Implementierung und Vergleich von Animationen und Interaktionen bei komplexen Modellen in 3D-Echtzeitanwendungen

Studiengang Medieninformatik

Bachelorarbeit

vorgelegt von

Christoph Damm
geb. in Gießen

durchgeführt bei
GM-W, Agentur für technische Kommunikation

Referent der Arbeit:	Prof. Dr. Cornelius Malerczyk
Korreferent der Arbeit:	M. Sc. Hans Christian Arlt
Betreuer bei GM-W:	Thomas Luh

Friedberg, 2019

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich im Laufe dieser Arbeit unterstützt haben.

Zuerst möchte ich mich bei Prof. Dr. Cornelius Malerczyk bedanken, der diese Arbeit betreut hat und stets für meine Rückfragen zur Verfügung stand.

Auch bei der GM-W möchte ich mich dafür bedanken, dass ich diese Arbeit im Rahmen eines Projektes in der Agentur schreiben durfte. Besonders allen Kollegen, die mir mit Rat und Tat zur Seite standen, möchte ich an dieser Stelle ganz herzlich danken.

Besonderer Dank kommt außerdem meinem Bruder, meiner Mutter sowie meinem Kommilitonen und guten Freund Dennis Heymann zu, die sich viel Zeit dafür genommen haben, mich durch fleißiges Korrekturlesen zu unterstützen.

Zuletzt möchte ich meiner Familie und meiner Freundin danken, die mir immer den Rücken gestärkt und mich motiviert haben.

Vielen, vielen Dank euch allen!

Selbstständigkeitserklärung

Ich erkläre, dass ich die eingereichte Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Buseck, Oktober 2019

Christoph Damm

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit beschäftigt sich mit der Erstellung von Echtzeitanwendungen mit komplexen Modellen. Es wird dabei untersucht, welche Animationen verwendet und wie sie implementiert werden können. Außerdem wird betrachtet, welche Möglichkeiten dem Nutzer geboten werden können, um mit der virtuellen Umgebung zu interagieren.

1.1 Motivation

Eine Echtzeitanwendung ist nach DIN ISO/IEC 2382: *„[...] ein Rechensystem, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind.“*[Deu05]

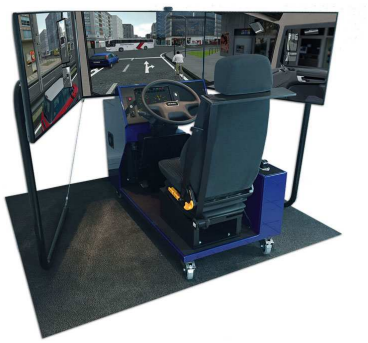

Abbildung 1.1: Eine 3D-Echtzeitanwendung, die als Fahrsimulator für LKWs und Busse zu Ausbildungszwecken eingesetzt wird. *(Quelle: https://www.live-akademie.de/index.php/leistungen/lkw-bus-simulator; Stand: 18.07.2019)*

Für 3D-Anwendungen bedeutet das, dass die einzelnen Frames nicht wie bei Videos oder Stillrenderings im Voraus berechnet werden, sondern erst zur Laufzeit. Dadurch hat die Anwendung die Möglichkeit, auf Nutzereingaben zu reagieren. So kann der Nutzer also mit einer virtuellen Umgebung interagieren. Als Videospiele kennen wir solche Anwendun-

gen schon lange. Die Möglichkeit eine Welt zu simulieren, in der der Nutzer verschiedene Vorgänge steuern und die Ergebnisse beobachten kann, macht solche Anwendungen jedoch auch in vielen anderen Bereichen einsetzbar. So zeigt Abbildung 1.1 beispielsweise einen Fahrsimulator der *LiVe-Akademie*[1], der zur Ausbildung von Bus- und LKW-Fahrern eingesetzt wird. Bei derartigen Anwendungen spricht man auch von *Serious Games*. Laut [MK10] hat sich zwar noch keine eindeutige Definition für Serious Games durchgesetzt, „es lässt sich jedoch festhalten, dass es sich bei Serious Games um Spiele oder spielähnliche Anwendungen handelt, die mit Technologien und Design aus dem Unterhaltungssoftwarebereich entwickelt werden und nicht primär bzw. ausschließlich der Unterhaltung dienen."

Besonders im industriellen Bereich wächst das Interesse an solchen Anwendungen, da sie vielfältig eingesetzt werden können: Sie können als virtueller Showroom, als Produktkonfigurator, als Schulungswerkzeug oder zu anderen Zwecken eingesetzt werden. Der entscheidende Vorteil gegenüber Videos oder Bildern als Produktvisualisierungen liegt dabei in der Interaktivität. So kann beispielsweise ein potenzieller Kunde ein Produkt in einer 360°-Ansicht betrachten und dabei Funktionen durch Animationen und Interaktionen individuell steuern. Dabei kann der Nutzer sogar Aktionen durchführen, die in der realen Welt nicht möglich wären. Wird beispielsweise eine solche Anwendung eingesetzt, um Technikern die Reparatur einer Maschine beizubringen, können sie hier verschiedene Schritte durchführen, ohne das Risiko einzugehen, eine reale Maschine zu beschädigen. Dennoch können die Folgen der Aktion vermittelt werden. Durch Visualisierung, einen Infotext oder sogar durch ein Punktesystem, kann dem Nutzer gezeigt werden, welche Reaktionen gewisse Aktionen nach sich ziehen. Durch diese Freiheiten kann der Effekt des Lernens und Verstehens mit dem Nutzen von Echtzeitanwendung deutlich verstärkt werden.

Kurt Squire, Professor an der *University of California, Irvine*, testete ein am Massachusetts Institute of Technology entwickeltes Serious Game (*„Supercharged!"*), in dem es um elektromagnetische Felder ging. In dem Spiel konnten die Teilnehmer elektrische Ladungen platzieren und umpolen, um sich so mithilfe der entstehenden elektromagnetischen Felder durch ein Labyrinth zu bewegen. Abbildung 1.2 zeigt einen Screenshot dieses Serious Games. Er ließ die Teilnehmer einen Test über elektromagnetische Felder absolvieren, bevor und nachdem sie die Anwendung nutzten. Dabei fand er heraus, dass ihre Leistungen sich um 28% verbesserten. Eine Kontrollgruppe, der die selben Lerninhalte durch konventionelle Lehrmethoden vermittelt wurde, verbesserte ihre Leistungen dagegen nur um 20%.

[1] https://www.live-akademie.de

Abbildung 1.2: Screenshot des Serious Games *„Supercharged!“*. Quelle: [SBMH04]

Das Buch *„Gamification in Education and Business“* beschäftigt sich mit dem Einsatz von Serious Games in der Ausbildung und im geschäftlichen Bereich. Darin wird festgestellt, dass interaktive Anwendungen eines der mächtigsten Werkzeuge sind, um das Verhalten und die Charakteristik von physikalischen Gesetzen, Prozessen oder Systemen zu lehren, lernen und zu verstehen.[RW15]

Ein weiterer Vorteil ist die große Menge an unterstützten Plattformen. Durch die starke Hardware, die mittlerweile selbst auf Mobilgeräten zur Verfügung steht und Technologien, wie WebGL für Echtzeitinhalte im Web, müssen 3D-Echtzeitanwendungen nicht als Programm auf einen Rechner installiert werden. Sie können auch als mobile App oder Webanwendung erstellt werden. *Unity* [2], neben der *Unreal Engine* [3] die größte Echtzeit 3D Plattform, erreicht nach eigenen Angaben über 3 Milliarden Geräte weltweit.[4] Diese Flexibilität und Vielfältigkeit machen Echtzeitanwendungen auch für die Industrie zunehmend interessant.

Der Kern dieser Anwendungen sind dabei die Animationen und Interaktionen. Animationen visualisieren verschiedene Abläufe oder Zusammenhänge und Interaktionen geben dem Nutzer die Möglichkeit, diese selbst zu entdecken. Aktuelle Game Engines wie *Unity* bieten verschiedenste Möglichkeiten, Animationen und Interaktionen zu erstellen und zu implementieren. Es stellt sich jedoch die Frage, welche dieser Möglichkeiten sich auch für komplexere Modelle eignen.

1.2 Problemstellung

Bei der Entwicklung von Echtzeitanwendungen sind hinsichtlich der Performance vor allem zwei Dinge zu beachten. Zum Einen werden für ein flüssiges Bild mindestens 30 Bilder pro Sekunde (Frames Per Second (FPS)) benötigt. Das bedeutet, dass die Bilder wesentlich schneller gerendert werden müssen, als es bei einer vorgerenderten Animation der Fall ist. Zum Anderen steht für Pre-Renderings in der Regel mächtige Hardware bereit. Oft werden Renderjobs auf mehrere Server verteilt, um schnellere Ergebnisse erzielen zu können. Eine Echtzeitanwendung dagegen läuft im besten Fall auf einem normalen Rechner ab, mögli-

[2] https://unity.com/de

[3] https://www.unrealengine.com/en-US/

[4] https://unity3d.com/de/public-relations?_{}ga=2.189216995.999677529.1563808046-1997544232.1563808046 Stand 22.07.2019

cherweise aber auch in einem Webbrowser oder sogar auf einem mobilen Gerät. Es ist also davon auszugehen, dass sowohl weniger Rechenleistung, als auch weniger Zeit zum Rendern zur Verfügung steht.

In der Regel werden die Modelle für Echtzeitanwendungen daher speziell für diese angefertigt. Beim Modellieren dieser Objekte wird stets darauf geachtet, so wenig Geometrie wie möglich zu verwenden, um die benötigte Rechenleistung so gering wie möglich zu halten. Denn große und komplexe Modelle führen in Echtzeitanwendungen schnell zu Einbrüchen der Performance. [Boe13]

Im industriellen Bereich jedoch werden Modelle oft nicht von Hand entworfen. Stattdessen werden sie in der Regel aus Konstruktionsdaten der Hersteller generiert. Es handelt sich dabei um technische Skizzen (2- oder 3D), die für die Fertigung der Produkte verwendet werden. Konstruiert werden diese Modelle in Computer Aided Design (CAD)-Programmen. Man spricht demzufolge auch von CAD-Daten. Werden diese Daten in Polygonmodelle überführt, entstehen sehr komplexe Modelle. Abbildung 1.3 zeigt beispielsweise einen Griff an der Maschine, die für den Prototyp verwendet wird. Man kann darauf sehr gut erkennen, wie komplex selbst simple Teile dargestellt werden. Der Screenshot wurde vor jeglichen Optimierungen aufgenommen. Zu diesem Zeitpunkt bestand der Griff aus über 39.000 Polygonen. Dementsprechend komplex ist auch der Rest des Modells. Die komplette Maschine besteht damit aus über 140 Millionen Polygonen. Im Vergleich dazu ist rechts in der Abbildung der Griff zu sehen, nachdem er für die Verwendung in Echtzeit optimiert wurde. Die Polygonanzahl wurde dabei von den ursprünglichen 39.000 auf nur 246 reduziert.

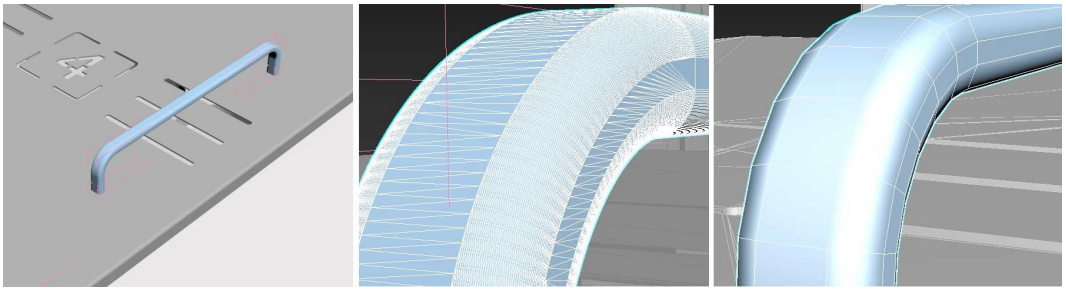

Abbildung 1.3: Screenshot eines Griffes an der Maschine aus 3DS Max. Links ist der komplette Griff zu sehen, in der Mitte ein Close-Up, der die ursprüngliche Tesselierung des Griffes zeigt. Rechts zum Vergleich das echtzeitfähige, vereinfachte Mesh.

Modelle, die aus CAD-Daten generiert wurden, werden im Folgenden der Einfachheit halber als *„CAD-Modelle“* bezeichnet. Da diese CAD-Modelle sehr komplex sind, kann allein ihre Visualisierung in Echtzeit ohne weitere Vorbereitung zu Problemen führen, da die Rechenleistung schnell nicht mehr für eine flüssige Darstellung ausreicht. Die Implementierung von Animationen und Interaktionen belastet die Performance zusätzlich. Wie bereits im Abschnitt Motivation geschildert, sind Animationen und Interaktionen allerdings der größte Vorteil von Echtzeitanwendungen. Aus diesem Grund muss untersucht werden, welche Animationen und

Interaktionen auch bei komplexen Modellen, wie sie bei der Konvertierung von CAD-Daten entstehen, anwendbar sind.

Es muss also zunächst betrachtet werden, welche Animationen im industriellen Bereich zum Einsatz kommen. Danach muss untersucht werden, welche dieser Animationen bei komplexen Modellen in Echtzeit implementiert werden können. Zuletzt muss überlegt werden, durch welche Interaktionsmöglichkeiten der Nutzer die Möglichkeit bekommt, die Funktionen der Anwendung zu bedienen. Dabei geht es um die Gestaltung der Benutzeroberfläche, aber auch um die generelle Bedienbarkeit der Anwendung.

1.3 Zielsetzung

Zu diesem Zweck wird ein Prototyp einer solchen Anwendung entwickelt. Dieser Prototyp wird im Rahmen eines Projektes der *GM-W*[5] als Beispielanwendung dienen, um einem Kunden die technischen Möglichkeiten von Echtzeitanwendungen zu demonstrieren. Als Testmodell wird ein aus CAD-Daten erzeugtes Modell einer Verpackungsmaschine verwendet. Der Entwicklungsprozess wird dabei dazu genutzt, auftretende Schwierigkeiten bei der Arbeit mit komplexen Modellen zu analysieren.

Der erste Schritt ist dabei Anforderungen zu erstellen, welche Animations- und Interaktionsmöglichkeiten die Anwendung beinhalten soll. Es soll sich dabei vor Allem um Animationen und Interaktionen handeln, die typischerweise im industriellen Umfeld zum Einsatz kommen. Zusätzlich werden aber auch Funktionen aufgenommen, bei denen es aus anderen Gründen interessant ist, die Implementierung bei komplexen Modellen genauer zu betrachten.

Die erstellte Anwendung ist dabei nicht an eine konkrete Plattform gebunden. Zur Erstellung der Anwendung wird *Unity* verwendet werden. Der Vorteil einer solchen Game Engine liegt darin, dass ein erstelltes Projekt auf verschiedene Plattformen exportiert werden kann. In dieser Arbeit soll es jedoch um die generelle Implementierung von Animationen und Interaktionen in Echtzeitanwendungen gehen, weshalb die Zielplattform eine untergeordnete Rolle spielt.

Bei der Erstellung der Anwendung wird untersucht, welche Probleme bei den jeweiligen Interaktionen auftreten und wie sie gelöst werden können. Das Ziel dieser Arbeit ist also nicht nur eine lauffähige Echtzeitanwendung. Stattdessen wird auch eine Übersicht erstellt, welche Schwierigkeiten und Probleme bei der Implementierung der verschiedenen Animationen und Interaktionen entstehen und wie diese zu lösen sind.

Zur Evaluation der Ergebnisse werden für solche Anwendungen typische Kennzahlen, wie Bilder pro Sekunde und Draw Calls ausgewertet. Auch der Speicherplatz, den Funktionen, Modelle oder die Anwendung belegen, soll betrachtet werden. Ein weiterer wichtiger Faktor, ist die Zeit und der Arbeitsaufwand, der zum Implementieren benötigt wird. Die so gewon-

[5] https://gm-w.de/

nenen Erkenntnisse sollen dabei nicht nur auf die Entwicklung dieser speziellen Anwendung bezogen sein. Stattdessen sollen die Ergebnisse dieser Arbeit so abstrahiert werden, dass sie als genereller Leitfaden zur Arbeit mit komplexen Modellen in Echtzeit dienen.

Kapitel 2

Grundlagen

Im folgenden Kapitel sind wichtige Informationen zusammengefasst, die grundlegend für diese Arbeit sind. Im ersten Abschnitt geht es dabei um Informationen zu CAD-Daten, während sich der zweite Teil mit Echtzeitanwendungen beschäftigt.

2.1 Was sind CAD-Daten?

CAD-Daten sind Daten, die in CAD-Systemen erstellt wurden. Diese Programme werden von Konstrukteuren, Ingenieuren oder Architekten genutzt, um Objekte zu konstruieren. Im Gegensatz zu Digital Content Creation (DCC)-Programmen wie *Autodesk Maya*[1] oder *3DS Max*[2] sind diese Programme darauf ausgelegt, korrekte Baupläne zu erstellen und keine Modelle zur Visualisierung. Dementsprechend sind in diesen Programmen andere Schwerpunkte gesetzt. Bei der Erstellung von Modellen für Renderings geht es um die Optik des Modells. In CAD-Systemen dagegen geht es um die korrekte Konstruktion. Das heißt, das selbst an Stellen, die bei einem Rendering oder einer Visualisierung gar nicht sichtbar sind, jegliche Geometrie vollständig vorhanden ist. Um Renderzeiten einzusparen, würde man so etwas beim Modellieren eines Objektes vermeiden.

Außerdem werden Modelle in CAD-Programmen anders repräsentiert als in DCC-Software. Im Bereich der DCC-Software wird in der Regel mit Polygonmodellen gearbeitet, während CAD-Programme mit Non-Uniform Rational B-Splines (NURBS)-Flächen arbeiten. Dabei handelt es sich um Flächen, die durch mathematische Formeln beschrieben werden. Das bedeutet, dass komplexere Daten einfacher dargestellt werden können. Der Unterschied wird in Abbildung 2.1 deutlich sichtbar. Um dieselbe Geometrie darzustellen, sind nur einige wenige NURBS-Flächen nötig. Das Polygonmodell dagegen ist sehr komplex, da Polygone immer planar sind. Um also eine Rundung nachzubilden, müssen viele kleine Polygone mit kleinem Winkel zueinander verwendet werden. Desto genauer das Polygonmodell die Rundung abbilden soll, desto mehr Polygone sind dazu nötig.

[1] https://www.autodesk.de/products/maya/overview
[2] https://www.autodesk.de/products/3ds-max/overview

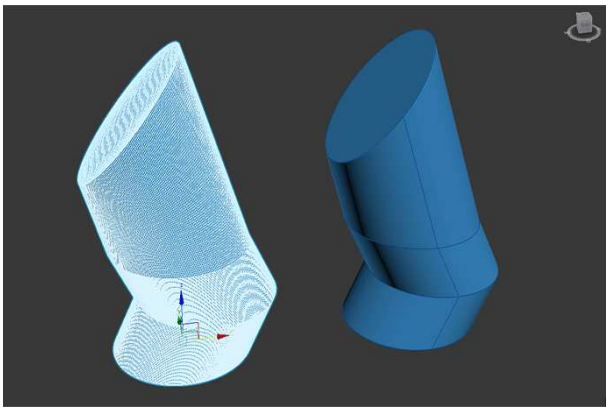

Abbildung 2.1: Vergleich der selben Geometrie als Polygonobjekt (l.) und aus NURBS-Flächen(r.). *Quelle: Ufo3D (https://ufo3d.com/wp-content/uploads/2018/03/polygon-modeling-vs-spline-View2.png)*

Bei der Umwandlung von CAD-Daten in Polygonmodelle spielt das eine große Rolle. Dabei werden die Flächen, die die CAD-Daten beschreiben, durch Polygonflächen aufgebaut. Diesen Vorgang bezeichnet man als *Tessellierung*. Bei der Tessellierung sind es vor Allem drei Parameter, die die Qualität und Größe des Modells maßgeblich beeinflussen. Auch wenn ihre Bezeichnungen sich in verschiedenen Softwares unterscheiden, sind ihre Funktionsweisen die Gleichen. Im Folgenden werden die Bezeichnungen *Chord Tolerance*, *Max Edge Length* und *Normal Tolerance* verwendet.

Die *Chord Tolerance* bezeichnet den maximalen Abstand, den die erzeugten Polygonen von den ursprünglichen NURBS-Kurven abweichen dürfen. Höhere Werte führen hier zu weniger Polygonen, aber auch weniger Details. In Abbildung 2.2 ist dies gut zu sehen.

Bei der *Max Edge Length* handelt es sich um eine maximale Kantenlänge, die die erzeugten Polygone nicht überschreiten sollen. Höhere Werte erzeugen hier wenigere aber größere Polygone, während kleinere Werte auch kleinere Polygone erzeugen. Wie in Abbildung 2.3 zu sehen ist, führt eine geringere Max Edge Length zwar zu mehr Polygonen, produziert dafür aber gleichmäßigere Unterteilungen.

