HAW HAMBURG

BACHELOR THESIS
Mirco Wiedenfeld

Ein CityGML-Parser für die Unreal Engine - ein Baustein für akustische Simulation im städtischen Raum

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Mirco Wiedenfeld

Ein CityGML-Parser für die Unreal Engine - ein Baustein für akustische Simulation im städtischen Raum

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Birgit Wendholt
Zweitgutachter: Prof. Dr. Philipp Jenke

Eingereicht am: 24. Oktober 2024

Mirco Wiedenfeld

Thema der Arbeit

Ein CityGML-Parser für die Unreal Engine - ein Baustein für akustische Simulation im städtischen Raum

Stichworte

CityGML, Akutische Simulationen, Unreal Engine

Kurzzusammenfassung

Lärm stellt vor allem in Städten ein zunehmendes Problem dar, das zu Schlafproblemen, Stress und weiteren gesundheitlichen Beeinträchtigungen führen kann. Um Lärmquellen gezielt reduzieren zu können, ist es notwendig, verschiedene Szenarien mit Lärmquellen nachzubilden, was durch akustische Simulationen in Virtual Reality (VR) immersiv erreicht werden kann. Zur Realisierung solcher Simulationen bieten sich Game Engines als Plattform an, da sie realistische 3D-Umgebungen mit hoher Interaktivität unterstützen. Für die Nachbildung städtischer Gebiete werden Geodaten im CityGML Format verwendet, da dieses realitätsnahe Gebäudemodelle in verschiedenen Levels of Detail (LoDs) bereitstellt. Ziel dieser Arbeit ist es, ein Tool zu entwickeln, das CityGML Daten in eine Game Engine importiert und so die Grundlage für akustische Simulationen in VR schafft. Dazu wurden die Konzepte CityGML, akustische Simulationen und Game Engines wissenschaftlich untersucht, um deren technische Voraussetzungen und Kompatibilität für dieses Vorhaben zu bewerten. Als Ergebnis dieser Arbeit wurde ein Plugin für die Unreal Engine entwickelt, mit dem CityGML Dateien in unterschiedlichen LoDs und Versionen in die Unreal Engine importiert werden können. Dieses Plugin kann im Projekt "Sound Reality" , in dessen Rahmen diese Arbeit geschrieben wird, als Basis für die Erstellung akustischer Simulationen in urbanen Gebieten eingesetzt werden.

Mirco Wiedenfeld

Title of Thesis

A CityGML-parser for the Unreal Engine - a building block for acoustic simulation in urban space

Keywords

CityGML, acoustic simulation, Unreal Engine

Abstract

Noise is an increasing problem, especially in urban areas, leading to sleep disturbances, stress, and other health issues. In order to effectively reduce noise, it is essential to simulate various scenarios with noise sources, which can be achieved immersively through acoustic simulations in virtual reality (VR). Game engines provide an ideal platform for acoustic simulations, as they support highly interactive and realistic 3D environments. Geodata in CityGML format is used to simulate urban areas, as it provides accurate and realistic building models in various levels of detail (LoDs). This thesis aims to develop a tool that imports CityGML data into a game engine, laying the foundation for VR-based acoustic simulations. To achieve this, the concepts of CityGML, acoustic simulations, and game engines were analyzed to evaluate their technical requirements and compatibility for this purpose. As a result of this work, a plugin for the Unreal Engine was developed, enabling the import of CityGML files in different LoDs and versions. This plugin serves as a foundational tool for the "Sound Reality" project within which this thesis was conducted and provides the basis for creating acoustic simulations of urban environments.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Zielsetzung . . . 2
1.3 Gliederung der Arbeit . . . 3

2 Verwandte Arbeiten **5**
2.1 CityGML . . . 5
2.2 CityGML in der Stadtmodellierung . . . 9
2.3 Akustische Simulation und Urbanisierung . . . 11
2.4 Game Engines in der Akustischen Simulation . . . 17
2.5 Zusammenfassung . . . 24

3 Anforderungsanalyse **26**
3.1 Einordnung in das Projekt . . . 26
3.2 Funktionale Anforderungen . . . 28
3.3 Nichtfunktionale Anforderungen . . . 29
3.4 Zusammenfassung . . . 30

4 Entwurf und Umsetzung **31**
4.1 Architektur der Zielumgebung . . . 31
4.2 CityGML Datenlage . . . 36
4.3 Funktionsweise des Plugins . . . 39
4.4 Meshgenerierung für die Unreal Engine . . . 42
4.4.1 FAN-Algorithmus . . . 42
4.4.2 Earcut-Algorithmus . . . 43
4.4.3 Beleuchtung und Texturierung . . . 45

4.5 Ergebnisse 46
4.5.1 Darstellungsvarianten der Hafencity 47
4.5.2 Skalierung auf die gesamte Stadt 48
4.5.3 Verarbeitung von CityGML Daten außerhalb Hamburgs 49
4.6 Probleme bei der Umsetzung und Erfahrungsbericht 51
4.7 Zusammenfassung 54

5 Fazit **56**
5.1 Optimierung und Ausblick 58

Literaturverzeichnis **61**

Selbstständigkeitserklärung **65**

Abbildungsverzeichnis

4.1 Unreal Engine Architektur Schaubild 32
4.2 Einbindung des CityGML-Parsers in die Architektur der Unreal Engine . . 34
4.3 Funktionsweise des CityGML-Parsers 40
4.4 Durch den FAN Algorithmus zerlegtes Polygon 43
4.5 Polygon FAN-Algorithmus Fehler 44
4.6 Earcut zerlegt konkaves Polygon 44
4.7 LoD 1 Haus 45
4.8 LoD 2 Haus 45
4.9 LoD 3 Haus 45
4.10 LoD 1 texturiertes Haus 46
4.11 LoD 2 texturiertes Haus 46
4.12 LoD 3 texturiertes Haus 46
4.13 Hafencity LoD3 47
4.14 Hafencity LoD2 47
4.15 Hafencity texturiert in LoD 2 48
4.16 Hamburg Vogelperspektive in LoD 2 48
4.17 Hamburg nördlich der Elbe in LoD 2 48
4.18 Berlin, Jüdisches Denkmal LoD 2 50
4.19 Dortmund Schnarhorst LoD 2 50
4.20 Stuttgart Hbf. von oben LoD 2 50
4.21 Haus mit einer fehlerhaften Wand LoD 3 52

1 Einleitung

1.1 Motivation

Lärm kann zu Schlafproblemen, Konzentrationsproblemen und allgemein Stressreaktionen führen. Die Reduzierung von Lärmquellen ist also wichtig, um eine hohe Lebensqualität zu gewährleisten. Um Lärmquellen reduzieren zu können, ist es hilfreich, verschiedene Szenarien mit Lärmquellen nachbilden zu können. Der klassische Weg, die Ausbreitung von Schall und Lärmpegel zu visualisieren, sind 2D Lärmkarten. Für die meisten Menschen ist es jedoch schwer, rein visuelle Lärmkarten auf die akustische Wahrnehmung zu übertragen. Ein immersiverer Ansatz zur Veranschaulichung von Lärmszenarien ist die akustische Simulation in der Virtual Reality (VR). Durch diesen Ansatz können die Geräusche der Umgebung nicht nur dargestellt, sondern auch realistisch erfahrbar gemacht werden. Dies ermöglicht es uns, potentielle Auswirkungen auf die Lebensqualität zu analysieren, indem nicht nur reine Tonaufnahmen abgespielt werden, sondern eine Interaktion mit der virtuellen Umgebung ermöglicht wird. In einer solchen Simulation wäre es möglich, sich frei in der Umgebung zu bewegen und die Geräuschkulisse aus verschiedenen Perspektiven zu erleben. So könnten Stadtplaner und Architekten Maßnahmen zur Lärmreduktion wie beispielsweise Schallschutzwände oder Begrünungen virtuell abbilden und bereits in der Planung testen und berücksichtigen, um städtische Räume den Bedürfnissen der Anwohner entsprechend zu gestalten.

Im Kontext der akustischen Simulationen gewinnen vor allem Game Engines an Bedeutung, da sie es ermöglichen, komplexe virtuelle Welten mit hoher Interaktivität zu erschaffen. Einerseits werden realistische und detailreiche 3D Umgebungen ermöglicht, was für die Simulation von Schallausbreitungsphänomenen Voraussetzung ist. Andererseits sind Game Engines leistungsstark und interaktiv, was besonders wichtig für die subjektive Wahrnehmung der Menschen ist, welche die Szenarien erleben und testen sollen. Das macht sie zu einer idealen Plattform für akustische Simulationen.

Daher wird im Projekt "Sound Reality", in dessen Rahmen diese Bachelorarbeit geschrieben wird, eine Game Engine als Plattform genutzt, um realistische akustische Simulationen für die Virtual Reality zu entwickeln. "Sound Reality" ist ein Projekt, welches sich mit Maßnahmen zur Lärmminderung in Städten beschäftigt. Es baut auf den Erkenntnissen der vorherigen Projekte "X-Eptance Impulse" (XEI), "X-Eptance Explore" (XEE) und "Open Citizen Soundwalks" (OCSW) auf. Aus dem Projekt XEI ging eine umfangreiche Geräuschdatenbank hervor, die im Projekt XEE genutzt wurde, um eine 3D-Nachbildung eines realen Windparks mit virtueller Akustik umzusetzen. Im dritten Projekt OCSW wurde die Umgebung, die aus dem Projekt XEE hervorging, genutzt, um die Akzeptanz der Bevölkerung gegenüber Windkraftanlagen zu erhöhen. Dafür wurden Soundwalks in VR durchgeführt und Befragungsergebnisse digital erfasst. Da wir in einer Zeit leben, in der die Urbanisierung schnell voranschreitet, soll nun "Sound Reality" als Nachfolgeprojekt von OCSW den Fokus von Windparks in den städtischen Bereich wechseln. Besonders im städtischen Bereich ist es wichtig, Gebäude mit möglichst hoher Präzision darzustellen, da Schall von Gebäuden reflektiert oder abgedämpft wird. Um die städtische Umgebung präzise zu simulieren, ist ein Tool erforderlich, das reale Gebäude in die Zielumgebung des Projekts "Sound Reality" integriert. Im nächsten Kapitel werden Ziele dieses Tools vorgestellt.

1.2 Zielsetzung

Es soll ein Tool entwickelt werden, welches den halbautomatischen Import visueller 3D Geometriedaten für den städtischen Raum in die Zielumgebung des Projekts "Sound Reality" ermöglicht, damit auf dieser Basis akustische Simulationen erstellt werden können.

Um nicht jedes Gebäude manuell modellieren oder geographisch beschreiben zu müssen, sollen frei verfügbare Geodaten für den Import genutzt werden. OpenStreetMap (OSM) und CityGML (City Geography Markup Language) bieten sich dafür an. Beides ist frei verfügbar und stellt 3D Gebäudedaten zur Verfügung. OSM wird von einer breiten Community gepflegt und aktuell gehalten. Wogegen CityGML von Fachleuten und Regierungsbehörden erstellt und aktualisiert wird. Dadurch sind CityGML Daten genauer und die Gefahr, dass Falschinformationen enthalten sind, ist geringer. Der Nachteil ist dabei, dass die Daten in vielen Regionen noch nicht existieren.

In diesem Fall soll CityGML benutzt werden, da auf dem Geoportal Hamburg jährlich aktualisierte Dateien zur Verfügung gestellt werden und die Beschaffung der Daten daher kein Problem darstellt. Die Verwendung von CityGML ermöglicht es zusätzlich, die Gebäude in unterschiedlichen Detailgraden durch die Unterstützung von Levels of Detail (LoD) darzustellen. Dies ermöglicht eine Abwägung zwischen der Genauigkeit der Gebäudeabbildung und der Performance, sowohl beim Import als auch während der Laufzeit, da höhere LoD-Stufen eine detailliertere Darstellung bieten, die jedoch mit einem höheren Rechenaufwand verbunden ist. Zusätzlich werden in CityGML auch Informationen wie Materialbeschaffenheit und strukturelle Eigenschaften beschrieben. Dadurch werden die Simulationen noch deutlich realistischer und die Aussagekraft der Auswertung von Simulationen steigt.

