformflächen beschrieben und fällt in den Bereich des rechnerunterstützten Konstruierens (engl. „*computer-aided design*“, kurz „CAD“) [VWB+09]. Als 3D-Modell für den Einsatz in einer Game Engine ist diese bis ins kleinste Detail ausmodellierte Darstellungsart jedoch nicht ausgelegt. Deshalb muss das CAD-Modell zunächst in ein diskretes Oberflächenmodell überführt werden. Da die resultierende Auflösung des Polygonnetzes sehr hoch

[4] `https://docs.blender.org/manual/en/latest/files/data_blocks.html` (abgerufen am 07.03.2022)

ausfallen kann, muss dieses vor der weiteren Verwendung optimiert werden, um dem ressourcensparenden Ansatz gerecht zu werden, der in dieser Arbeit verfolgt wird (siehe Abschnitt 2.4).

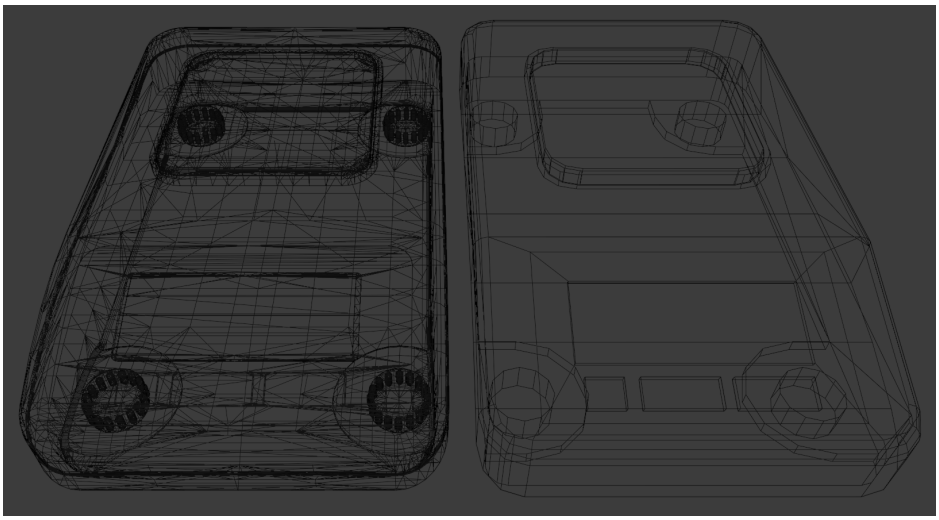


Abbildung 4.4: Vergleich (hochaufgelöstes gegen optimiertes Polygonnetz)

Die Überführung kann sowohl aufseiten des verwendeten Modellierungswerkzeugs (*Blender*)[5] als auch aufseiten der verwendeten Game Engine (*Unity*)[6] durch integrierbare Softwaremodule automatisiert werden. Diese Automatisierungen liefern allerdings nur in seltenen Fällen Ergebnisse, die ohne weiteres verwendbar sind. Eine direkte Einbindung in die Game Engine würde zudem Anpassungen im Sinne der Konformität zum restlichen Workflow ausschließen. Außerdem kann auch der Fall bestehen, dass nur das bereits überführte CAD-Modell vorliegt, ohne die Ausgangsdatei zur Verfügung zu haben. Aus diesen Gründen ist eine manuelle Optimierung unumgänglich, nicht zuletzt um die Hierarchie des 3D-Modells den zuvor aufgestellten Kriterien anzupassen. Wie aufwändig dieser Prozess ist, hängt jedoch vom Ergebnis der Überführung ab und kann vom Entfernen überflüssiger Geometrie bis hin zum kompletten Neuaufbau des 3D-Modells reichen. Im letzteren Fall wird das hochaufgelöste Oberflächenmodell als direkte Referenz genutzt, um Schlüsselpunkte des Objekts und Größenrelationen ablesen zu können. So wird der Modellierungsprozess beschleunigt, auch wenn das überführte Modell nicht direkt weiterverwendet werden kann.

[5] `https://docs.blender.org/manual/en/latest/addons/import_export/scene_dxf.html` (abgerufen am 05.03.2022)

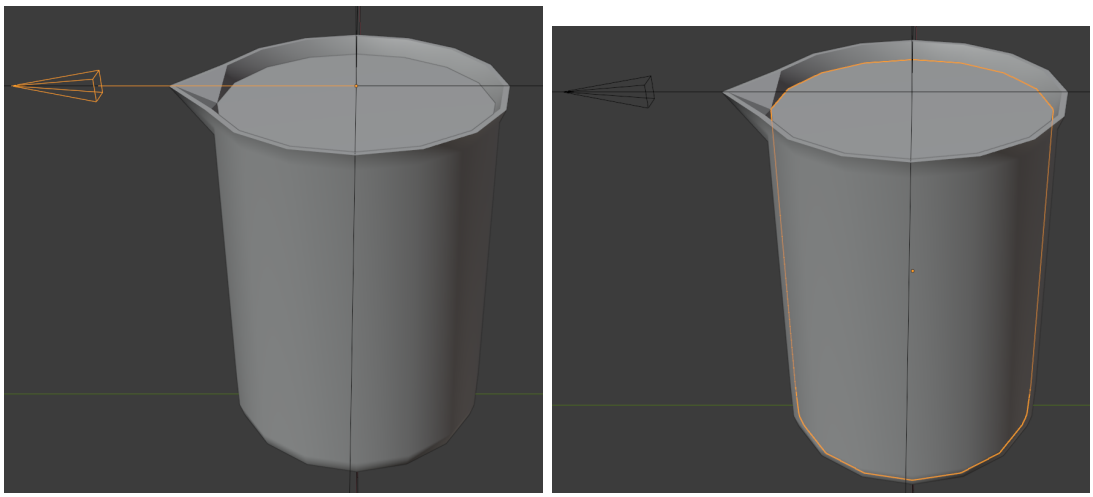
[6] `https://unity.com/products/pixyz` (abgerufen am 05.03.2022)

4.2.6 Flüssigkeitsbehälter

Der Umgang mit Flüssigkeiten stellt einen zentralen Punkt im HoT des *A-CINCH* Projekts dar. Dementsprechend ist es unumgänglich, eine dafür geeignete Darstellungsmöglichkeit zu finden. Die Lösungsansätze der numerischen Strömungsmechanik (engl. „*computational fluid dynamics*“, kurz „CFD“) liefern zwar realistische Ergebnisse, sind aber für den Einsatz in einer Echtzeitanwendung mit potentiell niedrigem Hardwareniveau nicht geeignet [Pow16]. Außerdem ist eine physikalisch korrekte Darstellung der flüssigen Chemikalien im Rahmen der Lehranwendung nicht erforderlich, um den Lehrstoff adäquat zu vermitteln, da dieser sich in erster Linie um die theoretischen Hintergründe der Flüssigkeiten dreht. Aus diesen Gründen wird die Optik und das Verhalten der im HoT verwendeten flüssigen Chemikalien lediglich approximiert. Im Zuge des Projekts wurde dafür ein spezieller Shader gewählt, der die gestellten Anforderungen erfüllen kann. Dazu zählen unter anderem die Darstellung eines gewissen Pegels im jeweiligen Behälter, das Umfüllen zwischen verschiedenen Gefäßen und die Farbänderung bei damit einhergehenden chemischen Reaktionen. Zur Umsetzung dieser Aufgaben bezieht sich der Quellcode des Shaders direkt auf folgende Bestandteile des 3D-Modells:

- **Ursprungspunkt**
  Jedes Polygonnetz besitzt einen dreidimensionalen Vektor, der dessen Position im 3D-Raum beschreibt. Dieser Ursprungspunkt (engl. „*origin point*“) dient als Koordinatenursprung für das lokale Koordinatensystem des Netzes und als Orientierung für alle Transformationen, die auf das Polygonnetz angewendet werden [Sim13]. Er kann während des Modellierungsprozesses nach Belieben angepasst werden, um bestimmte Aufgaben zu erfüllen[7].
  Im Falle der Flüssigkeitsdarstellung dient er als Referenzpunkt für die lineare Interpolation des Pegels beim Befüllen des Polygonnetzes und muss entsprechend positioniert werden. Der Shader geht davon aus, dass der Ursprungspunkt auf der Mitte der Strecke vom niedrigsten bis zum höchsten Punkt des Polygonnetzes im Bezug auf die Gravitationskraft liegt, der das 3D-Modell im HoT ausgesetzt wäre. Durch das Manipulieren der Interpolationsvariable im Intervall von 0 (leer) bis 1 (voll) kann der Pegel in der Game Engine variiert und somit animiert werden.

[7] `https://docs.blender.org/manual/en/latest/scene_layout/object/origin.html` (abgerufen am 01.03.2022)



(a) Vorwärtsvektor des 3D-Modells **(b)** Flüssigkeitskörper

Abbildung 4.5: Flüssigkeitsbehälter

- **Vorwärts gerichteter Vektor**
  Für eine effizientere Implementation des Umfüllens von Flüssigkeiten ist es von Vorteil, das Verhalten von Flüssgkeitsbehältern zu vereinheitlichen. Zu diesem Zweck wurde die Konvention getroffen, in jedem davon betroffenen 3D-Modell einen leeren Markierungsknoten am Punkt der Rotation zu platzieren, um den sich das Objekt beim Gießvorgang drehen würde. Dazu zählen z. B. die Spitze einer Tülle oder der Rand eines Reagenzglases. Da die Rotation eines 3D-Modells durch diese Maßnahme alleine noch einen ungewünschten Freiheitsgrad besitzen würde, wird der in Abschnitt 4.2.3 beschriebene vorwärts gerichtete Vektor genutzt, um die korrekte Rotationsrichtung zu erkennen. Aus diesem Grund ist es bei der Modellierung wichtig, den späteren Anwendungszweck des 3D-Modells zu berücksichtigen und es in die entsprechende Richtung aufzubauen. Durch diese analytische Vorüberlegung kann der Aufwand der Implementierung der Interaktion reduziert werden [PP10].

4.3 Materialien

Nachdem das Oberflächenmodell aufgebaut wurde und voraussichtlich keinen topologischen Änderungen mehr unterliegt, können dessen Flächen mit Materialien

versehen werden. Wie in Abschnitt 2.4 beschrieben, bestimmt ein Material nicht die physikalischen Eigenschaften, sondern lediglich das Erscheinungsbild der zugewiesenen Fläche. Die dafür benötigten Texturen werden im Anschluss an die Materialzuweisung erstellt.

4.3.1 Materialzuweisung

Damit die Flächen eines 3D-Modells in der späteren Anwendung dargestellt werden können, muss jeder Fläche eines Oberflächenmodells genau ein Material zugewiesen sein. Nur so kann im Zusammenspiel mit Kamera und Beleuchtung der virtuellen Szene der letztendliche Farbwert der Pixel im Ausgabebildschirm berechnet werden [NFHS19]. Entsprechend der in Abschnitt 2.4 getroffenen Vorüberlegungen zur Reduzierung des Ressourcenverbrauchs von Materialien, sollte die zugewiesene Anzahl nach Möglichkeit gehalten werden. Um diesem Anspruch gerecht zu werden, gilt es zunächst, organisatorische Aspekte innerhalb der Datei und des Projektverzeichnisses zu berücksichtigen. Es bietet sich eine Unterteilung nach Art des verwendeten Materials an [Kum20].

Prozedurale Materialien

Labormaterialien, die universell Anwendung bei den darzustellenden Objekten finden, sollten auch durch das selbe Material in der Entwicklungsumgebung repräsentiert werden. Darunter fallen grundlegende Stoffe wie z. B. Metall, Plastik oder Glas. Diese Stoffe zeichnen sich durch eine Oberflächenbeschaffenheit aus, die sich nach physikalischen Prinzipien durch einen Shader beschreiben lässt (engl. „*physically based rendering*“, kurz „PBR“). PBR-Materialien verwenden Algorithmen, Texturen und andere Eingabeparameter, um das natürliche Verhalten der realen Stoffe prozedural auf dem 3D-Modell zu approximieren [Kum20].

Durch diese generalisierte Funktionsweise ist es möglich, Flächen des 3D-Modells unabhängig von ihrer Größe oder Ausrichtung mit dem selben Material zu versehen. Der Detailgrad ist dabei beliebig skalierbar und das Erstellen von nahtlosen Oberflächen ist ebenfalls möglich [Kum20]. Da nur ein Material für alle Flächen des selben Stoffes verwendet wird, können die Flächen von 3D-Modellen,

die diese Grundstoffe repräsentieren, schnell und effizient mit Materialien versehen werden. Bei eventuell anfallenden Änderungen des Materials muss zudem nur ein einzelner Shader angepasst werden. So wird zudem ein einheitlicher Grafikstil gewährleistet. Ein weiterer Vorteil von prozeduralen Materialien wird in Abschnitt 4.3.2 näher erläutert.