Die *Normal Tolerance* gibt einen maximalen Winkel an, den zwei Polygone zueinander haben dürfen. Würde dieser Winkel überschritten werden, wird ein weiteres Polygon eingefügt. Durch diesen Wert lässt sich also vor allem einstellen, wie rund Rundungen dargestellt werden. Ein höherer Wert erzeugt dabei eckigere Rundungen. Aus Abbildung 2.4 ist ersichtlich, wie dieser Wert den Grad der Unterteilung bei Rundungen steuert. [Epi]

Die Abstimmung dieser drei Parameter bestimmt wesentlich die Qualität und Komplexität des generierten Modells. Es gibt dabei keine Musterlösung, die bei jedem Modell anwendbar ist. Es muss immer beachtet werden, wie die ursprünglichen CAD-Daten aussehen und wofür das Modell verwendet werden soll. Grundsätzlich gilt es immer, einen für den Verwendungszweck angemessenen Kompromiss zwischen Komplexität und Qualität anzustreben.

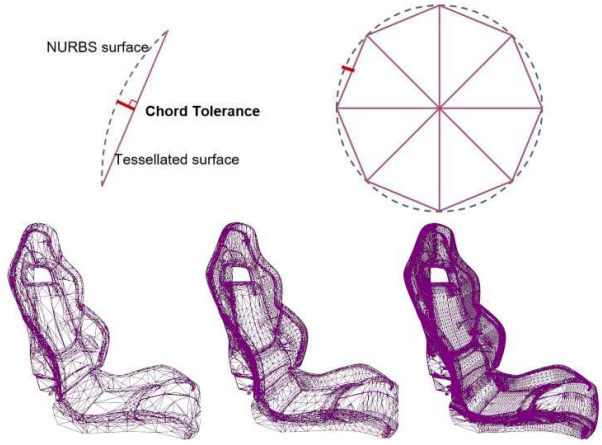

Abbildung 2.2: Oben im Bild: Eine Skizze zur Erklärung der Chord Tolerance. Darunter das Mesh eines Autositzes, der mit verschiedenen Werten tesseliert wurde. Von links: 10mm → 13.500 Polygone; 0,5mm → 37.500 Polygone; 0,1mm → 134.000 Polygone. *(Quelle: https://docs.unrealengine.com/en-US/Studio/Datasmith/SoftwareInteropGuides/CAD/index.html Stand: 08.08.2019)*

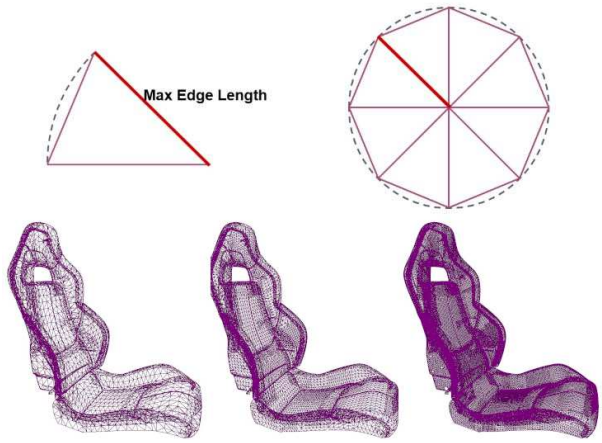

Abbildung 2.3: Oben im Bild: Eine Skizze zur Erklärung der Max Edge Length. Darunter das Mesh eines Autositzes, der mit verschiedenen Werten tesseliert wurde. Von links: 40mm → 21.000 Polygone; 20mm → 43.700 Polygone; 10mm → 128.000 Polygone. *(Quelle: https://docs.unrealengine.com/en-US/Studio/Datasmith/SoftwareInteropGuides/CAD/index.html Stand: 08.08.2019)*

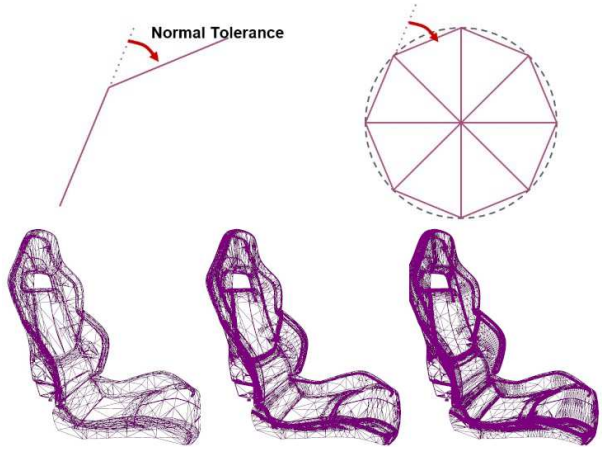

Abbildung 2.4: Oben im Bild: Eine Skizze zur Erklärung der Normal Tolerance. Darunter das Mesh eines Autositzes, der mit verschiedenen Werten tesseliert wurde. Von links: 40° → 21.500 Polygone; 10° → 100.000 Polygone; 5° → 295.000 Polygone. *(Quelle: https://docs.unrealengine.com/en-US/Studio/Datasmith/SoftwareInteropGuides/CAD/index.html Stand: 08.08.2019)*

2.2 Was sind 3D-Echtzeitanwendungen?

Bei der Arbeit mit Echtzeitanwendungen gibt es einige Unterschiede zur Arbeit mit Szenen für Pre-Renderings. Im folgenden Abschnitt sind die wichtigsten davon zusammengefasst.

2.2.1 Echtzeit-Rendering

Wie bereits in Motivation erläutert, muss eine Echtzeitanwendung Verarbeitungsergebnisse in einem vorgegeben Zeitrahmen liefern können. Bezogen auf den 3D-Bereich sind die Verarbeitungsergebnisse die Frames. Der vorgegebene Zeitrahmen wird durch die gewünschten FPS festgelegt. Werden beispielsweise 30 FPS angepeilt, bleibt also für die Anzeige eines einzelnen Frames ein Zeitrahmen von circa 0,03 Sekunden. Im Gegensatz zu der Dauer, die ein Frame in einem Pre-Renderer wie *V-Ray*[3] zur Berechnung benötigt, ist das ein sehr kurzer Zeitraum. Aus diesem Grund werden zum Rendern in Echtzeit andere Verfahren eingesetzt, als beim Pre-Rendering. Pre-Renderer arbeiten häufig mit Ray-Tracing. Bei diesem Verfahren werden Lichtstrahlen von der Kamera durch die Szene bis zum Auftreffen auf ein Objekt verfolgt. Von dort aus werden weitere Strahlen ausgesendet, die die Szene nach anderen Objekten oder Lichtquellen abtasten. Dieser Vorgang wird für jeden Pixel des zu erzeugenden Bildes durchgeführt. Aus den Ergebnissen, die die Strahlen liefern, werden Beleuchtung, Schatten, Reflektionen und vieles weitere errechnet. Auf diese Weise können sehr realistische Ergebnisse erzielt werden. Da für ein so erzeugtes Bild jedoch sehr viele Lichtstrahlen verfolgt werden müssen, ist es ein sehr rechenaufwändiges Verfahren. Für Echtzeit-Rendering ist Ray-Tracing daher nicht geeignet. [Whi05]

Stattdessen werden hier einfachere Verfahren verwendet. Das am weitesten verbreitete Verfahren für Echtzeitrendering ist die Rasterung(*Rasterization*). Es gibt heute viele verschiedene Rasterungsalgorithmen, das grundlegende Prinzip ist dabei jedoch gleich. Die Idee der Rasterung ist, das Polygon, das gerendert werden soll, auf eine 2D-Ebene zu projizieren, die das Bild beziehungsweise den Bildschirm repräsentiert. Man spricht dabei von einer Überführung von World- oder Object Space in Screen Space. Außerdem wird ein *Frame-Buffer* angelegt, bei dem es sich um ein Array handelt, dessen Größe der Auflösung des finalen Bildes entspricht. Nun wird die Ebene, auf die das Polygon projiziert wurde, in dieselbe Anzahl Rasterelemente zerteilt. Jedes Rasterelement entspricht also einem Pixel. Dann wird für jeden Pixel überprüft, ob er innerhalb oder außerhalb des projizierten Polygons liegt. Befindet sich ein Pixel innerhalb eines Objekts, so wird die entsprechende Information (zum Beispiel Sichtbarkeit, Helligkeits- und Farbwert etc.) für diesen Pixel im Frame-Buffer hinterlegt. So befinden sich nach dem Iterieren über alle Polygone die Informationen für das fertige Bild im Frame-Buffer. [Cat]

Diese Technik ist weitaus weniger rechenaufwendig als das Ray-Tracing Verfahren und ist aus diesem Grund bis heute das Standardverfahren für Echtzeitrenderings, hat jedoch auch Nachteile. Da jedes Polygon einzeln gerastert und in den *Frame-Buffer* geschrieben wird, können Effekte wie Schatten oder Reflektionen mit diesem Verfahren nicht errechnet wer-

[3] https://www.chaosgroup.com

den, da diese das Zusammenspiel mehrerer Polygone erfordern. Um trotzdem ein realistisches Renderergebnis zu erhalten, müssen diese Effekte nachgebildet werden. Dazu stehen verschiedene Möglichkeiten zur Verfügung, auf die später in dieser Arbeit noch einmal eingegangen werden wird. [Wal05] In Abbildung 2.5 ist zu sehen, wie ein Frame mittels der verschiedenen Verfahren erzeugt wird und wie sich die Ergebnisse qualitativ unterscheiden.

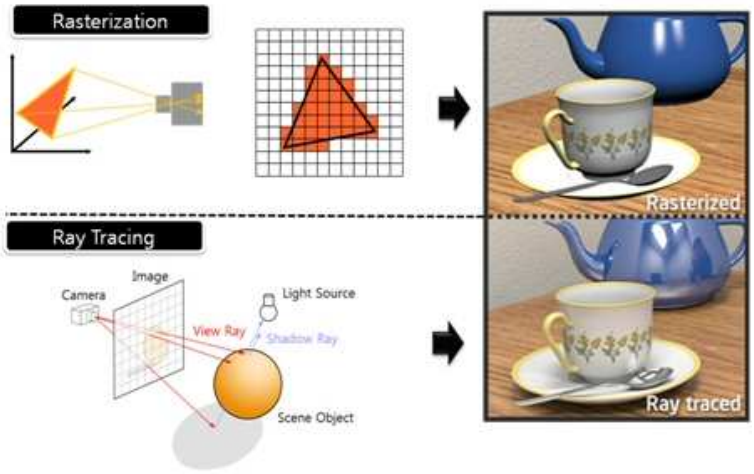

Abbildung 2.5: Die Abbildung zeigt die Unterschiede in der Erzeugung und der Qualität eines Bildes beim Einsatz von Ray Tracing und Rasterung. *(Quelle: http://mulsi.kaist.ac.kr/research/multimedia-processor/ray-tracing Stand: 21.08.2019)*

Es gibt allerdings auch einen weiteren Faktor, der Einfluss auf die Renderzeiten nehmen kann. Im Gegensatz zu einem Film, bei dem die einzelnen Frames und ihre Abfolge vorher festgelegt sind, reagiert eine 3D-Echtzeitanwendung auf Eingaben des Nutzers. Verschiebt der Nutzer beispielsweise ein Objekt, muss das Objekt im nächsten Frame auf der neuen Position erscheinen. Der Renderprozess kann also erst beginnen, wenn die Eingaben des Nutzers verarbeitet wurden. Die Verarbeitung der Nutzereingaben erfolgt dabei durch die Implementierung verschiedener Skripte. Erfolgt diese Verarbeitung zu langsam, bricht die Performance der Anwendung ein. Das Problem liegt dann nicht darin, dass das Rendering zu viel Zeit erfordert. Stattdessen muss der Renderer immer wieder auf die Ausführung eines Skriptes warten, woraufhin die Framerate ebenfalls sinkt.

Treten also beim Entwickeln einer 3D-Echtzeitanwendung Performanceeinbrüche auf, ist es wichtig, zuerst zu untersuchen, woher diese stammen. Braucht der Renderer zu lange, um ein Frame bereitzustellen, muss der Renderprozess optimiert werden. Dazu kann beispielsweise die Geometrie vereinfacht oder ein simplerer Shader verwendet werden. Verlangsamt die Ausführung eines Skriptes die Anwendung, muss das jeweilige Skript genauer betrachtet werden. Hier können beispielsweise unnötige Schleifendurchläufe vermieden oder effizientere Funktionen implementiert werden.

2.2.2 Beleuchtung

Eine weitere Besonderheit bei Echtzeitrenderings ist die Beleuchtung. Im Gegensatz zu einem Pre-Renderer, gibt es hier zwei Möglichkeiten, Licht zu berechnen. Die erste ist die Beleuchtung in Echtzeit. Dabei wird das Licht, wie bei einem Pre-Renderer, bei jedem Frame neu berechnet. Da dies bei vielen Lichtquellen oder vielen beleuchteten Objekten sehr viel Rechenleistung erfordern kann, gibt es außerdem die Möglichkeit, des *Lightmappings*. Lightmapping bezeichnet ein Verfahren zur statischen Per-Texel-Beleuchtung. Dabei wird für jede Fläche eines Objektes der gesamte Lichtwert berechnet. Diese Werte werden dann in einer Textur für das Objekt gespeichert. Diese Textur nennt man Lightmap. [Del04]

Das Erzeugen der Lightmaps wird auch als „Baking“ bezeichnet. Um eine Lightmap erzeugen zu können, muss das betreffende Objekt neben der UV-Map für die Texturierung eine weitere UV-Map besitzen. In dieser werden die berechneten Lichtwerte gespeichert. Diese UV-Map darf keine Überlappungen besitzen und muss uniform sein. Das bedeutet, dass die einzelnen UV-Shells im selben Maßstab vorliegen müssen. Abbildung 2.6 zeigt eine solche Lightmap.

Zu beachten ist jedoch, dass sich Lightbaking nur für statische Objekte eignet. Game Engines wie *Unity* bieten hier die Möglichkeit, Light Probes zu verwenden, um das „gebakede“ Licht auch auf dynamische Objekte zu übertragen. Durch den Einsatz von vorberechnetem Licht fallen aufwendige Echtzeit-Lichtberechnungen zur Laufzeit weg, wodurch der Rechenaufwand reduziert wird.

Wie bereits in Echtzeit-Rendering erwähnt, können durch das Rasterungsverfahren keine Schatten direkt berechnet werden. Stattdessen werden sogenannte *Shadow Maps* erzeugt. Es handelt sich dabei um Texturen, die Informationen zur Sichtbarkeit der Objekte aus Sicht einer Lichtquelle enthalten. Objekte, die für die Lichtquelle „sichtbar“ sind, sind beleuchtet, verdeckte Objekte sind schattiert. Erzeugt werden diese Texturen ebenfalls mithilfe des Rasterungsverfahrens. Dabei ist allerdings nur die Information zur Sichtbarkeit eines Objektes relevant, nicht aber jegliche Informationen zum Shading. [Wil78]

Es ist allerdings zu beachten, dass diese *Shadow Maps* pro Lichtquelle und vor dem Rendering des finalen Bildes erzeugt werden müssen. Das bedeutet, dass viele Lichtquellen den Renderprozess alleine durch die Erzeugung der *Shadow Maps* verlangsamen können. Insbesondere wenn die Szene viele Objekte enthält, die für die Erstellung der *Shadow Maps* berücksichtigt werden müssen, kann dies schnell zu spürbaren Einbrüchen in der Performance führen.

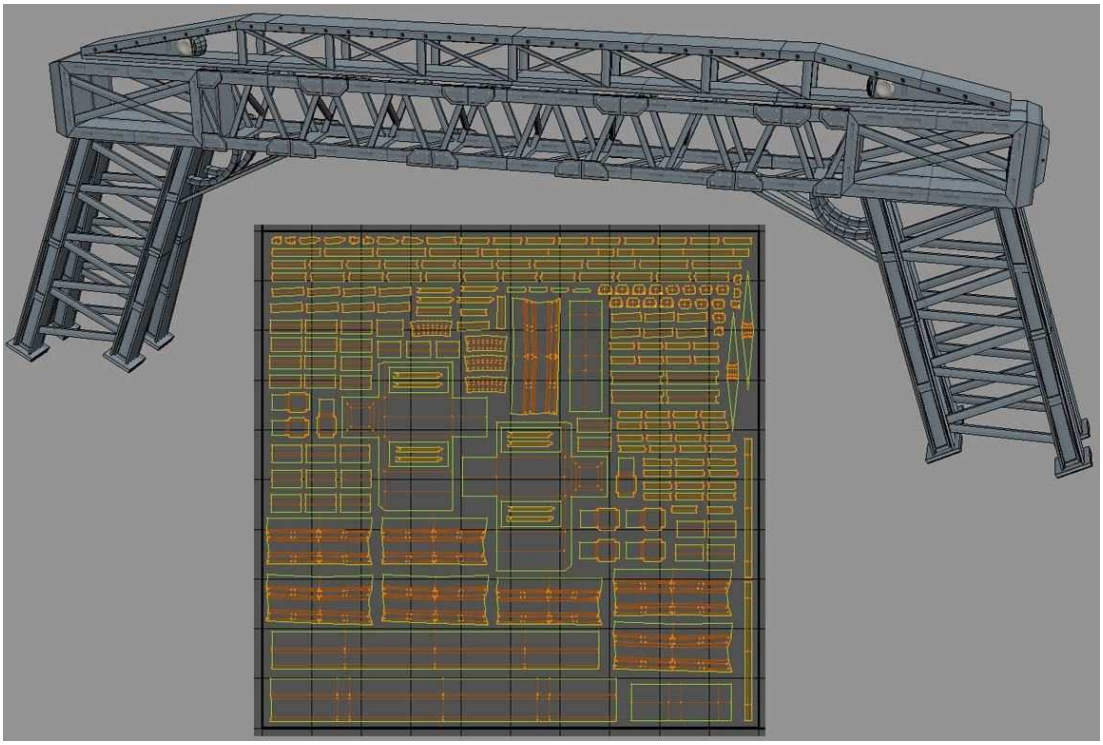

Abbildung 2.6: Die Abbildung zeigt ein das Mesh eines Metallgerüsts sowie das zugehörige Lightmap UV-Layout. *(Quelle: Unreal Dokumentation zu Lightmapping (https://docs.unrealengine.com/en-US/Engine/Content/Types/StaticMeshes/LightmapUnwrapping/index.html) Stand: 02.09.2019)*

Kapitel 3

Stand der Technik

Die Problematik dieser Arbeit lässt sich in zwei Teile einteilen: Die Aufbereitung der CAD-Daten sowie die Konzipierung und Implementierung von Animationen und Interaktionen. In diesem Kapitel werden zu diesen Themen bereits gewonnene Erkenntnisse zusammengefasst und betrachtet.

3.1 Aufbereitung von CAD-Daten

Die Verwendung von CAD-Daten im Bereich der 3D-Visualisierungen ist schon lange ein wichtiges Thema. So beschäftigte sich bereits 1999 Julien Berta mit der Visualisierung von CAD-Daten in einer Virtual-Reality Anwendung und damit in Echtzeit. In seiner Arbeit stellt er fest, dass CAD-Daten für die Verwendung in Echtzeit in geeignetere Formate überführt werden müssen. Bei dieser Konvertierung stellt er vier grundlegende Probleme fest. So treten bei konvertierten Modellen häufig Fehler im Modell auf, wie verdrehte Flächen (Back-Faces), fehlende Polygone oder fehlerhafte Topologie. Ein weiteres Problem ist der Verlust der Szenenhierarchie beim Export. Abhängig vom Zielformat werden Objekte oft kombiniert oder als einzelne Dateien exportiert. So gehen Informationen zur Szenenhierarchie verloren. Außerdem ist ein Problem, dass technische Daten, die in CAD-Formaten eingebettet sind, beim Export in andere Formate oft verloren gehen. Auch in CAD-Programmen angelegte Verhaltensweisen wie Kinematik werden nur sehr selten mit exportiert.[Ber99]
Für diese Arbeit sind dabei jedoch nur die ersten beiden Probleme von Bedeutung, da für die Implementierung von Animationen weder technische Daten der Teile, noch vordefinierte Verhaltensweisen benötigt werden.