Das Ziel dieser Bachelorarbeit ist, öffentlich verfügbare Geodaten (CityGML Dateien) in die Game Engine des bestehenden Projektes "Sound Reality" importieren zu können. Der Importprozess soll effizient gestaltet werden, da eine CityGML Datei nur einen Quadranten einer Stadt bzw. Region abdeckt und daher oftmals mehrere CityGML Dateien importiert werden müssen. Darüber hinaus soll eine stabile Performance zur Laufzeit gewährleistet sein, da eine schlechte Performance die Immersion der Simulation erheblich beeinträchtigen würde. Es soll außerdem experimentell entschieden werden, bis zu welchem Level of Detail die CityGML Dateien geladen werden sollen. Die Mischung aus möglichst hohem Detailgrad, welcher sich erheblich auf die User Experience auswirkt, und einer stabilen Performance während der Laufzeit ist dabei entscheidend.

1.3 Gliederung der Arbeit

Die Gliederung der Arbeit teilt sich wie folgt auf: Zuerst werden in Kapitel 1 die Motivation und Zielsetzung der Arbeit beschrieben. Kapitel 2 widmet sich den theoretischen Grundlagen und erläutert die für diese Arbeit relevanten Konzepte CityGML, akustische Simulationen und Game Engines, die dabei in den entsprechenden wissenschaftlichen Kontext eingeordnet werden. In Kapitel 3 folgt eine Anforderungsanalyse, die sowohl die funktionalen als auch nicht funktionalen Anforderungen des CityGML-Parsers beschreibt. Basierend auf diesen Anforderungen werden in Kapitel 4 der Entwurf und die Umsetzung des CityGML-Parsers erläutert. Hierbei wird auch auf die erzielten Ergebnisse, identifizierten Probleme und mögliche Erweiterungen eingegangen. Abschließend fasst

Kapitel 5 die Ergebnisse der Arbeit zusammen und bietet einen Ausblick auf mögliche Optimierungen und die zukünftige Anwendbarkeit des CityGML-Parsers.

2 Verwandte Arbeiten

In diesem Kapitel werden relevante Arbeiten und Forschungsergebnisse vorgestellt, die den theoretischen Hintergrund dieser Bachelorarbeit bilden. Zunächst wird die City Geography Markup Language (CityGML) als zentrales Datenmodell für die Darstellung von städtischen 3D-Geodaten erläutert (2.1). Anschließend wird die Anwendung von CityGML in der Stadtmodellierung behandelt, wobei die Erstellung digitaler Zwillinge und die damit verbundenen Vorteile und Herausforderungen beleuchtet werden (2.2). Im dritten Unterkapitel (2.3) werden Systeme/Anwendungen zur akustischen Simulation im urbanen Raum behandelt. Es wird gezeigt, wie unterschiedliche Bereiche der akustischen Simulation dafür genutzt werden, Lärmkarten zu erstellen oder Klanglandschaften in Städten zu verbessern. Das nächste Unterkapitel (2.4) diskutiert die Eignung von Game-Engines für akustische Simulationen. Es wird darauf eingegangen, was Game-Engines sind, welche Schwächen deren Sound-Engines haben und wie diese verbessert werden können. Danach wird erläutert, wieso Unreal Engine eine gute Wahl für die akustische Simulation ist, einschließlich einer Analyse von Projekten sowie deren Erfolgen. Zum Schluss werden die wichtigsten Aussagen zusammengefasst (2.5).

2.1 CityGML

City Geography Markup Language (CityGML) ist ein offenes Datenmodell und ein XML-basiertes Dateiformat zur Darstellung von städtischen 3D-Geodaten. Es basiert auf Geography Markup Language Version 3.1.1 (GML3) und wurde entwickelt, um komplexe städtische Strukturen und ihre Attribute in einer standardisierten und interoperablen Weise zu beschreiben.[16]

CityGML hat viele Anwendungsbereiche, wie zum Beispiel die Stadtvisualisierung, Stadt- und Verkehrsplanung, Solarpotentialanalyse, Hochwasser-, Windkraft- und Geräuschsimulationen. Bevor es CityGML gab, hat es eine Vielzahl von Insellösungen, um einzelne

der eben genannten Anwendungsbereiche abzudecken, gegeben. Für die meisten Anwendungen oder Projekte, die sich in ihrem Zweck oder einfach geographisch unterschieden, gab es ein eigenes Modell zur "Stadtmodellierung". Diese Stadtmodelle wurden nach Abschluss oder Abbruch des Projekts fast nie wiederverwendet, da die Modelle genau auf das jeweilige Projekt zugeschnitten waren. Das hat zu viel redundanter oder zumindest ähnlicher Arbeit für neue Projekte geführt. Aus diesem Grund wurde in Nordrhein-Westfalen die Special Interest Group 3D (SIG3D) gegründet. Die SIG3D setzt sich aus zahlreichen Vertretern aus den Bereichen Wirtschaft, öffentlicher Verwaltung und Forschung zusammen und arbeitet seit ihrer Gründung im Jahr 2002 an CityGML.[16]

Im Jahr 2008 wurde CityGML 1 veröffentlicht. CityGML fasst die grundlegenden Definitionen von Entitäten, Attributen und Relationen eines 3D-Stadtmodells im "CityGML-Core" zusammen. Dieser bildet die Grundlage für die verschiedenen Erweiterungsmodule, auch *Themes* genannt. Die *Themes* stellen die semantische Modellierung für verschiedene Anwendungsbereiche dar. Die *Themes* decken die Bedürfnisse der meisten Nutzer von Stadtmodellen ab. Darüber hinaus haben die Mitarbeiter von SIG3D viel Zeit in die Definition der Core-Elemente investiert, so dass es für viele Anwendungsbereiche einfach ist, ihre Domäne durch Modellierung hinzuzufügen. Dadurch ist CityGML in der Lage, viele verschiedene Zielgruppen anzusprechen. So konnte sich CityGML in den letzten Jahren als Standard durchsetzen und im September 2021 wurde mit CityGML 3.0 die bisher neueste Version veröffentlicht.[16]

Beispielsweise gibt es das *Theme* "Transportation", das verwendet wird, um die Transporteigenschaften einer Stadt darzustellen. Damit sind Straßen, Bahnstrecken und Ähnliches gemeint. Das *Theme* "CityFurniture" ist für nicht bewegbare Objekte wie Straßenlaternen, Verkehrsschilder und Straßenbänke, aber auch für Sensoren, Kameras und Mikrofone da, die in der Stadt verteilt sind, um bestimmte Werte wie z. B. die Lautstärke zu messen. Von den insgesamt 11 *Themes* sind für diese Arbeit die *Themes* "Generics" und "Appearance" von Interesse. Mit "Appearance" kann das Aussehen von CityGML-Eigenschaften modelliert werden; also hauptsächlich beobachtbare Eigenschaften von Oberflächen, aber auch Infrarotstrahlung, Lärmdurchlässigkeit oder Erdbebenfestigkeit. Da CityGML als anwendungsunabhängiges Modell konzipiert ist, spezifische Anwendungen aber immer zusätzliche Informationen benötigen, gibt es in CityGML die Möglichkeit, über das *Theme* "Generics" weitere Attribute und Eigenschaften zu modellieren. Dazu muss die Klasse CityObject aus dem CityGML-Core um Name-Wert-Paare erweitert werden, die sich auf das Objekt beziehen, dem sie hinzugefügt werden sollen.[12]

Generische Attribute haben jedoch auch Nachteile. Sie können beliebig in CityGML-Dokumenten vorkommen, ohne dass es eine formale Spezifikation des Namens, des Datentyps oder der Multiplizität (Beziehung) gibt. Daher gibt es für Anwendungen keine Garantie, dass Attribute in Instanzen vorkommen, und es können keine Aussagen über Kardinalitäten gemacht werden. Zudem ist es nicht möglich, Attribute als Liste oder als benutzerdefinierten Datentyp zu modellieren. Diese Attribute werden nicht durch einen XML-Parser validiert, was zu semantischen Interoperabilitätsfehlern führen kann. Darüber hinaus ist die Struktur der generischen Attribute sehr eingeschränkt, da nur eine kleine Anzahl von vordefinierten Datentypen für generische Attribute verwendet werden kann. Aufgrund dieser Nachteile gibt es eine andere Möglichkeit, CityGML um Attribute und zusätzlich auch um Objekte zu erweitern, nämlich über eine Application Domain Extension (ADE).

Ein ADE ist ein zusätzliches formales XML-Schema, das den CityGML Schemadefinitionen folgt. Dadurch können die Dokumente sowohl gegen CityGML als auch gegen das ADE-Schema validiert werden, wodurch Interoperabilität gewährleistet wird. Der "Namespace" von ADE muss sich von allen anderen "Namespaces" unterscheiden, um Doppelbenennungen und die damit verbundenen Interoperabilitätsfehler auszuschließen. Im XML-Dokument können Objekte und zusätzliche Attribute für bestehende Objekte mit beliebigem Datentyp definiert werden. Beispielsweise ist es möglich, ein ADE für Lärmkarten zu erstellen, indem zwei Erweiterungen definiert werden: eine für alle Gebäude, da diesen nun zusätzliche Attribute hinzugefügt werden müssen, und eine für Lärmschutzwände, einem weiteren Objekttyp, der für Lärmkarten benötigt wird. Im XML-Schema wird nun ein Namensraum angegeben, die neuen Attribute für die Gebäude, d. h. der Rauschreflexionskorrekturwert und die Einwohnerzahl, werden hinzugefügt und das neue Objekt "Lärmschutzwand" wird definiert.[12]

ADE's machen die Verwendung von CityGML zwar komplexer, aber für die Möglichkeit, interoperable Erweiterungen für entweder vollständig neue oder erweiterte Eigenschaften hinzuzufügen, gibt es keine Alternativen in CityGML.

Weil sich CityGML als Standard für die Repräsentation und Verteilung von 3D Städte Modellen durchsetzen konnte, wird es heute weltweit eingesetzt. In Deutschland hat jedes Bundesland die Gebäude seines Landes modelliert. Weltweit wird es in vielen Städten in Nordamerika, Australien, Europa und Asien eingesetzt. Die Datensätze können in unterschiedlicher Qualität vorliegen. Das hängt vom Level of Detail (LOD) ab. Es gibt 5 LOD, wobei 0 sehr einfach und 4 am komplexesten ist. LOD 0 ist eine 2,5-dimensionale Dar-

stellung der Landschaft. Bei LOD 1 werden Gebäude dargestellt, aber nur als Rechteck ohne Dach. Mit LOD2 kommen einige Details an den Außenwänden der Gebäude hinzu, wie z. B. ein Dach, Balkone und Treppen. Ab LOD3 kommt die Architektur des Hauses hinzu (Fenster, Türen und detaillierte Wände). LOD4 fügt die Inneneinrichtung durch Räume mit Möbeln, aber auch Treppen und Aufzüge hinzu. Die meisten Datensätze sind bis LOD 2 verfügbar, da LOD 3 und 4 nicht so häufig verwendet werden wie LOD 1 und 2. Die meisten Gebäude können bis LOD 2 auch automatisch generiert werden. Ab LOD 3 müssen Gebäude teilweise vollständig manuell mit einer Modellierungssoftware wie CAD erstellt werden, was einen hohen manuellen Aufwand bei geringem Nutzen bedeutet.[16][20]

Bevor Projekte mit CityGML durchgeführt werden können, müssen zunächst die Daten für CityGML gesammelt werden. Generell gibt es verschiedene Möglichkeiten, diese Daten zu erheben, z. B. durch Luftbilder, Architekturzeichnungen, prozedurale Modellierung, aus bestehenden Modellen oder Laserscanning. Für ein Projekt in Nordrhein-Westfalen, bei dem Lärmkarten erstellt werden sollen, wurden diese Daten von verschiedenen Geodatenbehörden des Landes zusammengetragen. In diesem Beispiel wurden verschiedene Modelle und Daten aus unterschiedlichen Quellen bezogen:

- Das Digital Terrain Model (DTM) wurde vom Vermessungsamt NRW mittels Laserscans erzeugt und für das Projekt bereitgestellt.
- Ein 3D-Blockmodell von 10 Millionen Gebäuden wurde von derselben Behörde bereitgestellt und konnte dadurch in das DTM integriert werden.
- Die Straßen- und Bahnstreckendaten stammen aus dem AKTIS-System (Authoritative Topographic and Cartographic Information System), das ebenfalls vom Vermessungsamt verwaltet wird. Es enthält bereits einige Attribute wie die Funktion der Straße, den Namen und die Fahrbahnbreite für CityGML.
- Informationen zu den Einwohnern wurden von den Gemeinden zur Verfügung gestellt oder es wurden Schätzungen eingetragen, wenn keine Daten vorhanden waren.
- Der Landesbetrieb Straßenbau Nordrhein-Westfalen hat weitere erforderliche Daten zu den Landesstraßen bereitgestellt, die in das Modell integriert wurden.