Ein Nachteil dieser prozeduralen Herangehensweise ist jedoch die fehlende Kontrolle über die akkurate Platzierung von Oberflächentexturen und die damit erzeugbaren Effekte [Kum20]. Für die effiziente Umsetzung bestimmter Objekte ist das manuelle Erstellen von Materialien inkl. der benötigten Texturen immer noch unumgänglich.

Klassische Materialien

Im Gegensatz zum prozeduralen Ansatz ist das Verwenden von statischen Bildtexturen die verbreitetere Herangehensweise beim Erstellen von Materialien [Pai19]. Anstatt die Oberflächenbeschaffenheit algorithmisch zu beschreiben, wird jeder Fläche des 3D-Modells eine bestimmte Fläche auf einer Bildtextur zugewiesen. Dieser Abbildungsprozess vom 2D-Raum der Textur auf den 3D-Raum des Modells (siehe Abbildung 3.2) wird als Texturierung bezeichnet [Vil22]. So ist es möglich, komplexe Inhalte auf dem 3D-Modell darzustellen, ohne das zugrundeliegende Polygonnetz erweitern zu müssen. Vor allem Objekte mit einer großen Varianz in der optischen Oberflächenbeschaffenheit (z. B. Eingabefelder, Plastikgehäuse oder mit Symbolen bzw. Logos bestückte Flächen) profitieren von dieser Form der Materialien. Anstatt jedem gefärbten Bereich ein entsprechendes PBR-Material zuweisen zu müssen, kann die komplette Oberfläche mit nur einer Bildtextur abgedeckt werden. Der Ersteller des Materials hat so die vollständige Kontrolle über die Optik des 3D-Modells und kann jederzeit manuelle Anpassungen vornehmen[8].

Da diese Art von Materialien auf Bildtexturen basiert, ist sie auch an die Einschränkungen gebunden, die diese mit sich bringen. Dazu zählt zum einen der höhere Verbrauch von Speicherplatz, der mit dem Hinterlegen von Bildtexturen einhergeht und zum anderen die festgelegte Maximalauflösung. So kann es

[8] `https://docs.blender.org/manual/en/2.79/editors/uv_image/uv/overview.html` (abgerufen am 19.02.2022)

bei bestimmten Zoomstufen zu Unschärfen auf dem 3D-Modell kommen. Die Artefaktbildung ist jedoch nicht nur auf die Auflösung der Textur beschränkt, sondern kann auch durch Deformationen des Polygonnetzes entstehen. Durch die manuelle Aufteilung der Bildtextur sind nahtlose Oberflächenstrukturen zudem schwerer umzusetzen [Pai19]. Diesen Artefakten kann mit einer sinnvollen Flächenzuweisung im 2D-Raum der Textur (engl. „*UV mapping*“) vorgebeugt werden.

4.3.2 UV-Mapping

Damit die in einer zweidimensionalen Textur gespeicherten Informationen auf die Flächen eines 3D-Modells abgebildet werden können, muss zunächst eine Menge an 2D-Koordinaten festgelegt werden, mit denen diese Abbildung durchgeführt werden kann. Dieser Prozess wird als *UV-Mapping* bezeichnet [Sim13]. Dazu muss das Polygonnetz des Oberflächenmodells in den 2D-Raum der Textur überführt und dessen Flächen so verzerrungsfrei wie möglich ausgebreitet werden. Damit diese Projektion gelingt, wird das Polygonnetz an bestimmten Stellen mit Nähten (engl. „*seams*“) versehen und entsprechend aufgetrennt. Die nach dem Auftrennen noch zusammenhängenden Flächenverbunde werden als *Inseln* bezeichnet. Je größer eine Insel ist, desto mehr Pixelinformation kann sie aus der verwendeten Bildtextur auslesen, wodurch ein höherer Detailgrad erzielt werden kann. Die Inselgröße im 2D-Raum sollte deshalb proportional zur Größe der Geometrie sein, die sie im 3D-Modell repräsentiert [CN15]. Da die Inseln nach der Projektion nicht mehr wie zuvor in Beziehung zueinander stehen und unabhängig voneinander transformiert werden können, ist es wichtig, darauf zu achten, alle Inseln der selben Region gleichförmig zu transformieren. So wird das Größenverhältnis untereinander bewahrt und es wird verhindert, dass sich der Detailgrad innerhalb zusammenhängender Regionen des 3D-Modells unterscheidet.

Abhängig von der Art des zugewiesenen Materials unterscheidet sich das UV-Mapping in Ausführung und Aufwand:

- **Prozedurale Materialien**
  Durch ihre automatisierte Grundstruktur sind diese Materialien nicht auf die Projektion einer Bildtextur angewiesen. Für bestimmte Techniken ist

es jedoch nötig, dass das 3D-Modell dennoch eine UV-Map besitzt (siehe Abschnitt 4.3.4). Um Effekte zu erzielen, die auf algorithmisch erstellten Bildtexturen beruhen, müssen die Inseln der UV-Map allerdings nicht unbedingt einer menschlichen Logik unterliegen. Aus diesem Grund kann das UV-Unwrapping für solche 3D-Modelle mit einem bereits in der Modellierungssoftware implementierten Algorithmus durchgeführt werden. Neben dem Geschwindigkeitsvorteil stellt dieser auch eine proportionale Inselgröße und eine effiziente Verteilung der Inseln auf der UV-Map sicher[9].

- **Klassische Materialien**
  Mit der Verwendung einer festen Bildtextur geht auch ein weitaus höherer Aufwand beim Erstellen der UV-Map einher. Das initiale UV-Mapping kann zwar von einer automatisierten Funktion übernommen werden, aber eine manuelle Bearbeitung der so entstandenen Inseln ist unumgänglich, da die automatisch generierte Verteilung zwar Verzerrungen vorbeugt, aber keinen semantischen Zusammenhang zwischen den Inseln herstellt. Deshalb können Inseln, die verbundene Regionen auf dem 3D-Modell darstellen, über die ganze UV-Map verteilt liegen. An diesen Stellen muss also interveniert werden, um zu gewährleisten, dass die später genutzte Bildtextur (siehe Abschnitt 4.3.3) effizient erstellt und bearbeitet werden kann [Vil22].

Da das Laden einer Bildtextur in den Grafikspeicher (engl. „*video random access memory*“, kurz „VRAM“) ein ressourcenintensiver Prozess für eine Echtzeitanwendung ist, spielt dessen Optimierung eine wichtige Rolle. Sowohl mobile und webbasierte aber auch hoch budgetierte Videospiele nutzen deshalb stark optimierte UV-Maps für ihre 3D-Modelle. Hauptaugenmerk liegt dabei auf der effizienten Verteilung der Inseln in Verbindung mit der Reduzierung ihrer Gesamtzahl. Um dieses Ziel zu erreichen, gibt es verschiedene Optimierungspraktiken, die unabhängig vom verwendeten UV-Editor anwendbar sind [Vil22].

Entfernung unnötiger Flächen

Jede einzelne Fläche eines Oberflächenmodells besitzt eine Ausrichtung im 3D-Raum. Dieser Vektor wird als *Normalenvektor* bezeichnet und steht orthogonal

[9] `https://docs.blender.org/manual/en/2.79/editors/uv_image/uv/editing/unwrapping/mapping_types.html` (abgerufen am 20.02.2022)

zur zugehörigen Fläche. Mit Hilfe dieser Richtungsinformation kann während des Renderings ermittelt werden, ob eine Fläche der Kamera zugewandt ist oder nicht. Sollte die Fläche vom Blickwinkel der Kamera aus nicht sichtbar sein, muss die Fläche während des Renderings nicht gezeichnet werden und wird deshalb schon im Vorfeld aussortiert. Diese Art der Auslese wird als *Backface Culling* bezeichnet und ist ein grundlegendes Optimierungsprinzip in der Computergrafik [NFHS19]. Indem ohnehin nicht sichtbare Flächen bei der Modellierung direkt entfernt werden, kann dieses Prinzip in die Optimierung des Polygonnetzes und damit der UV-Map mit einbezogen werden. Betroffen davon sind z. B. Flächen, die an der Unterseite des Objekts positioniert sind oder während des HoT nicht erreicht werden können. So wird der Platz auf der UV-Map nicht von unnötigen Inseln blockiert und relevante Inseln können mehr Platz einnehmen, was den Detailgrad erhöht und mehr Spielraum für das Arrangement der Inseln bietet.

Platzierung der Nähte

Um ein kontrolliertes UV-Mapping zu ermöglichen, müssen zunächst bestimmte Kanten des Polygonnetzes als Nähte markiert werden. An diesen Stellen wird das Netz aufgetrennt und anschließend auf der 2D-Fläche ausgebreitet. Damit dabei möglichst wenige Inseln entstehen, sollten so wenig Nähte wie möglich gesetzt werden, während gleichzeitig so viele Nähte wie nötig gesetzt werden müssen, um Verzerrungen bei der Projektion gering zu halten [Sim13]. Generell sollte versucht werden, die Übergänge zwischen den Inseln auf der 3D-Oberfläche zu kaschieren, indem die Nähte an unauffälligen Stellen des Polygonnetzes platziert werden[10]. Der beste Ort zum Platzieren der Nähte kann dabei von Modell zu Modell variieren, allerdings bieten sich Regionen an, die Umbrüche in ihrer Oberflächenbeschaffenheit aufweisen, wie z. B. Täler und Falten im Polygonnetz [Sim13]. So wird die optische Qualität des 3D-Modells weniger beeinträchtigt. Im Falle der Laborutensilien bieten sich Orte wie Rückseiten und Oberflächen, die zur Wand oder zur Arbeitsoberfläche gerichtet sind, zum Platzieren der Nähte an (siehe Abbildung 4.6). Diese bekommt der Nutzer im Normalfall nicht zu sehen und durch die meist statische Natur dieser Objekte (z. B. Schränke oder geschlossene Arbeitsbereiche) ist eine Animation, die diese Bereiche zeigt, unwahrscheinlich.

[10] `https://docs.blender.org/manual/en/2.79/editors/uv_image/uv/editing/unwrapping/seams.html` (abgerufen am 20.02.2022)

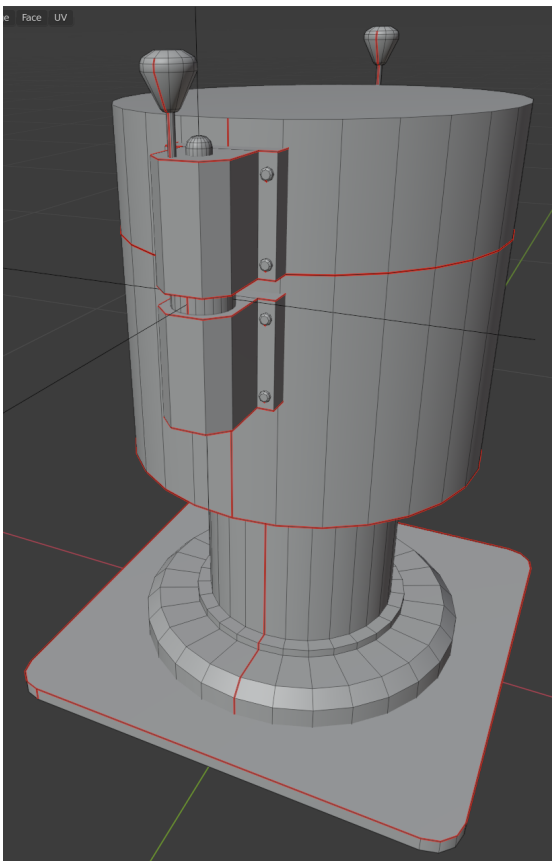


Abbildung 4.6: Platzierte Nähte auf der Rückseite (*rot*)

Arrangement der Inseln

Das Verteilen der Inseln im beschränkten Raum der UV-Koordinaten ist der letzte Schritt des UV-Mappings. Abhängig vom Projekt kann es in einer Entwicklungspipeline Konventionen geben, die bestimmte Ausrichtungen und Positionen für die Inseln vorschreiben [Vil22]. Im Zuge dieser Arbeit werden jedoch keine objektübergreifende Konventionen definiert, da die betroffenen 3D-Modelle keinem einheitlichen Aufbau folgen, wie es z. B. beim UV-Mapping von humanoiden Charaktermodellen der Fall wäre. Dieser Aufwand hätte also keinen Mehrwert im Bezug auf die darzustellenden Laborutensilien. Innerhalb verschiedener Ausführungen eines bestimmten Geräts wird das Arrangement der Inseln aber einheitlich gehalten, um sich auf unterscheidende Regionen konzentrieren zu können. Außerdem können deckungsgleiche Bereiche von Texturen so 1:1 wiederverwendet werden.