In einem Konferenzpapier zu einer Virtual-Reality Anwendung zur Visualisierung von CAD-Daten namens ENVIRON (**En**vironment for **vir**tual **O**bject **N**avigation) werden weitere Probleme aufgeführt, die bei der Verwendung von CAD-Daten in Echtzeit relevant sind. So enthalten CAD-Daten oft viele einzelne Objekte. Eine hohe Objektkonzentration auf engem Raum führt in Echtzeit jedoch schnell zu Performanceeinbrüchen, da viele Objekte auf einmal gerendert werden müssen. Zusätzlich führt die Tesselierung von CAD-Daten häufig zu einer hohen Polygonanzahl, wie bereits in Abschnitt 2.1 erläutert. Ein weiteres Problem ist

das Fehlen von Informationen zur Texturierung von Objekten, die für eine Visualisierung jedoch wichtig sind. [CRS+]

Um das Problem der hohen Komplexität zu lösen, müssen die CAD-Daten zunächst vereinfacht werden. 2015 beschäftigte sich ein Team der Technischen Universität Chemnitz um Mario Lorenz mit einem automatisierten Ansatz zur Vereinfachung solcher Daten. Dabei stellten sie fest, dass das Reduzieren von CAD-Daten ein sehr komplexes Problem ist. Daher könne ein teilweise oder vollständig automatisierter Prozess nicht die gleiche Qualität erreichen, wie eine manuell durchgeführte Reduktion eines Experten. Dennoch konnte mit geeigneter Software eine automatisierte Reduktion der Daten um rund 50 Prozent erreicht werden, ohne dabei an optischer Qualität einzubüßen.[LSR+16]

Bei den Verfahren zur automatischen Tesselierung und Reduktion von CAD-Daten gibt es jedoch Unterschiede. Im Rahmen der Entwicklung einer neuen Methode zur Tesselierung und Reduktion wurden 2003 verschiedene, bereits existierende Verfahren untersucht. Dabei wurde ein wichtiges Unterscheidungskriterium festgestellt: Acht der neun untersuchten Verfahren behandelten das Modell uniform. Das bedeutet, dass jedes Teil des Objekts mit denselben Parametern behandelt wird. Die Qualität des Objekts nach der Reduktion ist dabei ein Ergebnis und kein Kriterium. Der neunte Ansatz dagegen bot die Möglichkeit, eine maximale Abweichung zu definieren. Jedes einzelne Teil wird dann solange reduziert, bis diese Abweichung erreicht ist. Ein solcher Ansatz ermöglicht eine höhere Qualität zu erreichen, ohne unnötig viele Polygone zu behalten.[PMT03]

Neben den Arbeiten, die sich speziell mit der Aufbereitung von CAD-Daten beschäftigen, sollten auch Möglichkeiten zur Reduktion von High-Poly Modellen im Allgemeinen betrachtet werden. Ein weit verbreitetes Verfahren ist dabei das sogenannte *Normal Baking*. Dabei wird ein komplexes Modell zunächst dupliziert und anschließend stark reduziert. Als Ergebnis entsteht so eine wesentlich weniger detaillierte und damit weniger komplexe Version des ursprünglichen Modells. Nun wird aus dem ursprünglichen Modell eine Normal-Map generiert und dem reduzierten Modell als Textur zugewiesen. Auf diese Weise können Details dargestellt werden, die in der Geometrie des Objektes nicht mehr vorhanden sind.[Web17] Abbildung 3.1 zeigt ein Beispiel für dieses Verfahren.

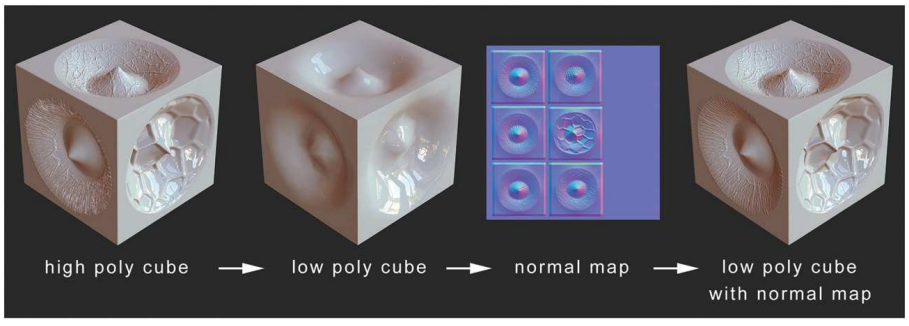

Abbildung 3.1: Darstellung der Funktionsweise des Normal Bakings. Quelle: [Web17]

3.2 Animationen und Interaktionen

Animationen werden in vielen Bereichen eingesetzt, um Zusammenhänge oder Abläufe zu erklären. Es gibt viele Abhandlungen darüber, wann und wie Animationen in edukativem Kontext eingesetzt werden sollten. In diesem Abschnitt werden einige für diese Arbeit relevante Erkenntnisse zusammengefasst.

So sind bereits einige Prinzipien bekannt, die bei der Erstellung von Animationen zu Aus- und Weiterbildungszwecken berücksichtigt werden sollten. [May14] definiert folgende Prinzipien:

1. **Das Prinzip der Aufnahme**
   Dieses Prinzip besagt, dass die Aussage einer Animation vom Betrachter sofort aufgenommen werden können soll. Das bedeutet, dass auf „kosmetische“ Extras, die nicht zur Vermittlung der Information beitragen, verzichtet werden sollte. Es besagt auch, dass Realismus nicht zwingend notwendig ist, um eine bestimmte Information zu vermitteln.

2. **Das Prinzip der Kongruenz**
   Eine Animation sollte eher dem konzeptionellen Modell als dem realen Phänomen entsprechen. Damit ist gemeint, dass eine Abstraktion oder Vereinfachung zum Verständnis eines Sachverhaltes beitragen kann, auch wenn es die tatsächlichen Abläufe nicht akkurat abbildet. Es sollte also stets im Vordergrund stehen, die gewünschte Information zu vermitteln.

3. **Das Prinzip der Interaktivität**
   Die Aussage dieses Prinzips ist, dass die mit einer Animation vermittelte Information leichter aufgenommen und verarbeitet werden kann, wenn der Nutzer die Möglichkeit hat, mit der Anwendung zu interagieren. Kann der Nutzer beispielsweise eine Animation pausieren und fortsetzen, bietet ihm das die Möglichkeit, die Animation in Teilstücken zu betrachten und die Informationen in einem selbst gewählten Tempo aufzunehmen.

4. **Das Prinzip der Aufmerksamkeitsführung**
   Bei einer Animation bewegen sich oft mehrere Objekte auf einmal. Der Benutzer muss also darauf hingewiesen werden, wann in welchem Bereich wichtige Änderungen vorgehen. Die Aufmerksamkeit des Nutzers muss also auf relevante Bereiche gelenkt werden, damit er keine wichtigen Informationen verpasst. Dazu können begleitende Texte oder die Einblendung von Hinweisen dienen.

5. **Das Flexibilitätsprinzip**
   In der Regel kann nicht vorhergesagt werden, wie weit der Benutzer in seinem Verständnis des behandelten Themas fortgeschritten ist. Es sollten also stets Möglichkeiten für den Nutzer bestehen, das Tempo und den Ablauf der Präsentation von Informationen zu steuern. So kann gesichert werden, dass sowohl Einsteiger als auch Fortgeschrittene einen Nutzen aus einer Anwendung gewinnen können.

Außerdem können Animationen auch einem rein kosmetischen Zweck dienen. In manchen Fällen kann der Betrachter durch ansprechende Animationen beispielsweise motiviert werden, sich weiter mit der Anwendung zu beschäftigen, auch wenn diese Animationen keinen informativen Mehrwert bieten. Es muss jedoch beachtet werden, das solche Animationen auch vom Wesentlichen ablenken können. Daher sollte ihr Einsatz gut überlegt sein.[WKM02]

Auch zur technischen Umsetzung von Animationen von CAD-Modellen gibt es bereits verschiedene Ansätze. Da CAD-Daten Konstruktionsdaten sind, werden sie häufig genutzt, um die Konstruktion eines Objektes zu simulieren beziehungsweise zu animieren. Dazu können sogenannte 4D-CAD-Tools verwendet werden. Dabei handelt es sich um CAD-Tools, die neben den 3 räumlichen Dimensionen auch die vierte Dimension – also die Zeit – berücksichtigt. Konkret bedeutet das, dass in diesen Systemen auch die zeitliche Abfolge der Konstruktion hinterlegt ist. Aus diesen Daten kann dann eine Animation der Konstruktion erstellt werden. [MKFH96]
Für diese Arbeit ist dieser Ansatz jedoch nicht geeignet, da nicht die Konstruktion des Objektes animiert werden soll, sondern Interaktionen mit dem fertig konstruierten Objekt. Außerdem setzt diese Vorgehensweise voraus, dass ein entsprechendes 4D-CAD-Tool verwendet zur Erstellung der Daten verwendet wurde und die erforderlichen Daten zum Konstruktionsablauf hinterlegt wurden.

Dieses Problem wird auch in einer Arbeit behandelt, die sich mit der Erstellung von Animationen eines Kransystems beschäftigt. Auch hier mussten neue Animationen für einen in CAD-Software konstruierten Kran erstellt werden. Die Autoren der Arbeit stellten fest, dass für diese Animationen weitere Software benötigt wird und entschieden sich aus diesem Grund dazu, die CAD-Modelle in 3DS Max zu exportieren, um dort die Animationen zu erstellen. [MAHTF07]
Diese Herangehensweise eignet sich auch für diese Arbeit, da 3DS Max auch in das FBX-Format exportiert, welches für die weitere Verwendung in Unity geeignet ist.

3.3 Verwandte Arbeiten

In diesem Abschnitt werden Projekte untersucht, die ein ähnliches Ziel verfolgten wie diese Arbeit. Es wird betrachtet, welche Ziele verfolgt wurden, wie sie erreicht wurden, welche Aspekte für diese Arbeit relevant sind und was diese Arbeit von ihnen unterscheidet.

3.3.1 Projekt EventSpaces

Es gab bereits einige Ansätze, CAD-Daten in spielähnlichen Umgebungen interaktiv zu visualisieren. Bei frühen Projekten zur Visualisierung von CAD-Daten wurden diese mithilfe der Virtual Reality Modeling Language (VRML) in Echtzeit visualisiert. Ein Beispiel hierfür ist das Projekt *EventSpaces*. Es handelte sich dabei um eine Web-Anwendung, die Studenten die Möglichkeit bot, gemeinsam eine virtuelle Welt zu bearbeiten. Zu Grunde lagen dabei Computer Aided Architectural Design (CAAD)-Daten, die mithilfe des VRML-Formats visualisiert werden konnten.[HGH+00]

Das VRML-Format ermöglichte allerdings nur begrenzte Möglichkeiten zum Rendern der Objekte. Die erzeugten Renderings waren also nicht fotorealistisch. Da VRML (heute *X3D*[1]) außerdem hauptsächlich für webbasierte Inhalte konzipiert wurde, ist es für diese Arbeit nicht geeignet.

3.3.2 Projekt ENVIRON

Bei dem bereits erwähnten Projekt *ENVIRON* wurde das TDGN-Format gewählt. Dabei handelt es sich um eine erweiterte Version des CAD-Formats DGN. Durch dieses Format war es möglich, semantische Inhalte der CAD-Daten bei der Überführung beizubehalten. Zur Verbesserung der Qualität wurde außerdem *3DS Max* verwendet, um beispielsweise Lightmaps zu generieren. Zur Visualisierung wurde Open Scene Graph (OSG)[2] verwendet. Die Kombination aus TDGN und OSG ermöglichte es, Primitive wie Spheren oder Würfel in parametrischer Form einzulesen. Daraus konnten dann einfachere Meshes zur Visualisierung generiert werden. Für auftretende Probleme wie T-Junctions musste dabei allerdings ein spezieller Algorithmus entwickelt werden.[CRS+]
Da sich diese Arbeit jedoch mit den Möglichkeiten zur Animation von komplexen Modellen beschäftigt, ist dieser Ansatz nicht optimal. Der Unterschied liegt dabei darin, dass die semantische Verbindung zu den originalen CAD-Daten nicht benötigt wird, wodurch ein einfacheres Format verwendet werden kann. Außerdem soll das komplette Modell als Polygonmodell vorliegen, um Animationen erstellen zu können.

3.3.3 Game-Engines

2010 beschäftigten sich Wei Yan, Charles Culp und Robert Graf mit einem Weg, architektonische Daten interaktiv zu visualisieren. Sie nannten auch Game Engines als Möglichkeit, Architekturvisualisierungen mit Interaktionen zu versehen. Mithilfe von Game-Engines wurden bereits einige interaktive Architekturvisualisierungen auf hohem grafischen Niveau erstellt.
Sie stellten jedoch auch fest, dass es keinen einfachen und schnellen Weg gäbe, um CAD-Daten direkt in Game-Engines zu überführen, weshalb sie sich in ihrer Arbeit mit der Erstellung dessen beschäftigen, was sie ein „Crossover-Modul“ nennen. Es geht dabei darum, die Lücke zwischen Konstruktionsdaten und für Echtzeitrendering geeigneten Modellen zu schließen.[YCG11]

2008 wurde ein virtueller Shop zur Analyse von Kaufverhalten auf der Basis des Source-Codes des First-Person-Shooters *Counter Strike*® aufgebaut. Als Grund nannte der Autor die hochwertige Grafikqualität und das hohe Maß an Interaktivität, das damalig aktuelle Game-Engines boten.[Moo08]

Für ein Projekt zur Visualisierung von urbanen Umgebungen basierend auf CAD-Daten wurde ebenfalls 2008 *Unity* verwendet. Die Gründe dafür waren auch hier unter anderem die gute Grafikqualität sowie die Möglichkeit zur Implementierung eigener Skripte für verschie-

[1] https://www.web3d.org/x3d/what-x3d
[2] www.openscenegraph.org

dene Interaktionen. [IS08]

Da sich auch diese Arbeit mit der Implementierung verschiedener Interaktionen beschäftigt und zudem ein hoher Anspruch an die grafische Qualität der Anwendung besteht, wird daher auch hier *Unity* verwendet.

Kapitel 4

Methodik

In diesem Kapitel wird die angewendete Vorgehensweise näher erläutert. Die einzelnen Entwicklungsstufen werden dabei in je einem eigenen Abschnitt betrachtet. Es beginnt mit der Entwicklung des Konzepts, also mit der Frage, welche Animationen beziehungsweise Funktionen implementiert werden sollten. Anschließend wird die Aufbereitung der CAD-Daten behandelt. Der dritte Abschnitt beschäftigt sich dann mit der eigentlichen Implementierung der Animationen und Interaktionen. In einem letzten Abschnitt wird außerdem beschrieben, wie ein Graphical User Interface (GUI) erstellt wird, das dem Nutzer die implementierten Funktionen zugänglich macht.

4.1 Das Konzept

Dieser Abschnitt behandelt das Konzept der Arbeit. Es beschäftigt sich mit der Planung und Vorüberlegungen zur Umsetzung des Prototypen.

4.1.1 Das Modell

Wie bereits im Abschnitt Zielsetzung erwähnt, werden für die Erstellung des Prototypen die Daten einer Verpackungsmaschine genutzt. Konkret geht es hier um eine Verpackungsmaschine, die Lebensmittel, wie Wurst- und Käsescheiben verpackt. Abbildung 4.1 zeigt diese Maschine.

4.1.2 Die Auswahl der Animationen

Zu Beginn der Arbeit muss festgelegt werden, welche Animationen und Interaktionen untersucht werden sollen. Wie bereits im Abschnitt Problemstellung erläutert, werden vor allem Animationen gewählt, die im industriellen Umfeld typischerweise verwendet werden. Außerdem werden Animationen und Interaktionen ergänzt, deren Untersuchung anderweitig sinnvoll ist. Im Folgenden werden die zu untersuchenden Animationen aufgelistet:

1. **Animierter Austausch von Teilen:**
   Unabhängig vom genauen Einsatz der Anwendung, ist ein Austausch von einzelnen

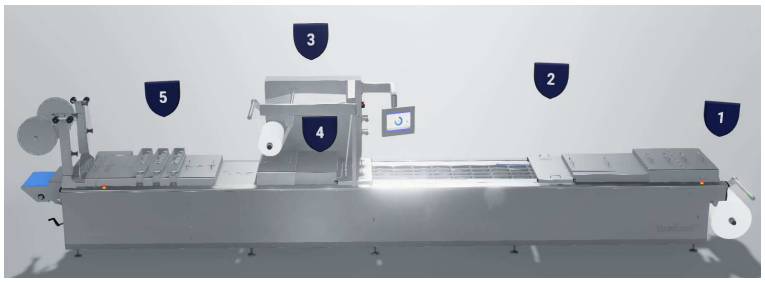

Abbildung 4.1: Screenshot des Modells der Maschine, die für die Erstellung der Anwendung verwendet wird.

Objekten mit Sicherheit eine grundlegende Funktion. Konfiguriert man beispielsweise ein Auto, möchte man die Felgen austauschen können. In der Prototypanwendung, die dieser Arbeit zugrunde liegt, soll eine große Verpackungsmaschine präsentiert werden, die in verschiedenen Ausführungen gebaut wird. Der Nutzer soll also die Möglichkeit haben, die Maschine in der Anwendung umzubauen. Diese Animation wird exemplarisch für jeden Anwendungsfall untersucht, bei dem einzelne Teile eines größeren Modells ausgetauscht werden sollen.

2. **Animation einzelner Arbeitsschritte:**
   Eine weitere sinnvolle Überlegung ist, dass es möglich sein soll, bestimmte Arbeitsschritte an oder von der Maschine detailliert darzustellen. Zum Einen könnten damit Wartungsarbeiten, wie beispielsweise das Wechseln von Verschleißteilen erklärt, zum Anderen aber auch Arbeitsabläufe in einem bestimmten Teil der Maschine genauer gezeigt werden. Untersucht werden sollen hiermit Animationen, bei denen es darum geht, einen bestimmten Vorgang detailliert darzustellen.

3. **Gesamtanimation des Objekts:**
   Zusätzlich zu detaillierten Informationen zu einzelnen Arbeitsschritten soll es auch die Möglichkeit geben, den kompletten Betriebsablauf der Maschine in einer Animation darzustellen. Hierbei steht im Vordergrund dem Nutzer einen visuellen Eindruck von der Maschine im Betrieb zu vermitteln. Eine solche Unterscheidung zu den detaillierten Animationen einzelner Arbeitsschritte macht in vielen Fällen Sinn. So können grundlegende Abläufe in einer vereinfachten Animation vermittelt werden. Für nähere Details stehen dann noch zusätzlich die detaillierteren Animationen zur Verfügung. Um es noch einmal am Beispiel eines Autos zu erklären, liegt der Unterschied darin, dass die Gesamtanimation zeigen kann, dass das präsentierte Auto ein Schiebedach eingebaut hat, während eine detaillierte Animation erklärt, wie es zu bedienen ist.

4. **Ausblenden von Modellbereichen:**
   In vielen Fällen ist es sinnvoll, einzelne Teile eines Objektes näher zu betrachten. Zu diesem Zweck sollte es möglich sein, die übrigen Objekte, die gerade nicht im Fokus des Betrachters stehen, auszublenden. Auf diese Weise wird die Aufmerksamkeit des Nutzers auf das gewünschte Objekt gelenkt, wie es das *Prinzip der Aufmerk-*

Abbildung 4.2: Explosionszeichnung eines Nissan ZEOD RC, die von Nissan anlässlich des Rennens von LeMans 2014 veröffentlicht wurde. *Quelle: Auto Bild (https://www.autobild.de/artikel/nissan-zeigt-le-mans-renner-als-explosionszeichnung-5132584.html)*

samkeitsführung beschreibt (siehe Animationen und Interaktionen). Da besonders bei komplexen Modellen, die aus vielen Einzelteilen bestehen, eine solche isolierte Ansicht eines einzelnen Teils sinnvoll ist, wird die Implementierung dieser Funktion untersucht.

5. **Austausch von Shadernetzwerken:**
   Durch das Austauschen von Shadern ist es möglich, das Aussehen der Oberfläche eines Objektes zu verändern. So könnte beispielsweise bei einem Konfigurator das Material eines Objektes zur Laufzeit geändert werden. Besonders im industriellen Kontext sind Ausführungen eines Objektes in verschiedenen Materialien keine Seltenheit. Daher wird untersucht, ob auch bei komplexen Modellen Shadernetzwerke zur Laufzeit ausgetauscht werden können.

6. **Animierte Explosionszeichnung:**
   Explosionszeichnungen werden häufig genutzt, um das Innere eines Objektes besser darstellen zu können (siehe Abbildung 4.2). Oft werden diese Zeichnungen auch als Montageanleitung genutzt oder um den Aufbau eines Objektes zu erklären. Aufgrund der häufigen Verwendung solcher Ansichten, besonders im technischen, industriellen Bereich, sollte auch die Verwendbarkeit bei komplexeren Modellen untersucht werden.

4.1.3 Vorüberlegungen zu den Animationen

Im folgenden Kapitel ist festgehalten, welche Überlegungen zu den einzelnen Animationen beziehungsweise Funktionen im Vorfeld der Implementierung angestellt wurden.

Teilaustausch

Um ein Teil animiert auszutauschen gibt es grundlegend zwei Optionen: Zum Einen kann das „alte“ Objekt aus dem sichtbaren Bereich bewegt – so dass es der Nutzer nicht mehr

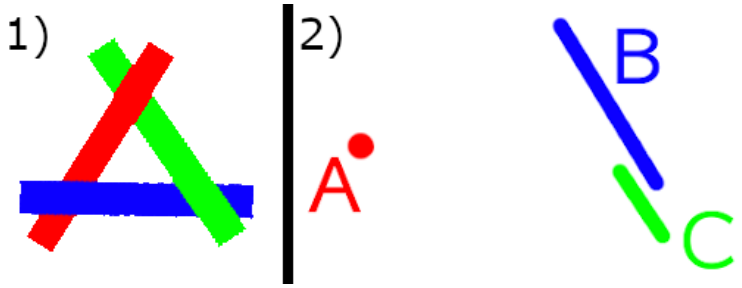

Abbildung 4.3: Die beiden Probleme, die Echtzeitrenderer mit Transparenz haben: A zeigt das Problem des *Triple Overlaps*, B zeigt die Problematik beim Finden eines Faktors zum Sortieren. *Quelle: OpenGL Dokumentation (https://www.khronos.org/opengl/wiki/Transparency_Sorting)*

sieht – und anschließend ausgeblendet werden. Anschließend wird das „neue“ Teil außerhalb des Bildes eingeblendet und eingefahren. Zum Anderen kann die Transparenz von Objekten animiert und das Objekt somit langsam ein- oder ausgeblendet werden (*Fade-In / Fade-Out*). Beide Optionen bringen ihre Problematiken mit sich.