Auch wenn die verantwortlichen Parteien hier gut zusammengearbeitet haben, kostet die Datenbeschaffung viel Zeit und damit Geld. Aus diesem Grund werden CityGML-Projekte oft nur dann gestartet, wenn bereits Geodaten vorhanden sind. Oft handelt

es sich dabei nicht um CityGML-Dateien, sondern um andere Formate wie .tiff oder .shape, die dann mit variierendem Aufwand in CityGML-Dateien umgewandelt werden können.[4][6]

Woher die Daten kommen, ist aber nicht das einzige Problem. Die nächste Frage ist, wie man die CityGML-Daten verwenden und hosten kann. Auch hier gibt es verschiedene Ansätze. Da CityGML von GML abstammt und es bereits viele WebTools für GML gibt, hat man hier eine Auswahl. Beispielsweise wurden im Projekt für die Erzeugung von Lärmkarten[4] standardisierte Webdienste des Open Geospatial Consortium (OGC) für umfangreiche landesweite Umweltdaten implementiert. Diese Architektur ermöglicht ein weitgehend automatisiertes Verfahren für den Zugriff auf Webdienstdaten, die Berechnung der Lärmimmissionen und die Bereitstellung der Ergebnisse über Portale. Sie basiert zudem darauf, dass CityGML als gemeinsames Austauschformat zwischen den 3D-Geodatendiensten genutzt wird.[4]

CityGML ist ein Standard für 3D-Visualisierungen (insbesondere in Städten). Es bietet die Möglichkeit, über das *Theme* "Generics" oder die ADEs für verschiedene Anwendungsfälle erweitert zu werden, wie z. B. die akustische Simulation. Für die Stadt Hamburg existieren zudem bereits CityGML-Dateien in LoD 1, 2 und 3, was für das Projekt "Sound Reality" ausreichend ist. Seit CityGML 3.0 ist es zudem möglich, akustische Sensordaten zu integrieren, wodurch auch Mikrofonaufzeichnungen eingebunden werden können. Somit ist CityGML für akustische Simulationen im urbanen Raum gut geeignet, und die Datensammlung, die für einen Raum wie Hamburg sehr zeitaufwändig wäre, ist bereits abgeschlossen.

2.2 CityGML in der Stadtmodellierung

Mehr als die Hälfte der Weltbevölkerung lebt in Städten. Bis 2050 sollen es 70% werden, was eine zusätzliche urbane Fläche von 1,2 Millionen Quadratkilometern bedeutet. Diese schnelle Ausbreitung erfordert neue Ansätze für die Stadtentwicklungsplanung. Digitale Zwillinge haben sich über die letzten Jahre immer mehr als geeignetes Mittel in diesem Kontext etabliert. Ein digitaler Zwilling ist eine virtuelle Repräsentation der physischen Stadt. CityGML ermöglicht nicht nur die Visualisierung von Städten, sondern auch die Modellierung einer Vielzahl von Zusatzinformationen. Dadurch können praktisch alle Aspekte einer Stadt in einen digitalen Zwilling integriert werden, wodurch eine effizientere und nachhaltigere Nutzung der Ressourcen für verschiedene Stadtentwicklungsprojekte

ermöglicht wird. Zudem können auch viele Probleme im Bereich des Krisenmanagements auf diese Art und Weise angegangen werden und es wird zusätzlich Transparenz für die Bürger geschaffen.[32]

Ein relevantes Beispiel für die Nutzung von CityGML ist das Projekt "Plateau" in Japan. Es wird nicht zur Modellierung einer einzelnen Stadt verwendet, sondern für eine große Fläche des Landes. In dem modellierten Gebiet leben 40% der japanischen Bevölkerung, das sind 49,7 Millionen Menschen. Für dieses Gebiet werden 18,3 Millionen Objekte in LoD 1 modelliert. Japan ist ein Erdbebengebiet und CityGML hat Katastrophenrisiko Informationen direkt in CityGML über das *Theme* "Land Use" hinterlegt. Deswegen, und weil im Jahr 2020, indem das Projekt ausgeführt wurde, CityGML schon als internationaler Standard mit vielen Praxisbeispielen etabliert war, wurde CityGML bei diesem Projekt genutzt.

Es wurden 150 Städte für insgesamt 40 Anwendungsbereiche modelliert. Im Gegensatz zu ähnlichen Projekten wurde aber nur eine große homogene Karte erzeugt, anstatt viele auf Städte oder Regionen aufgeteilte Einzelkarten zu erstellen. Mit CityGML ist die Abdeckung der Gebäude im digitalen Zwilling höher als in früheren Versuchen, die beispielsweise mit OpenStreetMap schon bestanden. In OSM waren zum Beispiel nur ein Drittel aller Bestandsgebäude modelliert. Im Rahmen des Projekts Plateau wurden in einigen Gebieten 100% der identifizierten Gebäude modelliert, beispielsweise in Tokio.

Die Karte ist über die Webseite "Plateau View" im Webbrowser einsehbar. Plateau View basiert auf FOSS4G (Free and Open Source Software for Geo-spatial), einer Open-Source-Software zur Verarbeitung und Visualisierung von geografischen Daten. Über die 3D-Stadtmodelle mit dem Webbrowser zugänglich gemacht werden.

In der zweiten Phase des Projekts wurden 57 Demonstrationsprojekte gestartet, um das Potential von "Plateau" zu demonstrieren und Anreize zu schaffen. Von den 57 Projekten, welche eine Vielzahl der 40 Anwendungsbereiche widerspiegeln sollten, wurden die meisten für die Städteplanung, Disaster-Simulationen und VR-Simulationen genutzt. Das Projekt wird alle 5 Jahre aktualisiert und die GUI wird laufend überarbeitet. Das ist vor allem deshalb wichtig, da bei solchen Projekten die Benutzerfreundlichkeit im Vordergrund stehen muss, damit möglichst viele potentielle Nutzer in der Lage sind, es einfach benutzen zu können.[24]

Im Gegensatz zum Fallbeispiel in Japan wurde in Sofia ein digitaler Zwilling pilotiert, der jedoch nicht einmal eine komplette Stadt modelliert. Das Projekt in Sofia wurde nach

dem Pilotversuch nicht weiterverfolgt, da 1/24 der Stadt zu wenig war, um Projekte zu entwickeln. Der ausgewählte Distrikt Sofia's wurde aufgrund seiner Vielfältigkeit ausgesucht, um zu zeigen, wie einfach es wäre, weitere Distrikte zu modellieren. Der Distrikt enthält zwei Flüsse, 30% Waldfläche und allgemein verschiedene Baustile. In diesem Projekt wurde modelliert, ohne Anwendung zu schaffen. Deswegen wurde das Projekt nicht weitergeführt. Im Fall des Projekts "Plateau" wurden die Kosten durch verschiedene Interessengruppen an den Anwendungsbereichen von Plateau View gerechtfertigt. In Sofia konnten die Kosten aufgrund fehlender Anwendungen nicht gerechtfertigt werden.[6]

Ein anderes Beispielprojekt, bei dem die Kosten durch den Nutzen gerechtfertigt werden können, ist die Lärmkarten-Erzeugung mithilfe von CityGML[4]. Das Projekt wurde umgesetzt, da die Europäische Union (EU) eine Umgebungslärmdirektive[10] erlassen hat. Diese verpflichtet Mitgliedstaaten dazu, alle fünf Jahre die Lärmemissionen der wichtigsten Straßen, Bahnstrecken, Flughäfen, industriellen Konzentrationen und städtischen Ballungsräume als Lärmkarten zu dokumentieren. Aufgrund der hohen Bevölkerungsdichte und der damit verbundenen Verkehrs- und Industriedichte ist NRW besonders betroffen. Die Berechnung der Ausbreitung von Lärmquellen wie beispielsweise Straßen und Bahnstrecken wird dabei auf Basis von akustischen Modellen durchgeführt, welche im CityGML Format vorliegen. So konnten, nachdem das Modell einmal initial fertiggestellt wurde, die Lärmkarten bei den Folgeiterationen, welche alle fünf Jahre stattfinden, um 77% günstiger erstellt werden.[4]

Die digitale Repräsentation von Städten wird in den kommenden Jahren zunehmend wichtiger. CityGML hat sich dabei als häufig verwendetes Format etabliert. Die genannten Beispiele zeigen, dass CityGML als Basis von Digitalen Zwillingen erhebliches Potential für die Stadtentwicklung bieten, sofern sie ausreichend groß angelegt sind und mit genügend konkreten Anwendungen einhergehen.

2.3 Akustische Simulation und Urbanisierung

Neben den Lärmkarten, die Lärm mit der physikalischen Größe Dezibel in diskreten Ausbreitungskorridoren visualisieren, hat sich in den letzten Jahrzehnten eine Forschungsrichtung etabliert, die Lärm als psychoakustische Größe versteht und in Form von Klanglandschaften modelliert. Klanglandschaften werden wie folgt definiert:

> Eine Klanglandschaft (engl. Soundscape) bezeichnet die Gesamtheit aller hörbaren Klänge in einer räumlich begrenzten Umgebung. Sie ist als gesamte akustische Hülle, die den Menschen in seinem Alltag umgibt, zu verstehen. Diese akustische Umwelt umfasst alle Arten von Klängen; natürliche, menschliche und technische, aber auch Musik. (...) Jeder Ort zeichnet sich durch seine spezifische Klanglandschaft aus – diese entsteht erst durch das Zusammenspiel aller vorhandenen Klänge.

[15] Bei der Modellierung von Klanglandschaften geht es meist um den Soundscape-Ansatz. Dieser zielt darauf ab, die akustische Umgebung zu betrachten und oftmals durch Intervention so umzugestalten, wie es gewünscht ist. Das kann heißen, dass unerwünschte Geräusche verringert und angenehme Geräuschquellen hinzugefügt werden. Allgemein geht es aber um eine bewusstere Betrachtung der akustischen Umgebung als üblich.