Ein weiterer wichtiger Punkt ist das Verhindern von Überlappungen beim Platzieren der Inseln. Da die selbe Texturinformation nicht doppelt auf das 3D-Modell projiziert werden soll, darf auch keine Insel die selbe Region auf der Textur abgreifen [CN15]. So werden perfekte Dopplungen der Oberflächenbeschaffenheit vermieden, die in der Natur nicht vorkommen würden. Außerdem profitieren selbst optisch identische Flächen von dieser Konvention, da sie sich an unterschiedlichen Stellen des 3D-Modells befinden und dementsprechend auch anderen selbst induzierten Schattenwürfen (siehe Abschnitt 4.3.4) unterliegen. Dieser Textureffekt greift bei der Texturgenerierung und der Darstellung durch den Shader auf die UV-Map zurück, was bei Überschneidungen von solchen Inseln zur Artefaktbildung führen würde.

4.3.3 Albedo Textur

Das am häufigsten eingesetzte Verfahren zum Versehen einer Fläche mit Feinstruktur ist der Einsatz einer Bildtextur (siehe Abbildung 3.2). Unter Verwendung der UV-Map werden die Farbwerte durch lineare Interpolation zwischen den Punkten des Polygonnetzes auf das Oberflächenmodell übertragen. Dieser Vorgang kostet verhältnismäßig wenig Rechenzeit [NFHS19]. Anstatt die Oberflächenstruktur mit nur einer Bildtextur darzustellen, die so viele Information wie möglich enthält, wird das Erscheinungsbild eines PBR-Materials jedoch durch die Kombination verschiedener Texturen erzeugt, die jeweils separate Aufgaben erfüllen. So können erweiterte Beleuchtungstechniken angewandt und mehr Einfluss auf das optische Erscheinungsbild genommen werden. Die Basis dafür bildet die sogenannte *Albedo* Textur, welche die grundlegenden Farbinformationen speichert [McD18]. Die Albedo Textur sollte im Allgemeinen keine Beleuchtungsinformationen wie z. B. Schatten enthalten, da diese später kontextabhängig hinzugefügt werden[11]. Es werden also nur die unverfälschten Farbwerte gespeichert, die auf dem Oberflächenmodell zu sehen sein sollen. Der Einsatz von Schrift ist davon allerdings klar getrennt, da diese aus verschiedenen Gründen erst später in der Game Engine hinzugefügt wird (siehe Abschnitt 5.3.2).

[11] `https://docs.unity3d.com/Manual/StandardShaderMaterialParameterAlbedoColor.html` (abgerufen am 03.03.2022)

4.3.4 Ambient Occlusion

Ein wichtiger Aspekt der Wahrnehmung der Plastizität eines Objekts ist die Wirkung von Schatten. Regionen, von denen weniger Licht reflektiert wird, erscheinen dunkler und tragen so zur Tiefenwahrnehmung bei. In der 3D-Modellierung wird dieser Effekt als *Ambient Occlusion* bezeichnet und kann durch verschiedene Techniken erzeugt werden [Fla10]. Eine zur Laufzeit der Anwendung ressourcensparende Umsetzung ist der Einsatz einer speziellen, im Vorfeld generierten Textur. Diese schwarz-weiße Textur (siehe Abbildung 3.2b) speichert die Schattenintensität an der korrespondierenden Stelle einer Fläche im Intervall von 0 (Schwarz) bis 1 (Weiß). Anschließend wird der diffuse Farbwert an der selben Stelle der Fläche mit diesem Faktor multipliziert, um so einen Schattenwurf zu simulieren [CN15]. Die Dateigröße der Textur selbst kann deshalb optimiert werden, indem kein Alpha-Kanal gespeichert und die Bilddatei beim Export als monochrom gekennzeichnet wird.

Da diese Art der Schattenberechnung statischer Natur ist, kann ihr Ergebnis zwar effizient aus der *Ambient Occlusion* Textur ausgelesen werden, allerdings unterliegt sie deshalb auch bestimmten Einschränkungen, die beim Erstellen der Textur berücksichtigt werden müssen:

- **Dynamische Polygonnetze**
  Teile eines 3D-Modells, die sich während der Interaktion im HoT bewegen, verändern ihre Position zum Rest des Oberflächenmodells. Die zuvor berechneten Schattenwürfe sind anschließend nicht mehr aktuell und müssten neu berechnet werden, was den Vorteil der *Ambient Occlusion* Textur negieren würde. Deshalb müssen statische und dynamische Polygonnetze während der Texturgenerierung separat behandelt werden [NFHS19].

- **Transparenz**
  Im Gegensatz zu opaken Polygonnetzen profitieren lichtdurchlässige Flächen nicht von dieser Art der Schattenbildung. Ihr Erscheinungsbild ist zu stark vom Kontext des Lichteinfalls in der Szene abhängig, als dass dieses im Vorfeld durch *Ambient Occlusion* Berechnungen in einer Textur abgebildet werden kann. Deshalb werden die transparenten Flächen eines 3D-Modells dabei ausgeschlossen und ihre Schattenbildung zur Laufzeit durch den zugewiesenen Shader berechnet (siehe Abschnitt 4.3.1).

- **Verhältnismäßigkeit**
  Da nicht alle Objekte im gleichen Maße von einer *Ambient Occlusion* Textur profitieren, muss beim Modellierungsprozess abgewogen werden, ob diese für das jeweilige 3D-Modell einen Mehrwert bietet. Gleichförmige Oberflächen, die keine markanten Vertiefungen oder abstehende Regionen aufweisen, ziehen keinen oder nur wenig Nutzen aus diesem Textureffekt. Von daher sollte im Sinne eines effizienten und schnellen Workflows darauf verzichtet werden, eine *Ambient Occlusion* Textur für diese 3D-Modelle zu generieren. So wird außerdem Speicherplatz gespart, da keine Texturen gespeichert werden müssen, die aus einheitlich weißen Regionen bestehen.

5 Anwendung des Workflows im Projekt

Dieses Kapitel beschreibt die Umsetzung des HoTs im Bezug auf die Verbindung von Modellierung und dem Einsatz des 3D-Modells in der Game Engine. Dazu wird zunächst der organisatorische Aufwand erläutert, der unternommen wurde, damit die Kollaboration der einzelnen Teammitglieder möglichst reibungslos vonstatten gehen kann. Anschließend wird die Integration eines 3D-Modells von der Modellierungssoftware in die Entwicklungsumgebung beschrieben.

5.1 Organisation

Wie für viele andere Probleme in der Informatik ist es auch für Modellierungsaufgaben sinnvoll, den Gesamtaufwand zunächst auf kleinere Unteraufgaben aufzuteilen. Durch dieses Teile-und-herrsche-Prinzip kann die komplexe Aufgabenstellung der Objektmodellierung in eine übersichtliche und leichter zu verfolgende Ausgangslage gebracht werden [Tho15].

5.1.1 Aufbau des Modellkatalogs

Entsprechend der in Abschnitt 3.3 angesprochenen Prinzipien zur Einführung der *Kanban*-Technik in ein Projekt, wurden die einzelnen Ablaufprotokolle des HoT zunächst ausgewertet, um den Gesamtumfang der zu modellierenden Objekte einschätzen zu können. Dazu wurde eine Liste erstellt, die all diese Objekte

	A	B	C	D	E	F	G	H
1	**Category**	**HoT**	**Object**	**Priority**	**Status**	**File Name**	**Notes**	**Image (Object)**
2	**0_Entry Lock**	**Enter / Leave**	Bench (Schuhschrank)		DONE			
3	**0_Entry Lock**	**Enter / Leave**	Personal Dosimeter, electronic (x ray)		REVIEW	personalDosimeter	https://www.mnt-int.com/gen/file/2010/04/personal-dosimeter.pdf	
4	**0_Entry Lock**	**Enter / Leave**	Personal Dosimeter, film badge (gamma rays, x rays and beta particles)		DONE	dosimeter_persona	https://de.wikipedia.org/wiki/Filmdosimeter	
5	**0_Entry Lock**	**Enter / Leave**	First Aid Box (to mount at a wall)		REVIEW	first_aid_kit	hierfür wäre Textur gut, damit man eindeutig erkennt was es ist	
6	**0_Entry Lock**	**Enter / Leave**	Full Body Monitor		IN PROGRESS	full_body_monitor	https://www.berthold.com/de/strahlenschutz/produkte/kontaminationsmonitore/personenkontaminationsmonitor-lb147-lb148/#	
7	**0_Entry Lock**	**Enter / Leave**	Gloves (Behälter)		BACKLOG			
8	**0_Entry Lock**	**Enter / Leave**	Kleiderschrank		DONE	wardrobe	geschlossener Kleiderschrank	
9	**0_Entry Lock**	**Enter / Leave**	Lab Coat		TO DO			

Abbildung 5.1: Auszug aus dem Modellkatalog

aufführt und entsprechend ihres zugehörigen Experiments sortiert. Dabei wurde gezielt darauf geachtet, Vorkommnisse gleicher Objekte mit einer deckungsgleichen Benennung zu versehen. Einträge für Objekte, die sich nur punktuell unterscheiden wie z. B. die gleiche Art eines Glasgefäßes mit unterschiedlichem Fassungsvermögen, beginnen mit der selben Benennung und erhalten ihre Spezifikation am Ende des Namens. Diese Konventionen ermöglichen die Verwendung eines automatisierten Skripts, das Änderungen an doppelten Einträgen synchronisiert. Außerdem können damit Sortierungen der Objektliste entsprechend der vorhandenen Spalten vorgenommen werden.

Jeder Eintrag im so zusammengetragenen Modellkatalog enthält folgende Datenfelder (siehe Abbildung 5.1):

(A) **Category**
Die fachliche Einordnung des Eintrags in das übergeordnete Themengebiet.

(B) **HoT**
Die Benennung das Experiments, aus dem der Eintrag stammt.

(C) **Object**
Der Benennung des zu modellierenden Objekts.

(D) **Priority**
Gibt an, ob der Eintrag eine erhöhte Priorität besitzt (rot eingefärbt) oder nicht.

(E) **Status**
Gibt den aktuellen Bearbeitungsstatus des Eintrags an (siehe Abschnitt 5.1.2).

(F) **File Name**
Der Dateiname des 3D-Modells, das für diesen Eintrag erstellt wurde und unter dem es im Repositorium zu finden ist.

(G) **Notes**
Ein offenes Textfeld, das für Anmerkungen aller Art genutzt werden kann (z. B. für Verlinkungen auf Referenzmaterial oder zum Hinterlassen von Feedback).

(H) **Image (Object)**
Ein Bild des zu modellierenden Objekts, um sicherzustellen, dass sich der Eintrag auf das korrekte Objekt bezieht. Bietet Teammitgliedern ohne Chemiekenntnisse eine Assoziation zum Eintrag.

Durch die Auflistung der Gesamtheit der Objekte ist nun die Grundlage geschaffen, um das *Kanban*-System anwenden zu können. Nach *Epping* [Epp11] wird dieses durch vier Elemente charakterisiert:

- **Pull**
  Im Zuge des Workflows werden die einzelnen Aufgaben nicht von einer Phase in die Nächste geschoben (engl. „*push*“), sondern vom jeweiligen Teammitglied selbst für sich herangezogen (engl. „*pull*“). Damit wird verhindert, dass sich die Aufgaben an einer Stelle im Prozess stauen, da eine Aufgabe nur dann gezogen wird, wenn auch Kapazitäten für deren Bearbeitung zur Verfügung stehen. So wird eine selbstorganisierte Arbeitsweise gefördert und die Zahl der momentan zu bearbeitenden Aufgaben limitiert.

- **Limitierte Mengen**
  Die Anzahl der Aufgaben, die gleichzeitig bearbeitet werden dürfen, ist begrenzt. Durch diese Vorkehrung wird eine Überlastung der Teammitglieder verhindert und die durchschnittliche Bearbeitungszeit einer Aufgabe reduziert [HU15], worauf der Hauptfokus des *Kanban*-Systems liegt. Außerdem wird die Mehrbelastung durch das Springen zwischen verschiedenen Aufgaben gesenkt.