Bei der ersten Option ist das Problem, dass der Nutzer die Kamera in einer Echtzeitanwendung frei bewegen kann. Es steht also nicht von Vornherein fest, was der sichtbare Bereich sein wird. Es müsste also ein Weg gefunden werden, sicherzustellen, dass der Nutzer die Ausblendung nicht sehen kann.

Bei der zweiten Option kommt ein im Echtzeitrendering weit verbreitetes Problem zum Tragen: das sogenannte *Transparency-Sorting*. Das Problem dabei ist, dass bei Echtzeitrenderings ein sogenannter *Depth-Buffer* oder *Z-Buffer* angelegt wird. Dieser Buffer enthält die Tiefeninformation der Szene, also welche Objekte vor- oder hintereinander liegen. Aufgrund dieses Buffers entscheidet der Renderer, welches Objekt welches andere verdeckt. Dem liegt die Annahme zugrunde, dass pro Pixel nur jeweils ein Objekt sichtbar sein kann. Transparenz jedoch stört diese Annahme, da mehrere Objekte auf einem Punkt sichtbar sein können. An dieser Stelle geraten Echtzeitrenderer schnell an ihre Grenzen, da die Verdeckung der Objekte nicht mehr korrekt bestimmt werden kann. Abbildung 4.3 zeigt die Problematiken genauer.

Teilabbildung **1)** zeigt dabei die Überlagerungen dreier Objekte: Rot liegt über Grün, Grün über Blau und Blau über Rot. Die meisten Sortieralgorithmen können diese Situation nicht auflösen, da A>B, B>C und C>A keine gültige Lösung hat. Infolgedessen determinieren die Algorithmen nicht. Teilabbildung **2)** dagegen zeigt ein anderes Problem. A sei in dieser Abbildung das Auge oder die Kamera. B und C seien transparente Objekte. Dem Betrachter ist klar, dass sich Objekt C vor Objekt B befindet. Programmatisch ist diese Tatsache allerdings schwer zu prüfen. Wenn man die Differenz von A zum nächsten Punkt von B und C misst, so liegt B näher an A, also vor C. Ein Vergleich der Entfernung zum Mittelpunkt des jeweiligen Objektes liefert dasselbe Ergebnis. Daher gibt es noch keine Lösungen für

auftretende Transparency-Sorting Fehler. [The06]

Es muss also getestet werden, ob einer der beiden Wege mit komplexen Modellen realisierbar ist.

Animation einzelner Arbeitsschritte

Bei dieser Animation, beziehungsweise Art von Animation, geht es darum den Betrachter die Funktionsweise eines einzelnen Arbeitsschrittes näher zu bringen. Das bedeutet konkret, dass hier der informative Gehalt der Animation der optischen Erscheinung übergeordnet ist. Hier findet also das in 3.2 erwähnte *Kongruenzprinzip* Anwendung.

Bei Animationen, die einen bestimmten Ablauf erklären sollen, handelt es sich oft um relativ komplexe Animationen. Im Fall der hier vorliegenden Maschine wurden diese Animationen bereits für ein vorhergehendes Projekt der *GM-W* erstellt. Es soll also hier konkret darum gehen, einen Weg zu finden, bereits erstellte Animationen von komplexen Modellen in eine Echtzeitanwendung zu überführen. Dazu gibt es zwei verschiedene Möglichkeiten, die abhängig vom konkreten Anwendungsfall eingesetzt werden können. Grundsätzlich ist die einfachste Methode, die in *3DS Max* erstellte Animation per FBX-Format in Unity zu importieren. Auf diese Weise kann die ablaufende Animation vom Nutzer gesteuert und in der 360-Grad Ansicht betrachtet werden. Ist diese Möglichkeit allerdings nicht praktikabel, kann die Animation auch gerendert werden und als Video in die Anwendung eingebunden werden. Diese Methode hat den Nachteil, dass zusätzliche Videos die Dateigröße der Anwendung erhöhen und dem Nutzer die Möglichkeit nehmen, die Animation aus verschiedenen Winkeln zu betrachten.

Es muss also untersucht werden, welche der beiden Optionen für Animationen an komplexen Modellen wann in Frage kommt und wie die jeweilige Option realisierbar ist.

Gesamtanimation

Im Gegensatz zur Animation eines einzelnen Arbeitsschrittes (4.1.3) soll diese Animation einen Eindruck davon verleihen, wie das präsentierte Objekt in Aktion versetzt wird. Es geht also eher um einen generellen Überblick über die Funktionsweise, als einen detaillierten Einblick in einen einzelnen Schritt. Dementsprechend soll hier auch die optische Abbildung der realen Abläufe der Informationsvermittlung übergeordnet sein.

Im Gegensatz zu der Animation eines einzelnen Arbeitsschrittes muss hier nicht jedes Detail animiert werden. Da der Sinn dieser Animation ist, die Funktion der Maschine darzustellen, reicht es, wenn der optische Eindruck von außen mit der realen Maschine übereinstimmt. Es ist also beispielsweise zwar wichtig, dass sich Laufbänder bewegen, nicht aber, dass sich im Inneren der Maschine kleine Zahnräder drehen. Aus diesem Grund ist diese Animation weniger komplex zu entwerfen, obwohl sie im Gegensatz zur Animation eines Schrittes die gesamte Maschine zeigt.

Da diese Animation noch nicht erstellt wurde, wird hier auch die eigentliche Erstellung der Animation genauer betrachtet. Dabei sollte darauf geachtet werden, dass die Animation für die Verwendung in Echtzeit geeignet ist.

Ausblenden von Modellbereichen

Diese Interaktion soll dazu dienen, dem Nutzer zu ermöglichen, gewählte Modellteile isoliert zu betrachten. Zu diesem Zweck müssen die Modellbereiche, die nicht betrachtet werden sollen, ausgeblendet werden. Dies kann auf zwei Arten erreicht werden.
Eine Möglichkeit wäre es, den Modellbereichen, die ausgeblendet werden sollen, eine hohe Transparenz zuzuweisen. Auf diese Art wären sie zwar noch leicht zu erkennen, so dass noch zu erkennen ist, wo in dem Modell sich der Teilbereich befindet, durch die hohe Transparenz stören sie jedoch nicht beim Betrachten des gewünschten Bereichs. Hierbei kommt allerdings wieder das bereits bei dem Teilaustausch beschriebene Problem des *Transparency Sortings* zum Tragen.

Eine zweite Möglichkeit ist, alle nicht gewünschten Teile vollständig auszublenden, indem man sie aus dem Renderprozess ausschließt. Dabei würde zwar der räumliche Bezug zum Rest des Objektes verloren gehen, dafür entstünden so keine durch Transparenz ausgelösten Renderprobleme.

Es muss also getestet werden, ob durch Transparenz ein optisch annehmbares Ergebnis erzielt werden kann. Sollte das nicht möglich sein, muss auf die zweite Lösung zurückgegriffen werden.

Austausch von Shadernetzwerken

Diese Funktion soll dem Nutzer ermöglichen, das Material eines Objektes zur Laufzeit der Anwendung zu ersetzen. Die Umsetzung dieser Funktion hängt stark davon ab, um welche Materialien es sich handelt. Soll beispielsweise roter Kunststoff durch grünen Kunststoff ersetzt werden, reicht es aus, lediglich eine Eigenschaft des Kunststoffmaterials zu verändern, die sogenannte *Albedo Color*, also die „Grundfarbe“. Andere Eigenschaften wie die Metallizität (*Metalness*) oder Rauhheit (*Roughness*) bleiben in diesem Fall unverändert. Anders verhält es sich, wenn beispielsweise Holz durch Metall ersetzt werden soll. In diesem Fall müssen mehrere Eigenschaften eines Materials geändert werden. Möglicherweise muss sogar der zugrunde liegende Shader gewechselt werden.

Animierte Explosionszeichnung

Diese Animation soll dem Nutzer nähere Informationen über den Zusammenhang und Aufbau einzelner Bauteile liefern. Abbildung 4.2 zeigt, wie diese Animation aussehen soll. Betrachtet man die Abbildung, so sieht man, dass dieser Effekt erreicht werden kann, indem alle Teile weg vom Ursprung der Szene bewegt werden. Dabei kommt es bei komplexen Modellen zu einem Problem. Wie bereits in Abschnitt 2.1 aufgeführt, enthalten CAD-Modelle oft sehr viele einzelne Teile. Im Falle der Maschine, die für diese Arbeit verwendet wird, sind das

über 8.000 einzelne Bauteile. Üblicherweise werden Animationen erstellt, indem *Keyframes* für jedes zu animierende Objekt gesetzt werden. Bei dieser Menge an Teilen wäre ein manuelles Setzen der *Keyframes* jedoch nicht nur sehr aufwändig, sondern auch möglicherweise unpräzise. Es sollte hier also ein Weg gefunden werden, diese Animation automatisch zu erzeugen. Außerdem stellt sich die Frage, ob es sinnvoll ist, jedes der 8.000 Teile einzeln zu bewegen. Wahrscheinlich ist, dass in diesem Fall nicht nur ein sehr unübersichtliches Ergebnis entsteht, sondern möglicherweise auch die Performance einbricht. Es ist also sinnvoll – wie im Beispiel der Abbildung – Gruppierungen bei der Explosion beizubehalten.

4.2 Aufbereitung der CAD-Daten

Dieser Abschnitt beschäftigt sich mit der Vorbereitung der CAD-Daten für die Verwendung in Echtzeit. Dabei geht es sowohl um die Komplexität der Datei im Sinne von Polygonanzahl und Dateigröße, als auch um sonstige Vorbereitungen wie UV-Mapping und Texturierung.

4.2.1 Reduktion der Datenmenge

Beschäftigt man sich mit der Reduktion der Datenmenge, die entsteht, wenn CAD-Daten in Polygonmodelle überführt werden, so ist der wichtigste Schritt dabei die Tessellierung. Die verwendete 3D-DCC Software für dieses Projekt ist *3DS Max*. Sie bietet die Möglichkeit, mithilfe des Autodesk Translation Framework (ATF), CAD-Formate, wie STEP, direkt zu importieren und zu tesselieren. Allerdings sind die Möglichkeiten zur Steuerung sehr begrenzt. So gibt es beim Import lediglich einen Regler für die *Mesh Resolution* des importierten Objektes. Auf einzelne Parameter der Tesselierung kann dabei kein Einfluss genommen werden. So ist es nicht möglich, Teile entsprechend ihrer Größe, Sichtbarkeit oder Relevanz verschieden zu tesselieren. Außerdem sind die Meshes, die der Tesselierungsalgorithmus des ATF erzeugt relativ komplex. Wird die Maschine für den Prototypen per ATF in *3DS Max* importiert, so entstehen selbst auf der niedrigsten Qualitätseinstellung knapp 7 Millionen Polygone.

Aus diesem Grund wurde die Software *PiXYZ Studio*[1] verwendet, um die CAD-Daten aufzubereiten. Es handelt sich dabei um eine Software, die speziell dazu entwickelt wurde, CAD-Daten für die Verwendung in Echtzeit vorzubereiten. Es können dabei CAD-Formate wie STEP oder IGES direkt importiert werden. Die Modelle werden dabei nicht direkt tesseliert, sondern weiterhin als CAD-Daten behandelt. Das bietet den Vorteil, das Teile einzeln und mit verschiedenen Einstellungen tesseliert werden können. Da die CAD-Daten im Hintergrund gehalten werden, ist es jederzeit möglich zum ursprünglichen Format zurückzukehren und die Tesselation neu zu starten. Außerdem bietet *PiXYZ* eine Vielzahl an weiteren Werkzeugen, um die entstandenen Daten weiter zu vereinfachen und zu reparieren, wie das Entfernen von nicht sichtbaren Teilen, das Korrigieren von Normalen oder das Erstellen von *Proxy Meshes*.

Die von *PiXYZ* Tesselator erstellten Meshes sind wesentlich weniger komplex als die Tessellierungsergebnisse, die *3DS Max* erzeugt. Zum Vergleich: Dieselbe Maschine, die bei der

[1] https://www.pixyz-software.com/studio

Tesselation durch *3DS Max* aus knapp 7 Millionen Polygonen bestand, hat bei der Tesselation durch *PiXYZ* lediglich etwas mehr als 2 Millionen Polygone. Exportiert man beide Modelle als FBX, so wird ein ähnlicher Unterschied auch in der Dateigröße ersichtlich. Die von *3DS Max* erzeugte Datei liegt bei circa 204 MB, während die von *PiXYZ* erzeugte Datei lediglich 80 MB belegt. Durch die Verwendung von *PiXYZ* konnten also 71% an Polygonen und 61% der Dateigröße eingespart werden. Abbildung 4.4 zeigt eine Umlenkrolle der Maschine aus beiden Meshes im Vergleich. Es fällt dabei auf, dass, besonders an schwierigen Stellen wie Kanten, auch der Edge Flow in dem von *PiXYZ* erstellten Modell sauberer verläuft.

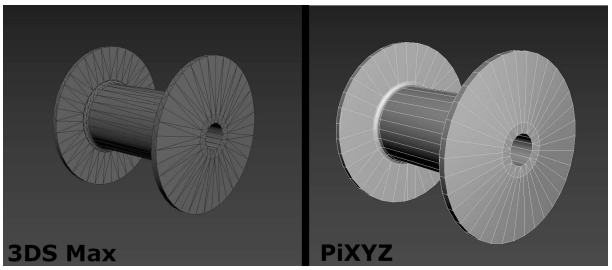

Abbildung 4.4: Screenshots einer Umlenkrolle der Maschine nach der Tesselation in *3DS Max* beziehungsweise *Pixyz*.

Eine weitere Maßnahme, um Dateigröße und Polygonanzahl zu verringern, ist, nicht benötigte Teile aus der Datei zu entfernen. Das für diese Arbeit verwendete Modell besteht aus über 8000 einzelnen Teilen. Davon liegen über 6000 Teile im Inneren der Maschine und sind damit für den Nutzer nicht sichtbar. Werden diese Teile aus der Szene entfernt, kann das Modell auf circa 1800 Teile reduziert werden, ohne dass ein Unterschied erkennbar ist. Die Dateigröße des Modells wird dabei von 140 MB auf 65 MB reduziert. Auch die Polygonanzahl des Modells wird dabei stark reduziert: Von 2,6 Millionen Polygonen verbleiben nach dem Ausräumen des Modells lediglich 1,4 Millionen. Um die innenliegenden Teile auszusortieren, kann *PiXYZ Studio* verwendet werden. Es bietet eine Funktion, die nicht sichtbare Teile automatisch aus dem Modell entfernt.

Im Fall des Prototypen, der für diese Arbeit entwickelt wird, ist das Entfernen der Teile jedoch nicht möglich, da eine Explosionszeichnung implementiert werden soll. Dabei werden nicht nur innenliegende Teile sichtbar, sie werden für diese Animation sogar zwingend benötigt. Daher bietet sich nur die Möglichkeit, die inneren Objekte in der Szene zu lassen, sie aber getrennt zu gruppieren. So können sie ausgeblendet werden, solange sie nicht benötigt werden und nur zum Abspielen der Explosionsanimation eingeblendet werden. Auf diese Weise wird die Dateigröße des Modells zwar nicht verringert, die Menge an Draw Calls und damit auch die Rechenlast wird so aber trotzdem reduziert, solange die Teile nicht eingeblendet werden. Dazu kann jedoch die automatische Funktion von *PiXYZ* nicht verwendet werden, da die Teile dabei komplett aus dem Objekt entfernt werden würden. Es muss also ein anderer Weg gefunden werden, um innenliegende Teile zu identifizieren und separieren.

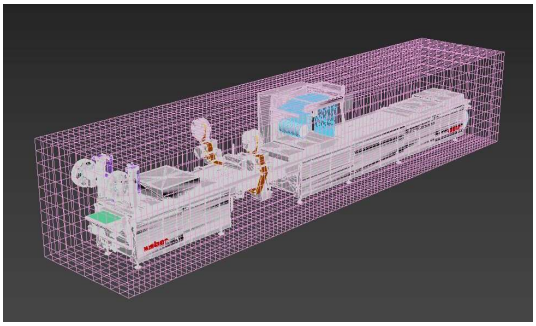

Abbildung 4.5: Screenshot eines Quaders, der die Maschine vollständig umgibt (in pink).

Da eine manuelle Sortierung der Teile zu aufwändig wäre, wurde im Rahmen dieser Arbeit ein Skript in *3DS Max* implementiert, das diese Aufgabe automatisiert. Um die Sichtbarkeit der einzelnen Teile zu überprüfen, können Raycasts aus verschiedenen Winkeln in Richtung der Objektmitte verwendet werden. Alle Teile, die als Erste von einem Strahl getroffen wurden, sind auch für den Nutzer aus diesem Winkel sichtbar. Diese Teile können in ein separates Layer verschoben werden, so dass nur Teile übrig bleiben, die nicht getroffen wurden und damit für den Nutzer nicht sichtbar sind. Um die Ausgangspunkte für die einzelnen Strahlen zu definieren, kann ein Objekt erstellt werden, das das Modell einschließt (siehe Abbildung 4.5). Anschließend kann jedes Vertex dieses Objektes als Ausgangspunkt für einen Strahl genutzt werden. Für die Richtung des Strahls kann der invertierte Normalenvektor jedes Vertices genutzt werden, da dieser senkrecht nach Innen zeigt (siehe Abbildung 4.6)).

Auf diese Weise kann die Dichte der verwendeten Strahlen und somit die Genauigkeit der Sortierung durch die Unterteilungen des umgebenden Objekts gesteuert werden. Desto mehr einzelne Vertices das Objekt hat, desto mehr Strahlen werden verwendet und desto enger liegen sie beieinander. Auf diese Weise kann das Risiko reduziert werden, dass einzelne Teile nicht von Strahlen getroffen werden, obwohl sie von außen sichtbar sind. Abbildung 4.7 zeigt, wie die Erhöhung der Unterteilungen Einfluss auf die Anzahl der verwendeten Strahlen nimmt.

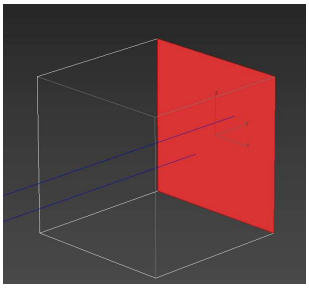

Abbildung 4.6: Die Abbildung zeigt in blau den Normalenvektor der rot markierten Seite des Würfels. Aufgrund der internen Behandlung von Quads als Tris werden hier zwei Vektoren dargestellt.

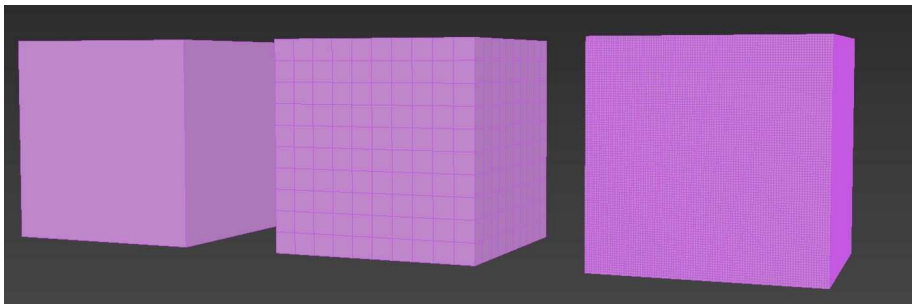

Abbildung 4.7: Die Abbildung zeigt einen Screenshot eines Würfels mit 1 (links), 10 (mittig) und 100 (rechts) Unterteilungen pro Seite.

Diese Herangehensweise hat allerdings auch Schwächen. Transparente Objekte werden von einem Raycast ebenfalls erkannt, was zur Folge hat, dass dahinter liegende Objekte als nicht sichtbar markiert werden, obwohl sie durch die Transparenz zu sehen wären. Ein weiteres Problem ist, dass das Hilfsobjekt sehr fein unterteilt sein muss, damit auch kleinste Objekte durch die Raycasts gefunden werden können. Eine hohe Unterteilung hat allerdings zur Folge, dass die Ausführungsdauer des Skriptes ansteigt. Es ist also nötig, die Ergebnisse des Skriptes zu kontrollieren und gegebenenfalls manuell zu verbessern.

4.2.2 UV- und Lightmapping

Um komplexe Modelle, wie die hier vorliegende Maschine, in Echtzeit verwenden zu können, reicht es jedoch nicht, nur die Datenmenge zu reduzieren. Ein weiteres Problem ist die Texturierung beziehungsweise Materialisierung. Da die CAD-Daten keine diesbezüglichen Informationen enthalten, müssen nach der Tesselierung Texturen und Materialien zugewiesen werden. Dazu ist es zuerst vonnöten, UV-Maps für die einzelnen Teile zu erstellen. Üblicherweise wird dazu ein *Unwrap* des Objektes erstellt. Das bedeutet, dass das Objekt zu einer 2D-Textur abgewickelt wird. Abbildung 4.8 zeigt dies am Beispiel eines Würfels.