Um die Klanglandschaft in den akustischen Simulationen realistisch darzustellen, müssen zunächst die Geräusche generiert werden, die sie prägen. Die akustische Simulation lässt sich in zwei Bereiche aufteilen: die numerisch akustische Simulation und die geometrisch akustische Simulation. Die numerische Simulation simuliert den tatsächlichen Schalldruck im Raum oder die Vibration einer Oberfläche. Es gibt verschiedene numerische Simulationsverfahren. Die Finite-Elemente-Methode (FEM) und die Boundary-Element-Methode (BEM) sind beispielsweise zwei sehr verbreitete Verfahren. Bei der FE-Methode muss das gesamte akustische Medium mit Volumenelementen in kleine Teile zerlegt werden, damit die Berechnung vom Computer effizient durchgeführt werden kann. Daher wird diese Methode hauptsächlich für die Untersuchung geschlossener Innenräume genutzt. Im Gegensatz dazu ist die Idee hinter der BE-Methode, alle physikalischen Vorgänge innerhalb eines akustischen Mediums auf dessen Oberfläche auszudrücken. Die beiden Methoden können auch gekoppelt werden, um beispielsweise verschiedene Materialien und Geometrien zu berücksichtigen. So können Bereiche mit unterschiedlichen Eigenschaften effizienter modelliert werden. Da diese Verfahren sehr genau sind, werden diese unter anderem zur Erzeugung von Lärmkarten genutzt. Allerdings sind die numerischen Simulationsverfahren sehr rechenintensiv und daher nicht für Echtzeit Berechnungen, also auch nicht für Echtzeit Simulationen geeignet.[8]

In dem Beispiel aus NRW (vgl. Kapitel 2.2) wurden dafür zuerst die Attribute der ausgewählten Straßensegemente wie beschrieben über die verschiedenen Ämter eingeholt. Dazu gehören die Lärminformationen (Verkehrsfluss, Schwerverkehrsanteil und Geschwindigkeitsbegrenzung) und die Informationen für die Gebäude, wie die Reflexionseigen-

schaften, die Ergebnisse der Lärmbelastung und die betroffenen Bewohner pro Gebäude. Die tatsächliche Berechnung wird dann von einer Software ausgeführt, welche über die Geschwindigkeit/Verkehrsfluss und die Entfernung zu allen Lärmquellen mathematisch Lärmwerte berechnet. Der Prozess ist sehr zeit- und rechenleistungsintensiv, aber wenn die Karte erstellt ist, ist der Prozess beendet, und daher braucht man hier keine Echtzeitberechnungen.[4]

Die geometrischen akustischen Simulationen hingegen ermöglichen es in Echtzeit, ein Audiosignal an einem bestimmten Ursprungspunkt im 3D-Raum zu approximieren. Am Ort des Hörers kann Schall aus allen Richtungen kommen und bildet lokal ein Schallfeld. Dieses Schallfeld besteht aus allen Schallquellen, welche direkt oder indirekt über Geräuschquellen an den Ort im Raum gelangen, der für die Simulation interessant ist. Die Schallquellen werden entlang der Schallstrahlen berechnet, indem der Weg mittels Raytracing von der Schallquelle bis zum Ziel verfolgt wird. Effekte wie Luftdämpfung, Reflektionen usw. werden dann entlang des Strahls auf das Quellsignal angewendet. Die Summe der im Schallfeld ankommenden Schallstrahlen bildet sozusagen eine Kugel um den Hörer. In dieser Kugel ist es möglich, sich umzuschauen, ohne dass das Schallfeld verändert wird. Anschließend kann es mithilfe von kopfbezogenen Transferfunktionen (HRTF) auf ein zweikanaliges Ohrsignal heruntergerechnet werden. Den gesamten Vorgang vom Bilden des lokalen Schallfeldes bis zum Runterechnen auf ein zweikanaliges Ohrsignal nennt man in der Raumakustik auch Auralisation.[27]

Eines der ersten Projekte, das für die akustischen Simulationen numerische und geometrische Verfahren anwendet, ist das SONORUS-Projekt, ein Forschungsprojekt der Europäischen Union, welches neue Ansätze und Werkzeuge für die städtische Klangplanung entwickelt hat. Es werden nicht nur traditionelle Lärmschutzmaßnahmen betrachtet, sondern auch die Steuerung und Gestaltung des gesamten Soundscapes in städtischen Gebieten. Dabei werden nicht nur die Reduzierung des Lärms berücksichtigt, sondern auch positive Geräuschquellen und architektonische Merkmale gefördert, um eine ausgeglichene Geräuschkulisse zu schaffen.

SONORUS unterscheidet drei verschiedene Klangumgebungen, die alle präzise manipulierbar sein müssen. Das Micro-Level, welches spezifische Details betrachtet, wie lokale architektonische Elemente. So werden dann Schlüsse gezogen, wie sich Form und Material auf die Umgebung auswirken und vor allem, wie Passanten diese Details wahrnehmen. Zum Beispiel können nach unten neigende und aus Glas bestehende Fassaden den Lärm um bis zu 12 dBA erhöhen. Bei dem Meso-Level geht es hauptsächlich um die Manipu-

lation des Verkehrs und das allgemeine Transportmanagement, um Straßenentfernungen für die Stadtplanung modellieren zu können. Zum Beispiel sollten Grünflächen besser verteilt werden, um Lärm zu reduzieren. Das Macro-Level beschäftigt sich mit der Struktur von Städten. Ziel ist auch hier die Reduzierung des Lärms mit der Hilfe von Grünflächen. Die Verteilung und die Art der Grünflächen (Parks, Gärten oder lokale Wälder) werden entscheidend durch die Struktur des Straßennetzes bestimmt. Daher werden auf diesem Level die Straßennetze analysiert und als linear, rasterartig oder ringförmig kategorisiert. Beispielsweise tendieren ringförmige Städte dazu, mehr Grünfläche zu besitzen als lineare Städte, jedoch besteht der Großteil der Grünflächen dabei aus Gärten. In linearen Städten gibt es weniger Grünflächen, diese sind aber effizienter beim Lärmschutz, da es wahrscheinlicher ist, dass es sich um kleine Wälder oder Parks handelt. Es gibt also keine bessere Struktur, aber die Analyse innerhalb der Kategorien hilft, Lärm zu reduzieren.[17]

Durch die Kontrolle aller drei Ebenen ist es möglich, Designentscheidungen in der städtischen Planung realitätsnah mithilfe der Simulationen zu testen. Auf diese Weise können passende Klanglandschaften für einen bestimmten Kontext entworfen werden und durch Soundwalks, Interviews und Beobachtung des Verhaltens der Testpersonen bewertet werden. SONORUS hat beispielsweise einige akustische Designstrategien für Innenstädte modelliert. Als Ergebnis entstanden Modelle mit fließendem Wasser in der Nähe von Straßen. Das laute Wasser war zwar ähnlich laut wie die Autos, wurde aber als deutlich weniger störend wahrgenommen. Auch Bäume vor Geräuschquellen wie Straßen und Bahngleisen wurden als sehr angenehm aufgefasst. Das lag nicht nur daran, dass ein Teil des Lärms abgefangen wurde, sondern auch daran, dass die Lärmquelle nicht direkt sichtbar war. Eine der Haupterkenntnisse war, dass eine Strategie alleine nicht ausreicht, um Lärmquellen zu eliminieren. Es werden immer mehrere kombinierte Lösungen benötigt.[17]

SONORUS führte konkrete Tests in vier europäischen Städten durch: Antwerpen, Göteborg, Brighton und Rom. In diesen Städten wurden jeweils unterschiedliche Probleme in Bezug auf den Lärm beschrieben und Lösungsansätze vorgestellt. Zunächst erfolgte eine Soundanalyse, um Problemgebiete und Störfaktoren zu identifizieren. In einem Park in Antwerpen beispielsweise sammelte das Team Aufzeichnungen von der Geräuschkulisse, welche Audio Signale und GPS Daten beinhalten. Gleichzeitig befragten sie Besucher über ihre Wahrnehmung der Geräusch Umgebung. Mit den Aufzeichnungen wurden Lärmkarten erstellt und durch die Interviews Statistiken erstellt. Die Statistiken und die Lärmkarte zeigten, dass eine Hauptstraße im Stadtzentrum als besonders störend identifiziert wurde. Im Rahmen des Projekts entwickelte das Team viele Vorschläge zur

Umgestaltung der Straße, um den Lärm für Fußgänger und Fahrradfahrer zu verringern. Zu den wichtigsten Maßnahmen gehören die Verlegung der Gehwege (eine Verringerung von 7 dBA), eine erhöhte vertikale Barriere (4–5 dBA) und eine Verringerung des Tempolimits (6 dBA). Es gibt weitere Vorschläge, die zusammen eine zusätzliche Verringerung der Lautstärke um 12 dBA erreichen könnten. In diesem Fall blieb es bei den Vorschlägen, da es nur indirekten Kontakt zum Planungsbüro gab. Allgemein sind die Erfolge in den vier Städten nicht so wie erhofft gewesen, da die verschiedenen Interessensgruppen und Akteure, die an der aktuellen Stadtplanung unserer Städte beteiligt sind, Lärm als noch kein allzu großes Problem sehen. Die erarbeiteten Vorschläge kosten nicht nur Geld, sondern auch kostbaren Raum im Stadtzentrum. Jedoch wächst das Bewusstsein in der Bevölkerung für das Lärmproblem, was zunehmend auch Städteplaner und andere Akteure dazu zwingt, sich aktiver mit dem Lärmproblem auseinanderzusetzen.

Virtual Reality (VR) bietet eine einzigartige Möglichkeit, Auralisierung und Visualisierung zu kombinieren, um Design-Entscheidungen effektiv zu kommunizieren. Das SONORUS Team konnte durch den Einsatz von VR einen Einfluss auf die Lärmwahrnehmung der Testpersonen feststellen. Durch das Aufsetzen des Headsets und der VR-Brille können Tester die Auswirkungen des Designs direkt erleben und Maßnahmen wie Lärmschutzwände auch auditiv nachvollziehen. Ein Beispiel, wo das SONORUS-Team Virtual Reality als Werkzeug verwenden konnte, ist die Turnhoutsebaan-Brücke in Antwerpen. Die Brücke verläuft über eine viel befahrene Straße und eine Bahnstrecke. Außerdem ist sie der einzige Weg vom Stadtzentrum aus in den Rivierenhof Park und damit stark frequentiert. Durch diese besondere Lage entschied sich das SONORUS Team hier für den Einsatz von Virtual Reality. Beispielsweise konnte deshalb beobachtet werden, dass Lärmschutzwände nicht nur einen Einfluss auf die Lautstärke haben, sondern auch auf die allgemeine Wahrnehmung der Umgebung, da sie die Geräuschquelle verstecken und gleichzeitig das Stadtbild aufwerten. Durch das Hinzufügen der VR-Landschaft wurde deutlich, dass die audiovisuelle Wahrnehmung im städtischen Bereich auch für Stadtplaner und Architekten vor allem in Zukunft wichtig sein wird, um eine lebenswerte Stadt zu schaffen.[17]

Im letzten Jahrzehnt sind VR-Systeme von Laborversuchen zu hilfreichen Werkzeugen geworden. In den meisten Fällen handelt es sich bei diesen Systemen allerdings noch um rein visuelle Applikationen. Das liegt hauptsächlich an dem höheren Interesse für Visuelles, verglichen mit der Akustik, aber auch an den verhältnismäßig teuren bzw. unpraktischen Hardware Lösungen, die für die Auralisierung existierten. Um die Auralisierung effizient zu gestalten, gibt es den Ansatz, die Simulationen durch Web Tools [28]

[25] umzusetzen und so die Klänge über Frameworks berechnen zu lassen und einfach über das Headset am jeweiligen PC der Benutzer auszugeben. Die technischen Anforderungen für einen solchen PC sind nicht zu hoch. Nahezu jeder Bildschirm und jedes Headset entsprechen den Anforderungen. Die Berechnungen müssen nicht clientseitig erledigt werden, und daher ist auch kein besonders guter Prozessor oder eine gute GPU erforderlich. Wenn die Berechnungen größtenteils serverseitig stattfinden, bleiben die Installation von Software und die Konfigurierung der Audio-Einstellungen als mögliche Schwierigkeit für die Nutzer. Dafür wurde ein Test mit 100 Freiwilligen durchgeführt und festgestellt, dass die Installation für die meisten Personen kein Problem darstellte. Trotzdem ist eine Website mit einfacher Bedienung einer Software, die man zuerst herunterladen muss, vorzuziehen. Zusammenfassend hat der Test gezeigt, dass speziell Online Versionen der Akustik-Simulationen ein großes Potential haben, aber vor allem, dass fast jeder in der Lage ist, daran teilzunehmen.[14]