- **Transparente Information**
  Damit die Arbeit reibungslos ablaufen kann, müssen alle relevanten Informationen für das Team einsehbar sein. Dazu zählen die Phasen, die eine Aufgabe durchläuft (siehe Abschnitt 5.1.2), die Gesamtheit aller Aufgaben, die momentan bearbeiteten Aufgaben (engl. „*work in progress*“), die gesetzte Limitierung dieser Aufgaben und die Personen, die an der Umsetzung der Aufgaben beteiligt sind. Zur Visualisierung dieser Informationen wird das *Kanban*-Brett in Form des zuvor beschriebenen Modellkatalogs genutzt. Dieser wird über einen Cloud-Dienst (*Google Drive*) allen Teammitgliedern zur Verfügung gestellt, sodass Änderungen direkt einsehbar sind und parallel vorgenommen werden können.

- **Kontinuierliche Verbesserung**
  Da sich die Rahmenbedingungen des Projekts (z. B. Teammitglieder, Termine oder Anforderungen) im Laufe der Bearbeitung ändern können, wird dieser Umstand im *Kanban*-System bewusst aufgegriffen. Um über diese Änderungen zeitnah informiert zu werden, ist eine regelmäßige Absprache innerhalb des Teams unverzichtbar. So können Anpassungen unter aktiver Teilnahme der betroffenen Personen vorgenommen werden, um Elemente und Techniken des Workflows stetig zu verbessern.

5.1.2 Bearbeitungsstatus und Review-Schleife

Um die Übersichtlichkeit und Transparenz der bearbeiteten Einträge zu wahren, wurde ein Ablaufplan zur Angabe des Bearbeitungsstatus erstellt, der den Grundelementen nach *Epping* [Epp11] entspricht (siehe Abbildung 5.2). Dieser besteht aus fünf Stufen, die folgende Zustände im Zyklus eines Objekts beschreiben:

1. **BACKLOG**
   In diesem Zustand befindet sich jeder Eintrag automatisch, sobald er in den Modellkatalog aufgenommen wird. Es wird signalisiert, dass das Objekt im Laufe des Projekts umgesetzt werden muss. Sobald das Objekt relevant für die Implementierung ist und zeitnah benötigt wird, wird es in Absprache mit den Entwicklern in den nächsten Zustand überführt.

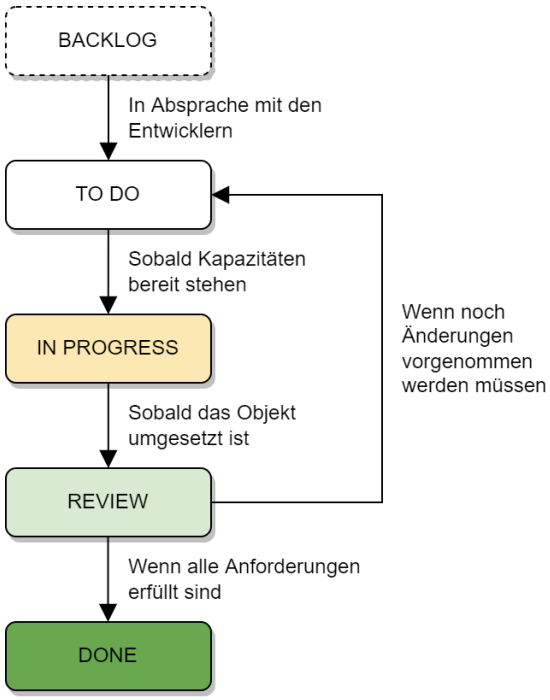


Abbildung 5.2: Flussdiagramm des Bearbeitungsstatus

2. **TO DO**
   Einträge, die sich in diesem Zustand befinden, stehen zur Umsetzung bereit. Sobald Kapazitäten seitens der Modellierer bestehen, wird ein solcher Eintrag in den nächsten Zustand gezogen. Dabei werden Objekte mit einer Markierung im Datenfeld *(D) Priority* bevorzugt behandelt.

3. **IN PROGRESS**
   Dieser Status wird dem Eintrag zuteil, der sich momentan in Bearbeitung befindet. Pro Modellierer ist dieser Zustand auf ein Objekt limitiert, um eine schnelle Durchlaufzeit zu gewährleisten [HU15]. Dadurch ist für jedes Teammitglied zu jeder Zeit ersichtlich, welches 3D-Modell als nächstes fertiggestellt wird.

4. **REVIEW**
   Wenn ein Objekt soweit umgesetzt wurde, dass keine weiteren Modellierungsziele mehr ersichtlich sind, wird der Eintrag in diesen Zustand übergeben. Hier angelangt prüft ein Entwickler, ob alle Anforderungen erfüllt sind. Je nach Ausgang dieser Begutachtung schickt er den Eintrag zurück in die Review-Schleife oder in den nächsten und finalen Zustand.

5. **DONE**
   Ein 3D-Modell, das alle Anforderungen erfüllt, erhält diesen Status und gilt somit als erfolgreich umgesetzt.

In Kombination mit wöchentlichen Absprachen zum erledigten Fortschritt wurde so das organisatorische Gerüst konstruiert, welches die zu modellierenden Objekte arrangiert, damit das Arbeitspensum von Woche zu Woche überschaubar ist. So konnten von insgesamt 166 Einträgen 72 Stück (43 %) abgeschlossen werden (*Done*). Noch einmal 31 Einträge (19 %) wurden so weit umgesetzt, dass sie sich zurzeit in der *Review*-Phase befinden.

5.2 Modellierungssoftware (Blender)

Für das Anfertigen der Vielzahl an Objekten im Katalog ist es sinnvoll, die identischen sich oft wiederholenden Abläufe in der Modellierungssoftware zu standardisieren. So kann eine identische Abwicklung der Einträge garantiert und Arbeitszeit eingespart werden.

5.2.1 Aufbau der Standard-Datei

Blender ist ein freies Open-Source-Modellierungsprogramm, das verschiedene Personalisierungsmöglichkeiten bietet[1]. Dazu zählt u. a. das Einrichten einer Szene, die beim Erstellen einer neuen Datei als Ausgangslage verwendet wird[2]. Diese Option wird genutzt, um eine an das Projekt angepasste Unterteilung der Szene einzurichten. Dabei wird mit den sogenannten *Sammlungen* (engl. „*collections*“) gearbeitet, die eine spezielle Möglichkeit der Organisation bieten, da sie keine Transformationsbeziehungen mit den Polygonnetzen in der Szene eingehen, wie es bei anderen Eltern-Kind-Beziehungen der Fall wäre (siehe Abschnitt 4.2.1 und 4.2.2)[3]. Sie unterteilen den Szene-Graphen in eigene benutzerdefinierte

[1] `https://www.blender.org/about/` (abgerufen am 25.03.2022)
[2] `https://docs.blender.org/manual/en/latest/editors/3dview/startup_scene.html` (abgerufen am 25.03.2022)
[3] `https://docs.blender.org/manual/en/latest/scene_layout/collections/collections.html` (abgerufen am 25.03.2022)

Abschnitte. Die erste Sammlung erhält den Titel „`references`“ und beinhaltet alle Bilder, Objekte oder Hilfslinien, die bei der Umsetzung des jeweiligen Objekts helfen. Da beim späteren Exportvorgang jedoch nur die relevanten Objekte aus der Szene exportiert werden sollen, wird eine Sammlung mit dem Prefix „`export_`“ erstellt gefolgt vom Dateinamen, der dem Datenfeld *(F) File Name* (siehe Abschnitt 5.1.1) entnommen wird. Prinzipiell kann eine Modelldatei auch mehrere *Export*-Sammlungen besitzen (z. B. bei einem Glasgefäß, das mit verschiedenen Volumina modelliert wird), aber die meisten 3D-Modelle kommen mit einer *Export*-Sammlung aus. Innerhalb dieser Sammlung wird ein leerer Knoten im Ursprung des globalen Koordinatensystem mit der Bezeichnung „`ctrl_root`“ erstellt. Der Knoten stellt die oberste Ebene in der Hierarchie des 3D-Modells dar. Alle anderen Teile des Objekts sind ihm untergeordnet. In der Game Engine wird dieser Knoten als Haupttransformationpunkt des Modells (engl. „*pivot point*“) dienen. Alle anderen Objekte, die standardmäßig in einer leeren Szene enthalten sind (Kamera, Lichtquelle und Beispielwürfel), werden entfernt.

5.2.2 Export

Für den Exportvorgang können ebenfalls Einstellungen als Vorlage gespeichert werden, die anschließend dateiübergreifend zur Verfügung steht. Die Konfiguration der relevanten Parameter, welche eine nahtlose Überführung in die Game Engine ermöglicht, wird in Abbildung 5.3 dargestellt. Sie beinhaltet folgende Punkte[4]:

Include

- Das Markieren der Option *Limit to Active Collection* weißt das Programm an, ausschließlich den Inhalt der momentan ausgewählten Sammlung beim Exportvorgang zu berücksichtigen. Hierfür wird die *Export*-Sammlung verwendet, welche in Abschnitt 5.2.1 angesprochen wurde.
- Die Auswahl der *Object Types* beschränkt die Art der zu exportierenden Objekte innerhalb der aktiven Sammlung. Da für die 3D-Modelle keine

[4] `https://docs.blender.org/api/current/bpy.ops.export_scene.html` (abgerufen am 27.03.2022)

Armaturen, d. h. Hilfsgerüste zur Animation skelettbasierter Polygonnetze, erstellt wurden und sowohl Lichtquellen als auch die Kamera seitens der Game Engine gehandhabt werden, werden nur die zwei Typen *Empty* und *Mesh* ausgewählt. *Custom Properties* finden ebenfalls keine Verwendung in diesem Workflow.

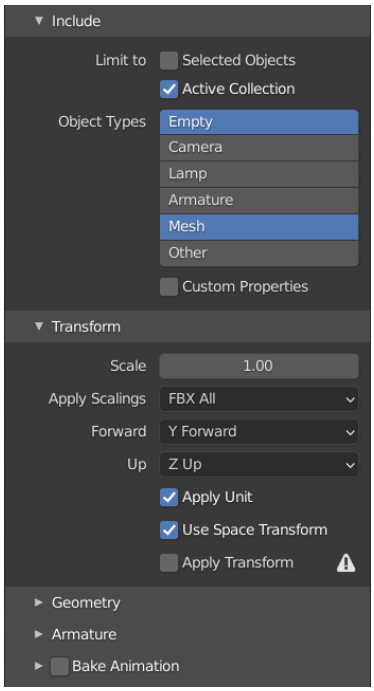


Abbildung 5.3: Exporteinstellungen (*Blender*)

Transform

- Der globale Skalierungsfaktor des 3D-Modells (*Scale*) bleibt unverändert bei einem neutralen Wert von 1. Da das Modell bereits im Maßstab 1:1 erstellt wurde, würden bei einem anderen Faktor ungewollte Skalierungen auftreten.
- Die Einstellungsmöglichkeit *Apply Scalings* gibt an, wie mit benutzerdefinierten und an Maßeinheiten orientierten Skalierungen umgegangen wird. *Blender* orientiert sich beim Import an der in der FBX-Datei gespeicherten internen Skalierung, um Maßstäbe korrekt abzubilden. Dieses Verhalten kann jedoch in vielen Anwendungen nicht als gegeben angesehen werden. Aus diesem Grund wird die Option *FBX All* gewählt, welche sämtliche

in *Blender* vorgenommenen Einstellungen im Bezug auf Skalierungen und Maßeinheiten in der Exportdatei speichert. So wird eine maßstabsgetreue Darstellung unabhängig von der verwendeten Software ermöglicht. Dort müssen jedoch gegebenenfalls noch bestimmte Einstellungen vorgenommen werden (siehe Abschnitt 5.3.1).

- Der vorwärts gerichtete Vektor des Modells (siehe Abschnitt 4.2.3) wird im Dropdown-Menü *Forward* angegeben. Bei der Erstellung der 3D-Modelle wurde dafür stets der positive Richtung der Y-Achse (*Y Forward*) genutzt.

- Der aufwärts gerichtete Vektor des Modells wird im Dropdown-Menü *Up* angegeben. Innerhalb von *Blender* ist das die positive Richtung der Z-Achse (*Z Up*). In Kombination mit dem Vorwärtsvektor ist die genaue Ausrichtung des 3D-Modells nun definiert.

