HOCHSCHULE
MITTWEIDA
University of Applied Sciences

Fakultät Angewandte Computer- und Biowissenschaften

Professur Medieninformatik

Bachelorarbeit

Umsetzung einer interaktiven, performanten Darstellung der historischen Entwicklung des Richard-Stücklen Baus

Nicole Tauber

Mittweida, den 3. April 2020

Erstprüfer: Prof. Dr. rer. nat. Marc Ritter
Zweitprüfer: Manuel Heinzig, M.Sc.

Tauber, Nicole
Umsetzung einer interaktiven, performanten Darstellung der historischen Entwicklung des Richard-Stücklen Baus
Bachelorarbeit, Fakultät Angewandte Computer- und Biowissenschaften
Hochschule Mittweida — University of Applied Sciences, April 2020

Abstract

Diese Arbeit thematisiert die digitale 3D-Modellierung des zur Hochschule Mittweida gehörenden Richard-Stücklen Baus. Hauptaugenmerk wird dabei auf die historische Entwicklung der veränderten Außenfassade und die einhergehende Frage, ob und inwieweit das das Gebäude umschließende Gitter performant umgesetzt werden kann, gelegt. Die Untersuchung der Frage erfolgt durch die Entwicklung eines beispielhaften Anwendungsfalls in Form einer interaktiven Software. Die Evaluation durch Performanztests legt dar, dass mit dieser Arbeit ein Weg gefunden wurde, den Richard-Stücklen Bau und sein Gitter durch eine eigens entwickelte Anwendung performant darzustellen.

Name: Tauber, Nicole
Studiengang: Medieninformatik und interaktives Entertainment
Seminargruppe: MI16w1-B
English Title: Implementation of an interactive, performant visualization of the historical development of the Richard-Stücklen Bau

Inhaltsverzeichnis

1 Einführung und Motivation

Beim *Richard-Stücklen Bau* handelt es sich um ein Gebäude der Hochschule Mittweida, das zurzeit von der Fakultät Computer- und Biowissenschaften genutzt wird. Schon seit langer Zeit existiert der Wunsch, einige Hochschulgebäude als 3D-Modelle zu erstellen, um sie in diversen Projekten, wie zum Beispiel von Studierenden entwickelten Spielen oder anderen interaktiven Anwendungen, einbinden zu können. Bisher gewagte Umsetzungsversuche scheiterten jedoch. Grund dafür ist vor allem die markante Außenfassade des Gebäudes, die aus einem goldenen Gitter mit großen Löchern besteht, wie in Abbildung 1.1 zu sehen. Das Gitter wurde während des Umbaus im Jahr 2010 hinzugefügt, dadurch konnten beispielsweise ein Wärmedämmsystem und ein verstecktes Notausgangssystem verwirklicht werden. Zuvor hatte das in der DDR als Schule genutzte Gebäude eine einfache weiße Fassade mit gelben Streifen, wie in Abbildung 1.2 zu sehen. Die Probleme bei der Umsetzung des neuen Gebäudes mit Gitter als 3D-Modell betreffen vor allem die Performanz. Um eine für den Nutzer angenehme Anwendung zu erhalten, darf diese nicht ständig ruckeln. Einer von vielen Gründen derartiger Performanzprobleme ist die Polygonanzahl der implementierten 3D-Modelle. Repetitive Muster, wie das Gitter, besitzen häufig eine recht große Polygonanzahl, was die Geschwindigkeit von Anwendungen reduziert.

Eine Gruppe aus Studenten und Dozenten arbeitet derzeit am Projekt *Historisches Mittweida*. Unter den Modulen *Digitalisierung spezifischer Lehrinhalte* und *Internationalisierung spezifischer Lehrinhalte* des Masterstudiengangs *Medieninformatik und interaktives Entertainment* wurde sich der Aufgabe angenommen die Geschichte der Hochschule erleb- und spielbar zu machen. Teil des Spiels, welches in der Gründungszeit der Hochschule spielt, ist es, für den Hochschulbetrieb relevante Gebäude zu bauen. Diese sind zum Teil in der Ego-Perspektive betret- und erkundbar. Für die Umsetzung werden also 3D-Modelle verschiedener Hochschulgebäude wie dem Richard-Stücklen Bau benötigt. Schön wäre es, nun einen Weg zu finden, das Haus trotz “Gitterproblematik“ realistisch darzustellen, ohne Einbußen in Optik oder Performanz hinnehmen zu müssen.

Abbildung 1.1: Vordere Fassade des Richard-Stücklen Baus im Jahr 2019 [Eigene Abbildung]

1.1 Aufgabenbeschreibung

Diese Bachelorarbeit nimmt den Bedarf an 3D-Gebäuden der Arbeitsgruppe *Historisches Mittweida* zum Anlass, den Richard-Stücklen Bau zu modellieren, um das entstandene Modell dafür, aber möglicherweise auch für diverse andere Projekte, zu nutzen. Um die Präsentation und Evaluation der Leistung zu ermöglichen ist es von Vorteil wenn das Modell nicht nur eigenständig existiert, sondern auch in einem realen Anwendungsfall Einsatz findet. Deshalb soll eine Software entwickelt werden, welche das Gebäudemodell beinhaltet. Da sich die Aufgabenstellung zum größten Teil aus genanntem Projekt ergibt, ist die spielerische und historische Thematik nicht zu vernachlässigen. Die Arbeit umfasst deshalb auch das Realisieren der Begehbarkeit des Gebäudes sowie das Darstellen der Entwicklung seiner Außenfassade durch den erfolgten Umbau. Zu betrachten ist das Vorgehen dabei stets unter dem Aspekt der Performanz.

Zusammengefasst sind die Themenschwerpunkte der Arbeit also die Erstellung des 3D-Modells des Richard-Stücklen Baus, die Entwicklung der Software, sowie die Einarbeitung in die Thematik der Performanzoptimierung.

Abbildung 1.2: Vordere Fassade des Richard-Stücklen Baus im Jahr 1999 [Hochschularchiv Mittweida, Bildarchiv, G_00195_003.jpg]

1.2 Zielsetzung und Aufbau der Arbeit

Durch die Entwicklung der Darstellung des Richard-Stücklen Baus soll die Frage beantwortet werden, ob und inwieweit das zur Aussenfassade gehörende Gitter des Gebäudes performant umgesetzt werden kann. Um das geplante Vorgehen bis zur Beantwortung dieser Frage besser zu verstehen, wird dieses in Abbildung 1.3 überblicksartig visualisiert.

Die Untersuchung der Frage geht mit der Modellierung des Baus und der Entwicklung der Anwendung einher. Zum Verständnis ersterens wird in Kapitel 2 zunächst auf die Möglichkeiten der 3D-Modellierung, spezifisch der von Gebäuden, eingegangen. Des Weiteren wird das Prinzip von Game Engines und deren Relevanz in der Umsetzung der Begehbarkeit des Baus erklärt und Möglichkeiten zur Performanzoptimierung erläutert. Auch Grundlagenwissen zu virtuellen Rundgängen ist nötig, um die konzeptionelle Umsetzung der Begehbarkeit zu begreifen und wird deshalb in diesem Kapitel dargelegt. Durch die nun vorhanden Grundkenntnisse können die konkreten Anforderungen an die Anwendung, welche in Kapitel 3 gelistet sind, besser nachvollzogen werden. Diesen muss entsprochen werden, um die genannte Aufgabe zu erfüllen und dem Ziel der Arbeit näher zu kommen. Kapitel 4 beinhal-

tet den Kern der Arbeit, den Implementationsprozess, inklusive einer detaillierten Beschreibung der verwendeten Werkzeuge, Methoden und Problemlösungsprozesse. Schließlich wird in Kapitel 5 bestimmt, inwieweit alle Anforderungen an die Anwendung erfüllt wurden. Ein Performanztest liefert dafür adäquate Daten. In Kapitel 6 wird das Vorgehen schließlich zusammengefasst und bewertet und ein Ausblick auf die Verwendung der nun vorhandenen Anwendung gegeben.

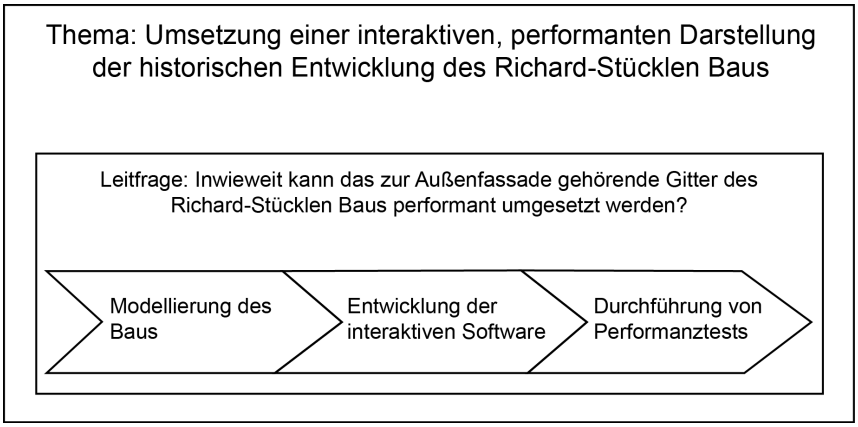

Abbildung 1.3: Überblick über das geplante Vorgehen [Eigene Darstellung]

2 Grundlagen

Nach der im vorigen Kapitel gegebenen Vorstellung und Einordnung des Themas werden in diesem Kapitel einige Grundlagen vermittelt, die zum Verständnis der Arbeit notwendig sind. Im Themenblock *Digitale Gebäudemodellierung* werden verschiedene Vorgehensweisen und gängige Software zur 3D-Modellierung von Gebäuden vorgestellt. Diese Erklärungen sollen ein späteres Nachvollziehen der genutzten Methoden und Werkzeuge ermöglichen. Um das Vorgehen bei der Implementierung des 3D-Gebäudes sowie der Integration von interaktiven Elementen zu verstehen, ist weiterhin ein grundlegendes Verständnis von *Game Engines* unumgänglich, welches im zweiten Theorieblock vermittelt wird. Ein Schwerpunkt der Arbeit besteht aus dem zu betrachtenden Performanz-Aspekt der zu erstellenden Anwendung, weshalb zusätzlich zur Erklärung der Funktionsweise von Game Engines auch die Performanzoptimierung thematisiert wird. Verschiedene Möglichkeiten zur Begehbarkeit von Gebäuden und der damit verbundenen Steuerungs- und Interaktionskonzepte werden im Kapitel *Virtuelle Rundgänge* erläutert. So können im Verlauf der Arbeit getroffene konzeptionelle Entscheidungen besser nachvollzogen werden.

2.1 Digitale Gebäudemodellierung

3D-Modelling bezeichnet das Erstellen eines virtuellen, dreidimensionalen Modells eines physischen Objekts mithilfe einer Software. 3D-Modelle haben heutzutage einen sehr großen Einsatzbereich, welcher sich von Computerspielen über Animationsfilme bis hin zu technischen Visualisierungen und Architekturplanung erstreckt. Auch 3D-Gebäude finden in vielen dieser Bereiche Anwendung. Aufgrund der umfangreichen Einsatzmöglichkeiten, die bestimmte Anforderungen an die Produkte stellen, gibt es diverse Herangehensweisen zur Erzeugung dieser. Zur Vereinfachung unterscheidet der Autor zwischen den beiden Methoden *Polygon-Modelling* und *Computer Aided Design (CAD)*, welche in den folgenden Absätzen genauer erklärt werden.

2.1.1 Polygon-Modelling

Das Polygon-Modelling ist ein zur 3D-Modellierung gehörendes Verfahren. In mancher Literatur werden die Begriffe gleichgestellt, wodurch Verständnisprobleme bei den Ausführungen innerhalb dieser Arbeit entstehen könnten. Deshalb wird das 3D-Modelling nun als Oberbegriff zum Polygon-Modelling betrachtet, welches wiederum als das Gegenstück zum später erläuterten Computer Aided Design gilt.

Beim Polygon-Modelling wird mithilfe von 3D-Modelling-Software die Oberfläche des zu erstellenden 3D-Objektes, welche auch Mesh genannt wird, so manipuliert, dass die gewünschte Form entsteht. Meist wird mit einer Grundform, wie zum Beispiel einem Würfel begonnen. Dieser wird durch das Verschieben, Rotieren und Skalieren von *Faces*, *Edges* und *Vertices*, verändert. Ein Face, auch: Polygon, ist ein flaches Stück der Oberfläche, eine Edge ist die Linie zwischen zwei Faces und ein Vertex ist der Punkt an dem sich zwei oder mehrere Edges treffen. Zur Verdeutlichung dient Abbildung 2.1. Um aus der Grundform ein komplexeres Objekt zu entwickeln, haben 3D-Programme neben den genannten Transformationstools auch weitere Werkzeuge, mit denen zum Beispiel Edges hinzufügt oder entfernt werden können. Dadurch kann der *Polycount*, welcher die Anzahl der Faces beschreibt, und in Folge dessen auch der Detailgrad des Modells verändert werden, was in Abbildung 2.2 deutlich gemacht wird. Normalerweise steigt mit zunehmendem Detailgrad auch die Komplexität des Modells und somit der Polycount [SKF08, S. 16].