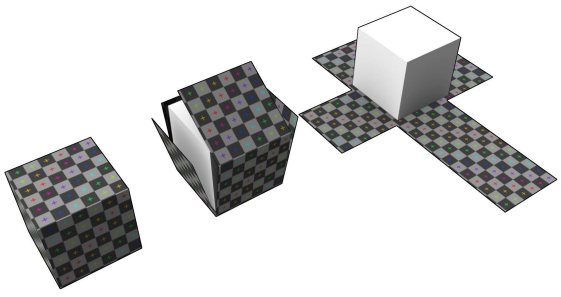

Abbildung 4.8: Visualisierung des Unwraps eines Würfels. Quelle: https://upload.wikimedia.org/wikipedia/commons/f/fe/Cube_Representative_UV_Unwrapping.png (Zephyris at en.wikipedia [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)])

Bei komplexeren Modellen werden diese Abwicklungen ebenfalls sehr komplex. Dabei kommt

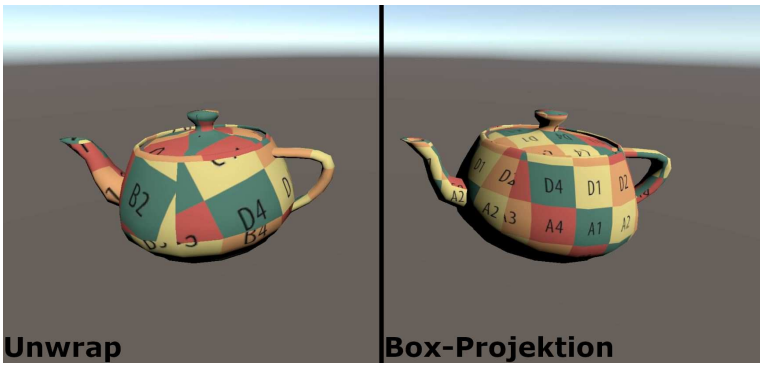

Abbildung 4.9: Screenshots eines 3D-Objektes mit Checker-Textur. Die UV-Map des linken Objektes wurde mithilfe eines Unwraps erzeugt, die des Rechten mithilfe einer Box-Projektion

es häufig zu Verzerrungen oder Verdrehungen der einzelnen *UV-Shells* zueinander, die dann manuell behoben werden müssen. Da das bei Modellen mit mehreren tausend Teilen ein zu großer Aufwand wäre, sollte hier mit einer UV-Projektion gearbeitet werden. Dabei werden die Texturen auf die Flächen projiziert, anstatt diese abzuwickeln. In diesem Fall wird eine Box-Projektion verwendet, die die Texturen von allen 6 Seiten projiziert. Abbildung 4.9 zeigt den Unterschied zwischen einem UV-Unwrap und einer Box-Projektion.

Eine Box-Projektion kann sowohl in *PiXYZ* als auch in *3DS Max* automatisch erstellt werden. Da eine Beleuchtung solch komplexer Modelle per Echtzeit-Beleuchtung in vielen Fällen zu aufwändig ist, bietet es sich an, an dieser Stelle auch eine Lightmap-UV zu erstellen. So kann das Licht später „gebaked“ werden, um die Performance zu verbessern, wie im Abschnitt 2.2 erläutert. Wie ebenfalls in diesem Kapitel erwähnt, handelt es sich bei Lightmaps um uniforme Unwraps eines Teils. Für die automatische Erstellung von solchen Maps bietet *3DS Max* allerdings keine eingebaute Lösung. Es wurden von Benutzern erstellte Skripte getestet, die eine solche Funktion bereitstellten. Bei Tests an CAD-Modellen führten sie jedoch immer zum Absturz des Programms. Aus diesem Grund sollte die Erstellung der Lightmaps in einem Programm vorgenommen werden, das für den Umgang mit solch komplexen Daten konzipiert wurde - in diesem Fall *PiXYZ*.

4.2.3 Materialisierung und Texturierung

Wie bereits in 3.1 erläutert, enthalten CAD-Daten keine Informationen zu den Materialien der Objekte. In der Regel sind jedoch den einzelnen Teilen verschiedene Farben zugewiesen, um beispielsweise verschiedene Werkstoffe zu kennzeichnen. Bei der Konvertierung in das FBX-Format werden diese lediglich als Standardmaterial mit entsprechender Base-Color interpretiert. Aus diesem Grund müssen neue Materialien erstellt und zugewiesen werden.

Die Zuweisung der Materialien auf die einzelnen Teile erfolgt normalerweise manuell. Arbeitet man jedoch mit komplexen Modellen mit mehreren Tausend Einzelteilen, muss ein anderer Ansatz gefunden werden. An dieser Stelle können die aus den CAD-Daten generier-

Abbildung 4.10: Ein Screenshot des Dialoges zum Ersetzen von Materialien in *Unity*. Links stehen die in dem Modell eingebetteten Materialien, rechts die Materialien, die zugewiesen werden sollen.

ten Standardmaterialien genutzt werden. Auch wenn diese Materialien selbst nicht nutzbar sind, so sind sie doch durch ihren Namen eindeutig identifizierbar. Es ist also möglich, diese Materialien per Skript durch neue Materialien zu ersetzen.

Das Ersetzen der Materialien wird durch *Unity* noch vereinfacht. Es gibt dort die Möglichkeit, für jedes Material eines Modells ein neues Material zu definieren. *Unity* wird dann beim Import des Modells die neuen Materialien korrekt zuweisen. Abbildung 4.10 zeigt den Dialog zum Ersetzen der Materialien. Dort ist zu sehen, dass beispielsweise das im Objekt eingebettete Material „Edelstahl“ durch das neue Material „steel_brushed“ ersetzt wird. Der Vorteil dieser Funktion ist, dass diese Einstellung bei jedem neuen Import des Modells angewendet wird. Wird also in *3DS Max* ein neues Teil hinzugefügt und mit dem Szenenmaterial „Edelstahl“ versehen, wird es beim Import in *Unity* das Material „steel_brushed“ erhalten. Auf diese Weise müssen nur ein Mal alle erforderlichen Materialien angelegt und zugewiesen werden. Die korrekte Materialisierung wird ab dann automatisch vorgenommen.

Um neue Materialien zuweisen zu können, müssen diese jedoch erst in *Unity* neu erstellt werden. *Unity* arbeitet mit Physically Based Rendering (PBR). Bei diesem Workflow geht es darum, den Verlauf des Lichtes physikalisch korrekt nachzubilden. Dazu werden, statt Werte wie Reflexivität festzulegen, Eigenschaften der Materialien wie *Metalness*(Metallizität) oder *Roughness*(Rauhheit) angegeben. Diese Eigenschaften können als fester Wert, oder durch eine Textur, eine sogenannte *Map* angegeben werden.

Um also ein Material zu entwerfen, muss für jede Eigenschafte eines Materials eine Map erstellt werden. Neben den Maps für Roughness und Metalness wird eine Map für die Grundfarbe (Albedo) sowie für die Struktur der Oberfläche eine *Height*- oder *Normalmap* benötigt. Grundsätzlich können diese Maps in jedem beliebigen Bildbearbeitungsprogramm erstellt werden. Sinnvoll ist es jedoch Software zu verwenden, die speziell für die Erstellung solcher Maps entwickelt wurden. Im hier vorliegenden Fall wurde dazu Allegorithmics *Substance Designer*[2] genutzt. Die Maps wurden dabei nicht im Rahmen dieser Arbeit erstellt, sondern für die Entwicklung des Prototypen zur Verfügung gestellt. Abbildung 4.11 zeigt die Maps

[2] https://www.substance3d.com/products/substance-designer

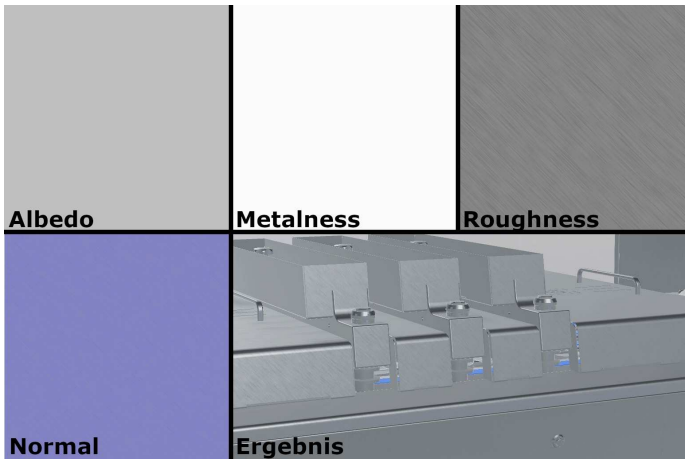

Abbildung 4.11: Screenshot des verwendeten Edelstahl Materials sowie die vier Maps, mit denen das Material erstellt wurde.

des Materials „steel_brushed“ und das fertige Material auf der Maschine.

4.3 Die Implementierung

Dieses Kapitel beschäftigt sich mit der eigentlichen Erstellung der Anwendung. Für jede Animation wird hier festgehalten, ob der geplante Weg erfolgreich war oder ob Probleme aufgetreten sind und wenn es Probleme gab, ob und wie sie gelöst werden konnten.

4.3.1 Animierter Austausch von Teilen

Diese Funktion soll es ermöglichen, den hinteren Teil der Maschine durch eine Variante mit beziehungsweise ohne eingebauten Etikettierer auszutauschen. Abbildung 4.12 zeigt die beiden Varianten. Es wird also der komplette hintere Teil der Maschine ausgetauscht.

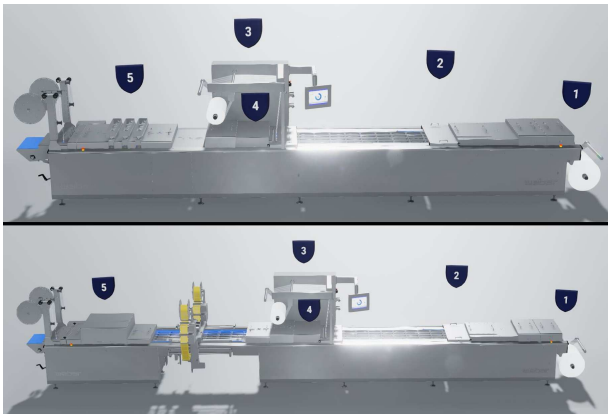

Abbildung 4.12: Oben im Bild ist die Maschine ohne Etikettierer zu sehen, unten die Variante mit Etikettierer

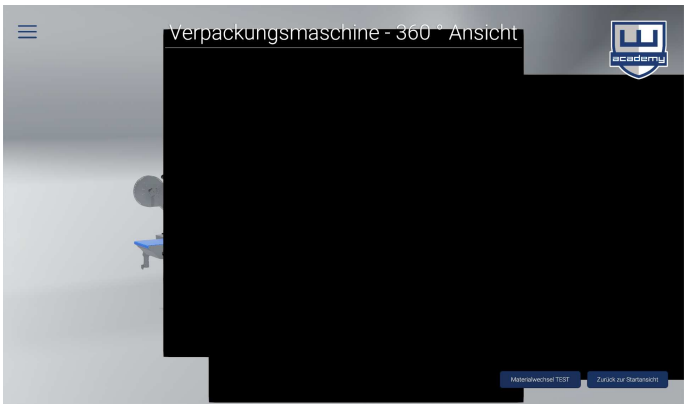

Abbildung 4.13: Screenshot des Renderfehlers, der entsteht, wenn die Transparenz eines Materials zur Laufzeit der Anwendung animiert wird.

Wie in 4.1.3 erklärt, gibt es zwei Wege, um einzelne Teile animiert auszutauschen. Um die auftretenden Probleme zu analysieren, werden beide Varianten getestet.

Fade-Out

Um einen Fade-In oder Fade-Out der Teile zu realisieren, müsste ein transparenter Shader verwendet werden. Dieser führt jedoch zu den bereits erläuterten Problemen beim Transparency Sorting. Es konnte keine Möglichkeit gefunden werden, diese Darstellungsfehler zu beheben. Daher ist die einzige Möglichkeit, den Shader zur Laufzeit auszutauschen. (Detaillierte Informationen zu diesem Vorgang sind in den Abschnitten 4.3.5, 4.1.3 und 5.2.5 zu finden.) Die Vorgehensweise dabei ist, zuerst einen Transparenz unterstützenden Shader zuzuweisen und anschließend die Transparenz des Materials mithilfe linearer Interpolation zu animieren. So konnte ein Fade-Out eines Modellbereichs zwar implementiert werden, allerdings führte die Animation der Transparenz zu Fehlern beim Rendering der Objekte, wodurch die in Abbildung 4.13 zu erkennenden Bildfehler entstanden.

Aus dem Bild bewegen

Ein Teil auszutauschen, indem man es aus dem sichtbaren Bereich bewegt, ist einfacher zu realisieren. Die Schwierigkeit dabei ist, dass sich der Nutzer der Anwendung frei umsehen kann, wodurch nicht feststeht, welchen Bereich der Nutzer sieht. Um also sicherzugehen, dass das Objekt nach dem Bewegen für den Nutzer verborgen ist, kann das Objekt außerhalb der Bühne platziert werden. Dazu bietet sich an, das Objekt zum Ausblenden nach unten zu bewegen. So befindet es sich unterhalb des „Bodens“ und ist somit für den Nutzer nicht zu sehen.

Um das Objekt zu bewegen, muss keine Animation erstellt werden. Stattdessen kann eine lineare Interpolation verwendet werden, um das Objekt animiert zu bewegen. Dazu wird folgender Code verwendet:

```
1 float timeToTravel = 1;
2 float currentTime = 0;
```

```
3  float normalizedTime;
4
5  while (timeToTravel >= currentTime)
6  {
7          currentTime += Time.deltaTime;
8          normalizedTime = currentTime / timeToTravel;
9          object.position = Vector3.Lerp(positionA, positionB,
               normalizedTime);
10         yield return null;
11 }
```

Die Funktion `Vector3.Lerp()` führt die lineare Interpolation von dreidimensionalen Vektoren durch. Da Positionen aus X-, Y- und Z-Koordinaten bestehen, kann diese Funktion hier verwendet werden. Die Dauer der Animation wird als `timeToTravel` festgelegt. In Zeile 8 wird die vergangene Zeit seit Beginn der Bewegung normalisiert, also in einen Wert zwischen 0 und 1 umgerechnet. 0 steht dabei für den Beginn der Bewegung, 1 für das Ende. Dieser Wert wird dann in der Interpolation genutzt, um die Position des Objektes auf eine relative Position zwischen Startposition (0) und Ausgangsposition (1) zu platzieren. Da diese Platzierung einmal pro Frame stattfindet, entsteht eine durchgehende Bewegung des Objektes. So kann ein animierter Austausch von Teilen realisiert werden.

Bei der Bewertung dieser Animation muss sich jedoch die Frage gestellt werden, ob und wann eine solche Animation zum Austausch von Teilen sinnvoll ist.

4.3.2 Animation einzelner Arbeitsschritte

Die Faktoren, von denen abhängt welche Methode verwendet werden sollte, sind der Umfang der Animation, wie die Animation in der verwendeten 3D-Software realisiert wird und wie die gewünschte Information vermittelt werden kann. Im Bezug auf die Anwendung, die dieser Arbeit zugrunde liegt, geht es um Animationen, die für verschiedene „Stationen“ innerhalb der Maschine die dort stattfindenden Vorgänge visualisieren. Diese Stationen sind beispielsweise die Folienzuführungen, das Zuschneiden der Folien oder die Versiegelung. Insgesamt gibt es dabei fünf verschiedene Stationen. In Abbildung 4.12 sind die Stationen durch die blauen Schilder markiert. Bei der Implementierung dieser Animationen ergeben sich folgende Problematiken:

Möchte man einen bestimmten Vorgang innerhalb der Maschine erklären, indem man die Animation in *Unity* importiert, so ist dies nur möglich, wenn die komplette Maschine animiert wird, da es in der Realität nicht möglich ist, nur einen Teil einer Maschine in Betrieb zu nehmen und der Nutzer während der Animation die komplette Maschine betrachten kann. Implementiert man nur ein gerendertes Video, so hat man die Möglichkeit, dem Nutzer den gewünschten Vorgang isoliert zu erklären, da hier der Rest der Maschine nicht gezeigt werden muss.

Auch die Möglichkeiten zur Erstellung der Animation sind zu berücksichtigen. So wurden

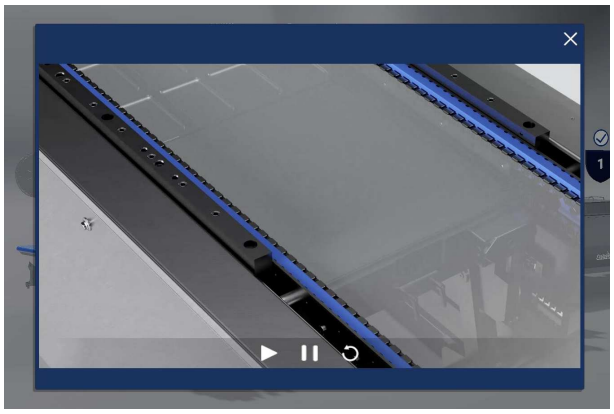

Abbildung 4.14: Screenshot des Videoplayers in der Anwendung.

für die Erstellung der Animationen einzelner Abläufe dieser Maschine verschiedene Modifier in *3DS Max* benötigt. Da das FBX-Format jedoch nur Animationen durch Keyframes oder Morph Targets unterstützt, können diese Animationen so nicht exportiert werden.

Des Weiteren mussten für manche Animationen Darstellungen gewählt werden, die ebenfalls nicht exportiert werden können. Beispiele dafür sind Querschnitte und schematische Abbildungen. Um also deren Verwendung zu ermöglichen, ist es nötig, die Animation vorzurendern und als Video einzubinden.

Dazu kann in *Unity* ein Videoplayer aus GUI-Elementen erstellt werden. Dazu wird ein leeres Bild verwendet, auf dem dann zur Laufzeit das gewünschte Video abgespielt werden kann. Außerdem können über Buttons Funktionen zur Steuerung des Videos implementiert werden. Abbildung 4.14 zeigt den Videoplayer in der Anwendung.

Um ein Video auf diese Art auf ein leeres Bild zu übertragen, kann die *Unity* Videoplayer-Komponente verwendet werden. Ihr kann das Bild als Rendertextur übergeben werden. So überträgt die Komponente, während sie ein Video abspielt, jedes Frame an das leere Bild.

Es gibt jedoch verschiedene Möglichkeiten, wie das Video in die Anwendung eingebunden werden kann. Eine Möglichkeit ist, das Video direkt in das *Unity* Projekt zu importieren. So wird es beim Build-Prozess normal in die Anwendung integriert. Auf diese Weise können die Videos allerdings nach dem Buildprozess nicht mehr ausgetauscht werden. Es ist jedoch auch möglich, die Videos im normalen Dateisystem abzulegen. In diesem Fall wird dem Videoplayer nicht direkt das Video, sondern der Pfad zum Speicherort und der Name des Videos übergeben. So können die Videos jederzeit ausgetauscht werden. Eine dritte Möglichkeit ist, die Videos auf einem Webserver abzulegen und dem Videoplayer die entsprechende URL zu übergeben. So fällt für die Anwendung kein weiterer Speicherbedarf an und auch ein Austausch der Videos ist jederzeit möglich. Der Nachteil dieser Methode ist jedoch, dass eine Internetverbindung zwingend notwendig ist, um die Videos anzusehen.

Welche dieser Methoden verwendet werden sollte, hängt in jedem Einzelfall von den Anforderungen an die Anwendung ab. Im hier vorliegenden Fall soll die Anwendung auch offline

nutzbar sein, weswegen die letzte Möglichkeit nicht anwendbar ist. Aufgrund der höheren Flexibilität durch Austauschbarkeit der Videos werden für diese Anwendung die Videos im lokalen Dateisystem abgelegt und über Pfadangaben referenziert.

4.3.3 Gesamtanimation des Objektes

Wie bereits in 4.1.3 erläutert, steht in diesem Fall im Vordergrund, abzubilden, wie die Maschine im Betrieb aussieht. Dazu ist es nicht notwendig, jedes Teil zu animieren. Es reicht aus sich auf die Teile zu beschränken, deren Bewegung von außen sichtbar ist. Bevor also die eigentliche Animation begonnen wird, muss überlegt werden, welche Teile animiert werden müssen. Im Falle der Maschine sind das die Folien, die Laufbänder und die Umlenkrollen der Folienzuführung.

Die Umsetzung der Animation beginnt in *3DS Max*. Hier können die beweglichen Teile der Maschine per Keyframe animiert werden. Eine Besonderheit tritt hier nur bei den Folien auf. An den Stellen, an denen sich die Rolle zur Einführung bewegt, muss das Mesh der Folie verformt werden. Da das FBX-Format *Morph Targets* unterstützt, können diese verwendet werden um die Verformung zu animieren. Dabei wird das Mesh in die Zielpose verformt. Bleiben dabei die Vertexindizes unverändert, kann zwischen den beiden Posen interpoliert werden. Diese Interpolation kann durch Keys gesteuert werden, wodurch eine Animation entsteht. Diese Animationen können nun in *Unity* importiert werden.

Die Folie selbst jedoch ist nicht animiert. Da sich an der Seite der Folie Markierungen befinden, fällt dieser Fehler sofort auf. Da es aber nicht möglich ist, die Folie tatsächlich zu animieren, kann sich hier des Texturscrollings bedient werden. Die Idee ist dabei, die Textur der Folie so zu verschieben, dass der optische Eindruck entsteht, die Folie würde sich bewegen. Abbildung 4.15 zeigt den gewünschten Effekt.