- Die Auswahl der Option *Apply Unit* weist dem Modell die zuvor eingestellte Maßeinheit zu. In Falle des HoTs ist das die Angabe der Längen in Meter. Wenn diese Option nicht markiert wird, würde eine abstrakte Längeneinheit verwendet werden, die keinen Bezug zu einer realen Maßeinheit besitzt.

- Mit dem Parameter *Use Space Transform* wird angegeben, dass die globalen Transformationswerte des 3D-Modells in der FBX-Datei gespeichert werden, anstatt die lokalen Werte zu nutzen. Da der Ursprung des globalen Koordinatensystems in *Blender* als Referenz für alle Modelle genutzt wird, ist diese Option in der Vorlage aktiviert.

- *Apply Transform* ist ein experimentelles *Blender*-Feature, das die Transformationswerte der einzelnen Teile des Oberflächenmodells in die Exportdatei integriert (engl. „*bake*“). Da dieser Vorgang jedoch automatisiert und bekannt dafür ist, ungewollte Transformationen hervorzurufen, wenn die Ausrichtung des Ziel-Koordinatensystems nicht mit dem Koordinatensystem von *Blender* übereinstimmt, wird dieses Feature deaktiviert. Der Umgang mit den Transformationswerten wird stattdessen während der Modellierung manuell vorgenommen (siehe Abschnitt 4.2.2).

Weitere Einstellungsmöglichkeiten

- Die Einträge im Reiter *Geometry* behalten ihre Werkseinstellungen bei. Sie sorgen u. a. dafür, dass die verwendeten Modifizierungen der Polygonnetze (z. B. Spiegelungen oder Glättungen der Oberfläche) im exportierten Modell eingearbeitet werden. Gleichzeitig bleiben sie aber in der Ursprungsdatei erhalten, was die Möglichkeit für spätere Änderungen offen hält.

- *Armature*-Objekte werden bei den 3D-Modellen für das HoT nicht verwendet und wurden bereits bei den *Object Types* abgewählt. Deshalb ist dieser Reiter in diesem Fall für den Export irrelevant.

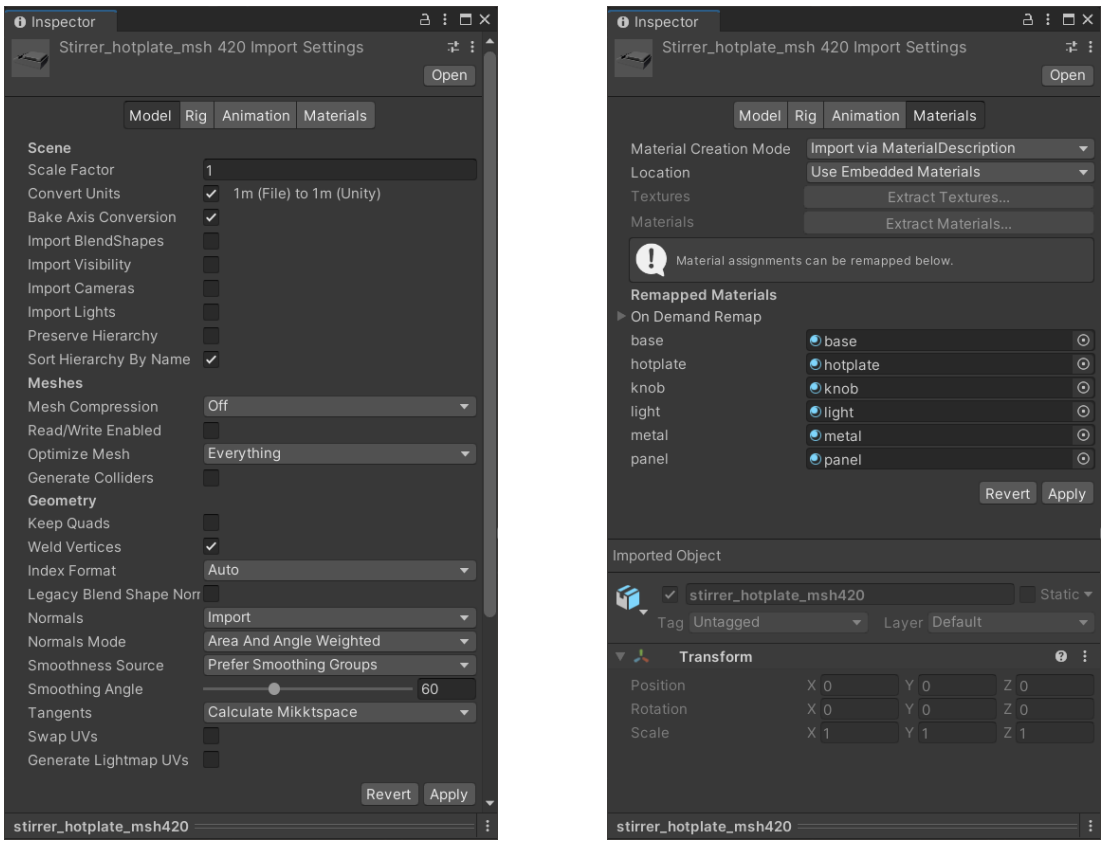
- Animationen werden erst in der Game Engine erstellt und müssen deshalb nicht exportiert werden. Aus diesem Grund wird die Option *Bake Animation* deaktiviert.

5.3 Game Engine (Unity)

Damit die modellierten Objekte in der Entwicklungsumgebung maßstabsgetreu dargestellt und korrekt verwendet werden können, sind beim Einbinden der 3D-Modelle bestimmte Aspekte zu beachten.

5.3.1 Import

Als Gegenstück zu den in Abschnitt 5.2.2 erläuterten Exporteinstellungen müssen in *Unity* die passenden Importeinstellungen gewählt werden. Nur so kann gewährleistet werden, dass die Inhalte der FBX-Dateien passend interpretiert werden. Die Einstellungen im *Unity*-Inspektor sind in vier Bereiche untergliedert (siehe Abbildung 5.4). Da es eine Vielzahl von Optionen gibt, wird im Folgenden nur auf die relevanten und veränderten Parameter eingegangen.



(a) *Model* **(b)** *Materials*

Abbildung 5.4: Importeinstellungen (*Unity*)

Model

- Der Skalierungsfaktor wird wie zuvor bei den Exporteinstellungen auf den neutralen Wert von 1 gesetzt.

- *Convert Units* wird aktiviert, damit das importierte 3D-Modell seinen Originalmaßstab beibehält.

- *Bake Axis Conversion* wird aktiviert, damit die Achsenausrichtung von Modellen aus Koordinatensystemen, die nicht mit der Standardausrichtung in *Unity* übereinstimmen, entsprechend angeglichen wird[5]. Diese Konvertierung ist essentiell, da sich die Achsenausrichtung in *Blender* unterscheidet. Das Aktivieren dieser Option lässt die Überführung der Ausrichtung auf alle Objekte in der Hierarchie des Modells anwenden. Andernfalls würde nur

das oberste Objekt in der Hierarchie konvertiert werden, was zu ungewollten Transformationen innerhalb des 3D-Modells führen würde.

- Beim Import von 3D-Modellen, auf dessen Polygonnetze von einem Skript aus zugegriffen werden soll, muss die Option *Read/Write Enabled* aktiviert werden, damit die nötigen Schreib- und Lesemöglichkeiten zur Verfügung gestellt werden. Das trifft z. B. auf alle Flüssigkeitsbehälter zu, da deren Interaktionen über ein Skript gesteuert werden (siehe Abschnitt 4.2.6). Wenn dies nicht der Fall ist, sollte diese Option deaktiviert bleiben, um den Verbrauch von Speicherplatz zur Laufzeit zu reduzieren[6].

Rig & Animation

Da die erstellten 3D-Modelle keine *Armaturen* oder *Rigs* enthalten und die Animationen erst nach dem Import in *Unity* erstellt werden, sind beide Reiter für diesen Workflow irrelevant und behalten ihre Standardeinstellungen bei.

Materials

Wie in Abbildung 5.4b zu erkennen ist, werden in diesem Reiter alle dem Oberflächenmodell zugewiesenen Materialien (siehe Abschnitt 4.3.1) aufgeführt. Um diese Materialien in das Projektverzeichnis aufzunehmen, wird die Schaltfläche *Extract Materials* verwendet. Nachdem ein Verzeichnis gewählt wurde, beziehen sich die in diesem Reiter aufgeführten Referenzen nun auf die entpackten Materialeinträge. Sollte ein Material bereits vorhanden sein, wird dieses genutzt und kein Duplikat erstellt. Somit wird die dateiübergreifende Verwendung von Materialien ermöglicht und Stück für Stück eine Materialsammlung im *Unity*-Projekt angelegt. Der Unterpunkt *On Demand Remap* bietet zusätzlich eine Suchfunktion, welche das Projektverzeichnis nach passenden Materialeintragen durchsucht

[5] `https://docs.unity3d.com/ScriptReference/ModelImporter-bakeAxisConversion.html` (abgerufen am 29.03.2022)

[6] `https://docs.unity3d.com/ScriptReference/Mesh-isReadable.html` (abgerufen am 28.03.2022)

und die Referenzierung bei Übereinstimmungen automatisch erzeugt. Ein erneuter Import oder Änderungen am 3D-Modell ändern diese Verbindungen nicht, sodass diese nach einer Überarbeitung des Modells nicht verloren gehen[7].

5.3.2 Einbau von Textfeldern

Bestimmte Objekte, die im HoT verwendet werden, tragen Aufschriften oder besitzen ein Display, auf dem verschiedene Inhalte angezeigt werden können wie z. B. die Digitalanzeige einer Waage. Die auf solchem Weg dargestellten Ziffern und Schriftzeichen müssen variabel und gut erkennbar sein, um ihren Zweck der interaktiven Informationsvermittlung zu erfüllen. Von daher wäre eine Umsetzung über die in Abschnitt 4.3.3 beschriebene *Albedo* Textur nicht sinnvoll, da die mit ihr dargestellten Inhalte statischer Natur sind. Weder könnten die Zeichen zur Laufzeit durch ein Skript geändert werden, noch wären sie von Artefakten verschont, die bei einer Texturprojektion unter näherer Betrachtung auftreten. Aus diesen Gründen wird der Einbau von Textfeldern mit *Unity*-UI-Komponenten umgesetzt. Diese können über die API angesprochen werden[8] und ihre Lesbarkeit ist durch ihre dynamische Auflösung immer gegeben[9]. Um diesen Vorgang zu unterstützen, wurden bei der Modellierung leere Knoten (siehe Abschnitt 4.2.2) an den relevanten Stellen positioniert. Sie dienen nun als Ankerpunkte, die als Zentrum des eingefügten Textfelds genutzt werden. Dazu muss das Feld dem leeren Knoten untergeordnet und seine Transformationswerte anschließend zurückgesetzt werden, um es in Relation zum neuen Elternknoten zu zentrieren. Anschließend muss nur noch die Größe des Textfelds an die Gegebenheiten des 3D-Modells und an die Anforderungen der umzusetzenden Interaktion des HoTs angepasst werden.

[7] `https://docs.unity3d.com/Manual/FBXImporter-Materials.html` (abgerufen am 28.03.2022)

[8] `https://docs.unity3d.com/Packages/com.unity.ugui@1.0/api/UnityEngine.UI.Text.html` (abgerufen am 28.03.2022)

[9] `https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-CanvasScaler.html` (abgerufen am 28.03.2022)

5.3.3 Animation

Zur Implementierung bewegter Objekte wird sich in *Unity* der gängigen *Keyframe*-Animation bedient, welche bereits in Abschnitt 4.2.2 erläutert wurde. Die namensgebenden *Keyframes* werden dabei in sogenannten Clips gespeichert, welche sie in Relation zu einem frei wählbaren Objekt in der Hierarchie aufnehmen[10]. Das gewählte Objekt muss lediglich auf der selben oder einer höheren Stufe in der Modellhierarchie liegen. Hier erfüllen die zuvor platzierten leeren Knoten ihren Zweck. Soll z. B. der Deckel einer Zentrifuge geöffnet werden, wird das entsprechende Hilfsknotenobjekt als Ausgangspunkt beim Erstellen der Animation gewählt. Dazu wird eine neue Animationsspur (engl. „*animation track*“) auf einer sogenannten *Timeline* erstellt. *Timelines* sind *Unity*-Komponenten, welche als Container und Bedienelement für Animationssequenzen fungieren[11]. Sie kombinieren *Animation Clips*, Audiodateien und andere abspielbare Elemente wie z. B. Partikelsysteme auf einem übersichtlichen Zeitstrahl. Wenn nun bestimmte Rotationswerte für das Hilfsobjekt in Form von *Keyframes* gespeichert werden, rotieren auch alle untergeordneten Objekte mit. Sollte sich der Aufbau des Zentrifugendeckels im Laufe des Projekts ändern, kann der aufgenommene *Animation Clip* trotzdem weiter verwendet werden, da sich die Rotationswerte ausschließlich auf das Hilfsobjekt beziehen.