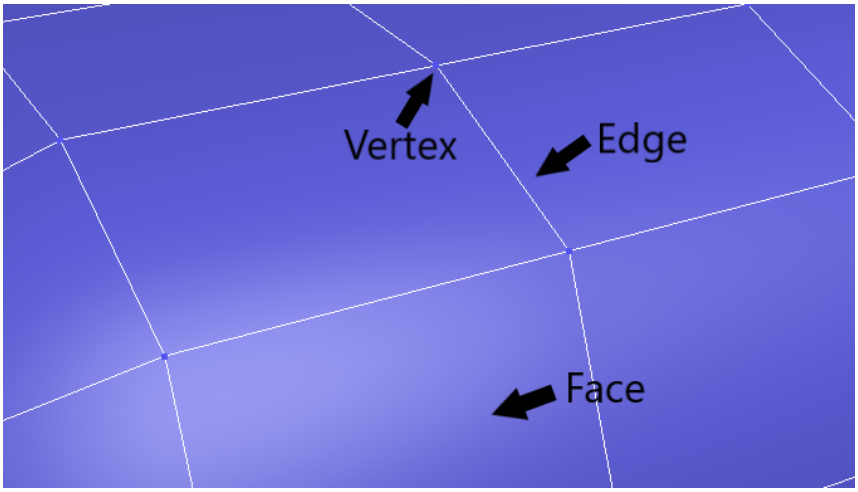

Abbildung 2.1: Vertices, Edges und Faces [Eigene Darstellung]

Der Polycount sollte generell so hoch wie nötig, jedoch so gering wie möglich gehalten werden [Gos16]. Durch eine höhere Polygonanzahl steigt der Rechenaufwand bei der Darstellung des Modells. Denn mit jedem Frame muss die Grafikkarte des Computers die dreidimensionalen Koordinaten aller Vertices in den zweidimensionalen Raum umwandeln, um sie in korrekter Position auf dem Monitor darstellen zu können. So ist es, mithilfe der berechneten Position und des Winkels der Kamera, möglich, dass nur die Polygone angezeigt werden, die sich auch tatsächlich im Sichtfeld der Kamera befinden. Je mehr Faces, desto mehr Vertex-Koordinaten müssen also pro Frame umgewandelt werden. Bei einem Spiel mit beispielsweise 30 Frames pro Sekunde (FPS) kann sich der Mehraufwand schnell aufsummieren, sodass das Spiel nicht mehr flüssig läuft. Es gilt also die Balance aus toller Grafik und Performanz zu halten [HZ14, S. 786-787]. Für eine gelungene Visualisierung ist es außerdem wichtig, dass alle 3D-Modelle, die Teil einer Gesamtszene sind, einen ähnlichen Detailgrad besitzen, da ein aus der Reihe fallendes Teil den homogenen, durchgängigen Bildeindruck zerstören würde [Haa05, S. 22].

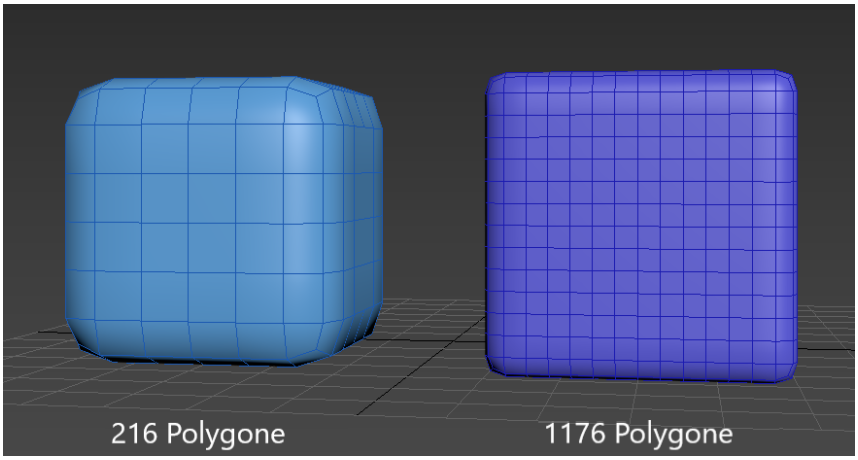

Abbildung 2.2: Polycount am Beispiel eines Würfels [Eigene Darstellung]

Oft werden mit Polygon-Modelling Inhalte für Spiele und Filme sowie für Special Effects und 3D-Druck kreiert [Tho18]. Es können organische Modelle, wie Menschen und Tiere, aber auch Objekte virtueller Welten, zum Beispiel Fahrzeuge, Pflanzen oder Gebäude, modelliert werden. Bekannte Programme für das Polygon-Modelling sind beispielsweise *Blender*, *Cinema 4D*, *Autodesk 3Ds Max* und *Autodesk Maya*.

2.1.2 Computer Aided Design

Im Gegensatz zum Polygon-Modelling wird Computer Aided Design vor allem für technische Planungen und Realisierungen von industriellen, mechanischen Objekten verwendet [Tho18]. In der Produktentwicklung ist CAD-Software heutzutage häufig bei der Kommunikation zwischen Designer und Kunde hilfreich [SZ01]. So benutzen beispielsweise Architekten CAD-Software, um maßstabsgetreue und möglichst realistische Visualisierungen von sich in Planung befindender oder bestehender Gebäude zu entwickeln [Ple19]. Vorteile für die Architekten, im Gegensatz zu traditionellen Methoden wie Zeichnungen oder von Hand gebastelten Gebäudemodellen, sind zum Beispiel die Flexibilität bei Änderungswünschen und die Möglichkeit der fotorealistischen Visualisierung zur besseren Veranschaulichung. Meist werden zuerst 2D-Grundrisse erstellt, welche automatisch in 3D-Modelle umgewandelt werden können [Ple19]. Im Gegensatz zum Polygon-Modelling, wo nur selten zu Beginn parametrisch gemodelt wird, spielen im CAD-Programm Maße eine große Rolle. Die direkte Transformation von Vertices, Edges und Faces ist dagegen in CAD-Software meist nicht möglich [Tho18].

2.1.3 AEC-Elemente in 3Ds Max

Architectural, Engineering and Construction (AEC) bezeichnet die Industrie des Konstruktions-, Ingenieur- und Bauwesen, in welcher CAD-Software häufig Anwendung findet [Cor01, S. 219].

3Ds Max ist eine Software des Unternehmens Autodesk, welche hauptsächlich für das Polygon-Modelling verwendet wird. Einzigartig für das Unternehmen ist die große Anzahl verschiedener 3D-Software. Dazu zählen auch verschiedene CAD-Programme, wie zum Beispiel *Auto-CAD*, *Revit* oder *Civil 3D*, welche im AEC-Sektor Anwendung finden. Es gibt einige Möglichkeiten 3D-Modelle aus CAD-Programmen zu exportieren und in Polygon-Modelling-Software einzubinden, um beispielsweise organische Elemente und Detail hinzuzufügen oder qualitative Renderings zu erstellen. Dies setzt jedoch bei kleinen Projekten die Fähigkeit der 3D-Artists heraus, beide Arten von Software bedienen zu können. Durch die Einbindung von AEC-Elementen ist es möglich Polygon- und CAD-Modelling innerhalb einer einzigen Anwendung zu kombinieren. Dies kann zum Beispiel für Entwickler von virtuellen Rundgängen durch Gebäude hilfreich sein, die die Maßgenauigkeit und Flexibilität in der

Grundrissgestaltung bei CAD schätzen, sowie das weit gefächerte Toolset einer 3D-Modelling-Software für Details nutzen möchten.

2.2 Game Engines

Game Engines können als der Motor von Spielen beschrieben werden. Sie sind essenziell für die Spieleentwicklung, da sie Abläufe automatisieren und der Entwickler sich durch ihre Verwendung viel Arbeit sparen kann. [Sch06, S. 21] Des Weiteren stellt eine Engine ein *Framework*, also eine Art Gerüst dar, das eine Reihe an grundlegenden Funktionen bietet, die für viele Spieleentwicklungen benötigt werden. Das können zum Beispiel physikalische Gesetze, die Möglichkeit der Darstellung von 3D-Modellen, Eingabeklassen oder Effekte sein [Sch06, S. 22]. Grundsätzlich ist bei der Entwicklung eines Spiels die Game Engine der zentrale Ort, an dem alle Inhalte zusammengefügt werden und ineinander greifen. So kann zum Beispiel ein Entwickler den selbst verfassten Code, die eigenen 3D-Modelle mit Animationen und seine Sounds in die Engine importieren und diese mittels der vorhandene Funktionen zu einem fertigen Spiel verknüpfen. Es ist möglich Engines selbst zu programmieren, jedoch benutzen viele Spieleunternehmen fertige Engines wie zum Beispiel das Portfolio der *Unity Engine*[1] zeigt. Weitere bekannte Engines sind beispielsweise *Anvil*, *GameMaker* und *Unreal*. Unity gilt jedoch als die am häufigsten verwendete Engine bei kleineren Spieleentwicklungen [Tof19].

Die Verwendung von Unity hat sich nicht nur in der Spieleentwicklung etabliert; auch Software, die nicht zur Unterhaltung dient, sondern beispielsweise im Gesundheitssektor Einsatz finden soll, wird häufig damit entwickelt [MKSB16, S.1]. Weitere Anwendungsmöglichkeiten sind *Architectural Walkthroughs*, also digitale Gebäudeführungen, interaktive Demonstrationen von technischen Konstruktionen, Training-Simulationen und Produktvisualisierungen. Die Verwendung der Unity Engine eignet sich auch für den Kontext dieser Arbeit, weshalb im Folgenden genauer auf ihre Funktionsweise und den Umgang mit ihr eingegangen wird. Im Anschluss werden Möglichkeiten der Performanzoptimierung in Unity erläutert.

[1] https://unity3d.com/de/games-made-with-unity, besucht am 18.02.20

2.2.1 Funktionsweise von Unity

Wie viele Engines stellt Unity eine graphische Benutzeroberfläche, die in Abbildung 2.3 zu sehen ist, bereit. Um die Verwendung grundlegend zu verstehen werden nun einige relevante in dieser Arbeit auftretende Begriffe erläutert und anhand der durchnummerierten Abbildung erklärt.

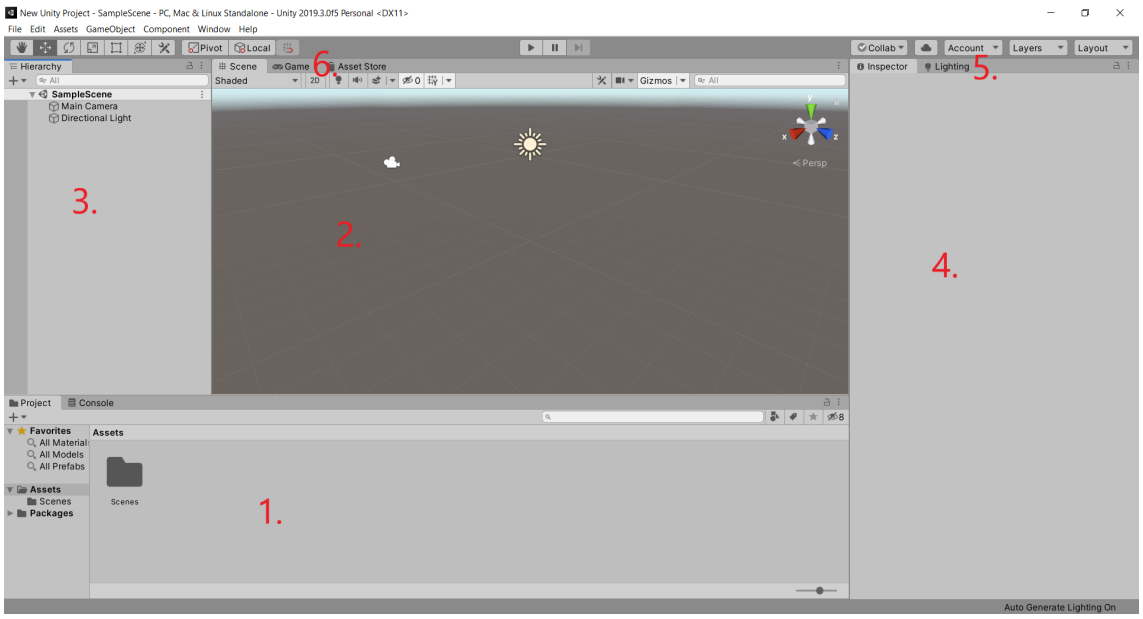

Abbildung 2.3: Benutzeroberfläche in Unity 2019.3 [Eigene Abbildung]

1. Projektfenster: Jede zu entwickelte Anwendung wird in einem einzelnen Projekt gespeichert, welche im Unity Hub[2] gemanaged werden. Das Layout der Benutzeroberfläche nach Erstellung eines neues Projektes ist in Abbildung 2.3 sichtbar. Alle Assets die zu einem Projekt gehören werden im Projektfenster dargestellt. Dies sind zum Beispiel importierte 3D-Modelle und Sounds oder selbst geschriebene Skripte.