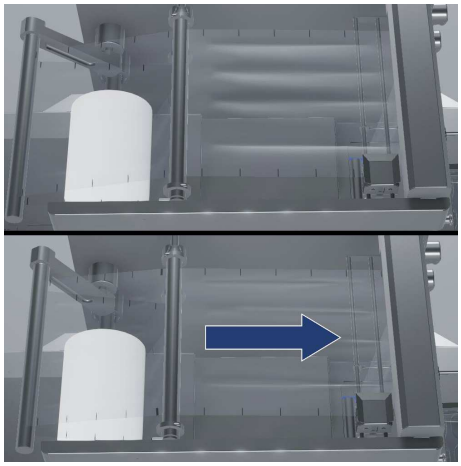

Abbildung 4.15: Bei Bewegung des Arms links im Bild, soll sich die Textur der Folie in Pfeilrichtung bewegen.

Am einfachsten kann diese Idee direkt in *Unity* realisiert werden. Per Skript kann hier der Versatz (*Offset*) einer Textur zur Laufzeit verändert werden. Auf diese Weise kann ohne

große manuelle Arbeit und ohne die Performance stark zu belasten eine Animation erstellt werden, die dem Nutzer zeigt, wie die Maschine im laufenden Betrieb aussieht.

4.3.4 Ausblenden von Modellbereichen

Um Modellbereiche auszublenden müssen sie aus dem Renderprozess ausgeschlossen werden. Dazu kann in *Unity* die *Mesh-Renderer* Komponente eines Objektes deaktiviert werden. Um diese Operation auf einen kompletten Modellbereich anzuwenden macht es Sinn, diese Modellbereiche bereits vor dem FBX-Export entsprechend zu gruppieren. So kann dann über die Gruppen iteriert und komplette Modellbereiche ausgeblendet werden. Auf diese Weise kann der Nutzer einen einzelnen Modellbereich isoliert betrachten. Dazu macht es Sinn, die Kamera auf das fokussierte Teil zu fixieren, damit der Nutzer nach dem Ausblenden nicht erst nach dem verbleibenden Teil suchen muss.

Um räumliche Bezüge zum Rest des Objektes nicht zu verlieren, ist es möglich die übrigen Teile nicht vollständig auszublenden, sondern sie stark transparent darzustellen. Dabei kommt es jedoch wieder zu den bereits erläuterten Problemen des Transparency Sortings. In diesem Fall können diese vermieden werden, indem ein Ersatzobjekt des kompletten Objektes erstellt wird, das aus nur einem Mesh besteht. So müssen keine verschiedenen Teile sortiert werden, wodurch auch keine grafischen Fehler mehr entstehen. Dieses Ersatzobjekt kann dann eingeblendet werden, wenn das eigentliche Objekt ausgeblendet wird. Wichtig dabei ist, dass das Ersatzobjekt die Teile, die nicht ausgeblendet werden, auch nicht enthalten darf, da es sonst zu Überlagerungen kommt. Es müssen also insgesamt drei Ersatzobjekte erstellt werden, da drei Modellbereiche einzeln betrachtet werden können.

Die Ersatzobjekte können auf verschiedene Arten erzeugt werden. Im Falle dieser Arbeit werden mittels *PiXYZ* Nachbildungen aus Voxeln erstellt. Abbildung 4.16 zeigt das Ergebnis dieses Verfahrens. In dem abgebildeten Fall soll das Ende der Maschine detailliert betrachtet werden. Wie zu erkennen ist, werden die restlichen Modellbereiche transparent dargestellt.

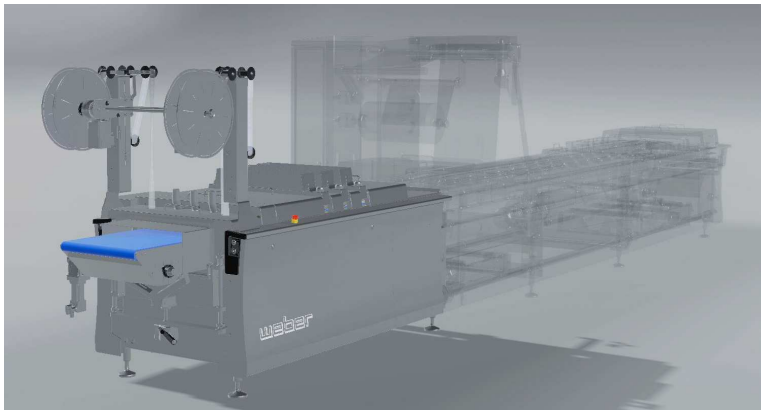

Abbildung 4.16: Screenshot der Anwendung mit ausgeblendeten Modellbereichen bei aktivierter Transparenz.

4.3.5 Austausch von Shadernetzwerken

Um in *Unity* den Shader eines Objektes zur Laufzeit auszutauschen, gibt es mehrere Möglichkeiten. Eine Möglichkeit ist, den Shader im gewünschten Material zur Laufzeit per Skript auszutauschen. Materialien sind in *Unity* grundsätzlich über ein Attribut des Renderers eines Objektes ansprechbar. Dabei gibt es zwei verschiedene Attribute, über die auf das Material zugegriffen werden kann. Eine Möglichkeit, um beispielsweise die Farbe eines Materials zu ändern, stellt folgende Codezeile dar:

```
1 Object.Renderer.material.color = Color.black;
```

Dabei ist *Object* das Objekt dessen Material geändert werden soll. Wird wie hier über die *material* Eigenschaft auf das Material zugegriffen, so wird die Änderung nur auf dieses eine Objekt angewendet. Dazu legt *Unity* automatisch eine Kopie des Materials an, auf dass die Änderungen angewendet werden. Iteriert man also mit einer Schleife über 100 Objekte, um jeweils den Shader auszutauschen, werden automatisch auch 100 Kopien des Materials im Speicher abgelegt. Problematisch ist dabei außerdem, dass Draw Calls für Objekte mit verschiedenen Instanzen des gleichen Materials nicht durch Batching zusammen gefasst werden können. So entstehen also zusätzliche Draw Calls und damit längere Renderzeiten. Um das zu vermeiden, kann auch die *sharedMaterial* Eigenschaft eines Objektes verwendet werden:

```
1 Object.Renderer.sharedMaterial.color = Color.black;
```

Dabei wird keine Kopie des Materials angelegt, sondern das Ausgangsmaterial direkt verändert. Das bedeutet, dass sich die Änderung auf alle Objekte auswirkt, die das selbe Material verwenden. Es ist jedoch zu beachten, dass die Änderungen auch nach einem Neustart der Anwendung erhalten bleiben, da das Material selbst verändert wurde anstatt einer Kopie. (Wird im weiteren Verlauf der Arbeit über das `material` oder `sharedMaterial` Attribut gesprochen, so ist darunter die jeweilige Zugriffsmethode auf das Material zu verstehen.) Eine weitere Möglichkeit ist, alle benötigten Materialien mit verschiedenen Shadern im Vorhinein anzulegen und zur Laufzeit lediglich das gewünschte Material auf das betreffende Objekt anzuwenden. Auf diese Weise entstehen keine zusätzlichen Materialinstanzen und Objekte mit gleichem Material können trotzdem getrennt behandelt werden.

Im Falle dieser Anwendung soll es die Möglichkeit geben, die Folien, die der Maschine zugeführt werden, auszutauschen. Dabei ändern sich, außer der Farbe des Materials, keine weiteren Eigenschaften. Daher reicht es auch, die Textur der Albedo Color zur Laufzeit auszutauschen. Dazu werden die benötigten Texturen in einem Array hinterlegt, sodass sie zur Laufzeit in die Materialien eingesetzt werden können. Da es in diesem Fall keine Objekte gibt, die ebenfalls das Folienmaterial nutzen und nicht geändert werden können, wird hier das `sharedMaterial` Attribut verwendet, um die Erzeugung zusätzlicher Materialinstanzen zu vermeiden.

Da jedoch die Vor- und Nachteile der verschiedenen Methoden getestet werden sollen, werden auch die anderen Optionen testweise implementiert, um die konkreten Auswirkungen zu

untersuchen. Anhand dieser Ergebnisse können dann Aussagen getroffen werden, wann die Verwendung welcher Option sinnvoll ist.

4.3.6 Animierte Explosionszeichnung

Wie in 4.1.3 festgestellt, muss ein Weg gefunden werden, die Explosionsanimation automatisch zu generieren. Dazu ist es sinnvoll, zunächst zu betrachten, was genau bei einer Explosionsanimation passiert. Betrachtet man noch einmal Abbildung 4.2, so sieht man, dass sich alle Bauteile von der Mitte wegbewegen. Im Kontext einer 3D-Software kann man also sagen, dass sich die Teile vom Ursprung des Objektes entfernen. Mathematisch bedeutet das, dass der Ortsvektor, also der Vektor, der die Position des Objektes ausgehend vom Ursprung beschreibt, vervielfacht wird. Die Idee ist also, ein Skript zu schreiben, dass durch eine Multiplikation des Ortsvektors jedes Teils die neue Position ermittelt und das Teil auf diese Position verschiebt. An dieser Stelle kann dann ein Keyframe gesetzt werden, um so die Animation zu erzeugen. Die Implementierung dieses Skriptes erfolgt in *3DS Max* mit der softwareeigenen Programmiersprache *MaxScript*. Der folgende Codeausschnitt zeigt, wie diese Idee technisch umgesetzt wurde:

```
1  exploderPos = selection.center
2          exploderPos = [exploderPos.x,exploderPos.y,0]
3          global sel = selection as array
4          for obj in sel do
5          (
6                  vec = obj.pos - exploderPos
7                  vec.x = vec.x * spn_x.value
8                  vec.y = vec.y * spn_y.value
9                  vec.z = vec.z * spn_z.value
10                 obj.pos = exploderPos + vec
11         )
```

In den Zeilen 1 und 2 wird der Mittelpunkt der Explosion festgelegt. In diesem Fall liegt er in der X-Y-Ebene auf Höhe des Mittelpunktes aller selektierten Objekte. Anschließend wird über alle selektierten Objekte iteriert (Zeile 3 und 4). Dabei wird für jedes Objekt der Ortsvektor bestimmt (Zeile 6), die neue Position durch Multiplizieren der X-, Y- und Z-Werte des Vektors mit vom Nutzer einstellbaren Werten errechnet(Zeilen 7-9) und zuletzt das Objekt an der errechneten Stelle platziert(Zeile 10). Abbildung 4.17 zeigt die Position der Objekte vor und nach Ausführung des Skriptes und damit die Keyframes für die Animation.

Die ursprünglich angedachte Gruppierung der Objekte für die Explosionszeichnung konnte mit diesem Skript jedoch nicht realisiert werden. Um die Explosionszeichnung mit Gruppen durchzuführen, müsste das Skript um eine Kollisionsüberprüfung ergänzt werden, um mögliche Überschneidungen zu vermeiden. Zusätzlich müsste ein Weg gefunden werden, die Verschiebung nicht auf einzelne Objekte, sondern auf definierte Gruppen anzuwenden. Dabei liegt das Problem, in der hierarchischen Struktur einer Szene. So sind verschiedene Baugruppen oft ineinander verschachtelt, was dazu führt, das auf manche Gruppen die

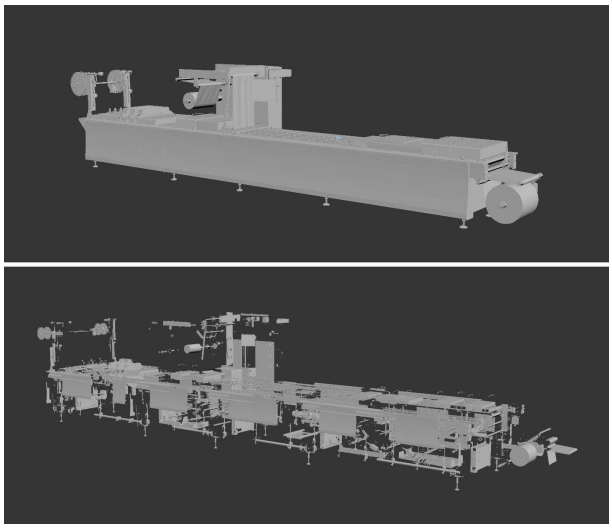

Abbildung 4.17: Screenshots der Maschine in *3DS Max* vor (oben) und nach (unten) der Ausführung des Skriptes. Gleichzeitig sind diese beiden Positionen die Keyframes der fertigen Explosionsanimation.

gewünschte Verschiebung mehrfach angewendet wird. Die einzelnen Baugruppen müssten also entweder auf eine Ebene gebracht werden, oder das Skript müsste eine doppelte Anwendung der Transformation erkennen und verhindern. Da jedoch diese Änderungen kompliziert zu implementieren waren und die Implementierung einer Explosionszeichnung von Einzelteilen die Machbarkeit einer solchen Animation bereits belegt, wurde auf die entsprechenden Änderungen des Skriptes im Rahmen dieser Arbeit verzichtet.

4.4 GUI

Um dem Nutzer die implementierten Animationen und Interaktionen zugänglich zu machen, muss ein Interface aufgebaut werden, über das der Nutzer die Anwendung bedienen kann. Abbildung 4.18 zeigt, wie dieses Menü in der finalen Anwendung aussieht.

Abbildung 4.18: Screenshot des finalen, kompletten User Interfaces der Anwendung

Abbildung 4.19: Screenshot des 3D-Objektes, das als World-Space-Button verwendet wird.

Um dieses Menü aufzubauen, wurde zunächst ein seitlich ausklappbares Menü implementiert. Über dieses Menü können die verschiedenen Interaktionen ausgelöst werden. Für die Animationen der einzelnen Stationen der Maschine wurde ein separater Reiter in diesem Menü angelegt, um zu vermeiden, dass das Menü überfüllt und unübersichtlich wirkt.

Damit der räumlichen Bezug der Animationen zu den Stationen der Maschine deutlich wird, wurden außerdem World-Space-Buttons zum Abspielen der Animationen implementiert. Es handelt sich dabei um 3D-Objekte, die in der Szene platziert und als Buttons genutzt werden. Abbildung 4.19 zeigt einen solchen Button.

Damit die World-Space-Buttons jederzeit der Kamera zugewandt sind, ist jedem dieser Objekte das folgende Skript zugewiesen, um die entsprechende Rotation des Objektes vorzunehmen:

```
1 Vector3 v = Camera.main.transform.position-transform.position;
2 v.x = v.z = 0.0f;
3 transform.LookAt(Camera.main.transform.position - v);
```

Die eigentliche Ausrichtung des Objektes findet durch die Funktion `LookAt(Vector3)` statt. Zeilen 1 und 2 sowie der Vektor v dienen hier lediglich dazu, die Y-Komponente des Arguments der Funktion gleich Null zu setzen, damit das 3D-Objekt sich nur um die Y-Achse fixiert dreht.

Da 3D-Objekten im Gegensatz zu UI-Elementen keine Click-Listener hinzugefügt werden können, musste ein Raycast implementiert werden, um einen Klick auf das Objekt zu erkennen. Dazu wurde folgender Code verwendet:

```
1 //Wenn Cursor über einem klickbaren Objekt liegt...
2 if ((Physics.Raycast(ray, out hit)) && (vpc.isPlaying == false)
    )
3         {
4         //...wird der Mauscursor geändert...
5         MouseOverWSGUI = true;
```

```
6           //..und bei Drücken der linken Maustaste die Position
               des Cursors gespeichert
7           if (Input.GetMouseButtonDown(0))
8           {
9                   mousePos = Input.mousePosition;
10          }
11          //Wird die Maustaste gehen gelassen ohne dass sie
               vorher bewegt wurde (Click oder Drag)? Ist der
               Raycast blockiert?
12          if ((Input.GetMouseButtonUp(0)) && (Input.mousePosition
                == mousePos) && (!guiControl.MouseOverUI()))
13          {
14                  //Auslösen der entpsrechenden Reaktion bei
                       Klick auf ein interaktives Teil
15                  switch (hit.collider.name)
16                  {
17                          case "Object005":
18                                  SwitchMonitor();
19                                  break;
20                          case "HotSpot1":
21                                  EnterObjective(0);
22                                  break;
23                          case "HotSpot2":
24                                  EnterObjective(1);
25                                  break;
26                          case "HotSpot3":
27                                  EnterObjective(2);
28                                  break;
29                          case "HotSpot4":
30                                  EnterObjective(3);
31                                  break;
32                          case "HotSpot5":
33                                  EnterObjective(4);
34                                  break;
35                  }
36          }
37  }
```

Wie zu sehen ist, wird also bei einem Klick der Name des getroffenen Objektes überprüft. Anschließend wird die entsprechende Funktion aufgerufen. `Object005` ist in diesem Fall der Monitor der Maschine, der mit der Funktion `SwitchMonitor()` an- und ausgeschaltet werden kann. Die Objekte `HotSpot 1-5` sind hier die World-Space-Buttons für die verschiedenen Stationen. Die Funktion `EnterObjective(X)` löst das Abspielen des Videos von Station X aus.

Anschließend wurden dem seitlichen Menü der Anwendung entsprechend nummerierte Symbole der World-Space-Buttons hinzugefügt, um eine direkte Zuordnung der Animationen zu erleichtern. Damit ergibt sich das in Abbildung 4.18 gezeigte, vollständige Interface.

Kapitel 5

Ergebnisse

In diesem Kapitel werden die Ergebnisse der Arbeit präsentiert. Im ersten Abschnitt werden dabei die Ergebnisse der Reduktion der CAD-Daten betrachtet. Im zweiten Abschnitt werden die Implementierungen der einzelnen Funktionen an der Verpackungsmaschine ausgewertet. Dabei wird zusammengefasst, wie erfolgreich die Ansätze zur Implementierung der einzelnen Animationen waren. Im letzten Abschnitt werden diese Ergebnisse dann abstrahiert, um Kenntnisse über den allgemeinen Umgang mit komplexen Modellen in Echtzeitanwendungen zu gewinnen.

Da die CAD-Daten – und somit auch das Modell der Verpackungsmaschine – selbst im Verhältnis zu anderen CAD-Daten und Modellen sehr komplex ist, ist davon auszugehen, dass Implementierungen und Vorgehensweisen, die bei diesem Modell erfolgreich sind, auch bei anderen Modellen anwendbar sein werden. Zudem wurden, wie in Abschnitt 4.1.3 erläutert, Animationen und Interaktionen gewählt, die sich auch auf andere Modelle und Anwendungsbereiche übertragen lassen.

5.1 Aufbereitung der CAD-Daten

In dieser Arbeit wurden verschiedene Schritte vorgestellt, die für die Vorbereitung von CAD-Daten für die Verwendung in Echtzeit vorgenommen werden können. Im folgenden Abschnitt werden die Ergebnisse dieser Schritte betrachtet und bewertet.

5.1.1 Komplexität der Modelle

Das größte Problem bei der Arbeit mit CAD-Modellen ist die große Menge an einzelnen Teilen und Polygonen. Durch verschiedene Optimierungen konnten diese stark reduziert werden. Der erste Schritt ist dabei eine gute Tesselierung der Objekte. Das wird deutlich, wenn die Ergebnisse von verschiedenen Tesselierungen hinsichtlich Dateigröße und Polygonanzahl verglichen werden. Es handelt sich bei den verglichenen Daten jeweils um dasselbe Modell der verwendeten Maschine. Die CAD-Daten wurden dabei jeweils im STEP-Format importiert, in verschiedenen Programmen mit verschiedenen Einstellungen tesseliert und zuletzt in das FBX-Format exportiert.

Wird das Modell in 3DS Max mit Standardeinstellungen tesseliert, hat das Objekt insgesamt 166 Millionen Polygone und eine Dateigröße von 4,10 GB. Mit dem Preset für minimale Qualität lässt sich ein Modell mit 7 Millionen Polygonen und einer Dateigröße von 205 MB erzeugen. Verwendet man stattdessen *PiXYZ Studio* für die Tesselierung, wird schon mit Standardeinstellungen ein Modell mit nur 2 Millionen Polygonen erzeugt, dass lediglich 30 MB Speicherplatz belegt. Wählt man ein Preset für geringere Qualität, kann das Modell auf 1 Million Polygone und 20 MB Dateigröße reduziert werden.
Natürlich kann nicht immer die geringste Qualität angestrebt werden. Abhängig vom Anwendungsfall muss für jedes Modell ein Kompromiss zwischen Qualität und Datenmenge gefunden werden. Dennoch zeigen diese Zahlen deutlich, wie wichtig eine effiziente Tesselierung von CAD-Daten ist, wenn die Modelle in Echtzeit verwendet werden sollen.

Auch das Entfernen von nicht benötigten Teilen hat einen maßgeblichen Einfluss auf die Dateigröße und Komplexität des Modells. Wie bereits in Abschnitt 4.2.1 erläutert, konnte durch das Entfernen der innenliegenden Teile die Polygonanzahl sowie die Dateigröße um etwa 50% verringert werden. Wie stark ein Modell durch das Entfernen nicht benötigter Teile vereinfacht werden kann, lässt sich allerdings nicht allgemein sagen. Abhängig von dem verwendeten Modell und dem Verwendungszweck können mehr oder weniger Teile aus dem Modell entfernt werden.

Da das Entfernen der Teile durch Skripts oder geeignete Software weitgehend automatisiert ablaufen kann, ist wenig manuelle Arbeit dafür aufzuwenden. Es ist also eine einfache aber effektive Lösung, um Speicherplatz und Polygone einzusparen.