Diese Allgemeingültigkeit geht aber auch über den Zentrifugendeckel hinaus und könnte auf alle Deckel mit ähnlichem Aufbau angewandt werden, welche die selbe Bewegung ausführen sollen. Dazu muss nur der jeweilige Hilfsknoten des Deckels mit dem gespeicherten *Animation Clip* kombiniert werden. Da deren Ausrichtung durch die getroffenen Konventionen einheitlich ist (siehe Gleichung 4.1), kann die Animation wiederverwendet werden. Dasselbe gilt auch für die Wiederverwendung von *Timelines*. Falls die gleiche Animationssequenz mit anderen Objekten ausgeführt werden soll, müssen lediglich die Referenzen der betroffenen Objekte in den jeweiligen Animationsspuren ausgetauscht werden[12].

[10] `https://docs.unity3d.com/Manual/AnimationClips.html` (abgerufen am 28.03.2022)
[11] `https://docs.unity3d.com/Packages/com.unity.timeline@1.2/manual/index.html` (abgerufen am 28.03.2022)
[12] `https://docs.unity3d.com/2018.3/Documentation/Manual/TimelineOverview.html` (abgerufen am 28.03.2022)

6 Performanzanalyse

Zur Überprüfung der Arbeitsergebnisse, die mit dem vorgestellten Workflow kreiert wurden, wird in diesem Kapitel eine gängige Szenerie im Labor des HoTs auf ihre Performanz untersucht. Die systematische Untersuchung der Leistung mit einem vergleichbaren Ergebnis wird in der Computergrafik als *Benchmark* bezeichnet. Sie erlaubt es, objektive und empirisch belegte Aussagen über die Performanz der untersuchten Komponenten zu treffen [LAM01].

6.1 Endgeräte

Damit die Ergebnisse des *Benchmarks* Relevanz für das zugrundeliegende Projekt besitzen, müssen sich die Rahmenbedingungen an den Gegebenheiten der verfügbaren Hard- und Software orientieren. Zu diesem Zweck wurde zunächst bei den Verantwortlichen des *CINCH*-Labors in Erfahrung gebracht, welche technischen Spezifikationen die Endgeräte haben, welche den Studenten momentan zur Verfügung stehen. Anschließend wurde eine Auswahl an Testgeräten getroffen, die diesen Systemen ähnelt.

Im Labor verfügbare Geräte

Die in Tabelle 6.1 aufgeführten Desktop PCs stellen die gängigen Arbeitsplätze dar, zu denen die Studenten Zugriff haben. Da nicht vorausgesetzt werden kann, dass sich ein Student privat mit besserer Hardware ausrüstet, werden diese PCs als Ausgangslage angesehen. Dabei handelt es ich um Systeme, die gemeinhin als *Office-PC* bezeichnet werden, da sie zwar genug Prozessorleistung und Arbeitsspeicher für alle Aufgaben des Büroalltags besitzen, aber nicht über eine

	Desktop PC 1	Desktop PC 2
OS	Windows 10	Windows 10
CPU	Intel Core i5-6500 4-Cores @ 3,20 GHz	Intel Celeron G3900T 2-Core @ 2,60 GHz
GPU	-	-
RAM	8 GB	8 GB

Tabelle 6.1: Gängige Endgeräte für Studenten

dedizierte Grafikkarte verfügen. Sie sind also nicht für grafisch aufwendige Echtzeitanwendungen ausgelegt.

Verwendete Testgeräte

In Anlehnung an die in Tabelle 6.1 aufgeführten PCs wurden fünf verschieden leistungsstarke Endgeräte für die Performanzanalyse ausgewählt. Zwei dieser Geräte sind portabler Natur in Form eines Laptops (siehe Tabelle 6.2) und die restlichen drei Geräte fallen in die Kategorie der Desktop PCs (siehe Tabelle 6.3). Damit wird sowohl die Anwendung an einem festen Arbeitsplatz als auch der steigende Trend zu einem mobilen Endgerät[1] abgedeckt.

Der Leistungsgrad der Systeme ist dabei sowohl unterhalb (siehe Laptop 1 und 2) als auch oberhalb der Referenzgeräte (siehe Desktop PC 2 und 3) angesiedelt, um nach der Analyse abschätzen zu können, ob Spielraum bei der für das virtuelle Labor zu empfehlenden Hardware besteht.

Softwareseitig wird auf allen Testgeräten der selbe Browser (*Mozilla Firefox*) für das *Benchmarking* verwendet, um eine vergleichbare Ausgangslage zu gewährleisten, indem die Zahl der sich unterscheidenen Faktoren reduziert wird. Außerdem ist dieser Browser auf jedem gängigen Bestriebssystem (engl. „*operating system*“, kurz „OS“) verfügbar und es ist kein zusätzliches Programm zum Aufsetzen eines lokalen Web Servers nötig, da das Ausführen einer *WebGL*-Anwendung ohne zertifizierte Herkunft unterstützt wird. Der Browser wird jeweils in seinen

[1] `https://de.statista.com/statistik/daten/studie/160925/umfrage/ausstattungsgrad-mit-personal-computer-in-deutschen-haushalten/` (abgerufen am 30.03.2022)

	Laptop 1	Laptop 2
OS	Windows 10 Pro	Ubuntu 21.10
CPU	AMD A8-7100 4-Core @ 1,70 GHz	Intel Core i5-4200U 2-Core @ 1,60 GHz
GPU	-	Nvidia GeForce GT 740M 2 GB
RAM	8 GB	12 GB

Tabelle 6.2: Testgeräte (Laptops)

	Desktop PC 1	Desktop PC 2	Desktop PC 3
OS	Windows 10 Pro	Windows 10 Pro	Windows 10 Pro
CPU	Intel Core i3-3220 2-Core @ 3,30 GHz	AMD Ryzen 9 3900X 12-Core @ 3,79 GHz	AMD Ryzen 7 5700G 8-Core @ 4,40 GHz
GPU	Nvidia GeForce GTX 650 1 GB	AMD Radeon RX 5700 XT 8 GB	Nvidia GeForce RTX 3060 12 GB
RAM	4 GB	32 GB	16 GB

Tabelle 6.3: Testgeräte (Desktop PCs)

Werkeinstellungen verwendet, damit evtl. installierte Erweiterungen das Ergebnis nicht verfälschen. Desweiteren wird die Hardwarebeschleunigung deaktiviert, um den Spezifikationen der Referenzgeräte (siehe Tabelle 6.1) zu entsprechen, welche selbst keine dedizierten Grafikkarten besitzen.

6.2 Aufbau des Benchmarks

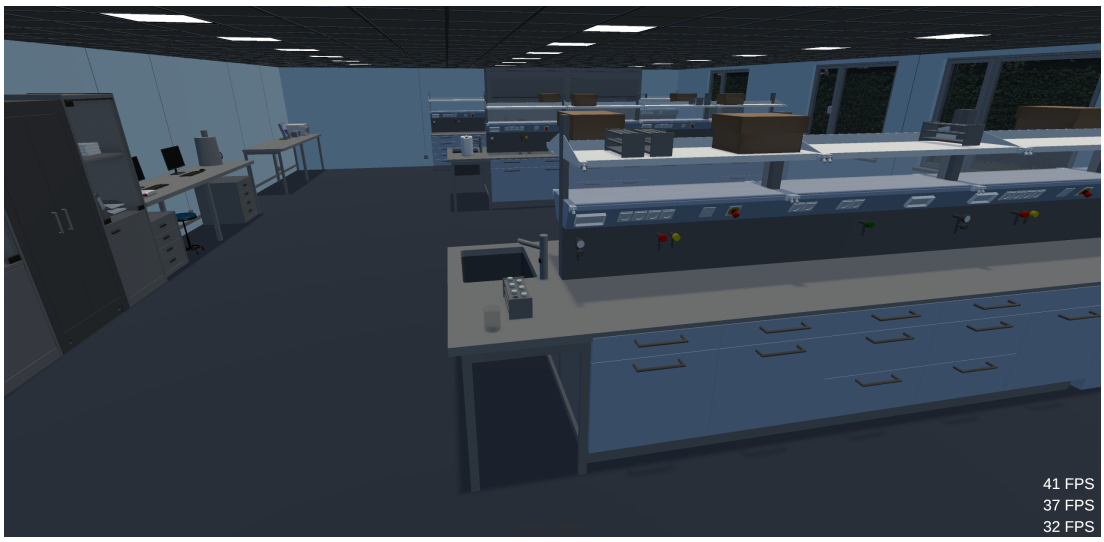
Um einen möglichst passenden Bezug zur späteren *WebGL*-Anwendung herstellen zu können, findet die Analyse im Hauptteil des virtuellen Labors statt, dem *Main Lab*. In diesem Bereich wurde eine feste Kameraperspektive gewählt, welche auf

der einen Seite die Ausmaße des Raums einfängt und auf der anderen Seite einen Arbeitsplatz fokussiert. Durch versteckte Bedienelemente lässt sich die Szene in die folgenden drei Zustände versetzen:

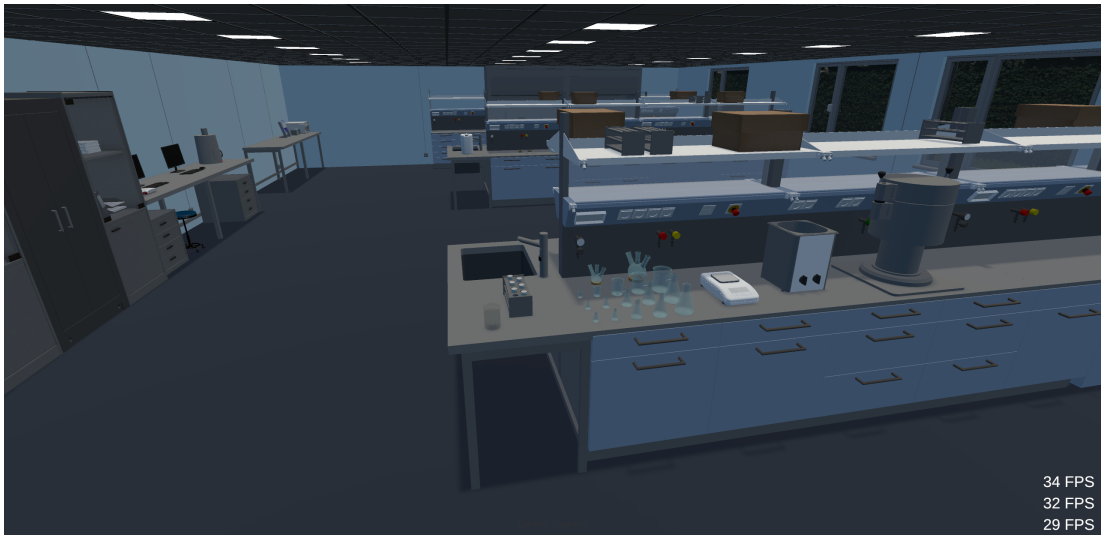
- **Empty**
  Dies stellt den Ausgangszustand der Szene dar. Er beinhaltet einen leeren Arbeitsbereich und die festen Umgebungsobjekte des Labors, welche zu allen Zeitpunkten des *Benchmarks* dargestellt werden (siehe Abbildung 6.1a). In diesem Zustand wird die grundlegende Performanz des Testgeräts bei der Darstellung des Labors getestet, um einen Referenzwert zu ermitteln, der mit den folgenden Zuständen verglichen werden kann.

- **High**
  Auf der Tischplatte des Arbeitsbereiches werden eine Vielzahl an Laborutensilien platziert. Dazu zählen vier Reihen an Glasgefäßen mit verschiedenen Volumina, eine Präzisionswaage, ein Ultraschallbad und ein Beta-Gamma-Spectrometer (siehe Abbildung 6.1b). Jedes dieser Objekte wurde zu Beginn dieser Arbeit als hochaufgelöstes Oberflächenmodell zur Verfügung gestellt und diente als Referenz bzw. Ausgangspunkt der Modellierung seines optimierten Gegenstücks. In diesem Zustand wird ermittelt, wie sehr die Performanz durch nicht adäquat optimierte 3D-Modelle beeinträchtigt wird.