2. Szenengraph: Ein Projekt besteht aus mindestens einer, meist jedoch einer Vielzahl an Szenen. Diese können durch das Projektfenster ausgewählt werden, um sie im Szenengraph anzuzeigen. In der Szene enthaltene Objekte können in diesem Fenster transformiert und angepasst werden.

[2]https://docs.unity3d.com/Manual/GettingStartedInstallingHub.html, besucht am 18.02.20

3. Hierarchie: Alle in einer Szene vorhandenen, als *GameObjects* bezeichneten Elemente werden in der Hierarchie gelistet. Sie können auch als Container für weitere GameObjects dienen wodurch eine Hierarchie mit vielen Unterebenen entstehen kann. Ein GameObject ist beispielsweise ein importiertes 3D-Modell eines Spielers, welches in die Szene gezogen wurde. Dem Spieler kann nun ein weiteres Objekt, wie beispielsweise eine Kamera untergeordnet werden, sodass die Position der Kamera der des Spielers folgt.

4. Inspector: Hier erhalten die GameObjects ihre Funktionen in Form von Komponenten. Dies können zum Beispiel Skripte oder physikalische Elemente sein.

5. Belichtung: Unterschiedliche Renderpipelines bieten verschiedene Belichtungsoptionen, welche sich in diesem Fenster befinden.

6. Game-Fenster: Hier kann die Anwendung in Echtzeit getestet werden.

2.2.2 Performanzoptimierung in Unity

Es existieren unterschiedliche Definitionen für den Begriff *Performanz*. In dieser Arbeit bezieht sich der Performanz-Begriff auf das durch eine hohe Framerate (deutsch: Bildfrequenz) erreichbare Gefühl der Immersion, also des Eintauchens in die Anwendung ohne “Ruckeln“, “Lags“ (deutsch: Verzögerungen) oder anderen Störungen in der graphischen Darstellung. Eine hohe Framerate im Zuge der Performanzoptimierung gilt als anzustreben. Für eine solche Optimierung gibt es folgende relevante Anhaltspunkte: Die graphische Darstellung einer Anwendung hängt von der Leistung des Prozessor (CPU) und die der Grafikkarte (GPU) des Computers ab, wobei sich die Möglichkeiten zur Performanzoptimierung bei diesen beiden Hardwarekomponenten unterscheiden.[3]

GPU-Optimierung:

- Um die GPU-Auslastung zu minimieren hilft es, wie bereits auf vorigen Seiten erwähnt, die Polygon- und damit einhergehenden Vertexanzahl, der in die Anwendung integrierten 3D-Elemente im Voraus niedrig zu halten. Um die Qualität eines bestehendes Modells nachträglich zu verbessern wurden viele

[3]Informationen für dieses Unterkapitel wurden aus der Unity Dokumentation zusammengetragen: https://docs.unity3d.com/Manual/OptimizingGraphicsPerformance.html, besucht am 19.02.20

verschiedene *Remesh*-Techniken entwickelt, durch die eine Verringerung der Polygonanzahl möglich ist [BKP+10, S. 85].

- Eine weitere Möglichkeit ist *Lightmapping* zu verwenden. Dabei werden die Lichtverhältnisse *gebaked*, was bedeutet, dass diese nicht während der Laufzeit berechnet werden müssen. Möglich ist dies hauptsächlich bei statischen Objekten in der Szene, da bei diesen meist keine durch Bewegung veränderte Belichtung vorkommt.
- Außerdem können möglicherweise verwendete Texturen der 3D-Modelle komprimiert werden, um weiteren Rechenaufwand zu sparen.

CPU-Optimierung:

- Für die Verringerung des CPU-Aufwands ist die Zusammenfügung von mehreren Objekten zu einem Einzelnen von Vorteil. Es gilt dabei: Je weniger Meshes, desto geringer der Aufwand.
- Auch das Sparen an verschiedenen Materialien kann die Performanz erhöhen.

Die erreichbare Framerate ist von diversen weiteren Faktoren abhängig, wie zum Beispiel die Kapazität des Arbeits- und Texturspeichers oder die Taktrate von Systembus und Speicher [Haa05, S. 21]. Weitere Ansätze zur Performanzoptimierung sind zum Verständnis dieser Arbeit jedoch nicht zwingend notwendig und werden daher nicht genauer erläutert.

2.3 Virtuelle Rundgänge

Ein virtueller Rundgang bezeichnet die Möglichkeit, sich in virtuellen Orten und Räumlichkeiten frei und interaktiv umzusehen. Heutzutage gibt es für diese Art von Anwendung viele Einsatzmöglichkeiten, beispielsweise um Kunden oder Besuchern von Einrichtungen wie Museen oder Universitäten die Räume nicht nur in der Realität sondern auch digital sichtbar und begehbar zu machen. Oft werden diese mit interaktiven Elementen oder zusätzlichem Material erweitert, um den Informationsgehalt für den Nutzer zu steigern. [Fin18, S. 1-2]

Aufgrund der vielen unterschiedlichen Definitionen für *Virtueller Rundgang* im Internet [Fin18, S. 1] gibt es auch keine einheitliche Kategorisierung der verschiedenen

Arten solcher Rundgänge. Im Folgenden werden die vom Autor als am relevantesten eingeschätzten Arten beschrieben.

2.3.1 Arten von virtuellen Rundgängen

- **Panorama-Rundgänge:**
  Es ist möglich, einzelne Räume oder Orte durch Panorama-Fotos darzustellen. Die einzelnen Fotos werden dann so verknüpft, dass dem Nutzer durch den Klick auf einen Hotspot, z.B. einem Pfeil, das nächste Foto angezeigt wird. Dieses Foto ist dann eine Aufnahme, die zum Beipiel aus einem anderen Winkel aufgenommen wurde. Dadurch wird eine Bewegung suggeriert, durch die das Erlangen eines Gesamtüberblicks über die Räumlichkeiten möglich ist. Ein Beispiel hierfür ist das bekannte Feature von *Google Street View*, welches in Abbildung 2.4 gezeigt wird. Bei den Fotografien kann es sich wie in diesem Beispiel um 360-Grad-Fotos handeln, aber auch andere Arten von Weitwinkel-Aufnahmen oder Renderings aus 3D-Welten sind möglich. Bei 360-Grad-Aufnahmen ist zusätzliche Interaktivität gewährleistet, da der Nutzer das Sichtfeld der Kamera in beide Achsen bewegen und somit kontrollieren kann was er sieht. Häufig werden durch die Einbindung von weiteren Elementen, wie zum Beispiel Soundeffekten, Musik oder Textelementen zusätzliche Informationen an den Nutzer vermittelt [PSL12, S. 100]. Panoroma-Rundgänge findet man im Internet sehr häufig, da viele Unternehmen oder Einrichtungen diese als Webversion für ihre Kunden anbieten, wie beispielsweise das *National Museum of Natural History*[4].

- **Video-Rundgänge:**
  Eine weitere Art von virtuellen Touren kann durch Videos realisiert werden. Diese Art ist weniger interaktiv als die vorig genannte, da der Ablauf der Tour schon im Vorfeld festgelegt ist. Es findet ein gewöhnlicher Videodreh statt, bei dem die Kamera durch die vorzustellende Einrichtung geführt wird. [PSL12, S. 101] Häufig kommt eine 360-Grad-Kamera zum Einsatz, um auch hier einen Rundblick während des Abspielens für den Nutzer zu ermöglichen. Ein Beispiel

[4] https://naturalhistory.si.edu/visit/virtual-tour, besucht am 03.04.20

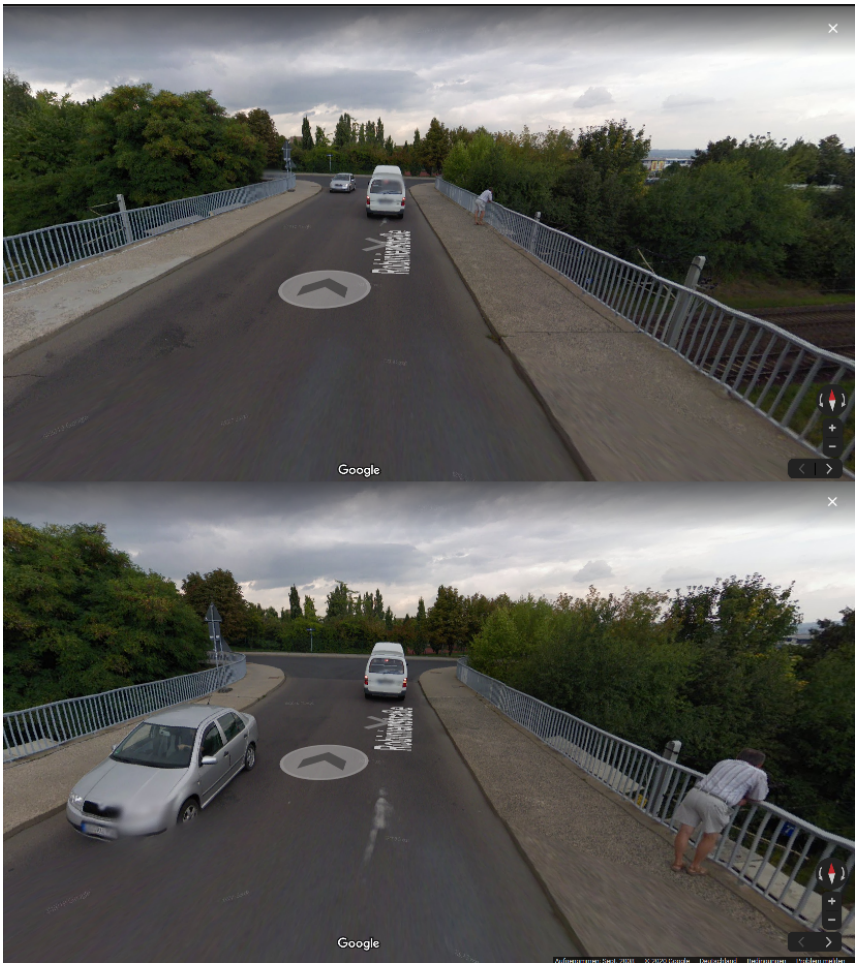

Abbildung 2.4: Google Street View [www.google.com/maps/]

eines solchen Videos ist ein Rundgang durch den Dresdner Fernsehturm, welcher auf dem Youtube-Kanal *Sächsische.de* zu finden ist[5]. Des Weiteren sind animierte Video-Rundgänge durch 3D-Welten möglich. Interaktivität kann zum Beispiel durch zusätzlich eingebundenes Material oder die Auswahl von Videosequenzen statt einem durchgängigen Video hinzugefügt werden. Auch ein Erzähler, der die Tour leitet, ist häufig vorhanden. Obwohl der Produktionsaufwand von Video-Rundgängen meist höher ist als der bei Panorama-Rundgängen, sind diese sehr beliebt [PSL12, S. 101].

[5]https://www.youtube.com/watch?v=TWrPfxkrGgw, besucht am 26.03.20

- **Echtzeit-Rundgänge:**
  Hierbei handelt es sich um eine Art von Rundgang der ausschließlich in 3D-Welten zum Einsatz kommt. Der Nutzer kann sich durch diese in Echtzeit bewegen und dabei auch die Kameraperspektive beeinflussen [LV02, S. 1]. Die Steuerung verhält sich also analog zu der in Spielen in der bekannten Ego-Perspektive, erlaubt also dem Nutzer sich so zu fühlen, als würde er tatsächlich durch das Gebäude gehen; dabei kann dieser beispielsweise auch Treppen steigen und aus Fenstern schauen [SKF08, S. 13]. Handelt es sich bei der 3D-Umgebung um Abbildungen realer Gebäude, so wird diese Art von virtuellem Rundgang im Internet häufig auch als *Architectural Walkthrough* bezeichnet. Um realitätsnahe 3D-Rundgänge zu erhalten, kann das Gebäude modelliert werden; es gibt jedoch auch die Möglichkeit mithilfe von *Image-Based-* oder *Video-Based-Rendering* 3D-Modelle aus Foto- oder Videoaufnahmen automatisiert zu erstellen [Pat12, S. 1]. Anwendung finden Echtzeit-Rundgänge zum Beispiel in der Immobilienbranche, um Kunden die Verkaufsobjekte schon vorher oder zusätzlich zu einer realen Besichtigung zu zeigen. Auch in der Bauindustrie finden virtuelle Rundgänge Anwendung, um Projekte sowohl in der Design- als auch der Bauphase zu visualisieren, um so realistischere Erwartungen an das Endprodukt zu gewährleisten [ST02, S. 3, S. 16].