Ein alternativer Ansatz zu Online Tools ist das AixCAVE VR System der RWTH Aachen. Dieses System, das 2012 installiert und zwischen 2013 und 2014 um ein Audiosystem zur Echtzeit-Auralisierung erweitert wurde, ermöglicht die Darstellung immersiver virtueller Umgebungen. AixCAVE ist ein hochauflösendes Projektionssystem, bei dem 24 Projektoren nahtlose Bilder an die vier Wände und die Decke werfen. Außerdem erfolgt die Erfassung der Position und Orientierung des Benutzers in Echtzeit. Für die Auralisierung kommen 4 Lautsprecher zum Einsatz. Die Berechnung der Geräusche, u. a. durch Ray Tracing, erfordert hohe Rechenleistung, weshalb sie auf mehrere PC-Cluster verteilt ist. Auch wenn das AixCave System ein hilfreiches Tool ist, ist es sehr teuer und auch nicht von jedem benutzbar. Daher steht es in einer Uni, wo möglichst viele Interessenten teilhaben können.[29][21]

VR Systeme sind eine interessante Mensch-Computer-Schnittstelle, die vielseitig eingesetzt werden kann. Gerade bei CAVE Systemen, also VR Systemen bei denen man durch z. B. vier Wände in der virtuellen Realität eingeschlossen ist, wird die Erfahrung besonders immersiv. Deshalb finden das AixCave System viel Anwendung. Es wird unter anderem genutzt, um Flugzeuglärm zu simulieren, die Akustik eines Konzertsaales widerzuspiegeln oder mittelalterliche Gesänge in ihrer ursprünglichen Umgebung wie beispielsweise einer Kirche zu simulieren. Im Beispiel mit dem Flugzeuglärm wird in dem Simulationsraum nicht nur ein visueller Geräuschteppich angezeigt, sondern es wird zu einem hörbaren Erlebnis in Echtzeit gemacht. Dadurch ist das Szenario erlebbar und für Forscher und Laien nachvollziehbarer. Genau das wird von akustischen Simulationen erwartet. Ein solches System ist jedoch, wie eben beschrieben, teuer und an der RWTH

Aachen muss man sich folglich das System teilen. Daher sind Software Tools, die mit einem einfachen PC benutzbar sind, zumindest noch eher von Interesse, um akustische Simulation verschiedener Art zu entwickeln. [18]

Zusammenfassend lässt sich sagen, dass die in diesem Kapitel behandelten Methoden zur akustischen Simulation wie die numerischen und geometrischen Ansätze wesentlich zur Zielsetzung der Arbeit beitragen. Der geometrische Ansatz bietet die Möglichkeit, Klanglandschaften in städtischen Umgebungen realitätsnah zu modellieren, was Stadtplanern und Architekten hilft, die akustischen Auswirkungen von Bauprojekten besser zu verstehen und Maßnahmen zur Lärmreduktion schon zur Planung des Projektes hin zu entwickeln.

2.4 Game Engines in der Akustischen Simulation

Individualsoftware und extrem teure Rechner stellen seit einiger Zeit nicht mehr die visuell besten oder authentischsten Simulationen. Die deutlich billigeren Game-Engines, welche auf normalen PCs laufen, bieten die Möglichkeit, realitätsnahe Virtual Reality (VR), Augmented Reality (AR) sowie komplexe Physik Simulationen zu erstellen und effizienter zu betreiben.

Nur die Militärbranche und wenige VR-Labore konnten es sich vor 25 Jahren leisten, in Richtung Computergrafik und Simulation zu forschen. Da die Gaming-Industrie seit den 70'er Jahren kontinuierlich wächst und Ende der 90'er sogar den Umsatz der Filmindustrie übertroffen hat, standen nun auch dieser Branche die finanziellen Mittel zur Verfügung, die visuelle Qualität der Spiele weiterzuentwickeln. Daher findet man heute die besten Rendering Pipelines auf Standard Gaming Grafikkarten und nicht mehr unbedingt auf hochmodernen Labor Computern. Für die Gaming-Industrie zahlt es sich aus, die Qualität der Spiele durch Forschung und Entwicklung besserer Hard- und Software anzuheben. Stand 2024 wird jährlich ein Umsatz von 300 Milliarden Dollar weltweit erzielt [5]. Nicht nur die besten Rendering Pipelines können heute vom Standard Nutzer gekauft werden, sondern auch die besten Tools und Plattformen, um Spiele zu entwickeln, gibt es für die Community teilweise gratis.[19]

Eines der meist verbreiteten Tools, um Videospiele zu entwickeln, ist die Game-Engine. Dabei handelt es sich um eine Software-Plattform oder ein Framework, welches auf die Videospielentwicklung ausgelegt ist, sich aber für viele Arten der Simulationserstellung

anbietet. Game-Engines abstrahieren alles, was Videospiele gemein haben, damit die Software wiederverwendet werden kann. Dafür wird eine Sammlung von Modulen und Werkzeugen bereitgestellt, welche eine Vielzahl von Funktionen bzw. Diensten bieten. Besonders relevant sind:

- Rendering Engine: Zuständig für die Darstellung von 2D- und 3D-Grafiken.
- Sound Engine: Für die Verarbeitung und Wiedergabe von Audiodaten.
- Physik Engine: Simuliert physikalische Gesetze wie Schwerkraft und andere Interaktionen zwischen Objekten.
- Skripting: Für die Programmierung von Spiellogik durch Skriptsprachen.

Game-Engines ermöglichen es wenigen Entwicklern, ein Spiel zu entwickeln und mit der Videospiel-Industrie mitzuhalten. Dabei können sich Entwickler mehr auf das kreative Design und Gameplay konzentrieren, da viele technische Aspekte abstrahiert werden. [1]

Auch wenn Videospiele die Hauptanwendung von Game-Engines bleiben werden, nutzen schon seitdem es Game Engines gibt auch viele andere Branchen diese für die eigenen Anwendungsbereiche. Das Militär benutzt Game Engines für Trainingseinheiten und um Simulationen von Fahrzeugen und Waffen in verschiedenen Einsatzumgebungen umzusetzen. In der Medizin können OPs simuliert werden, aber auch Piloten-Trainings-Simulationen und andere Arten von simulationsbasiertem Training werden allmählich zum Standard. In all diesen Beispielen werden Game-Engines genutzt, um eine virtuelle Realität zu schaffen, da einfache Simulationen auf einem Monitor, wie es in Spielen Standard ist, nicht die Immersion schaffen können, die bei solchen Simulationen gewünscht ist. Trainingssimulationen von OPs können in VR beispielsweise realisiert werden, indem eine VR-Brille den OP-Raum und einen Körper abbildet und reale Werkzeuge, welche zur Bewegungserfassung getrackt werden, mit in die Simulation eingebunden werden. Game Engines werden immer mehr für Anwendungen, die auch virtuelle Realität oder erweiterte Realität umfassen, genutzt, weil der Bedarf an Anwendungen mit VR-Technologie steigt und die Integration von VR-Hardware in Game Engines meist schon existiert.

Bei Projekten mit Game Engines steht zunächst die Entscheidung an, welche Engine zum Einsatz kommen soll. Es gibt eine ganze Reihe an Game Engines (Unreal Engine, Blender, Unity3D, CryEngine und viele mehr). Einige sind so flexibel, dass nahezu jede Anwendung damit geschrieben werden kann. Andere sind auf wenige Funktionen und Bereiche

spezialisiert. Da es (noch) keine Game-Engine speziell für Akustik-VR-Simulationen oder für Architektur-Simulationen gibt, ist es ratsam, die großen Game-Engines zu analysieren. Diese haben oft weit entwickelte Sound-Engines, einiges an VR-Unterstützung und bieten im Allgemeinen gute Graphik und Performance. Besonders groß und beliebt sind Unreal Engine und Unity. Zwar existieren auch andere Game Engines wie CryEngine oder Godot, die über eine große Community verfügen und bereits erfolgreiche Anwendungen hervorgebracht haben. Zur Vereinfachung der Analyse beschränkt sich diese Bachelorarbeit jedoch auf die beiden größten Game Engines.

Unity wurde 2005 von Unity Technologies veröffentlicht und brachte Videospiele wie Pokemon Go und Among Us hervor. Um mit Unity Spiele zu entwickeln, muss kein Geld gezahlt werden. Erst wenn ein Umsatz von 100 000 USD pro Jahr gemacht wird, werden Gebühren fällig. Primäre Programmiersprache bleibt bei Unity C#, auch wenn UnityScript und Bolt dazu verwendet werden können, Anwendungen mit Unity zu programmieren. Unreal Engine wurde 1998 von Epic Games veröffentlicht. Die letzte große Version ist 2021 mit Unreal Engine 5 herausgekommen und Unreal Engine brachte Videospiele wie Fortnite, Rocket League und viele andere hervor. Auch bei Unreal Engine kostet die Entwicklung von Anwendungen nichts, solange diese unter einem gewissen Betrag bleiben. Für nicht kommerzielle Anwendungen kostet es dadurch nichts. Hier wird für die Anwendungsentwicklung C++ verwendet, es gibt aber auch die visuelle Skriptsprache Blueprint. Diese macht es für Einsteiger besonders leicht, Anwendungen zu entwickeln, da dabei keine Programmiersprachen gelernt werden müssen.

Beide Game Engines sind gut geeignet, um akustische Simulationen zu simulieren. Das liegt hauptsächlich daran, dass beide eine große Community haben, welche Plugins zur Verfügung stellen, Programme und Erweiterungen auf dem neuesten Stand halten und vor allem anderen Entwicklern helfen können. Sowohl Unity als auch Unreal Engine unterstützten die wichtigsten Plattformen (Windows, verschiedene Konsolen usw.). Bei beiden Game Engines wird gemutmaßt, sie würden in naher Zukunft so realitätsgetreu werden, dass sogar Weltraum- und Militärsimulationen mit ihnen entwickelbar wären.

Allgemein ist Unreal Engine für bessere Grafik und insgesamt bessere Renderingqualität bekannt und bietet Werkzeuge für physikalisch basiertes Rendering und auch Tools für eine realistische Sound-Umgebung. Dafür ist Unity in den meisten Tests performanter und benötigt weniger Ressourcen, unabhängig von der Plattform. Die primäre Programmiersprache von Unreal Engine C++ bietet Entwicklern eine hohe Kontrolle und Leistung der Programme, die geschrieben werden. Die direkte Integration von C++ ermöglicht es,

tief in die Engine einzutauchen und komplexe Systeme zu entwickeln, auch wenn das mit einer steileren Lernkurve als bei höheren Programmiersprachen wie bei C# für Unity einhergeht. Unity ist bedienerfreundlicher, die Blueprints von Unreal Engine tragen zwar zur einfachen Entwicklung bei, ändern aber nichts an der hohen Komplexität von Unreal Engine. So wird für kleinere Projekte meist Unity verwendet, wohingegen Unreal Engine öfter für komplexe Applikationen zum Einsatz kommt.[1] [30][3]

Besonders relevant für die Entscheidung zwischen Unreal Engine und Unity sind in diesem Fall die akustischen Eigenschaften der Engines. Daher werden im Folgenden, wie in Tabelle 2.1 zusammengefasst ist, die verschiedenen Prinzipien der Audioansätze verglichen und dann der Bezug zu den Game Engines hergestellt. Stereo Mix wird zur Aufnahme aller Audioausgaben eines Computers verwendet. Dafür wird der gesamte Audiooutput eines Systems über zwei Kanäle erfasst und gemischt. Auch wenn es heute noch für die Musikwiedergabe und Ähnliches genutzt wird, ist es nicht besonders relevant für Game Engines, da die räumliche Wahrnehmung mit anderen Audioansätzen besser funktioniert. Als Nächstes erfolgt die Betrachtung von Spatial Audio. Es handelt sich um einen fortschrittlichen Audioansatz, welcher komplexe Algorithmen verwendet, um Klänge dreidimensional im Raum zu positionieren. Dadurch sind Töne nicht mehr nur von links oder rechts, wie bei Stereo, oder von hinten und vorne, wie bei Surround-Sound, hörbar, sondern aus jedem Punkt des 3D-Raums. Verwendung findet Spatial Audio in der Virtual- und Augmented-Reality, aber auch bei modernen Videospielen und in der Musikwiedergabe. Unreal Engine bietet umfassende Unterstützung für Spatial Audio durch Integrationen wie Steam Audio oder Resonance Audio. Auch Unity unterstützt Spatial Audio durch Erweiterungen wie FMOD und Wwise. Für Unreal Engine existiert die Möglichkeit, Spatial Audio Ansätze zu integrieren, jedoch schon länger als für Unity. Sowohl geometrische akustische Simulation als auch wellenbasierte akustische Simulation fallen unter den Überbegriff Spatial Audio, da beide Ansätze Techniken verwenden, um räumliche Klänge zu erzeugen (vgl. Kap. 2.3). Alle oben genannten Tools zur Integration in Unity und Unreal Engine verwenden geometrische Audioverfahren, es gibt aber auch Tools für wellenbasierte Audio, die in beide Game Engines integriert werden können. Dazu gehört beispielsweise Projekt "Acoustic" von Mircosoft. Dieses Tool ermöglicht beiden Game Engines, präzise physikalische Simulationen in komplexen Umgebungen zu modellieren.