5.1.2 UV-Mapping und Texturierung

Um die CAD-Daten in Echtzeit mit Texturen beziehungsweise Materialien versehen zu können, mussten bei der Vorbereitung der Daten zunächst UV-Maps angelegt werden. Es zeigte sich, dass die Erstellung der UV-Maps für die Texturierung der Objekte sehr unkompliziert ist, da eine einfache Box-Projektion verwendet werden kann. Diese Projektion kann von den meisten 3D-Softwares automatisch erstellt und zugewiesen werden, weshalb an dieser Stelle kein zusätzlicher manueller Aufwand entsteht.

Schwieriger gestaltet sich dagegen die Erstellung der Lightmaps. Bei ihnen handelt es sich um uniforme Unwraps der einzelnen Teile. Zwar können auch Unwraps automatisch erstellt werden, dieser Vorgang ist jedoch wesentlich komplizierter und damit aufwändiger als die Zuweisung einer Box-Projektion. Das führte dazu, dass die für diese Arbeit verwendete 3D-Software (*3DS Max*) bei jedem Versuch, das Unwrapping der einzelnen Teile des Modells zu automatisieren, abstürzte. Durch die Menge an Einzelteilen in CAD-Daten ist es jedoch in den seltensten Fällen praktikabel, für jedes Teil einzeln einen Unwrap zu erstellen. Daher muss entweder eine geeignetere Software verwendet oder ein anderer Ansatz zum automatisierten Unwrapping gefunden werden.

Die eigentliche Texturierung des Modells geschieht erst beim Import in *Unity*. Dabei werden wie bereits in Abschnitt 4.2.3 erläutert die bereits zugewiesenen Materialien durch in *Uni-*

ty definierte, neue Materialien ersetzt. Abgesehen von der Erstellung dieser Materialien ist dazu keine manuelle Arbeit nötig. Das Ersetzen der Materialien läuft bei jedem Import des Modells automatisch ab.

Die Schwierigkeit liegt also eher darin, die Platzhaltermaterialien zuzuweisen, die später ersetzt werden sollen. Die Verwendung der Materialien, die bereits durch die in den CAD-Daten hinterlegten Farbinformationen automatisch angelegt werden, erweist sich hier als sehr nützlich. Per Skript können für jedes dieser Materialien alle Objekte selektiert werden, die es verwenden und diesen das entsprechende Material zugewiesen werden. So wird das Selektieren von Teilen mit gleichem Material erleichtert, was den manuellen Aufwand verringert. Abhängig davon, wie viele verschiedene Farben in den CAD-Daten verwendet wurden, werden jedoch auch entsprechend viele Materialien beim Import erstellt, weshalb das Ersetzen der Materialien trotz des Skriptes einen gewissen Arbeitsaufwand in Anspruch nehmen kann.

5.2 Implementierung der Funktionen

In diesem Abschnitt werden die Ergebnisse betrachtet, die durch die angewandten Lösungen erreicht werden können. Für jede Funktion wird dabei der Arbeitsaufwand sowie die erreichte Performance betrachtet.

5.2.1 Teilaustausch

Der animierte Austausch von Teilen gestaltet sich insgesamt schwierig. Ein Wechsel durch Fade-Out ist oft nicht umsetzbar, wie das Beispiel dieser Anwendung zeigt. Besonders bei Modellen, bei denen sich viele Teile auf engem Raum befinden, können die durch Transparenz ausgelösten Probleme eine Implementierung eines Fade-Outs verhindern. Aus diesem Grund bleibt in diesen Fällen nur eine Bewegung des Teils aus dem sichtbaren Bereich des Nutzers.

Im Fall der hier behandelten Anwendung konnte die Bewegung des Modellteils erfolgreich implementiert werden und führte auch zu keinen Beeinträchtigungen der Performance der Anwendung. Ein Problem bleibt allerdings, das Objekt aus dem sichtbaren Bereich des Nutzers zu bewegen. Eine Position außerhalb der Bühne ist die einzige Möglichkeit sicherzustellen, dass der Nutzer das Objekt nicht mehr sieht. Dazu muss das Objekt allerdings durch den Boden bewegt werden, was den angestrebten Realismus der Anwendung stört. Aus diesen Gründen wurde ein animierter Austausch der Modellbereiche zwar implementiert, kann in den Optionen der Anwendung jedoch auch deaktiviert werden.

Allgemein ist jedoch zu sagen, dass die Entscheidung, welche Methode genutzt werden sollte, immer vom jeweiligen Anwendungsfall abhängt. Sollen beispielsweise nur wenige Teile ausgeblendet werden, kann möglicherweise auch ein Fade-Out ohne Probleme realisiert werden. In speziellen Fällen kann auch eine Bewegung des Objektes Sinn machen, wenn beispielsweise gezeigt werden soll, wie ein bestimmtes Teil ausgetauscht wird. Es muss also grundsätzlich

betrachtet werden, welchen Zweck der Austausch der Teile erfüllen soll und welche Probleme bei den jeweiligen Methoden auftreten können.

5.2.2 Animation einzelner Arbeitsschritte

Verwendet man Videos anstelle von Animationen in Echtzeit, erhöht sich die Dateimenge abhängig von der Qualität der Videos. Im Beispiel dieser Anwendung wurden 5 Videos in Full HD mit einer Länge von jeweils rund 30 Sekunden verwendet. Diese belegten insgesamt circa 200 MB. Auf die Performance der Anwendung hat diese Methode keine Auswirkungen, da das Video nicht mehr gerendert wird und da der Nutzer die Anwendung nicht bedient, während er das Video betrachtet.

Ein direkter Vergleich von Videos zu Animationen der einzelnen Arbeitsschritte kann in diesem Fall nicht gezogen werden, da die Animationen, wie bereits erläutert, nicht ins FBX-Format exportierbar waren. Generell ist jedoch festzuhalten, dass der Speicherbedarf von Videos und Animationen durch verschiedene Faktoren bedingt ist und die Entscheidung, welcher Ansatz verwendet werden soll, stets vom jeweiligen Anwendungsfall abhängt. Einfach ausgedrückt lässt sich sagen, dass für kurze, simplere Arbeitsschritte Animationen und für komplexere, längere Vorgänge Videos verwendet werden sollten. Im Detail hängt diese Entscheidung jedoch von den in 4.1.3 erläuterten Kriterien und deren Gewichtung im jeweiligen Projekt ab.

5.2.3 Gesamtanimation

Durch die Verwendung von Texturscrolling und das Reduzieren der Animationen auf die von außen sichtbaren Teile konnte der Aufwand für diese Animation sehr gering gehalten werden. Unity bietet einen eingebauten Profiler an, um die Dauer einzelner Aufgaben auf der CPU zu überwachen. Damit ließ sich feststellen, dass lediglich eine Dauer von 0,05 bis 0,07 Millisekunden für die Animation aufgewendet werden musste (siehe Abbildung 5.1).

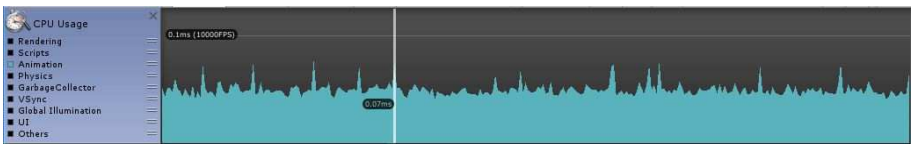

Abbildung 5.1: Die Abbildung zeigt einen Screenshot des Unity-Profilers. Darin ist die Dauer zu sehen, die die CPU des Rechners für Animationen benötigt. Der Screenshot wurde bei laufender Animation aufgenommen.

5.2.4 Ausblenden von Modellbereichen

Das Ausblenden von Modellbereichen stellt für sich kein Problem dar. Die einzige Bedingung ist eine Gruppierung der Teile in die gewünschten Modellbereiche. Da in CAD-Daten zu Konstruktionszwecken Baugruppen angelegt sind, die beim Konvertieren ins FBX-Format

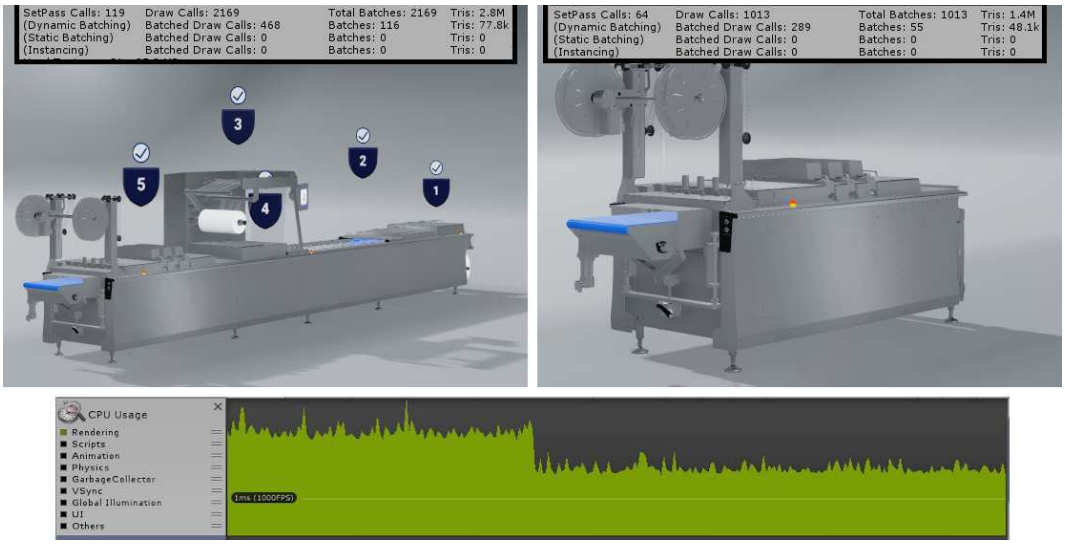

Abbildung 5.2: Die Abbildung zeigt, welche Auswirkungen das Ausblenden von Modellbereichen auf die Performance hat. Links im Bild ist die komplette Maschine eingeblendet, rechts nur der hintere Modellbereich. Oben im Bild sind jeweils die resultierenden Draw Calls, sowie *Tris* aufgelistet. Unterhalb ist zu sehen, dass sich die Renderzeit infolgedessen auf weniger als ein Drittel reduziert.

automatisch zu Helper-Objekten konvertiert werden, ist zur Gruppierung nur noch wenig manuelle Arbeit erforderlich.

In der Anwendung wird das Ausblenden von Modellbereichen durch das Deaktivieren des jeweiligen *MeshRenderers* realisiert. Das bedeutet, dass die ausgeblendeten Objekte nicht mehr gerendert werden, also ausgeblendet sind. Dieser Vorgang hat keine negativen Auswirkungen auf die Performance der Anwendung. Tatsächlich verbessert sich sogar die Reaktionszeit, da weniger Teile gerendert werden müssen.

Wie Abbildung 5.2 zeigt, reduzieren sich beim Betrachten des hinteren Teils der Maschine die Draw Calls von rund 2200 auf circa 1000. Auch die Zeit, die die CPU für das Rendering benötigt, verkürzt sich dabei von 3 Millisekunden auf circa 1,5 Millisekunden.

Möchte man jedoch die übrigen Modellbereiche nicht vollständig ausblenden, sondern transparent darstellen, sodass die Position der Teile zueinander deutlich bleibt, treten Probleme auf. Da es aufgrund des Transparency Fightings nicht möglich ist, alle Einzelteile der Maschine transparent darzustellen, muss ein Ersatzobjekt verwendet werden, dass die ausgeblendeten Teile ersetzt. Das bedeutet, dass für jedes Teil, das separat betrachtet werden kann, ein Ersatzobjekt erstellt werden muss, dem das entsprechende Teil fehlt, um Überlagerungen zu vermeiden. Im hier vorliegenden Fall sind das drei einzelne Ersatzobjekte, die jeweils zwischen 5 und 10 MB Speicher belegen und aus 200.000 bis 300.000 Polygonen bestehen. Bei Verwendung dieser Objekte ist keine relevante Veränderung der Renderzeiten

messbar. Da jeweils nur eins dieser Objekte gleichzeitig sichtbar ist, ist davon auszugehen, dass auch bei komplexeren Projekten die Performance nicht zu einem Problem werden wird. Zu beachten ist jedoch, dass je nach Qualität und Menge der benötigten Ersatzobjekte, der benötigte Speicherplatz der Anwendung variiert.

5.2.5 Austausch von Shadernetzwerken

Wie bereits in 4.3.5 erläutert, wird zum Ändern der Folie mithilfe des `sharedMaterial` Attributs die Albedo-Textur des Materials gewechselt. Wie erwartet werden dabei zur Laufzeit keine neuen Materialinstanzen angelegt. Werden die Materialien dagegen über das `material` Attribut verändert, entstehen 26 neue Materialinstanzen nach dem ersten Wechsel. Anschließend steigt die Materialanzahl nicht weiter an, da die Kopie des Materials weiterhin verändert wird. Da in diesem Anwendungsfall nur 26 Teile betroffen sind, entstehen auch nur 26 neue Materialinstanzen, was noch keinen feststellbaren Einfluss auf die Renderzeiten oder den Speicherbedarf zur Laufzeit der Anwendung hat.

Ist jedoch eine größere Anzahl von Objekten betroffen, wird das Problem deutlich: Zu Testwecken wurde die Möglichkeit implementiert, das Material aller Teile, die zu einem Modellbereich der Maschine gehören, einzufärben. In diesem Fall ist mit dem Unity Profiler festzustellen, dass sich die Anzahl der Materialien von 130 auf über 4.000 erhöht. Der benötigte Speicher für Materialien steigt dabei von 240 KB auf 8,5 MB an (siehe Abbildung 5.3).

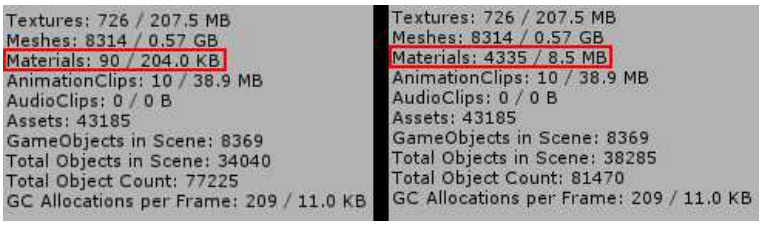

Abbildung 5.3: Screenshot der Ausgabe des Unity Profilers vor der Erzeugung der Materialinstanzen (links) und danach (rechts).

Um die Erstellung dieser Materialinstanzen zu verhindern, kann in diesem Beispiel jedoch nicht das `sharedMaterial` Attribut verwendet werden, da es noch weitere Objekte in der Szene gibt, die dasselbe Material verwenden und nicht verändert werden sollen. In diesem Fall bleibt also nur die Möglichkeit, ein neues, vorher erstelltes Material zuzuweisen. Auf diese Weise entstehen keine zusätzlichen Instanzmaterialien und nur die Materialien der gewünschten Objekte werden verändert.

Allgemein lässt sich also sagen, dass das `material` Attribut genutzt werden sollte, wenn nur wenige Objekte angepasst werden sollen oder wenn das Ausgangsmaterial nicht verändert werden soll, weil beispielsweise andere Objekte dieses Material ebenfalls nutzen. Sollen alle Objekte mit dem betreffenden Material von der Änderung betroffen sein, kann das `sharedMaterial` Attribut genutzt werden. Dabei ist jedoch zu beachten, dass das Material dauerhaft geändert wird. Sind also viele Objekte von einer Materialänderung betroffen und das Material darf nicht geändert werden, so können die benötigten Materialien angelegt wer-

den und den gewünschten Objekten zugewiesen werden. Das Erstellen des Materials kann dabei im Vorhinein oder zur Laufzeit per Skript geschehen.

5.2.6 Explosionszeichnung

Die größte Schwierigkeit beim Implementieren der Explosionsanimation ist die Erstellung der Animation. Durch die Menge an Teilen ist es nicht möglich, Keyframes manuell zu setzen, weshalb Möglichkeiten zur Automatisierung gefunden werden müssen. Das in 4.3.6 vorgestellte Skript löst nur Teile dieses Problems. Es ermöglicht zwar die Erstellung einer Explosionsanimation mit nur wenigen Klicks, hat aber auch noch einige Schwächen. So werden beispielsweise keine Überschneidungen zwischen den Objekten bei der Explosion erkannt, was eine aufwendige manuelle Korrektur nötig macht. Außerdem wird die Ursprungsposition der Objekte nicht automatisch gespeichert. Das führt dazu, dass die Ausführung des Skriptes nicht rückgängig gemacht werden kann, wenn die Ursprungsposition nicht als Keyframe gespeichert wurde.

Innerhalb Unity unterscheidet sich die Implementierung der Animation nicht von anderen Animationen. Selbst zur Laufzeit hat die Animation kaum Auswirkungen auf die Performance: Im Profiler ist festzustellen, dass die Animation weniger als eine Millisekunde Bearbeitungszeit benötigt. Ein Problem ist jedoch, dass wie in 4.2.1 erwähnt, innenliegende Teile der Maschine eingeblendet werden müssen, wenn die Explosionsanimation gestartet wird. Das Einblenden dieser Teile erhöht die Draw Calls von rund 2.000 auf knapp 9.000. Dies führt besonders auf Mobilgeräten oder Rechnern ohne leistungsstarke Hardware schnell zu Einbrüchen der Framerate.

Da auch eine Gruppierung der Objekte bei der Explosion nicht umgesetzt werden konnte, ist die Explosionszeichnung so nicht sinnvoll. Die Explosionsansicht ist unübersichtlich und führt zu starken Einbrüchen der Performance. Allgemein ist daher eher zu empfehlen, Explosionszeichnungen nur für einzelne Baugruppen zu implementieren. Diese können dem Nutzer dann aus der isolierten Ansicht der einzelnen Modellbereiche zugänglich gemacht werden. Auf diese Weise wird verhindert, dass sich zu viele Teile gleichzeitig in der Szene befinden, was sowohl der Übersichtlichkeit als auch der Performance zuträglich ist.

5.3 Gewonnene Erkenntisse

Da sich diese Arbeit nicht nur mit der Erstellung und Analyse des Prototypen beschäftigt, muss betrachtet werden, welche allgemeinen Erkenntnisse sich für die Arbeit mit komplexen Modellen aus den Ergebnissen dieser Arbeit ergeben. Im Wesentlichen sind es drei Punkte, die bei der Verwendung von komplexen Modellen beachtet werden müssen. Zum Einen muss die Komplexität der Daten schon vor dem Import in eine Echtzeitplattform so weit wie möglich reduziert werden. Zum Anderen müssen einige Arbeitsschritte, die üblicherweise manuell erfolgen können, bei komplexen Modellen aufgrund der hohen Anzahl an Einzelteilen automatisiert erfolgen. Außerdem muss im Allgemeinen die Komplexität des Modells

beim Entwickeln der Anwendung berücksichtigt werden, da sie einige spezielle Überlegungen erforderlich macht. Im Folgenden wird auf diese Besonderheiten genauer eingegangen.

5.3.1 Vereinfachung der Daten

Wie bereits in 4.2.1 aufgeführt, konnten die CAD-Daten für diese Anwendung vor dem Import um mehr als die Hälfte reduziert werden. Trotzdem sind in der Entwicklung der Anwendung mehrere Probleme durch die immer noch sehr hohe Komplexität aufgetreten. Diese Tatsache zeigt, dass es unbedingt notwendig ist, CAD-Modelle vor ihrer Verwendung in Echtzeit weitestgehend zu vereinfachen. Dabei sind zwei Faktoren aus unterschiedlichen Gründen von Bedeutung.

Der erste wichtige Faktor ist die Anzahl der Polygone. Desto mehr Polygone in Echtzeit gerendert werden müssen, desto höher sind die Anforderungen der betreffenden Anwendung an CPU und GPU. Für ein *iPad* der vierten Generation beispielsweise empfiehlt *Unity* eine maximale Polygonanzahl von 30.000 bis 40.000[Bla11]. Natürlich können moderne leistungsstarke Rechner mit wesentlich größeren Polygonmengen problemlos umgehen. Trotzdem ist es auch bei Anwendungen, die auf leistungsstarke Hardware abzielen zu empfehlen, die Menge der Polygone in einem Modell so gering wie möglich zu halten, da die Dateigröße eines Modells maßgeblich durch die Anzahl der Polygone bestimmt wird.

Der zweite wichtige Faktor ist die Anzahl der einzelnen Teile. Auch wenn eine hohe Anzahl einzelner Teile durch Verfahren wie Batching nicht zwangsläufig die Performance beeinträchtigt, ist es aus anderen Gründen sinnvoll, möglichst wenige einzelne Teile in einem Modell zu verwenden. Viele Funktionen, die sowohl für die Vorbereitung, als auch zur Laufzeit von Echtzeitanwendungen auf ein Modell angewandt werden, müssen für jedes Teil des Modells einzeln durchgeführt werden. Dazu zählt beispielsweise das Zuweisen von Materialien oder das Erstellen von Lightmaps in der Vorbereitung sowie das Generieren von Shadow Maps oder die Kollisionsprüfung eines Raycasts zur Laufzeit. Mehr Einzelteile in einem Modell führen also auch zu mehr Arbeits- oder Rechenaufwand bei der Vorbereitung und Verwendung des Modells.