2.3.2 Steuerung und Interaktion bei Echtzeit-Rundgängen

Die Bewegung des Nutzers in einem Echtzeit-Rundgang ähnelt der Bewegung in einem First-Person-Computerspiel. Der Nutzer kann sich wie in der realen Welt frei bewegen und es wird ein Gefühl der tatsächlichen Präsenz suggeriert. Dadurch, dass es sich bei einem solchen Rundgang um einen *Walkthrough* handelt, bei welchem man im Gegensatz zu einem *Flythrough* nicht schweben kann, sollte ein Schwerkraft-Effekt umgesetzt werden. Der Nutzer “geht“ also durch die Räume, kann sich umschauen, Treppen steigen und theoretisch auch herunterfallen. Auch Drehungen bis zu 360 Grad auf der vertikalen Achse sind dem Spieler möglich, was häufig durch die Möglichkeit die Kamera zu bewegen umgesetzt wird. Diese entspricht dem Sichtfeld des Nutzers und kann auch auf der horizontalen Achse mit Begrenzungen, die der von realen Kopfbewegungen entsprechen, rotiert werden.

Generell kann die Interaktion von Mensch und Computer auf verschiedene Arten geschehen, zum Beispiel durch Kommandos über die Tastatur, Mausklicks oder Sprachsteuerung [SKF08, S. 13]. Auch Gamepads werden häufig benutzt und sind bei kontrollierbarer Geschwindigkeit häufig die beste Alternative [LV02, S. 1], was bei virtuellen Touren jedoch selten relevant ist, da der Fokus auf den zu besichtigenden Räumen liegt. Weiterhin bietet eine grafische Benutzeroberfläche zum Beispiel durch Buttons die Möglichkeit, Interaktionen mit der Maus durchzuführen. Wichtig ist dabei, dass die Bedienelemente intuitiv sind, um den Nutzer, der möglicherweise ein Laie ist, nicht abzuschrecken. Bei der Navigation durch 3D-Gebäude innerhalb eines virtuellen Rundgangs wird auf dieselben Möglichkeiten der Interaktion zurückgegriffen. Durch die Analogie zu Computerspielen in der Ego-Perspektive wird häufig die klassische Steuerung über die Pfeiltasten der Tastatur genutzt. Auch die Kamerabewegung wird wie in den meisten First-Person-Spielen durch die Bewegung der Maus umgesetzt.

3 Anforderungen und Spezifikation

Im folgenden Kapitel werden Anforderungen an die Anwendung gestellt, welche für die Erreichung der Ziele der Arbeit notwendig sind; auch einige Inhalte, deren Umsetzung optional ist, werden genannt. Das Thema wird dabei in einzelne Anforderungsbereiche eingeteilt, welche anschließend detailliert erläutert werden. Die durch Kapitel 2 vorhandenen Grundkenntnisse helfen dabei, die folgenden Ausführungen besser nachvollziehen zu können. Des Weiteren wird die Relevanz der vorig vorgestellten Vorgehensweisen und Werkzeuge für diese Arbeit bestimmt.

Eine Aufschlüsselung des Themas dieser Arbeit führt zu folgenden Teilbereichen:

- Die **historische Entwicklung** des Richard-Stücklen Baus soll dargestellt werden.
- Elemente der **Interaktivität** soll vorhanden sein.
- Die **Performanz** der Anwendung soll sichergestellt werden.

Diese Teilanforderungen stellen die drei Anforderungsbereiche dar, die in den folgenden Unterkapiteln detailliert beschrieben werden.

3.1 Historische Entwicklung

Der erste Anforderungsbereich, der aus dem Thema hervorgeht, ist die Darstellung der Historischen Entwicklung des Richard-Stücklen Baus. Die Darstellung des Gebäudes soll als digitales 3D-Modell erfolgen, welches in mindestens einer zu entwickelnden Desktop-Anwendung für Windows 10 für Nutzer sichtbar gemacht wird. Das 3D-Modell muss dabei nicht maßstabsgetreu sein; der Realismusgrad richtet sich nach Augenmaß. Die historische Entwicklung wird dabei auf die der Außenfassade des Hauses begrenzt, da das während eines Umbauprojektes im Jahr 2010 hinzugefügte goldene Gitter als das markantestes Element des Gebäudes gilt. Aufgrund der

im ersten Kapitel geschilderten Motivation, das Projekt im Anwendungsfall *Historisches Mittweida* nutzen zu können, bietet es sich an, die durch den Umbau erfolgte Veränderung der Fassade darzustellen. Das Modell darf daher nicht nur als zusammenhängendes 3D-Objekt existieren, sondern muss modular aufgebaut werden. Es sollen sowohl die alte als auch die neue Fassade modelliert werden und der Tausch innerhalb der zu entwickelnden Anwendung möglich sein, um die Veränderung dieser für den Nutzer sichtbar zu machen. Die Fassade sollen mindestens einfarbige Materialien aufweisen, um die Angleichung an den realen Richard-Stücklen Bau zu unterstützen. Graphische Anforderungen an das Innenleben werden nicht gestellt, da die Ausarbeitungen den Umfang dieser Arbeit übersteigen würden und für das Ziel der Arbeit nicht relevant sind.

Konkrete Anforderungen:

- Ein digitales 3D-Modell, das den Richard-Stücklen Bau von Außen und Innen so realistisch wie mit Augenmaß möglich darstellt, soll vorhanden sein.
- Das 3D-Modell soll so aufgebaut sein, dass theoretisch zwischen zwei verschiedenen Fassaden gewechselt werden kann.
- Es soll sowohl ein Fassaden-Modell von vor, als auch eines von nach dem Umbau des Hauses in Jahr 2010 existieren.
- Mindestens eine Desktop-Anwendung für Windows 10, in welcher das Modell sichtbar ist, soll vorhanden sein.

Geplante Vorgehensweise:

Die Modellierung des 3D-Modells soll mit der Software 3Ds Max entstehen. Die im Grundlagenkapitel diskutierten Vorteile, sowie die durch private- und studiumsbezogene Projekte erlangte Erfahrung des Autors mit der Software, versprechen eine kurze Einarbeitungszeit und somit eine einfache Umsetzung der Anforderungen. Die Nutzung der vorgestellten AEC-Elemente, die in der Software enthalten sind, erscheint intuitiv und schnell erlernbar. Dies ist von Vorteil, da der Autor bisher keinerlei Erfahrung mit CAD-Software hat. Die Möglichkeit der Kombination von parametrischen Elementen und Polygon-Modelling ist dieser Arbeit zuträglich.

Grund hierfür sind die folgenden beiden notwendigen Schritte, um den Anforderungen an die historische Entwicklung gerecht zu werden:

- Zum Einen ist die Umwandlung des realen Richard-Stücklen Baus in ein 3D-Modell gefordert, was die Notwendigkeit parametrischer Objekte erklärt. Zwar wurde die Übertragung nach Augenmaß als ausreichend eingestuft, doch ist es effektiver, einige grobe Messungen des Gebäudes durchzuführen und diese in den Maßstab des 3D-Modells umzuwandeln, um grobe Referenzen zu den Größenverhältnissen zu erhalten. Auch die Erstellung des umfangreichen Innenlebens des Baus soll mithilfe von AEC-Elementen vereinfacht werden. Durch die Modularität und die schnelle Platzierung von kompletten Wänden, Türen, Fenstern und Treppen, die sich von der AEC-Element-Nutzung erhofft wird, soll der 3D-Grundriss schnell aufgebaut werden können.
- Ein weiterer Schritt ist die Erstellung des goldenen Gitters: Das Gitter ist ein repetitives Muster, welches aus einzelnen Teilen besteht, die im Idealfall in der 3D-Software isoliert modelliert und dann in großen Mengen dupliziert werden können. Für das Erstellen einer realitätsnahen Form ist Polygon-Modelling erforderlich. Die Nutzung von 3Ds Max wirkt somit auch für die Modellierung des Gitters adäquat.

In Unterkapitel 2.3 wurden einige Ansätze zur Umsetzung virtueller Touren vorgestellt. Da der Fokus dieser Arbeit auf dem zu erstellenden 3D-Gebäude liegt fällt die Wahl auf einen Echtzeit-Rundgang. Zur Entwicklung der Desktop-Anwendung empfiehlt sich die Verwendung der Unity Engine. Auch hier ist die Erfahrung des Autors durch das Studium sowie die in Kapitel 2.2. beschriebene Popularität unter privaten Entwicklern ausschlaggebend. Durch die Möglichkeit, Anwendungen für diverse Plattformen zu erstellen, eignet sich die Engine für die Entwicklung der geforderten Desktop-Version für Windows 10 sowie eventuell erwünschte mobile Versionen.

Auf die Texturierung der Fassaden wird kein besonderes Augenmerk gelegt, stattdessen kann sich diese auf das Hinzufügen von einfarbigen Materialien in Unity beschränken. Dazu kann ein von Unity standardisierter Shader verwendet werden.

3.2 Interaktivität

Der zweite Bereich befasst sich mit der zu entwickelnden Interaktivität in der Software. Dabei sind zwei Elemente essenziell: Die Möglichkeit einen Fassaden-Tausch durchzuführen und die Begehbarkeit des Gebäudes. Ersteres soll in Kombination mit den Anforderungen an die historische Entwicklung interaktiv umgesetzt werden. Der Nutzer soll mithilfe eines Userinterfaces oder Tastatureingaben in der Lage sein, selbst zu steuern, welche der beiden Außenwände sichtbar ist. Die Begehbarkeit unterstützt das interaktive Erlebnis in einem spielerischen Ansatz, indem sich der Nutzer im Stil eines Echtzeit-Rundgangs in Ego-Perspektive um und durch das Modell des Richard-Stücklen Baus bewegen kann, um es aus verschiedenen Perspektiven zu betrachten.

Konkrete Anforderungen:

- Die beiden Fassaden-Varianten sollen in der Anwendung vom Nutzer getauscht werden können.
- Der Nutzer soll sich in der Anwendung eigenständig durch das Gebäude-Modell bewegen können.

Geplante Vorgehensweise:

Innerhalb der zu benutzenden Unity Engine ist das Hinzufügen von interaktiven Elementen möglich. Für Fassaden-Tausch und Begehbarkeit empfiehlt es sich eigenen Code in Form von in Unity einzubindenden Skripten zu verfassen. Die Umsetzung der interaktiven Elemente soll sich an folgenden geplanten Vorgehensweise orientieren:

- Die Bewegung des Spielers kann durch die in der *Character Controller*-Klasse enthaltenden Methoden durchgeführt werden. Dem zu erstellenden Spieler wird eine Character-Controller-Komponente zugewiesen. In einem Movement-Skript sollen die Tastatureingaben mithilfe der *Input*-Klasse abgerufen und in Variablen gespeichert und schließlich als Parameter der Move()-Methode der Character-Controller-Klasse übergeben werden. Die Geschwindigkeit soll leicht anpassbar sein, weshalb dafür eine öffentliche Variable zu erstellen ist. Auch die Schwerkraftseinwirkung soll durch die Nutzung der Move()-Methode geschehen.

- Zusätzlich zur Spielerbewegung soll auch eine horizontale und vertikale Kamerarotation umgesetzt werden, sodass der Nutzer seinen "Kopf" bewegen und das Gebäude komplett betrachten kann. Dafür soll für die Mauseingaben wieder die Input-Klasse verwendet werden. Die erhaltenen Werte können schließlich durch die aktuellen Rotationswerte der Kamera ersetzt werden.
- Das Wechseln der Fassaden kann mithilfe eines if-else-Statements durch das direkte Aktivieren und Deaktivieren dieser Objekte umgesetzt werden. Die Methode in der dieser Wechsel passieren soll, kann möglicherweise einfach über die onClick()-Methode des zu erstellenden Buttons aufgerufen werden. Eine zusätzlicher Wechsel durch Klicken der Leertaste ist auch hier mit einer Wertabfrage über die Input-Klasse möglich.

3.3 Performanz

Die Entwicklung des Modells und der Anwendung erfolgt stets unter Betrachtung des Performanz-Aspekts. Die Desktop-Anwendung soll auf einem modernen, gängigen Laptop mit Windows 10 ohne Unterbrechungen durch größeres "Ruckeln" laufen. Falls es der Zeitrahmen der Arbeit zulässt wird zusätzlich zur Desktop-Version eine Version für Android-Geräte angefertigt, um Unterschiede in der Performanz erkennen und analysieren zu können. Damit muss möglicherweise eine Detailreduzierung des Goldgitter-Modells einhergehen.

Konkrete Anforderungen:

- Die Desktop-Anwendung soll auf einem modernen Laptop mit Windows 10 störungsfrei laufen.
- Nach Möglichkeit kann zusätzlich eine Anpassung der Gitters und der Anwendung für Android-Geräte durchgeführt werden.

Geplante Vorgehensweise:

Während des Modellierungsvorgangs ist darauf zu achten, dass die Polygonanzahl möglichst niedrig gehalten wird ohne Optik-Einbußen hinnehmen zu müssen. Die

Polygonanzahl kann in 3Ds Max angezeigt werden. Zu beachten ist, dass das separat modellierte Modell eines einzelnen Gitterelements im nächsten Schritt um ein Vielfaches dupliziert wird, wodurch eine große Anzahl an Polygonen entsteht.