Ansatz	**Stereo Mix**	**Spatial Audio**	
		geometrisch	**wellenbasiert**
Komplexität	Niedrig	Hoch	Sehr hoch
Rechenanforderungen	Niedrig	Hoch	Sehr hoch
Räumliche Darstellung	Basis	Sehr gut	Hervorragend
Anwendungsbereich	Musik, einfachere Spiele	Realistische Spiele, Simulationen	Forschung, High-End-Simulationen
Beispiele	Standard Audio Player	Steam Audio, Wwise	Projekt Acoustics

Tabelle 2.1: Vergleich der Sound-Ansätze

Aktuell stehen beiden Game Engines die (teilweise gleichen) Tools zur Verfügung, um geometrische und wellenbasierte akustische Simulationen zu erzeugen. Unreal Engine und Unity wechseln sich in Bezug auf die realitätsnähere Grafik zwar immer wieder ab. Zum Zeitpunkt der Entscheidung zwischen Unreal und Unity hatte Unreal Engine aber die Oberhand, und da auch Spartial Audio Plugins bereits früher für Unreal Engine zur Verfügung standen, wurden in Vorgängerprojekten schon auf Unreal Engine gesetzt. Damit fällt die Entscheidung auch hier schlussendlich auf die Unreal Engine.

Unreal Engine hat, wie nahezu jede der bekannten Game Engines, eine eigene Sound Engine. Der Ansatz der Unreal Engine Sound Engine (kurz UE Sound-Engine) besteht darin, Audioeffekte zu reproduzieren und nicht selbst physikalisch zu simulieren. Sie nutzt eine Kombination aus Sound-Cues und Sound-Mixes zur Wiedergabe. Sound-Cues und Sound-Mixes bestehen aus Sammlungen von im Studio aufgenommenen und bearbeiteten Sound-Dateien, die nach Unreal importiert werden. Diese Dateien werden bei bestimmten Events wie Timern, Spieleraktionen oder definierten Triggern ausgelöst. Bei Eintritt eines Events entscheidet ein Prioritätssystem, welche der Audio-Dateien aus den Sound-Cues abgespielt wird. Effekte wie Reverb oder Delay können auf Sound Cues angewendet werden und Veränderungen durch Spielereignisse werden in Echtzeit berechnet. Der Sound wird in UE4 in Bezug auf seine Position im Raum in Echtzeit berechnet. Die Dämpfung des Schalls wird bei der Ausbreitung in den Berechnungen berücksichtigt und speziell Luftabsorption wird realistisch simuliert, während andere akustische Phänomene weitgehend ungeachtet bleiben. Komplexe Schallreflektionen, die Absorptionsfähigkeit von Material und allgemein die Interaktion von Sound und Objekten begrenzen die physikalische Genauigkeit der Sound Engine. Bei sehr niedrigen ($<$125 Hz) oder zu hohen ($>$8000 Hz) Frequenzen zeigt die UE Sound Engine auch Leistungsschwächen. Unreal

Engine kann diese Nachteile aber ausgleichen, da es eine Vielzahl von Tools gibt, welche genau diese Nachteile kompensieren. Beispielsweise durch eine Integration einer Middleware wie Wwise[2] oder das Plugin [26]Steam Audio wird die Funktionalität der UE Sound Engine erheblich erweitert. Steam Audio ermöglicht wie im letzten Abschnitt beschrieben fortgeschrittene Raumakustik und ist seit Unreal Engine 2.6 als fester Bestandteil in der Sound-Engine integriert.[11]

Da bereits einige akustische Simulationen mit der Unreal-Engine realisiert sind, folgt im Folgenden eine Vorstellung relevanter Beispiele. Als Erstes soll ein Projekt von Massimiliano Masullo [11] vorgestellt werden, in dem ein dreidimensionales Akustik-Modell für Verkehrslärm entwickelt wurde. Dabei handelt es sich um eine Software, die 2020 in Irland entwickelt wurde, welche es Nutzern ermöglicht, durch eine modellierte Wohnsiedlung zu gehen. Dabei erlebt man den Verkehrslärm in Echtzeit und bekommt Werkzeuge zur Verfügung gestellt, um diesen zu regulieren. Diese Software wurde geschrieben, damit Stadt- und Raumplaner das Ausmaß von Lärmbelästigung verstehen und die bestimmten Gebiete entsprechend gestalten.

Das Projekt wurde in der Unreal Engine entwickelt und richtet sich dabei nach den CNOSSOS-EU [9] (Common NOise aSSessment methOdS in Europe)-Dokumentationen. Diese enthalten Gleichungen, mit denen der Schallemissionspegel von Fahrzeugs- bzw. Verkehrsgeräuschen definiert wird. Mit der visuellen Programmiersprache Blueprint wurden diese Gleichungen für jede Fahrzeugkategorie umgesetzt. Jedem modellierten Fahrzeug in der Simulation wird ein Geräuschgenerator angeheftet und mithilfe der Gleichungen passende Geräusche abgespielt. Der Verkehrslärm wird daher über die Summe der individuellen Fahrzeuggeräusche berechnet, welche im betrachteten Straßenabschnitt fahren. Eine wichtige Anmerkung ist, dass sich die Fahrzeuge dabei nicht in der Simulation bewegen. Die Geräusche werden berechnet und eine Geräuschquelle folgt dem Nutzer, während dieser sich bewegt. Damit die Geräusche in Unreal Engine realitätsgetreu reflektiert, absorbiert und übertragen werden können, wird Steam Audio verwendet. Das Steam-Audio-Plugin schwächt nicht nur Geräusche durch Hindernisse ab oder blockiert diese. Es bietet auch die Möglichkeit, den Abschwächungsmodus auf frequenzbasierte Übertragung zu setzen und so das Material und die Geometrie des Hindernisses mit einzurechnen. Damit ist der Hochfrequenzverlust vom angegebenen Material abhängig und die Simulation ist realistischer.

Insgesamt zeigt das Projekt, dass Verkehrslärm schon realistisch modelliert werden kann. Der Nutzer kann die Anzahl der Fahrzeuge sowie deren Kategorie nach Belieben ver-

ändern, um den Verkehrslärm zu variieren. Zudem können Lärmschutzwände platziert und in ihrer Größe und Form verändert werden, um mit dem Lärm und entsprechendem Schutz zu experimentieren. Die Geräuschbelastung wird mit Unreal-Engine und dem Steam Audio-Plugin durch die Gleichungen, welche durch die CNOSSOS-EU Dokumentation festgelegt worden sind, realistisch dargestellt. Die einzigen Nachteile dieses Projekts sind, dass einige Variablen, wie verschiedene Straßenbeläge, Temperaturunterschiede oder ob ein Fahrzeug bergauf oder bergab fährt, nicht implementiert worden sind und dass das Projekt auf ein Modell beschränkt ist. Das Projekt zeigt, dass Unreal Engine und Steam Audio zusammen leistungsfähig sind und realitätsnahe akustische Simulationen ermöglichen. Zusätzlich ist es durch wenige Änderungen am Projekt möglich, das entwickelte Plugin in ein Programm für architektonisches Design zu integrieren. Durch diese Änderung würde das Ziel des beschriebenen Papers, nämlich die Bereitstellung eines Werkzeugs für Städteplaner zur Bildung von mehr Lärmbewusstsein, erreicht werden.[11]

Die zuvor beschriebene Software findet bereits Anwendung in der Architektur und der Innenarchitektur. Ein Architekturbüro mit Sitz in Kopenhagen hat Akustik-Simulationen verwendet, um die Designentscheidungen für eine Universität in den USA zu treffen. Klassenzimmer können speziell ausgestattet werden, damit der Raum möglichst wenig nachhallt. Dafür wurden mehrere Modelle der Klassenzimmer erzeugt, mit jeweils unterschiedlich guter akustischer Behandlung. Zuerst ein Klassenzimmer, ohne dass auf Akustik geachtet wurde, dann eines mit absorbierenden Wänden, ein weiteres mit zusätzlich absorbierenden Decken und ein letztes mit allem, was man sich für gute Akustik wünschen kann. Der Nachhall wurde immer weniger und es wurde, wie zu erwarten, immer teurer. Einerseits liegt der Erfolg dieses Projekts darin, dass die akustischen Simulationen korrekt den Nachhall berechnen konnten und somit jeder Beteiligte die verschiedenen Arten von Klassenzimmern selbst erleben konnte. Andererseits haben diese Simulationen die Kommunikation zwischen den Architekten und der Uni vereinfacht, da so klar wurde, was die Architekten vorschlugen und wieviel die Maßnahmen zur Nachhallreduzierung bringen könnten. So konnten informierte Entscheidungen mit dem Kunden zusammen getroffen werden, was auch zu höherer Zufriedenheit bei allen Beteiligten beiträgt.[23]

In diesem Kapitel wurde gezeigt, dass der Einsatz von Game Engines eine vielversprechende Methode zur akustischen Simulation im urbanen Kontext darstellt. Durch ihre leistungsfähigen Render- und Sound-Engines ermöglichen diese Werkzeuge in Verbindung mit modernen Audio Plugins eine realitätsnahe Modellierung akustischer Umgebungen. Die vorgestellten Projekte verdeutlichen, dass Game Engines durch die Integration von

Plugins eine hohe Flexibilität bieten und nicht nur technisch machbar sind, sondern auch bereits praktische Anwendungen finden.