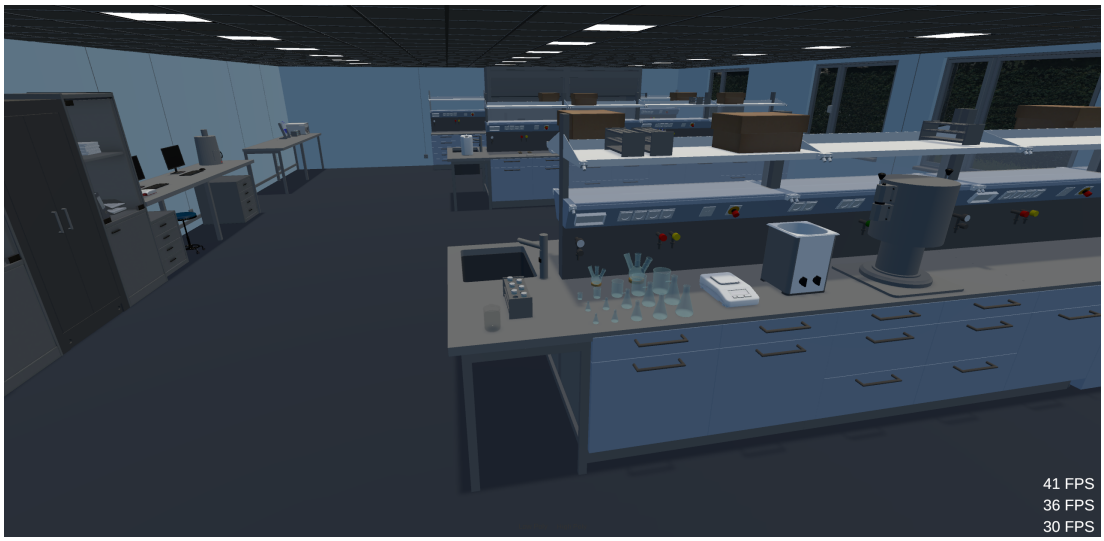
- **Low**
  Die zuvor platzierten hochaufgelösten Oberflächenmodelle werden durch ihre optimierten Gegenstücke ersetzt. Durch den direkten Vergleich wird zunächst sichtbar, wie sehr sich die 3D-Modelle optisch unterscheiden (siehe Abbildung 6.1c). Die in diesem Zustand ermittelten Werte dienen als Vergleichswert und zeigen, wie sehr sich die Performanz zum vorherigen Zustand (*High*) verbessert hat. Je näher die Messwerte am Ausganszustand (*Empty*) sind, desto besser sind die 3D-Modelle für den Einsatz in der *WebGL*-Anwendung optimiert.



(a) *Empty*



(b) *High*



(c) *Low*

Abbildung 6.1: Zustände der *Benchmark*-Szene

6.3 Ergebnisse

Zur Ausgabe der Messwerte wurde der *Benchmark*-Szene ein UI-Element hinzugefügt, welches drei verschiedene Zahlenwerte anzeigt (siehe Abbildung 6.1). Die oberste Zahl gibt dabei den höchsten gemessenen FPS-Wert an, die mittlere Zahl gibt den durchschnittlichen FPS-Wert an und die unterste Zahl gibt den niedrigsten FPS-Wert an. Zum Auslesen der Prozessorauslastung und der Belegung des Arbeitsspeichers wurde die jeweilige Funktionalität des Betriebssystems verwendet (*Task Manager*). Sie beinhalten die Grundleistung, die aufgebracht werden muss, um den Browser zu betreiben, da ohne ihn die *WebGL*-Anwendung nicht ausführbar wäre. Alle Werte wurden über einen Zeitraum von einer Minute erhoben und gegebenenfalls gemittelt.

6.3.1 Bilder pro Sekunde (FPS)

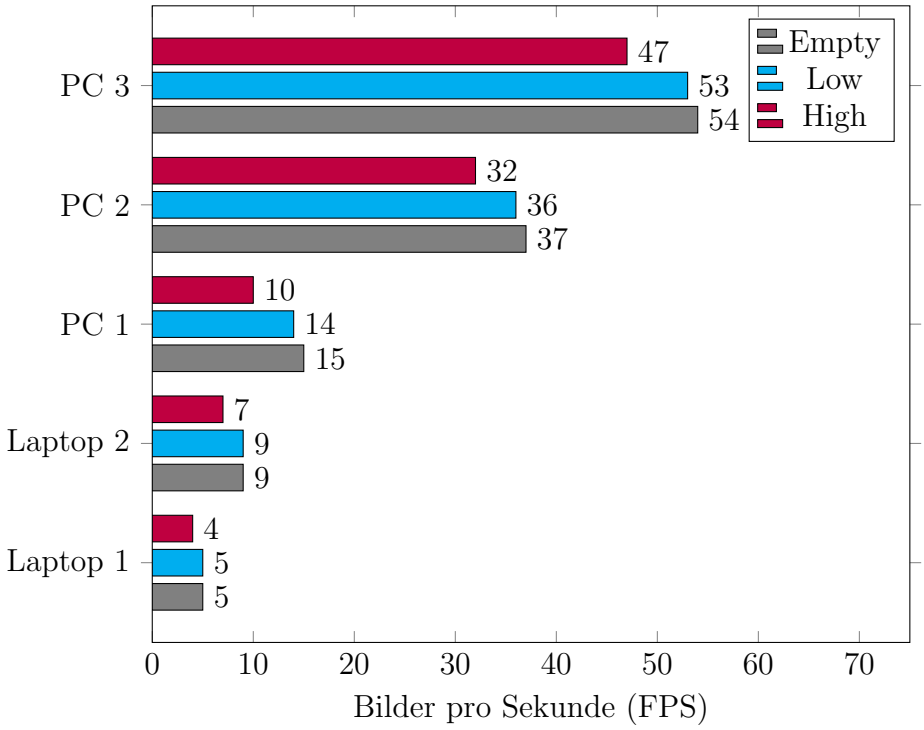


Abbildung 6.2: *Benchmark*-Ergebnis (FPS)

Diese Messgröße beschreibt die Anzahl der durchschnittlich berechneten Bilder pro Sekunde. Von ihm ist abhängig, wie flüssig die Darstellung des HoTs wahr-

genommen wird. Sollte er besonders niedrige Werte annehmen (< 30 FPS), sinkt nicht nur die Motivation des Nutzers. Es kann auch zu Beeinträchtigungen der zu erledigenden Aufgaben führen [WB94]. Aus diesem Grund ist dieser Wert eine gute Richtlinie für die Performanz einer Anwendung. Prinzipiell wird ein Wert von 30 FPS oder höher angestrebt. Wie jedoch in Abbildung 6.2 zu erkennen ist, bieten nur zwei Testgeräte ein zufriedenstellendes Ergebnis. Die Grundperformanz des virtuellen Labors sollte also weiterhin optimiert werden, was jedoch über den Rahmen dieser Arbeit hinausgeht. Im Bezug auf die 3D-Modelle ist allerdings deutlich zu erkennen, dass der Einsatz der hochaufgelösten Modellvarianten eine signifikante Reduzierung der Bilder pro Sekunde verursacht (13 % bis 33 % im Vergleich zum Ausgangszustand (*Empty*) und 11,1 % bis 28,6 % im Vergleich zu den optimierten Modellen (*Low*)). Im Gegensatz dazu wirken sich die optimierten 3D-Modelle nur in geringem Maße auf die Performanz aus (0 % bis 6,7 % im Vergleich zum Ausgangszustand (*Empty*)).

6.3.2 Prozessorauslastung (CPU)

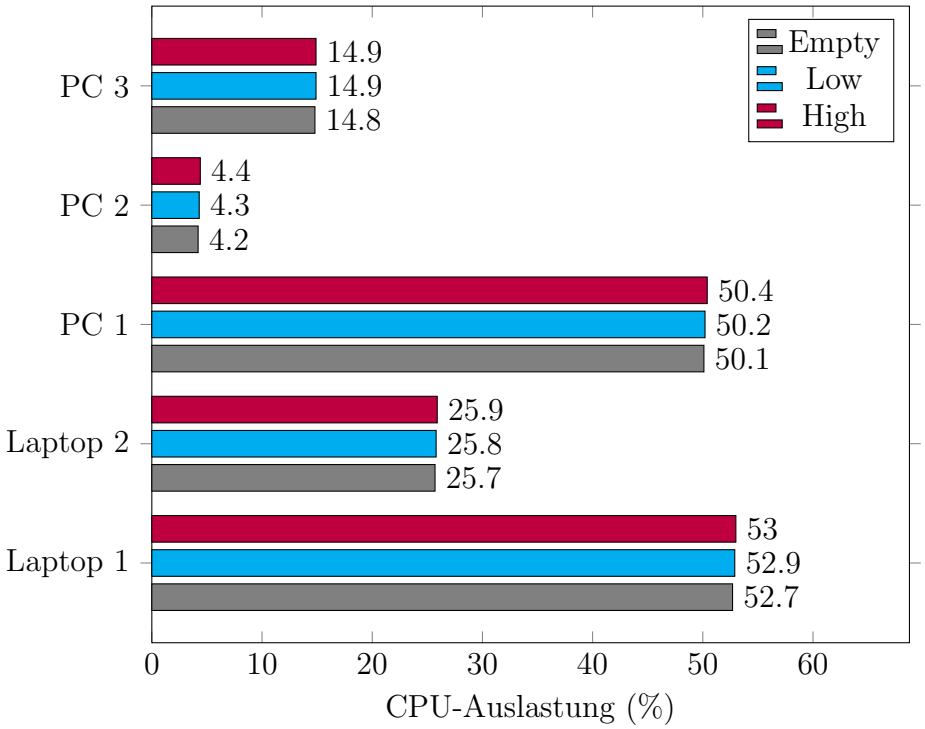


Abbildung 6.3: *Benchmark*-Ergebnis (CPU)

Da die GPU-gestützte Hardwarebeschleunigung während des *Benchmarks* deaktiviert wurde, trägt der Prozessor die gesamte Last der Bildberechnung. Je mehr die CPU damit beschäftigt ist, die *WebGL*-Anwendung auszuführen, desto höher ist der in Abbildung 6.3.2 dargestellte Prozentwert. Bei einer zu hohen Auslastung kann es zu Einbrüchen in der Performanz bis hin zu Abstürzen des gesamten Systems kommen. Es wird daher angestrebt, den Prozessor so effizient wie nötig zu nutzen und so wenig wie möglich zu belasten. Die im *Benchmark* ermittelten Werte zeigen jedoch, dass die verwendete Art von Oberflächenmodellen wenn überhaupt nur einen minimalen Einfluss auf die Prozessorauslastung hat. Diese Unterschiede sind so marginal (0 % bis 0,3 %), dass sie nicht direkt auf die 3D-Modelle zurückführbar sind.