Nach der Implementierung in Unity sind weitere Verfahren in Betracht zu ziehen, die, wie schon im Grundlagenkapitel erläutert, den Rechenaufwand verringern können. Es bietet sich, an einen Test über den Performanz-Unterschied bei Verwendung von Box- statt Meshcollidern durchzuführen. Des Weiteren wäre es von Vorteil die Möglichkeit des Bakings zu testen, um möglicherweise weitere Einsparungen des Rechenaufwands während der Laufzeit machen zu können.

Sollte es im Umfang dieser Arbeit möglich sein eine Anpassung der Anwendung an mobile Endgeräte vorzunehmen, so sind wahrscheinlich Remesh-Maßnahmen, um die Polygonanzahl zu verringern oder gegebenfalls eine erneute Modellierung des Gitters mit graphischen Einschränkungen, notwendig. Beide Verfahren können in 3Ds Max erfolgen.

4 Umsetzung

Nachdem im vorigen Kapitel die Anforderungen an die Anwendung spezifiziert wurden, werden in diesem Kapitel notwendigen Schritte zur Erfüllung dieser erläutert. Dabei sollen nicht nur verwendete Werkzeuge und Arbeitsabläufe bei der Umsetzung der Software Inhalt sein, sondern auch aufgetretene Probleme analysiert werden. Die Dokumentation des schrittweisen Vorgehens wird in Unterkapitel gegliedert.

4.1 Datengewinnung

Die Anforderung an das 3D-Modell des Richard-Stücklen Baus gibt eine möglichst realitätsnahe Darstellung des Baus vor, wobei diese nicht maßstabsgetreu sein muss, sondern nach Augenmaß entwickelt wird. Der Grund hierfür wurde bereits im vorigen Kapitel erläutert. Es muss daher keine umfangreiche Vermessung des vollständigen Gebäudes durchgeführt werden. Um einen Richtwert zu den Größenverhältnissen zu erhalten und so die Modellierung zu vereinfachen wurden dennoch einige Maße genommen. Dabei handelt es sich um Länge und Breite des Gebäudes, einige Abmessungen des Flurs, des Treppenhauses, der Fenster und der Türen, sowie verschiedene Deckenhöhen. Da die Vermessung keinen relevanten Teil der Arbeit darstellt wurde diese mit einem einfachen Meterstab mit einer Messtoleranz von 50 cm durchgeführt.

In jedem Stockwerk des Gebäudes befinden sich Notausgangspläne, welche sich als Grundrissvorlage für das 3D-Modelling der Wände eignen. Dazu wurden die Pläne fotografiert und die gemessenen Werte digital eingetragen. Die Pläne konnten so später in 3Ds Max importiert und verwendet werden, was im nächsten Unterkapitel genauer erläutert wird. Durch diese Werte wurde sichergestellt, dass die Verhältnisse, die zum Beispiel durch den durch schiefes Fotografieren entstandenen Parallaxenfehler verzerrt werden können, den Anforderungen genügen.

Für eine realitätsnahe Modellierung wurden relevante Teile des Gebäudes fotografiert. Um eine korrekte Darstellung von Fassade und Innenleben zu entwickeln, müssen die grobe Anordnung von Gebäudelementen sowie Details wie das Goldgitter genau begutachtet werden. Durch die vorhandenen Fotos konnte dieser Prozess nicht nur vor Ort, sondern auch während des Modellierungvorgangs erfolgen.

Die Arbeit sieht eine Darstellung der historischen Entwicklung des Richard-Stücklen Baus vor. Um die alte Fassade des Gebäudes zu modellieren waren auch Fotos dieser notwendig. Hierfür wurde das Hochschularchiv kontaktiert, welches einige alte Fotos des Richard-Stücklen Baus von der Zeit vor dem Umbau bereitstellen konnte.

4.2 Grundrissmodellierung

Wie schon im Grundlagenkapitel erläutert, gibt es verschiedene Möglichkeiten der digitalen Modellierung von Gebäuden. In der Anforderungsanalyse wurde der Entschluss gefasst, sich die vielversprechenden AEC-Funktionen in 3Ds Max zu Nutzen zu machen, wofür die Software-Version 2020 verwendet wird. Zuerst wurden die Fluchtpläne als Texturen in 3Ds Max importiert und Planes zugewiesen, welche als Vorlage für Wände, Türen und Treppen verwendet wurden.

Die Nutzung von AEC-Elementen in 3Ds Max 2020 ist intuitiv und erfolgt durch das *Erstellen*-Tab im *Vorgabe*-Menü. Hier lassen sich im Dropdown-Menü verschiedene Kategorien mit jeweils verschiedene Objekttypen auswählen, deren Benutzung analog zu der von gewöhnlichen Grundkörpern funktioniert. In Abbildung 4.1 ist dies am Beispiel der Kategorie *Türen* dargestellt. Jeder Objekttyp hat individuelle anpassbare Eigenschaften.

In Abbildung 4.2 wird deutlich, wie die Linien des Fluchtplans mit dem AEC-Wand-Tool realitätsgetreu nachgezogen werden konnten. Das Tool erlaubt es, während der Erstellungsphase den Höhenparameter und die Ausrichtung der Wand anzupassen.

Durch Auswählen der Wand-Funktion kann mithilfe von Mauseingaben eine Wand in der gewünschten Position “aufgezogen“ werden. Durch einen anschließend Linksklick wird diese fest platziert. Im Anschluss ist es möglich diesen Vorgang beliebig oft zu wiederholen, um weitere Wände zu erstellen. Alle gesetzten Wände befinden sich nach Beendigung des Wand-Erstellungmodus, durch beispielsweise einen Rechtsklick, in einem einzigen Objekt. Um separate Wandobjekte zu erhalten ist

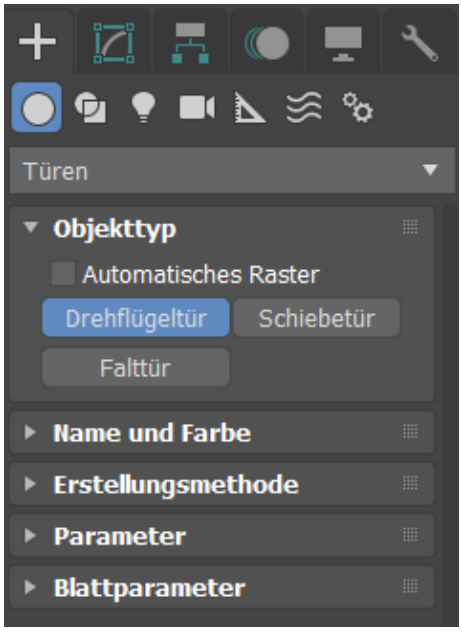

Abbildung 4.1: Einbindung von AEC-Elementen im Create-Tab in 3Ds Max 2020 am Beispiel von Türen [Eigene Darstellung]

also eine Unterbrechung des Vorgangs nach jeder Wandsetzung nötig. Durch die Reduktion der Anzahl an Objekten hat man zwar einen besseren Überblick in der Hierarchie und eventuelle Einsparungen in der Polygonanzahl, doch ist diese in dem Fall so gering, dass die Nachteile der Objektvereinigung überwiegen. Der größte Nachteil ist die dadurch fehlende Modularität, die im Folgenden erklärt wird: Nach fortgeschrittener Erstellung des 3D-Grundrisses musste festgestellt werden, dass es bei AEC-Elementen keine Funktion gibt, mit der man ein Objekt wieder teilen kann. Möchte man nun einige Korrekturen der Positionen einzelner Wände vornehmen, wie es in diesem Projekt der Fall war, so muss möglicherweise ein aus einer Vielzahl von Wänden bestehendes Objekt neu erstellt werden, was einen großen Zeitaufwand erfordert.

In diesem Fall bot sich als Lösung zuerst die Umwandlung in ein *Editable Poly* an. Dabei handelt es sich um eine Modifizierung der Wandobjekte, durch welche die Faces, Edges und Vertices der Wand transformiert werden können. Es findet eine Übertragung der parametrischen Objekte in das im Kontext dieser Arbeit häufig erwähnte Polygon-Modelling statt. Die Umwandlung bringt eine Einschränkung in der Nutzung der AEC-Funktionen mit sich, denn ein einfaches Einbauen von Fenstern

Abbildung 4.2: Wanderstellung mit AEC-Elementen [Eigene Abbildung]

und Türen, welches in den Anforderungen als Grund für die Wahl der AEC-Nutzung genannt wird, ist so nicht mehr möglich.

Da jedoch auch die Neuentwicklung eines Arbeitsablaufs zeitintensiv gewesen wäre und der Autor weiterhin von den möglichen Vorteilen der AEC-Elementen überzeugt war, fiel die Entscheidung den Vorgang nochmal zu wiederholen. Es wurden alle Wände erneut erstellt, die Wandabschnitte dabei jedoch kleiner gewählt, sodass zum Beispiel jede Seite eines Raums ein separates Objekt darstellt. Zusätzlich zum Vorteil der einfacheren Durchführung möglicherweise erforderlicher Positionskorrekturen der einzelnen Wandabschnitte besteht auch die Möglichkeit diesen unterschiedliche Höhenwerte zuzuweisen.

Im Anschluss an die korrekte Übertragung der Wände in die Dreidimensionalität wurden diesen schließlich Türen hinzugefügt. Das AEC-Türen-Werkzeug zeichnet sich dadurch aus, dass durch die korrekte Platzierung der Türen in der Wand automatisch eine Boolean-Operation ausgeführt wird. Der durch die Tür ersetzte Wandteil wird sozusagen “ausgestanzt“. Auch die Treppen konnten als AEC-Elemente hinzugefügt und durch die verschiedenen Parameter wie Länge, Breite, Höhe und Stufenanzahl angepasst werden. In Abbildung 4.3 ist das Gebäude nach abgeschlossener Modellierung des Grundrisses zu sehen.

Für die zwei unterschiedlichen Fassaden wurden die Vorder- und die Rückwand des Modells dupliziert und diesen die entsprechenden Fensterfronten und andere markante Details hinzugefügt. Das Anwenden der AEC-Fenster-Funktion entspricht dabei der der Türen-Funktion mit dem erwähnten Boolean-Operator. Ausgewählt

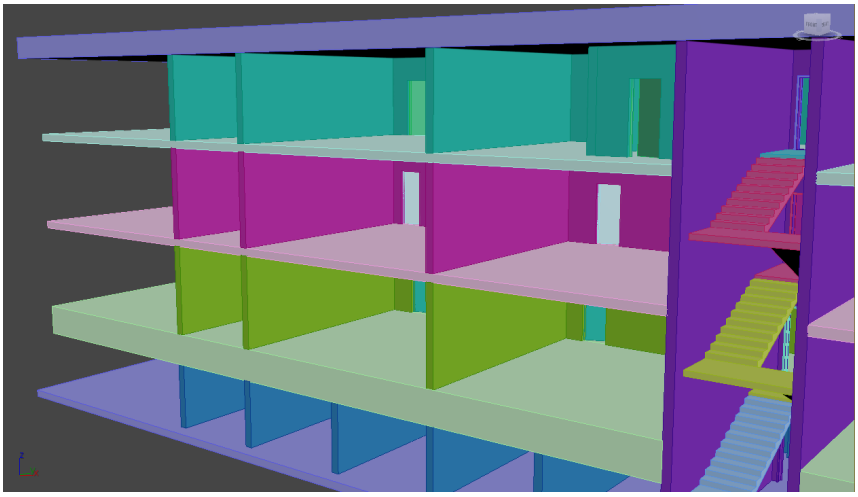

Abbildung 4.3: 3D-Grundriss des Richard-Stücklen Baus [Eigene Abbildung]

wurde die *fixed*-Version, was einem Fenster ohne Öffnungsmöglichkeit entspricht. Grund hierfür ist der Verzicht auf Animationen, weshalb eine mögliche Öffnung keine Rolle spielt.

4.3 Gittermodellierung

Die Modellierung des Gitters wurde durch Polygon-Modelling umgesetzt. Es wurde zuerst ein einzelnes Gitterteil modelliert, indem eine rechteckige Box in ein Editable Poly umgewandelt wurde und dieser durch den Befehl *Chamfer* neue Edges hinzugefügt wurden. Die so neu entstandene Polygone und Vertices konnten bewegt und rotiert werden. Anhand von Nahaufnahmen des Gitters wurde die zu erstellende Form analysiert und in 3D übertragen. Während des Vorgangs galt es das Performanzkriterium zu beachten und deshalb nur so viele Polygone, wie zur Erreichung der gewünschten Form nötig, entstehen zu lassen. Abbildung 4.4 zeigt den Vergleich zwischen den realen und den in 3D erstellten Gitterelementen.

Nach der Modellierung des Gitterelements konnte dieses passend rotiert und mehrfach dupliziert werden, um das vollständige Gitter zu erhalten. Dieses wurde um die äußeren Wände des Gebäudes positioniert und angepasst. Um die großen Aussparungen, die sich im Gitter des Richard-Stücklen Baus befinden darzustellen, wurden nach Augenmaß einige der Einzelteile wieder herausgelöscht.