2.5 Zusammenfassung

CityGML ist ein Datenmodell und XML-basiertes Dateiformat zur Darstellung von städtischen 3D-Geodaten. Es bietet standardisierte und interoperable Beschreibungen städtischer Strukturen und deren Attribute, kann aber durch das *Theme* "Generics" oder ADEs erweitert werden. Die Datenbeschaffung kann sehr viel Aufwand bedeuten, doch für die Stadt Hamburg gibt es bereits die nötigen CityGML-Dateien in LoD 1, 2 und 3, um das Projekt "Sound Reality" umzusetzen. Es gibt bereits zahlreiche Projekte, die mit CityGML umgesetzt wurden, auch im Bereich der Akustik. Akustische Simulationen finden im urbanen Raum vielfältige Anwendungen. Es werden mithilfe von numerischen akustischen Methoden Lärmkarten erzeugt, und mit geometrischen Akustikmethoden können in Echtzeit Klanglandschaften simuliert werden. Durch die Simulation von Klanglandschaften kann Lärm realitätsnah erlebt werden, und Lärmquellen können identifiziert und bekämpft werden. Das SONORUS-Projekt entwickelte neue Ansätze und Werkzeuge für die urbane Klangplanung. Besonders interessant ist, dass auch akustische VR Ansätze dabei als Werkzeug genutzt wurden. So konnte der Lärm von allen Beteiligten erlebt werden, wie es auch im Projekt "Sound Reality" geplant ist. Für diese Art von Simulationen wird eine Plattform benötigt, welche leistungsstark genug ist, alle Berechnungen für die Auralisierung und Visualisierung zu bewältigen. Game Engines bieten sich aus mehreren Gründen dafür an. Sie sind billiger und leistungsstärker als herkömmliche Simulationssoftware, aber vor allem deutlich flexibler. Game Engines sind zwar für Videospiele ausgelegt, können aber für jede andere Art von Simulation benutzt werden. Unreal Engine ist besonders für akustische VR-Simulationen geeignet, da umfangreiche VR-Unterstützung geboten wird, es eine große Community an Entwicklern gibt und die Nutzung bei nichtkommerzieller Anwendung der Software kostenlos ist. Ein zentrales Feature ist die Sound-Engine, die durch vorhandene Plugins, wie Steam Audio, erweitert werden kann, um realitätsnahe akustische Simulationen zu ermöglichen. Unreal Engine konnte in Kombination mit Steam Audio schon in mehreren akustischen Simulationsprojekten ihre Eignung unter Beweis stellen. Diese Erfolge umfassen die Nachbildung von materialabhängiger Geräuschabsorbierung sowie eine akkurate Simulation von Verkehrslärm und dessen Abschwächung durch Lärmschutzwände. Diese Funktionalitäten sind für

das Projekt "Sound Reality" wichtig, da sie die Visualisierung, Auralisierung und Analyse im urbanen Kontext ermöglichen. Im Rahmen dieser Bachelorarbeit wird untersucht, wie die öffentlich verfügbaren CityGML-Daten durch einen halbautomatischen Import in die Game Engine integriert werden können, um die Voraussetzung für eine sowohl detailgetreue als auch realistische Simulation urbaner Klanglandschaften zu ermöglichen. Die Untersuchung zielt darauf ab, die Grundlage für die Erzeugung akustischer Simulationen zu schaffen, welche eine immersive Erkundung urbaner Umgebung erlaubt.

3 Anforderungsanalyse

In diesem Kapitel werden die Anforderungen an den CityGML-Parser formuliert. Dazu beschreibt Kap. 3.1 zunächst das Projekt sowie die Zielumgebung, in die die Artefakte des Parsers zu integrieren sind. Daraufhin werden die funktionalen Anforderungen in Kap. 3.2 und die nicht funktionalen Anforderungen in 3.3. vorgestellt. Die Anforderungsanalyse wird durch eine Zusammenfassung in Kap. 3.4 abgeschlossen.

3.1 Einordnung in das Projekt

Der CityGML-Parser wird im Rahmen des Projekts "Sound Reality" entwickelt, das sich mit Maßnahmen zur Lärmminderung in Städten und der effektiven Kommunikation dieser Maßnahmen gegenüber den Bürger*innen beschäftigt. Dafür soll ein Tool entstehen, welches Teile der Stadt Hamburg mittels VR Simulationen realitätsnah widerspiegelt. Besonders relevant ist dabei die Auralisation, also die Hörbarmachung von Geräuschen innerhalb der Simulation.

Dem Projekt "Sound Reality" gehen 3 Projekte voran, namentlich "X-Eptance Impulse" (XEI), "X-Eptance Explore" (XEE) und "Open Citizen Soundwalks" (OCSW). Diese hatten das Ziel, die visuellen und akustischen Auswirkungen von Windparks in VR zu simulieren, um die Reaktionen der Anwohner*innen auf geplante Windparks besser zu verstehen. Dabei wurden Werkzeuge und Methoden entwickelt, welche durch kleinere Erweiterungen die Grundlage für "Sound Reality" bilden.

Im Projekt XEI wurden umfangreiche akustische Messungen durchgeführt, um eine Geräuschdatenbank mit Geräuschquellen und deren relevanten akustischen Parametern zu füllen. Diese Daten haben die Grundlage für die Simulationen im Projekt XEE gebildet. Dabei wurde die Schallausbreitung in einem realen Windpark physikalisch modelliert und in einer VR-Umgebung nachgebildet. Die Modellierung des Windparks ist größtenteils automatisiert worden und konnte auch mit öffentlich zugänglichen 3D Geodaten

realisiert werden. So ließen sich die Geräusche und die visuelle Umgebung realitätsnah darstellen.

Darauf aufbauend hat sich das Projekt OCSW verstärkt mit der Validierung dieser Simulationen auseinandergesetzt. Es wurde geprüft, inwiefern die virtuelle akustische und visuelle Wahrnehmung der realen Umwelt entspricht. Dafür wurden "Soundwalks" durchgeführt, bei denen Menschen durch den realen Windpark gingen, um die akustischen Eindrücke zu erfassen. Danach konnten die Eindrücke mit denen in der virtuellen Nachbildung verglichen werden, um die Genauigkeit der Simulation zu bewerten. Dazu wurden innerhalb des Projekts Tools für die Speicherung, systematische Bewertung und Auswertung der Befragungsergebnisse entwickelt.

In den vorherigen Projekten konnten Shapefiles, Openstreetmap- und Geotiff-Dateien ausgelesen werden und alle nötigen Informationen liefern. Die Datenformate waren vollkommen ausreichend, um ländliche Umgebungen zu beschreiben, da die Häuser vereinzelt oder am Rand des Modellierungsgebiets lagen und der Detailgrad dabei irrelevant war.

Im Projekt "Sound Reality" wird der Fokus von Windparks jedoch auf urbane Lärmsituationen verlagert. Hier spielt der Detailgrad der Gebäudemodelle sowohl für die visuelle Darstellung als auch für die Auralisation eine größere Rolle. Die Geometrie der Gebäude wird für akustische Reflektionen, Absorptionen und Beugungseffekte verwendet. Daher ist der Detailgrad der aus Openstreetmap Daten generierten Gebäude nicht mehr ausreichend.

Die Stadt Hamburg stellt auf ihrem öffentlichen Geoportal[13] Gebäudemodelle in höherem Detailgrad im CityGML Format bereit. Um diese nutzen zu können, wird der CityGML-Parser benötigt. Dieser soll die Geometrien aus den CityGML Daten in Gebäude in der Unreal Engine umwandeln. So kann die aufwändige, manuelle Modellierung vermieden werden. Der Import der CityGML Daten in die Unreal Engine wird somit zur Grundlage für die akustischen Berechnungen und Simulationen, die Gegenstand des Forschungsprojektes Sound Reality sind und außerhalb des Fokus dieser Arbeit liegen.

Zusätzlich zum Detailgrad sind auch die Materialeigenschaften der Gebäude von Interesse, da unterschiedliche Materialien den Schall auf unterschiedliche Weise beeinflussen. Die Materialeigenschaften sind in den vorliegenden CityGML Dateien leider nicht direkt verfügbar, könnten jedoch potentiell aus den vorliegenden Texturen abgeleitet werden. Auch das fällt jedoch nicht mehr in den Rahmen dieser Arbeit.

3.2 Funktionale Anforderungen

Nachdem nun das Projekt noch einmal beschrieben und die Rolle des Parsers erklärt wurde, lassen sich daraus folgende Anforderungen an den Parser ableiten:

- *Import von CityGML Dateien für Hamburg*

  Der Parser muss in der Lage sein, CityGML Dateien, die aus dem Geoportal Hamburg [13] stammen, einzulesen und in renderbare Unreal Meshes umzuwandeln. Dafür müssen die Dateien im XML oder GML Format vorliegen und der Import soll mehrere Dateien gleichzeitig umfassen können.

- *Auslesen der relevanten Daten*

  Der Parser muss CityGML Dateien erkennen und zwischen den LoDs 1, 2 und 3 unterscheiden können. Aus den LoD 1 bis 3 sollen Geometrien und Metadaten ausgelesen werden. Zu den Geometriedaten gehören die Koordinaten der Gebäudeflächen, das Koordinatensystem von CityGML sowie ein geografischer Referenzpunkt. Die Metadaten umfassen je nach vorliegendem LoD verschieden viele Attribute. Es muss pro Gebäude mindestens die ID extrahiert werden und, soweit es das LoD zulässt, zusätzlich die Adressdaten und die Texturkoordinaten.

- *Umwandeln der CityGML Koordinaten in die Unreal Engine und Skalierung*

  In den CityGML Dateien kann ausgelesen werden, welches Koordinatensystem verwendet wird. Dieses muss in das rechtshändige kartesische Koordinatensystem der Unreal Engine umgewandelt werden. Zudem muss der Parser eine Skalierung vornehmen. Die CityGML Koordinaten werden in Metern angegeben, während die Unreal Engine in Zentimetern arbeitet. Der Koordinatenursprung soll aus dem zuvor extrahierten Referenzpunkt berechnet werden können, damit die Gebäude sinnvoll im Raum platzierbar sind.

- *Normalen und Tangentenberechnung*

  Die erzeugten Gebäude sollen in der Unreal Engine korrekt beleuchtet werden. Dazu muss der Parser die Normalenvektoren und Tangenten der Gebäudeflächen berechnen, um eine realistische Beleuchtung und Schattenberechnung zu ermöglichen.

- *Berechnung des Texturkoordinaten Defaults*

  Da nicht mit jedem Level of Detail auch Texturkoordinaten und Texturen selbst mitgegeben werden, sollen die Texturkoordinaten berechnet werden, um so die spätere Texturierung zu vereinfachen und jedem Gebäude eine Standard Textur zuzufügen.

- *Benutzeroberfläche (UI)*

  Der CityGML-Parser muss in der Unreal Engine über das Hauptmenü zugänglich sein, damit der Benutzer über eine Menüoption eine oder mehrere CityGML Dateien zum Import auswählen kann. Außerdem soll es eine Rückmeldung nach dem Import geben, welche dem Benutzer die Anzahl der geladenen Dateien anzeigt, oder eine entsprechende Fehlermeldung mit dem Dateipfad zu der nicht eingelesenen Datei.

3.3 Nichtfunktionale Anforderungen

- *Performance*

  Die importierten Meshes sollen in der Simulation nicht zu Performance Einbußen führen. Um das zu gewährleisten, soll es die Option geben, entweder ein Mesh pro Gebäude oder ein Mesh pro Import zu erstellen. Die zweite Option soll eine stabile Framerate sicherstellen.

- *Erweiterbarkeit*

  Der Parser soll so aufgebaut sein, dass er in Zukunft einfach auf weitere LoDs erweitert werden kann, ohne dass umfangreiche Anpassungen des bestehenden Codes notwendig sind.

  Darüber soll er auf weitere Metadaten, beispielsweise Gebäudedaten wie Funktion des Gebäudes oder Dachtyp, einfach erweiterbar sein. Der Parser soll dafür so gestaltet sein, dass Metadaten klar von Geometriedaten getrennt sind.

- *Proof of Concept*

  Es soll gezeigt werden, dass der CityGML-Parser für ausgewählte, nicht triviale Bereiche des Hamburger Stadtgebiets erkennbare Umgebungen in der Unreal Engine erzeugen kann.

3.4 Zusammenfassung

In diesem Kapitel wurde zunächst der Kontext des Projekts "Sound Reality" beschrieben, in dessen Rahmen der CityGML-Parser zur Visualisierung von Hamburger Stadtteilen in einer VR Simulation eingesetzt werden soll. Der Parser spielt dabei eine zentrale Rolle, indem er CityGML Dateien in Geometriemodelle für die Unreal Engine umwandelt und so eine realitätsnahe Darstellung der Stadt ermöglicht. Dazu müssen relevante Geometrie und Metadaten aus verschiedenen LoDs extrahiert und so weit verarbeitet werden, dass die Unreal Engine daraus beleuchtete Gebäude rendern kann. Außerdem soll der Parser performante Gebäudestrukturen erstellen und für zukünftige LoDs sowie das Auslesen von weiteren Metadaten leicht erweiterbar sein.