6.3.3 Belegter Arbeitsspeicher (RAM)

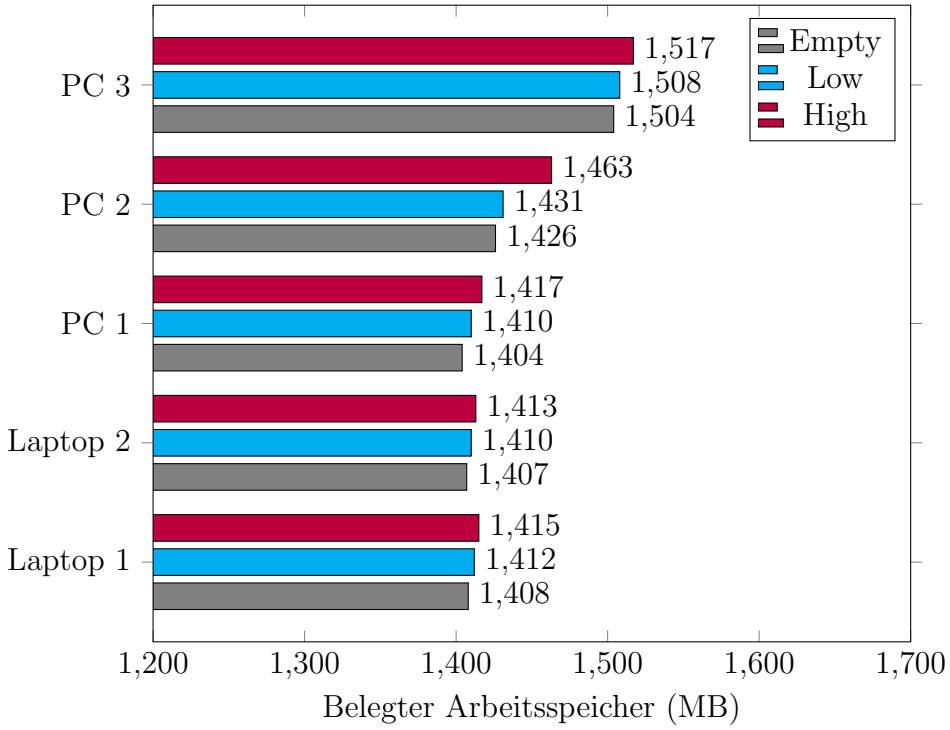


Abbildung 6.4: *Benchmark*-Ergebnis (RAM)

Arbeitsspeicher (RAM) dient der temporären Speicherung von Daten, auf die der Prozessor (CPU) zugreifen kann. Je mehr RAM ein Computer besitzt, desto

größer sind dessen Kapazitäten im Bezug auf die Verarbeitung von großen Dateien und die Ausführung von komplexen Programmen. Wenn mehr RAM zur Verfügung steht, erhöht sich auch die Performanz des Systems [Aca13]. Es ist daher erstrebenswert, die Belegung des RAMs möglichst gering zu halten, um Performanzeinbrüchen vorzubeugen. Wie in Abbildung 6.4 zu erkennen ist, schlägt sich der Optimierungsgrad der 3D-Modelle auch auf die Speicherkapazitäten nieder. So erhöht sich die Belegung im Vergleich von *High* zu *Empty* um 0,4 % bis 2,5 %. Dagegen wird der RAM im Vergleich von *Low* zu *Empty* nur um 0,2 % bis 0,4 % mehr beansprucht.

7 Fazit

7.1 Zusammenfassung

Die Arbeit hat einen angepassten 3D-Modellierungs-Workflow vorgestellt, der sich an den Anforderungen und Gegebenheiten einer *WebGL*-Anwendung orientiert. Im Zuge dessen wurden von der Referenz-Phase bis zum Einsatz in der Game Engine verschiedene Optimierungsansätze abgewogen und an einem komplexen Modellkatalog erprobt. Durch Übertragung des bewährten *Kanban*-Systems auf diese Aufgabenliste und der gezielten Wiederverwendung von Oberflächenmodellen konnten von insgesamt 166 Einträgen 72 Stück (43 %) von nur einem 3D-Modellierer abgeschlossen werden. Weitere 31 Einträge (19 %) wurden so weit umgesetzt, dass die bis dato bekannten Modellierungsziele erfüllt sind und sie sich zurzeit in der *Review*-Phase befinden. Durch die Performanzanalyse in Form eines *Benchmarks* wurde zudem gezeigt, dass die erstellten 3D-Modelle sehr gut an die Verwendung im virtuellen Labor angepasst sind, da sie sich nur geringfügig negativ auf die Zahl der Bilder pro Sekunde (FPS) auswirken (0 % bis 6,7 %).

7.2 Diskussion

Die Performanzanalyse hat gezeigt, dass die *WebGL*-Anwendung zum jetzigen Stand der Entwicklung noch keine zufriedenstellende *Frame Rate* für die angestrebten Endgeräte aufweisen kann. Da das vorgestellte *Benchmark* in Anlehnung an die vorhandenen Geräte jedoch ohne den Einsatz einer dedizierten Grafikkarte durchgeführt wurde, steckt in diesem Aspekt noch ein hohes Verbesserungspotential. Für den weiteren Verlauf der Entwicklung wäre es daher ratsam, die

Voraussetzung einer GPU in Betracht zu ziehen.
Im Bezug auf die 3D-Modellierung können ebenfalls noch weitere Optimierungen vorgenommen werden. So gibt es z. B. noch keine genaue Richtlinie dazu, wie hoch ein 3D-Modell aufgelöst sein muss in Abhängigkeit zur minimalen Sichtdistanz, auf der es verwendet werden soll. Dies könnte durch die Gegenüberstellung verschiedener Reduktionsgrade eines Modells im Bezug auf die Auflösung der Anwendung ermittelt werden. Außerdem könnten und bestimmten Voraussetzungen verschiedene Texturen (z. B. *Albedo* und *Ambient Occlusion* Textur) miteinander kombiniert werden, um noch mehr Ressourcen einzusparen.

7.3 Ausblick

Da noch nicht alle Einträge im Modellkatalog bearbeitet wurden, das virtuelle Labor sich aber noch mitten in der Entwicklung befindet, werden die bis jetzt angefertigten 3D-Modelle dem nachrückenden Teammitglied zur Verfügung stehen. Damit wird auf die Probe gestellt, ob der Workflow und die damit aufgestellten Konventionen ihren Zweck erfüllen. Im Sinne von *Dere* et al. [DSI10] wäre es darüber hinaus möglich, die erstellten 3D-Modelle in Absprache mit den Verantwortlichen in einem öffentlichen Online-Repositorium zur Verfügung zu stellen, das über das *A-CINCH*-Projekt hinaus geht. So könnten noch mehr Suchende im akademischen Umfeld davon profitieren.
Aufgrund der Tatsache, dass viele der verwendeten Glasgefäße einem systematischen Aufbau folgen, wäre es weiterhin denkbar, ein automatisiertes Skript zu implementieren, das durch Angabe bestimmter Parameter (wie z. B. dem Verlauf eines Glaskörpers in Form einer Kurve und die Angabe der Wandstärke) ein Polygonnetz generiert. So könnten diverse Versuchsaufbauten realisiert werden, für die es momentan noch einer engen Absprache zwischen den Entwicklern und den Laboranten bedarf.

Literaturverzeichnis

[Aca13] Academy, Cisco: *IT Essentials: PC Hardware and Software Companion Guide*. Cisco Press, 2013.

[BAS13] Bhawar, Pooja, Nitin Ayer Sameer Sahasrabudhe: *Methodology to Create Optimized 3D Models Using Blender for Android Devices. IEEE International Conference on Technology for Education*, 5, 139–142, 2013.

[BDRA11] Berthelot, Rozenn Bouville, Thierry Duval, Jérôme Royan Bruno Arnaldi: *Improving Reusability of Assets for Virtual Worlds while Preserving 3D Formats Features. Journal of Virtual Worlds Research*, 4, 2011.

[Bri18] Brito, Allan: *Blender Quick Start Guide: 3D Modeling, Animation, and Render with Eevee in Blender 2.8*. Packt Publishing Ltd., 2018.

[Büh21] Bühler, Peter: *3D mit Blender: Modeling – Animation – Rendering*. Springer Fachmedien Wiesbaden, 2021.

[CN15] Caudron, Romain Pierre-Armand Nicq: *Blender 3D By Example*. Packt Publishing Ltd, Birmingham, 2015.

[DDKN11] Deterding, Sebastian, Dan Dixon, Rilla Khaled Lennart Nacke: *From Game Design Elements to Gamefulness: Defining Gamification*. International Academic MindTrek Conference: Envisioning Future Media Environments, 9–15, 2011.

[DSI10] Dere, Shruti, Sameer Sahasrabudhe Sridhar Iyer: *Creating Open Source Repository of 3D Models of Laboratory Equipments*

using Blender. International Conference on Technology for Education, 149–156, 2010.

[EG11] Ericsson, Robin Anna Granlöf: *The effects of Kanban in software development teams - a study of the implementation at Sandvik.* DALARNA University College, 2011.

[Epp11] Epping, Thomas: *Kanban für die Softwareentwicklung.* Informatik im Fokus. Springer-Verlag Berlin Heidelberg, 2011.

[Fla10] Flavell, Lance: *Beginning Blender: Open Source 3D Modeling, Animation and Game Design.* Apress, New York, 2010.

[GKB07] Gratch, Jonathan, Janet Kelly Curtis Bradley: *Science Simulations: What Do They Contribute to Student Learning? Proceedings of Society for Information Technology Teacher Education International Conference*, 3422–3425. Association for the Advancement of Computing in Education (AACE), 2007.

[GPF09] Golovinskiy, A., J. Podolak T. Funkhouser: *Symmetry-Aware Mesh Processing. Mathematics of Surfaces XIII*, 170–188. Springer Berlin Heidelberg, 2009.

[GS15] Gerlach, Stefanie Inga Squarr: *E-Learning: Methodenhandbuch für Softwareschulungen.* Springer-Verlag Berlin Heidelberg, 2015.

[HU15] Hauser, John R. Glen L. Urban: *From Little's Law to Marketing Science: Essays in Honor of John D.C. Little.* The MIT Press, 2015.

[Kum20] Kumar, Abhishek: *Beginning PBR Texturing: Learn Physically Based Rendering with Allegorithmic's Substance Painter.* Springer eBook Collection. Apress, Berkeley, CA, 2020.

[LAM01] Lext, Jonas, Ulf Assarsson Tomas Moller: *A Benchmark for Animated Ray Tracing.* IEEE Computer Graphics and Applications, 21(2):22–31, 2001.

[LJ00] LaViola Jr., Joseph J.: *A discussion of cybersickness in virtual environments.* ACM SIGCHI Bulletin, 32:47–56, 2000.

[LM21] Liesen, Jörg Volker Mehrmann: *Algebraische Strukturen*. Springer Berlin Heidelberg, 2021.

[Mar19] Mari, Jean-Luc: *Geometric and Topological Mesh Feature Extraction for 3D Shape Analysis*. ISTE Ltd., 2019.

[McD18] McDermott, Wes: *The PBR Guide*. Allegorithmic, 2018.

[MDS17] Munafo, Justin, Meg Diedrick Thomas A. Stoffregen: *The virtual reality head-mounted display Oculus Rift induces motion sickness and is sexist in its effects*. Experimental Brain Research, 235:889–901, 2017.

[NFHS19] Nischwitz, Alfred, Max Fischer, Peter Haberäcker Gudrun Socher: *Computergrafik*. Springer Vieweg, 2019.

[OGE+20] Ortiz, Jessica, Bryan Guevara, Edison Espinosa, Jaime Santana, Lenin Tamayo Víctor Andaluz: *3D Virtual Content for Education Applications. Iberian Conference on Information Systems and Technologies (CISTI)*, 15, 1–5, 2020.

[Pai19] Pai, Hong-Yi: *Texture designs and workflows for physically based rendering using procedural texture generation. IEEE Eurasia Conference on IOT, Communication and Engineering (ECICE)*, 195–198, 2019.

[Par14] Parisi, Tony: *Programming 3D applications with HTML5 and WebGL*. O'Reilly Media, 2014.

[PD93] Prieto-Diaz, R.: *Status Report: Software Reusability*. IEEE Software, 10(3):61–66, 1993.

[Pow16] Powell, Gretchen: *Computational Fluid Dynamics (CFD): Characteristics, Applications and Analysis*. Mechanical Engineering Theory and Applications. Nova Science Publishers, Inc., 2016.

[PP10] Peng, Guobin Runhong Peng: *Strategic Thinking of Modeling Method and Modeling. International Conference on System Science, Engineering Design and Manufacturing Informatization*, 2, 71–73, 2010.

[Sim13] Simonds, Ben: *Blender Master Class: A Hands-On Guide to Modeling, Sculpting, Materials, and Rendering.* No Starch Press, San Francisco, CA, 2013.

[Sis01] Sisson, G.R.: *Hands-On Training: A Simple and Effective Method for on the Job Training.* Publication in the Berrett-Koehler organizational performance series. Berrett-Koehler Publishers, 2001.

[SS20] Saake, Gunter Kai-Uwe Sattler: *Algorithmen und Datenstrukturen.* dpunkt, 6th edition , 2020.

[Sto13] Stoecker, Daniela: *eLearning - Konzept und Drehbuch, Handbuch für Medienautoren und Projektleiter.* Springer-Verlag Berlin Heidelberg, 2013.

[Tho15] Thorn, Alan: *Practical Game Development with Unity and Blender.* Cengage Learning, 2015.

[Vil22] Villanueva, Nova: *Beginning 3D Game Assets Development Pipeline: Learn to Integrate from Maya to Unity.* Apress, 2022.

[VWB+09] Vajna, Sandor, Christian Weber, Helmut Bley, Klaus Zeman Peter Hehenberger: *CAx für Ingenieure: Eine praxisbezogene Einführung.* Springer Berlin Heidelberg, 2009.

[WB94] Ware, Colin Ravin Balakrishnan: *Reaching for Objects in VR Displays: Lag and Frame Rate.* ACM Transactions on Computer-Human Interaction, 1(4):331–356, 1994.