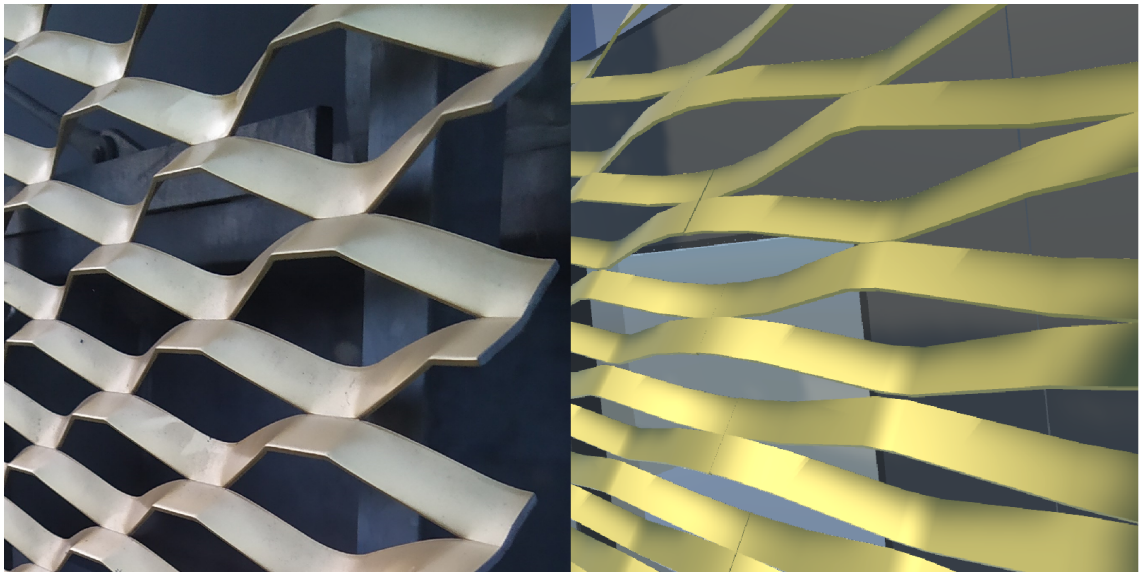

Abbildung 4.4: Links: Reales Gitter, Rechts: Modell des Gitters [Eigene Abbildung]

Um die Übersichtlichkeit in der Projektstruktur zu wahren wurde zunächst versucht, die große Menge an einzelnen Objekten durch den Befehl *Attach* zu einem einzigen Objekt zusammenzufügen. Durch dadurch entstandene unvorhersehbare Probleme mit verdrehten Edges wurde sich jedoch für die Gruppierung der Objekte entschieden. Dies erwies sich zudem bei späteren Anpassungen der Gitteraussparungen als vorteilhaft, da die einzelnen Objekte einfach gelöscht wurden konnten.

4.4 Implementierung

Die Implementierung erfolgte, wie in der Spezifikation diskutiert, durch die Unity Engine. Es empfiehlt sich dabei stets die aktuellste Version zu nutzen, um auf alle neuen Funktionen zugreifen zu können, weshalb die Unity-Version 2019.3 verwendet wurde. Es wurde sich in den Projekteinstellungen für die Benutzung der neuen, im Grundlagenkapitel vorgestellten, Universal Render Pipeline[1] entschieden, um ihre Vorteile gegebenfalls für diese Arbeit nutzen zu können.

Die Erstellung einer ersten Szene leitet die Entwicklung der Anwendung ein. Danach folgt der Import des Gebäudemodells, wobei die naheliegendste Methode zur Hinzu-

[1] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@7.1/manual/index.html, besucht am 18.02.20

fügung eines 3D-Modells aus 3Ds Max in ein Unity-Projekt der direkte Import der 3Ds Max-Projektdatei ist. Jedoch ergibt sich folgendes Problem: Sollte das Unity-Projekt später an einem anderen Computer ohne 3Ds Max-Installation geöffnet werden, kann es zu Problemen mit der Darstellung des Modells kommen. Der Autor entschied sich deshalb für einen Export aus der Modellingsoftware in das gängige FBX-Format, in welchem die Datei schließlich in Unity importiert wurde. Das Modell wurde schließlich passend in der Szene platziert.

4.5 Materialien

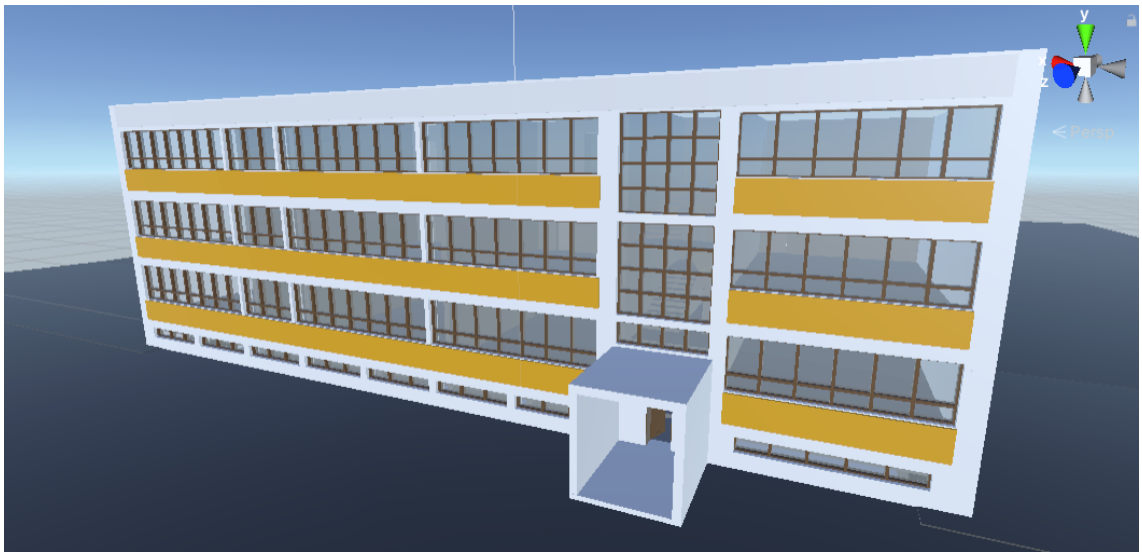

Abbildung 4.5: 3D-Modell des alten Richard-Stücklen Baus mit Materialien [Eigene Abbildung]

Dem Umfang dieser Arbeit entsprechend wurden für die Texturierung einfarbige Materialien benutzt. Durch die verwendete Universal Render Pipeline standen einige dem Autor unbekannte Shader zur Verfügung, weshalb eine kurze Recherche notwendig war, um den für diesen Projekt passenden zu finden. Es wurde sich für den Lit-Shader entschieden, da dieser als universeller Shader von Unity empfohlen wird. Erstellt wurden mehrere verschiedenfarbige Materialien deren Metallic- und Emissionwerte angepasst werden konnten, um unterschiedliche Oberflächen anzudeuten. Des Weiteren wurden für die Darstellung von Fensterglas die sich in den

Farbeinstellungen befindenden Werte des Alphakanals reduziert. Anschließend konnten sie nach Vorlage des realen Richard-Stücklen Baus den verschiedenen Elementen des Gebäudemodells zugewiesen werden. In Abbildung 4.5 und 4.6 sind die fertigen Ergebnisse der mit Materialien ausgestatteten Fassaden zu sehen.

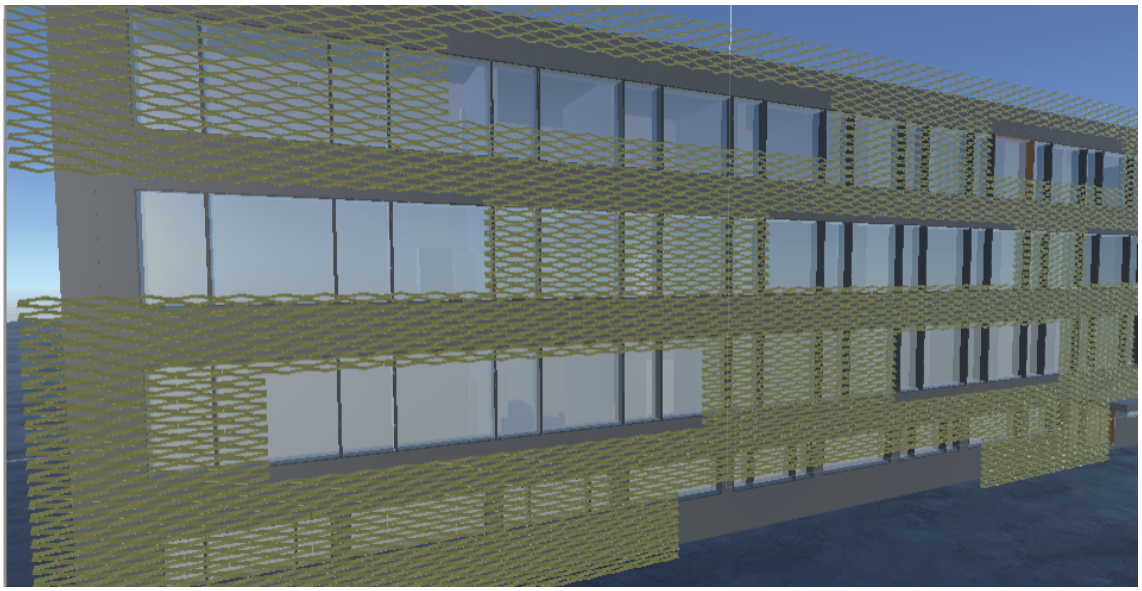

Abbildung 4.6: 3D-Modell des neuen Richard-Stücklen Baus mit Materialien [Eigene Abbildung]

4.6 Begehbarkeit

Die Realisierung der Begehbarkeit des Baus ging mit folgenden Schritten einher:

Zuerst musste ein Player erstellt werden, welcher vom Nutzer gesteuert werden kann. Dazu wurde ein Gameobject erstellt, welchem in der Hierarchie die Hauptkamera untergeordnet wurde. Die Anpassung der Höhenposition der Kamera, sodass diese im korrekten Verhältnis zur Gebäudehöhe steht, wurde später nach den ersten Tests durchgeführt.

Nun wurde ein Skript für die Bewegung benötigt, welches dem Player-Objekt zugeordnet wurde. Im Code-Listing 4.1 befinden sich die wichtigsten Codezeilen, die eine unkomplizierte Umsetzung der Bewegung bieten. In der Update()-Methode werden

in Zeile 9-10 zuerst mögliche Eingaben über die horizontale und vertikale Achse abgerufen und in Variablen gespeichert. Durch die CharacterController-Komponente, die dem Spieler-Objekt zugewiesen wurde, ist es in Zeile 12 möglich die Move()-Methode aufzurufen, um sie mit den Eingabewerten und einer Geschwindigkeitsvariable zu multiplizieren. In den Zeilen 15-16 wird dem Spieler schließlich noch die Einwirkung von Schwerkraft hinzugefügt. Zu beachten ist die Multiplikation mit Time.deltaTime an mehreren Stellen im Code: Dieser Wert liefert die Zeit, die seit dem jeweils letzten Frame vergangen ist. Durch die Verwendung dieser Werte kann sichergestellt werden, dass Geschwindigkeiten, wie die der Spielerbewegung, bei unterschiedlichen Framerates, die beispielsweise aus verschiedenen Hardewareelementen resultieren, immer gleich bleibt.