5.3.2 Automatisierung

Die hohe Anzahl an Teilen in CAD-Modellen erfordert außerdem eine weitere Besonderheit. Wie bereits erwähnt, gibt es bei der Vorbereitung und Verwendung von Modellen in Echtzeitanwendungen einige Arbeitsschritte, die für jedes Teil des Modells durchgeführt werden müssen. So muss beispielsweise für jedes Teil eines Modells ein UV-Unwrap angelegt werden, um eine Lightmap zu erstellen. Im Beispiel des Modells der Maschine besteht das Modell jedoch auch nach den Optimierungen noch aus rund 1800 Einzelteilen. Wenn derart komplexe Modelle verwendet werden, würde es einen enormen Aufwand bedeuten, jedes Teil manuell zu bearbeiten.

Aus diesem Grund müssen Möglichkeiten gefunden werden, solche Vorgänge zu automatisieren. Dazu können zum Beispiel Skripte entwickelt werden, wie bei der Erstellung der

Explosionsanimation in dieser Arbeit. Auch Softwares wie *PiXYZ*, die für die Bearbeitung komplexer Modelle ausgelegt sind, bieten aus diesem Grund zahlreiche Möglichkeiten zur Automatisierung verschiedener Aufgaben, wie beispielsweise das Ausblenden innenliegender Teile oder das automatische Unwrapping der einzelnen Teile.

5.3.3 Berücksichtigung der Komplexität

Die hohe Polygonanzahl und vor Allem die große Menge an einzelnen Teilen erfordern außerdem einige zusätzliche Überlegungen beim Entwickeln einer Echtzeitanwendung. So gibt es einige Funktionen und Möglichkeiten, die erst bei komplexen Modellen zu einem Problem werden. Ein Beispiel hierfür ist die Funktion `camera.main`, die *Unity* mit seinen Standardbibliotheken zur Verfügung stellt. Dieser scheinbar einfache Befehl liefert als Rückgabewert die Hauptkamera der Szene. Im Hintergrund wird bei Verwendung dieses Befehls jedoch über die komplette Szene iteriert und das Tag jedes Objekts betrachtet.

Bei Szenen, in denen sich Tausende Einzelteile befinden, kann es daher schnell zu spürbaren Performanceeinbrüchen kommen, wenn eine solche Funktion oft aufgerufen wird. Es ist also wichtig, auch bei der Erstellung von Skripten und insbesondere beim Verwenden von Bibliotheken darauf zu achten, wie genau eine Funktion arbeitet und ob bei komplexeren Modellen besondere Probleme auftreten können.

Kapitel 6

Zusammenfassung und Ausblick

In diesem Kapitel wird die Arbeit und ihre Ergebnisse noch einmal kurz zusammengefasst. Außerdem werden hier auch Ideen zu weiterführenden Arbeiten vorgestellt.

6.1 Zusammenfassung

Echtzeitanwendungen nehmen mittlerweile einen immer größer werdenden Stellenwert ein. Aufgrund der hohen Flexibilität und möglichen Interaktivität werden sie auch im industriellen Kontext immer interessanter. Dort können sie beispielsweise als Showroom, Konfigurator oder Lehrmittel eingesetzt werden.

Problematisch ist dabei, dass im industriellen Bereich oft Konstruktionsdaten verwendet werden, um Modelle für 3D-Visualisierungen zu erzeugen. Diese Modelle sind jedoch sehr komplex und damit für die Verwendung in Echtzeit schlecht geeignet. Daher wird in dieser Arbeit untersucht, ob und wie auch komplexere Modelle durch Aufbereitung und Optimierung in Echtzeitanwendungen verwendet werden können.

Zu diesem Zweck wird ein Prototyp einer solchen Echtzeitanwendung erstellt. Als Modell dient eine Verpackungsmaschine, die in Konstruktionsdaten vorliegt. Dieses Modell soll in einer Echtzeitanwendung verwendet und mit verschiedenen Animationen und Interaktionen, deren Einsatz im industriellen Kontext üblich oder sinnvoll ist, versehen werden.

Diese Arbeit beschäftigt sich dabei mit der Konzeption der Anwendung, der Aufbereitung der Daten, sowie der eigentlichen Implementierung der Anwendung. Dabei werden Probleme, die beim Entwicklungsprozess auftreten untersucht und mögliche Lösungen vorgestellt. Auf diese Weise kann festgestellt werden, welche Besonderheiten sich bei der Arbeit mit komplexen Modellen in Echtzeit ergeben.

So wurde festgestellt, dass bereits die Tesselierung bei der Überführung von Konstruktionsdaten in Polygonmodelle ein entscheidender Schritt ist, da an dieser Stelle die Komplexität des entstehenden Modells maßgeblich beeinflusst werden kann. Bei dem Modell der Verpackungsmaschine konnte in diesem Schritt 70% der Polygone und damit 60% der Da-

teigröße eingespart werden.

Als weiterer entscheidender Punkt stellte sich die Automatisierung erforderlicher Aufgaben heraus. Bei der Verwendung von Modellen für Echtzeitanwendungen müssen mehrere Arbeitsschritte durchgeführt werden. Dabei geht es sowohl um die Vorbereitung des Modells, wie beispielsweise bei dem Zuweisen von UV-Maps und Materialien, als auch um die eigentliche Implementierung der Anwendung, zum Beispiel beim Erstellen von benötigten Animationen. Durch die hohe Anzahl an einzelnen Teilen in CAD-Modellen benötigen einige dieser Arbeitsschritte oft einen wesentlich erhöhten Arbeitsaufwand. Es müssen also Möglichkeiten gefunden werden, diese Aufgaben zu automatisieren, um einen immensen manuellen Aufwand zu vermeiden.

So konnte in dieser Arbeit beispielsweise die Erstellung der Animation einer Explosionszeichnung oder auch die Sortierung einzelner Bauteile automatisiert werden.

Weitere Besonderheiten ergeben sich bei der Implementierung und Verwendung von verschiedenen Funktionen. Hier treten oft Probleme auf, die durch die hohe Anzahl an Polygonen und Einzelteilen ausgelöst werden. Als Beispiel hierfür ist das Bearbeiten von Materialien zur Laufzeit zu nennen, da hierbei für jedes betroffene Objekt eine neue Materialinstanz angelegt wird, wodurch mehr Speicherplatz belegt wird und die Anzahl der Draw Calls steigt. Ein weiteres Beispiel ist die in Abschnitt 5.3.3 vorgestellte Funktion `camera.main`, die über alle Objekte einer Szene iteriert.

Diese Beispiele zeigen, dass manche Faktoren, die bei simpleren, für Echtzeit optimierten Modellen vernachlässigbar sind, bei komplexen Modellen zu Problemen führen können. Bei der Verwendung von komplexen Modellen sollte also grundsätzlich noch stärker darauf geachtet werden, möglichst effiziente Funktionen zu implementieren beziehungsweise zu verwenden.

6.2 Ausblick

In diesem Abschnitt werden Ideen für zukünftige Arbeiten vorgestellt. Dabei handelt es sich zum Einen um Problematiken, die im Laufe dieser Arbeit aufgetreten sind und nicht oder nicht vollständig gelöst werden konnten. Zum Anderen handelt es sich um Fragen, die mit dem Thema dieser Arbeit verwandt sind oder darauf aufbauen

Eine Problematik, die im Zuge dieser Arbeit nicht vollständig untersucht werden konnte, ist die Umwandlung von CAD-Daten in echtzeitfähige Meshes. Zwar konnten die Konstruktionsdaten weit genug vereinfacht werden, um in Echtzeit verwendet zu werden, der Umwandlungsprozess ist dabei jedoch noch optimierbar. Um die Entwicklung umfangreicherer Echtzeitanwendungen auf Basis von Konstruktionsdaten zu ermöglichen beziehungsweise zu erleichtern, wäre es sinnvoll, eine Arbeit durchzuführen, die sich mit der optimalen Konvertierung von CAD-Daten in für Echtzeit optimierte Polygonmodelle beschäftigt.

Ein besonders für die Zukunft interessantes Thema, das auf dieser Arbeit aufbaut, ist die Verwendung von komplexen Modellen in Augmented-, Virtual- oder Mixed-Reality Anwendungen. Diese Anwendungen sind ebenfalls Echtzeitanwendungen, in denen der Nutzer in einer virtuellen (VR) oder augmentierten (AR) Realität mit den Objekten interagiert. Diese Funktionalitäten bringen weitere Besonderheiten im Bezug auf die Modelle, sowie die Implementierung der Anwendung mit sich. Da Anwendungen dieser Art auch für industrielle Zwecke durchaus interessant sein können, wäre es interessant zu untersuchen, ob und wie auch für diese Anwendungen komplexe Modelle verwendet werden können.

Ein weiteres Thema, das sich mit der Arbeit mit komplexen Modellen in Echtzeit beschäftigt, ist die Verwendung von Real-Time Ray-Tracing. Wie in Abschnitt 2.2.1 dieser Arbeit erläutert, werden in Echtzeitanwendungen Rasterungsverfahren genutzt, um Bilder zu rendern. Seit 2018 gibt es jedoch die Möglichkeit, auch in Echtzeit Bilder mit Ray-Tracing zu rendern. Auf diese Weise können wesentlich realistischere Bilder erzeugt werden. Bisher gibt es jedoch nur wenige spezielle Grafikkarten, die für dieses Verfahren geeignet sind, weswegen Real-Time Ray-Tracing noch nicht weit verbreitet ist. In Zukunft ist allerdings damit zu rechnen, dass mehr geeignete Hardware eingeführt wird und sich Ray-Tracing auch im Echtzeitbereich durchsetzt. Daher wäre es interessant zu untersuchen, welche Veränderungen die Verwendung eines neuen Renderingverfahrens im Bezug auf komplexe Modelle in Echtzeitanwendungen nach sich zieht.

Glossar

Begriff	Beschreibung	Seiten
Batching	Batching bezeichnet das Zusammenfassen mehrerer Draw Calls zu einer einzigen Anweisung. Auf diese Weise kann der Overhead auf der CPU, der durch große Mengen an Draw Calls entsteht, verringert werden. [Tec19]	39, 52
Draw Call	Als Draw Call bezeichnet man eine Anweisung zum Rendern. Diese Anweisung wird von der CPU erstellt und an die GPU übermittelt, die dann das eigentliche Rendering übernimmt.	5, 28, 39, 49, 51
Light Probes	Light Probes werden eingesetzt, wenn dynamische Objekte in Szenen mit vorberechnetem Licht genutzt werden sollen. Es handelt sich dabei um Punkte, die son in der Szene platziert werden, dass sie ein Volumen ergeben. In diesen Punkten werden dann die Lichtinformationen an dieser Stelle gespeichert. Bewegt sich nun ein Objekt durch das von den Light Probes definierte Volumen, so kann die korrekte Beleuchtung dieses Objektes aus den umliegenden Light Probes berechnet werden. [Uni17]	12
Quad	Als Quad wird im Kontext von Polygonmodellierung ein vierseitiges Polygone, also Viereck, bezeichnet.	29
Raycast	Speziell im Bezug auf Game-Engines ist ein Raycast ein Strahl mit Startpunkt und Richtung, der durch die Szene „geschossen“ wird. Dabei wird in der Regel auf Kollisionen mit Objekten in der Szene geprüft, um bestimmte Ereignisse auszulösen.	29, 30, 42, 52

Begriff	Beschreibung	Seiten
Shader	„Ein Shader ist ein benutzerdefiniertes Programm, das auf einer bestimmten Stufe eines Grafikprozessors ausgeführt werden kann. Ihr Zweck ist es, eine der programmierbaren Stufen der Rendering-Pipeline auszuführen.“ [Ope19]	11, 23, 26, 34, 39
T-Junction	Als T-Junction wird in der 3D-Modellierungen eine Stelle bezeichnet, an der zwei Polygone entlang der Kante eines dritten Polygons aufeinander treffen. Eine so entstehende „T-Verbindung“ kann zu Fehlern beim Rendering des Objektes führen.	19
Tri	Ein Tri (oder *Triangle*) bezeichnet ein dreiseitiges, dreieckiges Polygon. Mit drei Vertices und Edges stellt es die minimalste Möglichkeit dar, eine Polygonfläche aufzuspannen.	29
Voxel	Voxel ist die Kurzbezeichnung für Volumenpixel. So wie ein Pixel ein einzelner Bildpunkt in einem zweidimensionalen Bild ist, so ist ein Voxel ein einzelnes Bildelement eines dreidimensionalen Elements und besitzt demzufolge neben einer Höhe und einer Breite auch eine Tiefe. [.2009])	38
WebGL	„WebGL ist ein plattformübergreifender, gebührenfreier Webstandard für eine 3D-Grafik-API auf niedriger Ebene, die auf OpenGL ES basiert und über das HTML5-Canvas-Element für ECMAScript verfügbar gemacht wird.“ [Khr11]	3

Abkürzungen

Begriff	Beschreibung
ATF	Autodesk Translation Framework
CAAD	Computer Aided Architectural Design
CAD	Computer Aided Design
DCC	Digital Content Creation
FPS	Frames Per Second
GUI	Graphical User Interface
NURBS	Non-Uniform Rational B-Splines
OSG	Open Scene Graph
PBR	Physically Based Rendering
VRML	Virtual Reality Modeling Language

Literaturverzeichnis

[.2009] *Voxel (volume pixel) :: Volumenpixel :: ITWissen.info*. https://www.itwissen.info/Voxel-volume-pixel-Volumenpixel.html. Version: 2009

[Ber99] Berta, J.: Integrating VR and CAD. In: *IEEE Computer Graphics and Applications* 19 (1999), Nr. 5, S. 14–19. http://dx.doi.org/10.1109/38.788793. – DOI 10.1109/38.788793. – ISSN 02721716

[Bla11] Blackman, Sue: *Beginning 3D Game Development with Unity*. Berkeley, CA : Apress, 2011. http://dx.doi.org/10.1007/978-1-4302-3423-4. http://dx.doi.org/10.1007/978-1-4302-3423-4. – ISBN 978–1–4302–3422–7

[Boe13] Boeykens, Stefan: *Unity for Architectural Visualization*. Birmingham : Packt Publishing, 2013 (Community experience distilled). http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10772102. – ISBN 9781783559060

[Cat] Catmull, Edwin ; National Technical Information Service (Hrsg.): *A Subdivision Algorithm for Computer Display of Curved Surfaces*. https://apps.dtic.mil/dtic/tr/fulltext/u2/a004968.pdf

[CRS+] Corseuil, Eduardo T. L. ; Raposo, Alberto B. ; Silva, Romano J. M. da ; Pinto, Marcio H. G. ; Wagner, Gustavo N. ; Gattass, Marcelo: *ENVIRON - Visualization of CAD Models In a Virtual Reality Environment*

[Del04] *DGL Wiki: Lightmaps*. https://wiki.delphigl.com/index.php/Lightmaps. Version: 2004

[Deu05] Deutsches Institut für Normung: *ISO/IEC 2382:2015-05: Informationstechnologie - Vokabularien*. https://www.beuth.de/de/norm/iso-iec-2382/235389372. Version: 2015-05

[Epi] *Unreal 4 Documentation: Using Datasmith with CAD File Formats*. https://docs.unrealengine.com/en-US/Studio/Datasmith/SoftwareInteropGuides/CAD/index.html

[HGH+00] Hirschberg, Urs (Hrsg.) ; Gramazio, F. (Hrsg.) ; Höger, K. (Hrsg.) ; Legendre, Liaropoulos (Hrsg.) ; Milano, M. (Hrsg.) ; Stöger, B. (Hrsg.): *EventSpaces. a Multi-Author Game and Design Environment*. Bauhaus-Universitat Weimar, 2000 (eCAADe: Conferences). – ISBN 0–9523687–6–5

[IS08] Indraprastha, Aswin ; Shinozaki, Michihiko: Constructing virtual urban environment using game technology. In: *26th eCAADe Conference: Architecture in Computro Antwerpen*, 2008, S. 359–366

[Khr11] *WebGL - OpenGL ES for the Web*. https://www.khronos.org/webgl/. Version: 2011

[LSR+16] Lorenz, Mario ; Spranger, Michael ; Riedel, Tino ; Pürzel, Franziska ; Wittstock, Volker ; Klimant, Philipp: CAD to VR – A Methodology for the Automated Conversion of Kinematic CAD Models to Virtual Reality. In: *Procedia CIRP* 41 (2016), 358–363. http://dx.doi.org/10.1016/j.procir.2015.12.115. – DOI 10.1016/j.procir.2015.12.115. – ISSN 2212–8271

[MAHTF07] Manrique, Juan D. ; Al-Hussein, Mohamed ; Telyas, Avi ; Funston, Geoff: Constructing a Complex Precast Tilt-Up-Panel Structure Utilizing an Optimization Model, 3D CAD, and Animation. In: *Journal of Construction Engineering and Management* 133 (2007), Nr. 3, S. 199–207. http://dx.doi.org/10.1061/(ASCE)0733-9364(2007)133:3(199). – DOI 10.1061/(ASCE)0733–9364(2007)133:3(199). – ISSN 0733–9364

[May14] Mayer, Richard E. (Hrsg.): *The Cambridge handbook of multimedia learning*. Second edition. New York : Cambridge University Press, 2014 (Cambridge handbooks in psychology). http://dx.doi.org/10.1017/CBO9781139547369. http://dx.doi.org/10.1017/CBO9781139547369. – ISBN 9781107035201

[MK10] Marr, Ann C. ; Kaiser, Ronald: *BIT online Innovativ*. Bd. 28: *Serious Games für die Informations- und Wissensvermittlung: Bibliotheken auf neuen Wegen: Teilw. zugl.: Stuttgart, Hochsch. der Medien, Masterarbeit, 2009 u.d.T.: Marr, Ann Christine: Einsatzbereiche und Potential von Serious Games*. Wiesbaden : Dinges & Frick, 2010. – ISBN 978–3–934997–31–8

[MKFH96] McKinney, Kathleen ; Kim, Jennifer ; Fischer, Martin ; Howard, Craig: Interactive 4d-cad. In: *Proceedings of the third Congress on Computing in Civil Engineering*, 1996, S. 17–19

[Moo08] Moorapun, CHUMPORN: Methods for validating the use of virtual simulation generated from a computer game for environment and behaviour research. In: *CAADRIA 2008 [Proceedings of the 13th International Conference on Computer Aided Architectural Design Research in Asia] Chiang Mai (Thailand)*, 2008, S. 642–646

[Ope19] OpenGL Wiki contributors ; OpenGL Wiki (Hrsg.): *Shader - OpenGL Wiki*. http://www.khronos.org/opengl/wiki_opengl/index.php?title=Shader&oldid=14593. Version: 09.10.2019

[PMT03] Paillot, D. ; Merienne, F. ; Thivent, S.: *CAD/CAE visualization in virtual environment for automotive industry*. http://dx.doi.org/10.1145/769953.769995. Version: 2003

[RW15] Reiners, Torsten ; Wood, Lincoln C.: *Gamification in Education and Business*. Cham : Springer International Publishing, 2015. http://dx.doi.org/10.1007/978-3-319-10208-5. http://dx.doi.org/10.1007/978-3-319-10208-5. – ISBN 978–3–319–10207–8

[SBMH04] Squire, Kurt ; Barnett, Mike ; M Grant, Jamillah ; Higginbotham, Thomas: Electromagnetism supercharged!: Learning physics with digital simulation games. In: *Proceedings of the 6th International Conference on Learning Sciences* (2004)

[Tec19] Technologies, Unity: *Unity - Manual: Draw call batching*. https://docs.unity3d.com/Manual/DrawCallBatching.html. Version: 03.10.2019

[The06] *OpenGL Wiki: Transparency Sorting*. https://www.khronos.org/opengl/wiki_opengl/index.php?title=Transparency_Sorting&action=history. Version: 2006

[Uni17] *Light Probes - Unity*. https://unity3d.com/de/learn/tutorials/topics/graphics/light-probes. Version: 2017

[Wal05] Wald, Ingo: Realtime ray tracing and interactive global illumination. In: Wagner, Dorothea (Hrsg.): *Ausgezeichnete Informatikdissertationen 2004*. Bonn : Gesellschaft für Informatik, 2005, S. 173–182

[Web17] Webster, Nicky L.: High poly to low poly workflows for real-time rendering. In: *Journal of visual communication in medicine* 40 (2017), Nr. 1, S. 40–47. http://dx.doi.org/10.1080/17453054.2017.1313682. – DOI 10.1080/17453054.2017.1313682

[Whi05] Whitted, Turner: An Improved Illumination Model for Shaded Display. In: *ACM SIGGRAPH 2005 Courses*. New York, NY, USA : ACM, 2005 (SIGGRAPH '05)

[Wil78] Williams, Lance: Casting Curved Shadows on Curved Surfaces. In: *ACM SIGGRAPH Computer Graphics* 12 (1978), Nr. 3, 270–274. http://dx.doi.org/10.1145/965139.807402. – DOI 10.1145/965139.807402. – ISSN 00978930

[WKM02] Weiss, Renée E. ; Knowlton, Dave S. ; Morrison, Gary R.: Principles for using animation in computer-based instruction: theoretical heuristics for effective design. In: *Computers in Human Behavior* 18 (2002), Nr. 4, S.

465–477. http://dx.doi.org/10.1016/S0747-5632(01)00049-8. – DOI 10.1016/S0747–5632(01)00049–8. – ISSN 07475632

[YCG11] Yan, Wei ; Culp, Charles ; Graf, Robert: Integrating BIM and gaming for real-time interactive architectural visualization. In: *Automation in Construction* 20 (2011), Nr. 4, S. 446–458. http://dx.doi.org/10.1016/j.autcon.2010.11.013. – DOI 10.1016/j.autcon.2010.11.013. – ISSN 0926–5805