Schließlich werden Collider benötigt, um aus den Wänden und Böden undurchdringliche Barrieren zu machen. In der Spezifikation wurde festgelegt, dass aus Performanzgründen nach Möglichkeit keine Meshcollider verwendet werden sollen, weshalb Wänden und Böden Boxcollider zugewiesen wurden. Zu dem Zeitpunkt war das in Unterkapitel 4.2 erläuterte Problem, welches die Trennung von AEC-Elementen betrifft, noch nicht bekannt. Auch hier war die Unterteilung eines Treppenobjekts in mehrere Objekte, also zum Beispiel einzelne Stufen, nicht möglich, weshalb die Nutzung von Boxcollidern einen großen Aufwand mit sich gezogen hätte. Auf ein einzelnes Treppenobjekt müssten eine Vielzahl an Collidern gelegt werden, die dann unübersichtlich im Editor angepasst werden müssten. Dies wurde teilweise bei den Wänden so umgesetzt, um die Möglichkeit, sich durch Türöffnungen zu bewegen, zu erhalten, was in Abbildung xx ersichtlich wird. Doch steht der Aufwand dabei in keinem Verhältnis zu dem beim Hinzufügen von Collidern für einzelne Treppenstufen. Aufgrund des Mehraufwands wurde sich dazu entschlossen, die Nutzung von Meshcollidern auszuprobieren. In Tests wurde keine sichtbare Veränderung der Performanz durch die Meshcollider festgestellt, was vermutlich aus ihrer zu geringen Anzahl folgt.

```
1
2 public class Movement : MonoBehaviour {
3
4     public CharacterController controller;
5     public float speed;
6     Vector3 velocity;
```

```
 7
 8       void Update(){
 9           float h = Input.GetAxisRaw("Horizontal");
10           float v = Input.GetAxisRaw("Vertical");
11
12           controller.Move((transform.right * h +
13           transform.forward * v) * speed * Time.deltaTime);
14
15           velocity.y += -9.81f * Time.deltaTime;
16           controller.Move(velocity * Time.deltaTime);
17       }
18   }
```

Listing 4.1: Skript für die Spielerbewegung

Als letzten Schritt für die vollständige Begehbarkeit galt es eine Möglichkeit, die Kamera zu bewegen, zu erschaffen, sodass der Nutzer das Gebäude aus allen Perspektiven betrachten kann. Dafür sorgt ein zweites Skript, welches in Listing 4.2 gezeigt wird. Dieses wurde dem Kamera-Objekt angehängt. In der Update()-Methode werden in den Zeilen 9-10 die Mauseingaben abgefangen, welche schließlich mit einem variablen Sensivitätswert und auch hier mit Time.deltaTime multipliziert werden. Die nun erhaltenen Werte werden in den Rotationsparameter von Spieler und Kamera eingesetzt, sodass der Spieler das Gefühl hat, seinen Kopf horizontal und vertikal bewegen zu können. In den Zeilen 22-27 findet schließlich eine Sperrung der vertikalen Bewegung durch begrenzende Werte statt, um eine realitätsnahe Kopfbewegung zu verwirklichen. Die Kamera wird dafür in der Szene passend ausgerichtet. In der Start()-Methode des Skripts wird außerdem der Mauszeiger gesperrt, sodass dieser beim Bewegen nicht als störend empfunden werden kann.

```
1
2   public class LookAround : MonoBehaviour {
3
4       public float sensitivity;
5       public Transform playerTransform;
6       public Transform cameraTransform;
7
```

```
 8      void Start(){
 9          Cursor.lockState = CursorLockMode.Locked;
10      }
11
12      void Update(){
13
14          float mousex = Input.GetAxis("Mouse X") *
15          sensitivity * Time.deltaTime;
16          float mousey = Input.GetAxis("Mouse Y") *
17          sensitivity * Time.deltaTime;
18
19          playerTransform.Rotate(Vector3.up * mousex);
20          cameraTransform.Rotate(Vector3.left * mousey);
21
22          Vector3 currentRotation
23          = transform.localRotation.eulerAngles;
24          currentRotation.x
25          = Mathf.Clamp(currentRotation.x, 10, 80);
26          transform.localRotation =
27          = Quaternion.Euler(currentRotation);
28      }
29  }
```

Listing 4.2: Skript für die Kamerabewegung

4.7 Fassadenwechsel

Um einen Wechsel zwischen der alten und der neuen Fassade umzusetzen, wurde ein weiteres Skript geschrieben, dessen Code sich in Listing 4.3 befindet.

```
1  public class RemoveFront : MonoBehaviour {
2
3      public GameObject oldFront;
4      public GameObject newFront;
```

```
 5
 6       void Start(){
 7           Button frontSwitch
 8           = gameObject.GetComponent<Button>();
 9           frontSwitch.onClick.AddListener(switchIt);
10       }
11
12
13       public void switchIt(){
14
15       if (newFront.activeSelf && !oldFront.activeSelf){
16               oldFront.SetActive(true);
17               newFront.SetActive(false);
18           }
19
20       else if (oldFront.activeSelf && !newFront.activeSelf){
21               newFront.SetActive(true);
22               oldFront.SetActive(false);
23           }
24       }
25
26       public void Update(){
27           if (Input.GetKeyDown(KeyCode.Space)) {switchIt();}
28       }
29   }
```

Listing 4.3: Skript für die Spielerbewegung

In Zeile 7 wird ein zuvor im Canvas erstellter Button abgerufen. Ihm wird anschließend mithilfe einer in Unity integrierten onClick()-Methode ein Listener hinzugefügt, welcher dafür sorgt, dass die selbst erstellte Methode switchIt() bei jedem Klick mit der Maus auf den Button aufgerufen wird. In der Methode wird dann überprüft, ob zum einen das Gameobject der alten Fassade aktiv, zum anderen das der neuen Fassade inaktiv ist. Ist dies der Fall, so wird die alte Fassade aktiviert und die neue deaktiviert. Im jeweiligen anderen Fall ist das Vorgehen das Gleiche. Das Aktivieren und Deaktivieren von Gameobjects sorgt dafür, dass diese in der Szene sicht- bzw. unsichtbar gemacht werden. Durch die im vorigen Unterkapitel erwähnte

Sperrung des Mauszeigers, die mit der Umsetzung der Kamerabewegung einhergeht, tritt das Problem auf, dass der Fassadenwechsel-Button nicht geklickt werden kann. Zusätzlich zum Button, der nun lediglich in der Android-Version, deren Entwicklung im nächsten Kapitel beschrieben wird, vorhanden ist, wurde ein Wechsel durch die Leertasteneingabe implementiert. Diese wird in Zeile 26 in der Update()-Methode abgefragt und ruft im positiven Fall die schon bekannte switchIt()-Methode auf. Durch die Vollständigkeit aller Funktionen konnte ein Build der Anwendung erstellt werden, wodurch die folgende Evaluation möglich ist.

4.8 Android-Version

Um weitere Vergleichsparameter für die Performanztests im nächsten Kapitel zu erhalten, wurde sich dazu entschieden, die in den Anforderungen als optional eingestufte Anpassung der Software für Android-Geräte durchzuführen. Für die Steuerung über den Touchscreen mussten nur kleine Änderungen der Spieler- und Kamerabewegungen erfolgen.

Der Code, der im Movement-Skript hinzugefügt wurde, ist in Listing 4.4 zu sehen. Um die fehlenden Tasten auf Mobilgeräten zu ersetzen wurde ein Vorwärts-Button implementiert. Ein *Event Trigger*-Komponent ruft ab, wann dieser gedrückt und wieder losgelassen wird und weist so der neu erstellten Boolean-Variable entweder den Wert *false* oder den Wert *true* zu. Diese Variable wird durch ein if-else-Statement in der Update()-Methode abgefragt und der Spieler entsprechend mithilfe die Move()-Methode bewegt.

```
1
2 void Update(){
3     if (ButtonPressed == true){
4         controller.Move(transform.forward *
5         mobileSpeed * Time.deltaTime);
6     }
7     else if (ButtonPressed == false){
8         controller.Move(transform.forward * 0 *
9         Time.deltaTime);
```

```
10      }
11  }
12
13  public void onPress(){
14          ButtonPressed = true;
15  }
16
17  public void onRelease(){
18          ButtonPressed = false;
19  }
```

Listing 4.4: Anpassung der Spielerbewegung für die Android-Version

Das Anpassen der Kamerabewegung erforderte kaum Aufwand. Lediglich der in Listing 4.5 genannte Befehl musste der Update()-Methode des LookAround-Skripts hinzugefügt werden, um ein Springen der Kamera an die Position des letzten Touchpoints auf dem Mobilgerät zu verhindern.

```
1
2  void Update(){
3      if (Input.GetMouseButtonDown(0)) return;
4  }
```

Listing 4.5: Anpassung der Kamerabewegung für die Android-Version

Auch aus der Android-Version konnte ein funktionierender Build erstellt werden, welcher im nächsten Schritt evaluiert werden kann. In den Abbildungen 4.7 und 4.7 ist das Userinterface der Android-Version sichtbar.

Abbildung 4.7: Fertige Android-Anwendung - alte Fassade [Eigene Abbildung]

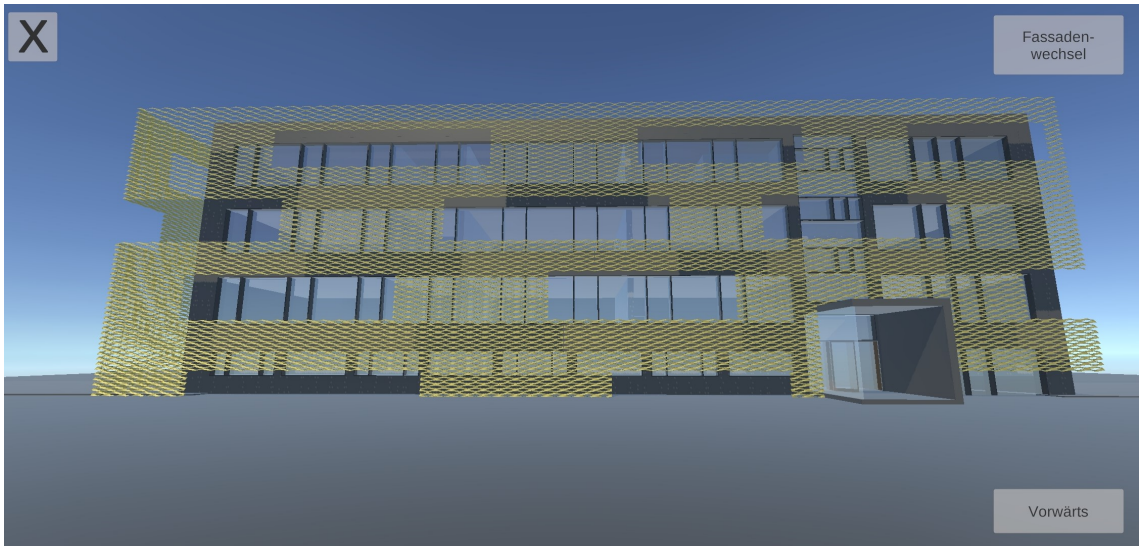

Abbildung 4.8: Fertige Android-Anwendung - neue Fassade [Eigene Abbildung]

5 Evaluation

Die entwickelte Anwendung erfüllt alle in der Spezifikation festgelegten Anforderungen. Ein 3D-Modell des Richard-Stücklen Baus wurde mit 3Ds Max erstellt und in Unity implementiert. Durch die Entwicklung der beiden Fassaden und das Hinzufügen von passenden Materialien ist die optische Annäherung des Modells an das Originalgebäude deutlich zu erkennen; die entsprechenden Vergleiche wurden der Arbeit hinzugefügt. Auch die historische Entwicklung des Gebäudes ist durch die alte und neue Fassade und die Möglichkeit, diese innerhalb der Anwendung zu wechseln, nachvollziehbar. Darüber hinausgehend wurde weitere Interaktivität durch die Begehbarkeit des Modells realisiert. Das markante Goldgitter des Baus wurde so modelliert, dass die Anwendung ohne größere Störungen arbeitet. Die Beeinträchtigung der Performanz durch das Gitter wird in den folgenden Unterkapiteln anhand Tests verdeutlicht. Zusätzlich zur geforderten Desktop-Version wurde die Anwendung auch für die Benutzung auf Android-Geräten angepasst. Durch die so entstandene Mehrzahl an Testgeräten kann eine umfangreichere Datenerhebung erfolgen.

5.1 Performanztest

Zur Bewertung der Performanz der Anwendung kommen zwei Verfahren zum Einsatz: Zum einen die subjektive Einschätzung des Autors und zum anderen die quantitative Bewertung anhand der Framerate. Die Framerate stellt ein wichtiges Maß dar, da sie für mögliches “Ruckeln“ verantwortlich ist. Um aussagekräftige Ergebnisse zu erhalten und statistische Ausreißer zu glätten, werden diese Werte anhand dreier unterschiedlicher Testszenarien gemessen. Zur Bewertung der Ergebnisse erfolgt ein Vergleich mit gängigen Referenzparametern.

Eine Auflistung der verwendeten Geräte und deren Hardware wird dem Test vorangestellt:

Windows-Testgerät:

- Modell: Lenovo Y520-15IKBN
- CPU: Intel Core i5-7300HQ @ 2.50GHz
- GPU: Nvidia GeForce 1050 Ti 4 GB
- RAM: 8.0 GB

Android-Testgerät:

- Modell: Xiaomi Redmi Note 7
- CPU: Qualcomm Snapdragon 660 @ 2.2GHz
- GPU: Adreno 512
- RAM: 4.0 GB

Die Messung der Framerate am Computer wurde durch die Software *Bandicam* durchgeführt. Der Bildschirm wurde während der Testszenarien mit der zugehörigen Framerate aufgenommen und anschließend die Niedrigst- und Höchstwerte entnommen und in den folgenden Tabellen dargelegt. Für die Messung der Werte am Smartphone konnte keine geeignete Software gefunden werden, weshalb sich die Ergebnisse dabei auf die subjektive Einschätzung des Autors beschränken.

Testszenario 1

Beschreibung:

Die Vorderseite des Gebäudes befindet sich zu Beginn zentral und vollständig im Sichtfeld der Kamera und füllt dieses dabei annähernd aus. Der Spieler bewegt sich geradlinig auf das Gebäude zu und bleibt unmittelbar vor der Fassade stehen ohne die Kamera zu rotieren. Es folgt eine Wiederholung des Vorgangs nach dem Fassadenwechsel.

Ergebnisse:

Die Anwendung läuft in der Desktop-Version ohne größeres Ruckeln. Dennoch ist zwischen den Fassadentypen ein Unterschied in der Framerate zu erkennen. Bei ausgeblendetem Goldgitter wird die Bewegung als deutlich “weicher“ wahrgenommen. Die Annäherung an die neue Fassade bei der Android-Version fühlt sich deutlich “ruckliger“ als in der Desktop-Version an. Mit der aktivierten alten Fassade läuft die Android-Version geschmeidig.

	FPS
Alte Fassade	59-62
Neue Fassade	17-24

Tabelle 5.1: Framerate in Testszenario 1 - Desktop-Version [Eigene Darstellung]

Die in Tabelle 5.1 aufgelisteten Framerates untermauern die subjektive Wahrnehmung des Autors. Zudem muss erwähnt werden, dass die Bildfrequenz positiv mit der Annäherung an das Gebäude korreliert, da immer weniger Gitterelemente im Sichtfeld der Kamera erscheinen, wodurch weniger Rechenleistung in Anspruch genommen wird.

Testszenario 2

Beschreibung:

In diesem Fall befindet sich die Rückseite des Gebäudes zu Beginn zentral und vollständig im Sichtfeld der Kamera und füllt dieses dabei annähernd aus. Der Spieler bleibt stehen. Die Kamera wird dreimal um im Uhrzeigersinn rotiert. Das Verfahren wird nach einem Fassadenwechsel wiederholt.

Ergebnisse:

Auch im zweiten Testszenario ist in in der Desktop-Version ein größeres Ruckeln wahrnehmbar. Bei aktiver Gitterfassade ist in der Desktop-Version eine kleine Verzögerung bei der Bewegung der Maus zu erkennen, diese verstärkt sich bei der Android-Version um ein Vielfaches. Bei aktiver alter Fassade sind immernoch keine Störungen zu erkennen.

	FPS
Alte Fassade	59-62
Neue Fassade	15-25

Tabelle 5.2: Framerate in Testszenario 2 - Desktop-Version [Eigene Darstellung]

Die in diesem Szenario getesteten FPS-Werte, welche in Tabelle 5.2 aufgelistet werden, weisen keine nennenswerte Unterschiede im Vergleich zu denen in Testszenario 1 auf.

Testszenario 3

Beschreibung:

Der Spieler befindet sich im zweiten Stock des Gebäudemodells und steht vor dem Treppenhaus-Fenster in Richtung Rückseite des Gebäudes. Die Kamera rotiert in beide Richtungen jeweils einmal.

Ergebnisse:

Der Unterschied zwischen eingeblendeter alter und neuer Fassade ist in diesem Testszenario kaum spürbar. Ein nennenswertes Ruckeln bleibt hier in allen Konstellationen aus.

	FPS
Alte Fassade	59-61
Neue Fassade	30-61

Tabelle 5.3: Framerate in Testszenario 3 - Desktop-Version [Eigene Darstellung]

In Tabelle 5.3 wird die Annahme des Autors bestätigt, dass die Framerate bei Aktivierung der neuen Fassade deutlich höher ist als in den vorigen Szenarios. Dabei erreicht sie teilweise sogar die hier immernoch konstant bleibenden FPS-Werte bei ausgeblendetem Gitter.

5.2 Auswertung der Ergebnisse

Die Einschätzung des Autors zur Performanz der Desktop-Anwendung ist durchaus positiv. Die Anwendung arbeitet selbst bei eingeschaltetem Gitter störungsfrei ohne “Ruckeln“ oder Verzögerungen. Framerates von bis zu 60 FPS stützen die Aussage, dass die Anwendung als performant betrachtet werden kann. Zwar sinkt die Rate bei Aktivierung des Goldgitters teilweise auf Werte von bis zu 17 FPS ab, doch handelt es sich dabei um Ausnahmefälle, die kaum wahrnehmbar sind. Laut Miliano [Mil99] ist eine Framerate von beispielsweise 3-4 FPS inakzeptabel für Präsentationszwecke, da die Orientation des Nutzers darunter leidet. Für die meisten Spiele wird heutzutage eine Framerate von mindestens 60 FPS angestrebt, jedoch werden alle Raten zwischen 30 und 60 FPS als gut spielbar betrachtet [Kla20]. Die oft über 30 FPS liegende Framerate in der vorliegenden Anwendung unterstützt die positive Einschätzung des Autors.

Bei der Android-Version gestaltet sich eine Bewertung schwieriger. Bei deaktiviertem Goldgitter ist das Gefühl auch hier positiv. Der Unterschied beim Testen mit der neuen Fassade ist jedoch deutlich stärker wahrnehmbar als in der Desktop-Version. Die Anwendung stürzt zwar nicht ab und hängt sich auch nicht komplett auf, doch fühlt sich die Bewegung etwas unangenehm und “rucklig“ an.

6 Zusammenfassung und Ausblick

Die im Titel geforderte Aufgabe konnte vollständig und in gegebener Zeit abgeschlossen werden. Nachdem in Kapitel 1 die Thematik vorgestellt und in Kapitel 2 grundlegende Informationen zum besseren Verständnis der Arbeit erläutert wurden, konnte in der Spezifikation in Kapitel 3 die zu entwickelnde Anwendung geplant werden. Der Prozess der Implementierung, welcher in Kapitel 4 dokumentiert wurde, führte im Ergebnis zu einer lauffähigen Anwendung, welche schließlich in Kapitel 5 evaluiert werden konnte. Die folgenden Zeilen dienen nun als kurze Zusammenfassung der entstandenen Arbeit. Außerdem wird ein Ausblick auf die Verwendung der Software und Ansätze zu möglichen Erweiterungen gegeben.

Das 3D-Modell des Richard-Stücklen Baus wurde zur möglichen Verwendung in Projekten der Hochschule Mittweida, wie zum Beispiel dem Projekt *Historisches Mittweida* entwickelt. Die entstandene Software beweist, dass der Einbau des Modells in diese möglich ist, ohne größere Einbußen in Performanz oder Optik hinnehmen zu müssen. Die zu Beginn gestellte Frage, inwieweit eine Applikation, die ein 3D-Modell mit repetitivem Muster enthält performant umgesetzt werden kann, ist damit beantwortbar.

Das Gebäudemodell kann für verschiedene weitere Anwendungsfälle problemlos angepasst und erweitert werden. Durch die Speicherung in gängigen Formaten und einer übersichtlichen Projektstruktur ist eine schnelle Einarbeitung möglich. Auch eine Erweiterung der entstandenen Software ist denkbar.

6.1 Verbesserungsmöglichkeiten

Es gibt viele Verbesserungsmöglichkeiten des Modells des Richard-Stücklen Baus und der Anwendung, deren Ansätze im Folgenden erläutert werden.

Eine grafische Ausarbeitung des Modells könnte durch das Anwenden von realitätsnahen Texturen stattfinden. Durch eine zusätzliche Einarbeitung in Shader und Belichtung in Unity wäre es möglich, das Gebäude innerhalb der Software noch realistischer wirken zu lassen. Auch das Innenleben des Gebäudes könnte in dem Zuge passend texturiert werden.

Des Weiteren wäre es möglich im Inneren des Gebäudes Details, wie zum Beispiel Türklinken, Geländer oder Sockelleisten hinzuzufügen. Zusätzlich wäre auch das Hinzufügen von Einrichtung denkbar, die sich möglicherweise auch an der historischen Entwicklung des Gebäudes orientiert und für das Projekt *Historisches Mittweida* von Vorteil sein könnte.

Eine weitere Möglichkeit wäre es die Anwendung immersiver zu gestalten. Dazu könnte zusätzlich zur erwähnten Belichtung ein passendes Terrain und eine Skybox erstellt werden. Auch die Möglichkeit Fenster und Türen zu öffnen und diese mit Animationen zu versehen erhöht die Chance aus der Software ein Erlebnis zu machen.

Durch weitere Forschung zum Thema Performanz bei mobilen Geräten würde es sich anbieten, die Android-Anwendung noch performanter umzusetzen.

Ein weiterer Meilenstein wäre die Übertragung der Anwendung in die virtuelle Realität, da die Unterschiede hinsichtlich der Performanz einen interessanten Vergleich zur jetzigen Anwendung bieten könnten.

6.2 Fazit

Die entstandene Software erfüllt alle anfänglich benannten Anforderungen. Das Vorgehen bei Planung, Entwicklung und Evaluation dieser wurde durch den schriftlichen Teil der Arbeit vollständig dokumentiert und durch Ansätze für Erweiterungsmöglichkeiten abgerundet.

Literaturverzeichnis

[BKP+10] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez und Bruno Lévy: *Polygon Mesh Processing*, A K Peters, Natick, MA, 3 Aufl., 2010, ISBN 9783446405967.

[Cor01] Clark A. Cory: *Utilization of 2D, 3D, or 4D CAD in construction communication documentation*, in *Proceedings Fifth International Conference on Information Visualisation* (herausgegeben von E. Elanissi, F. Khosrowshahi, M. Sarfraz und A. Ursyn), S. 219–224, IEEE, 3 Park Avenue, 17th Floor, New York, NY 10016-5997 USA, Juli 2001.

[Fin18] Alina Finger: *Konzeption und Realisierung eines virtuellen Rundgangs*, Bachelorarbeit, Hochschule Hannover, März 2018.

[Gos16] Phil Gosch: *3D-Modelloptimierung für mobile Endgeräte*, Okt. 2016, URL: https://codefluegel.com/de/3d-modelle-modelloptimierung-fuer-mobile-endgeraete/, besucht am 16.02.20.

[Haa05] André Oliver Haas: *Virtueller Museumsführer am Beispiel der Stiftskirche Stuttgart. Eine VR-basierte Edutainment-Anwendung in der CAVE^{TM}*, Diplomarbeit, Hochschule der Medien, März 2005.

[HZ14] Peng Hu und Kai Zhu: *Strategy research on the performance optimization of 3D mobile game development based on Unity*, *Journal of Chemical and Pharmaceutical Research*, Bd. 6(3):S. 785–791, 2014.

[Kla20] Michael Klappenbach: *Understanding and Optimizing Video Game Frame Rates*, Febr. 2020, URL: https://www.lifewire.com/optimizing-video-game-frame-rates-811784, besucht am 01.04.20.

[LV02] Jean-François Lapointe und Norman G. Vinson: *Effects of JoystickMapping and Field-of-View on Human Performance in Virtual Walkthroughs*, in *Proceedings. First International Symposium on 3D Data Processing Visualization and Transmission* (herausgegeben von Bob Werner), S. 490–493, IEEE, 3 Park Avenue, 17th Floor, New York, NY 10016-5997 USA, Juni 2002.

[Mil99] Vito Miliano: *Application of a 3D Game Engine to Enhance the Design, Visualization and Presentation of Commercial Real Estate*, 1999.

[MKSB16] Farouk Messaoudi, Adlen Ksentini, Gwendal Simon und Philippe Bertin: *Performance Analysis of Game Engines on Mobile and Fixed Devices*, *ACM Transactions on the Web*, Bd. 9(4), März 2016, Artikel 39.

[Pat12] Nikul P. Patel: *Survey On 3D Interactive Walkthrough*, *International Journal of Engineering Research Technology*, Bd. 1(9), 2012.

[Ple19] Kat Plewa: *CAD vs 3D modeling software: what is the difference?*, März 2019, URL: https://www.sculpteo.com/blog/2019/03/19/cad-vs-3d-modeling-software-what-is-the-difference/, besucht am 16.02.2020.

[PSL12] Behrang Parhizkar, Kanammal A/P Sandrasekaran und Arash Habibi Lashkari: *Motion Detection Real Time 3D Walkthrough in Limkokwing University of Creative Technology (ModetWalk) using Kinect XBox*, *IJCSI International Journal of Computer Science Issues*, Bd. 9(6):S. 100–109, 2012.

[Sch06] David Scherfgen: *3D-Spiele-Programmierung*, Carl Hanser Verlag, München and Wien, 3 Aufl., 2006, ISBN 9783446405967.

[SKF08] Mohd.Fairuz Shiratuddin, Kevin Kitchens und Desmond Fletcher: *Virtual Architecture. Modeling and Creation of Real-Time 3D Interactive Worlds*, Lulu Press, USA, 2008, ISBN 978-1-4357-5642-7.

[ST02] Mohd.Fairuz Shiratuddin und Walid Thabet: *Virtual Office Walkthrough Using a 3D Game Engine*, Virginia Polytechnic Institute and State University, 2002.

[SZ01] Mohd.Fairuz Shiratuddin und Abdul Nasir Zulkifli: *Making Virtual Reality a Reality: Bringing CAD and Game Engine together*, in *Proceedings of the International Conference on Information Technology and Multimedia at UNITEN*, Malaysia, Aug. 2001.

[Tho18] Andrew Simon Thomas: *CAD vs. Modeling: Which 3D Software to Choose?*, Jan. 2018, URL: https://www.shapeways.com/blog/archives/27653-cad-vs-modeling-which-3d-software-to-choose.html, besucht am 16.02.2020.

[Tof19] Marcus Toftedahl: *Which are the most commonly used Game Engines?*, Sept. 2019, URL: https://www.gamasutra.com/blogs/MarcusToftedahl/20190930/350830/Which_are_the_most_commonly_used_Game_Engines.php, besucht am 16.02.20.

Selbstständigkeitserklärung

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

Mittweida, den 3. April 2020

Nicole Tauber