4 Entwurf und Umsetzung

In diesem Kapitel stehen der Entwurf und die Umsetzung im Fokus. Zunächst behandelt Kap. 4.1 die für das Projekt relevante Architektur der Unreal Engine und zeigt auf, wie sich der CityGML- Parser in diese Architektur eingliedert. Anschließend folgt in Kapitel 4.2 eine Vorstellung der Struktur der CityGML Dateien sowie eine Darstellung der spezifischen Datenlage für Hamburg. Mit dem Wissen, wie das Plugin in die Unreal Engine integriert wird sowie dem Aufbau der CityGML Dateien, wird in Kapitel 4.3 die Funktionsweise des Plugins erläutert. Danach richtet sich der Blick in Kapitel 4.4 auf die Meshgenerierung für die Unreal Engine, die zur Darstellung von 3D Objekten nötig ist. Kapitel 4.5 widmet sich den Ergebnissen, bevor Kapitel 4.6 Probleme und einen Erfahrungsbericht vorstellt. Abschließend fasst Kapitel 4.7 die wichtigsten Erkenntnisse des Kapitels zusammen.

4.1 Architektur der Zielumgebung

In der Zielumgebung nimmt die Unreal Engine eine zentrale Rolle als Plattform für die (akustische) Simulation des Projekts ein. Daher ist es sinnvoll, sich die Architektur bzw. Programmierumgebung, die die Unreal Engine bietet, genauer anzuschauen. Die Architektur der Unreal Engine 4 ist komplex; sie besteht aus zahlreichen Modulen und Werkzeugen. Diese übernehmen jeweils spezifische Aufgaben und ermöglichen es, vielfältige Systeme zu integrieren. Abbildung 4.1 zeigt die grundlegende Struktur der Unreal Engine und verdeutlicht die Beziehungen zwischen relevanten Komponenten.

Im Rahmen dieser Arbeit ist nur ein Teil der vielen Module relevant, weswegen hier kein Anspruch auf Vollständigkeit erhoben wird.

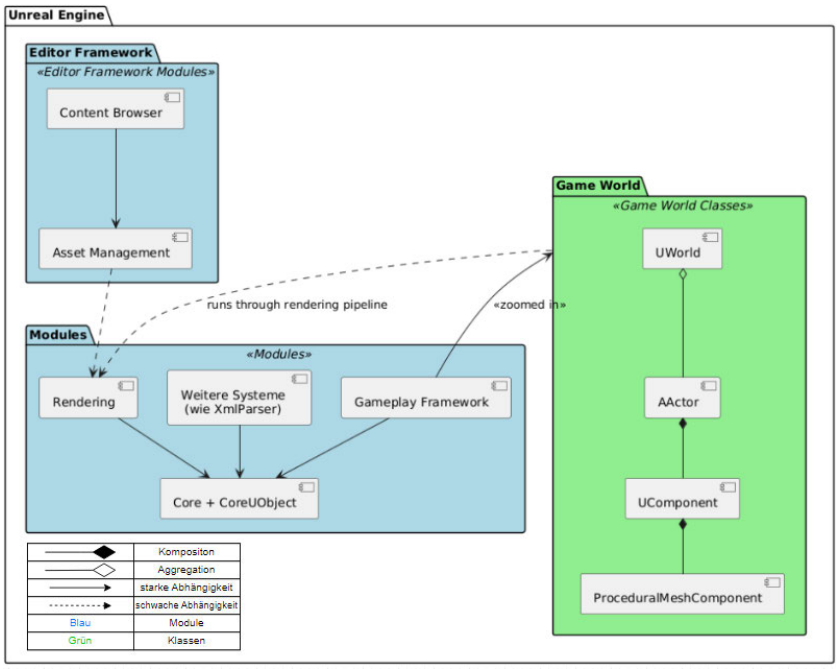

Abb. 4.1: Unreal Engine Architektur Schaubild

Die internen Komponenten der Unreal Engine bestehen aus Modulen, welche spezifische Funktionsbereiche abdecken und dabei fest im Projekt integriert sind. Die für den CityGML-Parser relevante Architektur der Unreal Engine lässt sich in drei Hauptbereiche unterteilen: die Game World, das Editor Framework und die Modules. Das Editor Framework und die Modules sind hier blau, da sie beide aus Modulen bestehen. Im Gegensatz dazu ist die Game World grün dargestellt, da sie einen Teil des Gameplay Frameworks abbildet, welches aus Klassen besteht. Der Hauptbereich Modules bildet die Grundlage der Unreal Engine. Darin enthalten ist das Core Modul, in welchem die grundlegendsten Funktionen definiert sind. Jedes Modul, nicht nur die, die hier dargestellt sind, baut auf dem Core auf und hat eine starke Abhängigkeit dahingehend. Darüber hinaus sind das Rendering Modul und das Gameplay Framework für diese Arbeit relevant. Das Rendering Modul stellt die Rendering Pipeline, welche durchlaufen wird um Objekte zu visualisieren und zu beleuchten. Das Gameplay Framework ist relevant, da es für die Spiellogik und die Game World verantwortlich ist.

Die Game World ist ein Teil des Gameplay Frameworks und verwendet `UWorld` als Hauptklasse, die die gesamte Spielumgebung repräsentiert. Die Klasse verwaltet die spielrelevanten Instanzen und Mechaniken. Innerhalb der Game World werden sämtliche Ob-

jekte als `AActors` implementiert und in die Szene integriert. Actors bilden die fundamentalen Bausteine, aus denen alle interaktiven, aber auch nicht-interaktiven Elemente wie beispielsweise Gebäude und Städte bestehen.

Jeder Actor kann mit Komponenten (`UComponent` Objekten) ausgestattet werden, welche zusätzliche Funktionalitäten hinzufügen. Das folgt dem Entity Component Entwurfsmuster, welches häufig in der Spielentwicklung verwendet wird. Dabei fungiert der Actor als Entity und die Komponenten als spezialisierte Datenmodule, die die spezifischen Eigenschaften bzw. Funktionalitäten hinzufügen. Eine solche Komponente ist beispielsweise die `ProceduralMeshComponent`, die es ermöglicht, dreidimensionale Meshes darzustellen und zur Laufzeit zu verändern. Die gesamte Game World durchläuft die Rendering Pipeline des Rendering Moduls, damit alle Objekte der Welt dargestellt werden.

Das Editor Framework spielt eine zentrale Rolle bei der Erweiterung der Unreal Engine durch die Entwicklung von Plugins. Plugins sind im Gegensatz zu Modulen kein interner Bestandteil, sondern modulare Erweiterungen der Unreal Engine. Diese ermöglichen funktionale Erweiterungen der Engine, ohne den Quellcode verändern zu müssen. Sie können, müssen aber nicht aus mehreren Modulen bestehen, welche spezifische Aufgaben entweder zur Entwicklungszeit im Editor oder zur Laufzeit übernehmen.

Das Editor Framework stellt Tools zur Verfügung, um mit der Unreal Engine zu interagieren und diese zu erweitern. Es ist in mehrere Module aufgeteilt. Beispielsweise sind das Blueprint System, welches eine visuelle Skripting Möglichkeit bietet, oder das Unreal Motion Graphics Modul wichtige Tools, welche hier aber nicht genutzt werden. Relevant sind hier das Asset Management und der Content Browser. Das Asset Management verwaltet alle spielbezogenen Ressourcen wie Texturen, 3D Modelle (wie Gebäude) und Sounds. Es ist mit dem Rendering System verknüpft, um die Assets effizient laden und rendern zu können. Der Content Browser stellt eine Oberfläche bereit, über die auf die Assets zugegriffen werden kann, um diese in das Projekt zu integrieren.

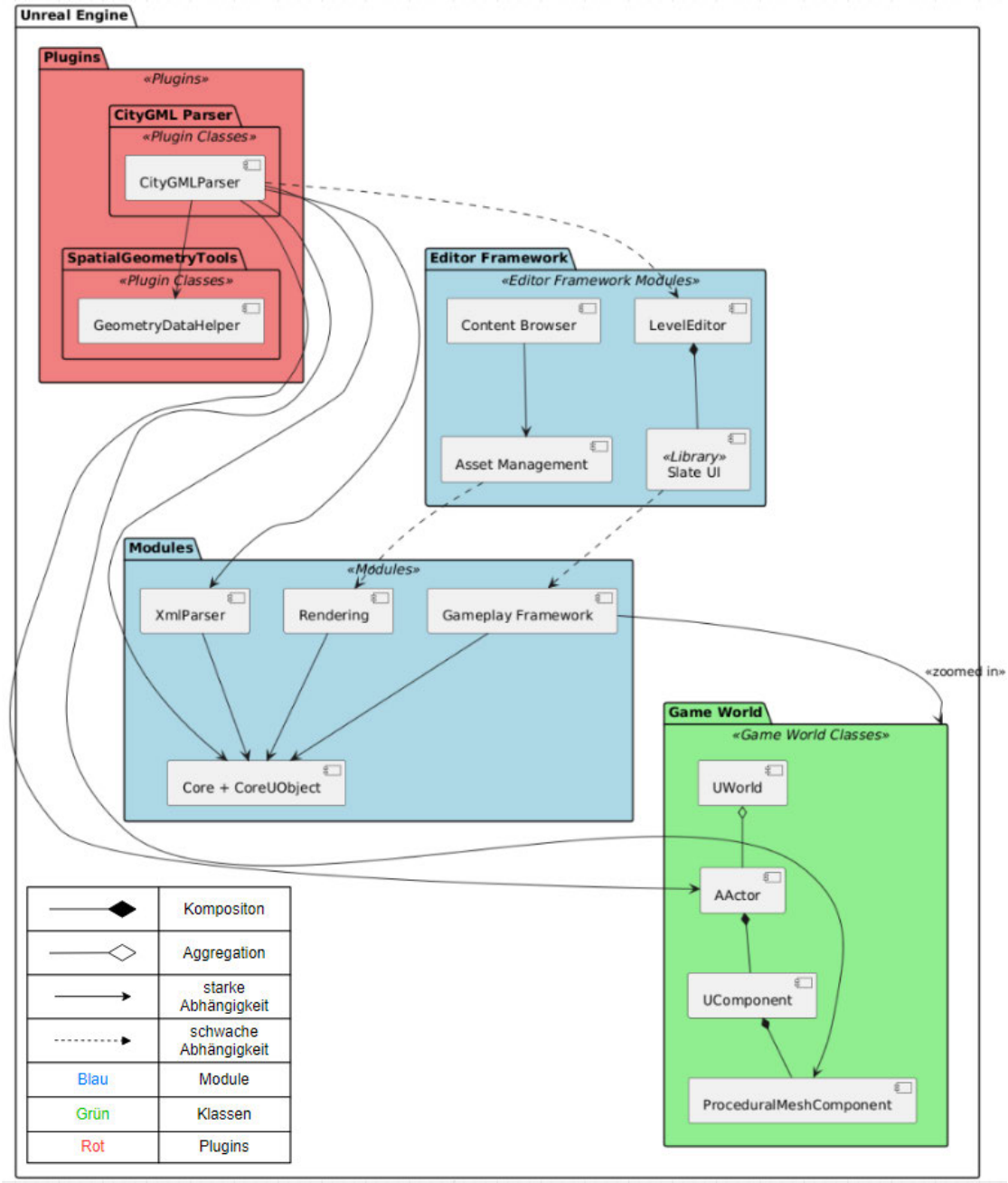

Abb. 4.2: Einbindung des CityGML-Parsers in die Architektur der Unreal Engine

In Abbildung 4.2 werden die Plugins als eigener Bereich in Rot dargestellt, um diese klar von den anderen Modulen zu trennen. In den Plugins werden jeweils die Klassen abgebildet, welche hier genutzt werden. Die hier abgebildeten Plugins sind einerseits das entwickelte CityGML-Parser Plugin und andererseits SpatialGeometryTools; ein Plugin, welches aus dem Vorgängerprojekt von Sound Reality stammt. SpatialGeometryTools wird vom CityGML-Parser für die Triangulation von Polygonen (vgl. Kap. 4.4) genutzt.

Der CityGML-Parser erweitert das Editor Framework der Unreal Engine, indem dem LevelEditor Modul eine Menüoption hinzugefügt wird, welche die Selektion und den Import von CityGML Dateien ermöglichen. Dafür wird die Slate UI Bibliothek verwendet. Diese ist in der Unreal Engine für die Erstellung von Benutzeroberflächen verantwortlich. Konkret wird ein neuer Button im Editor hinzugefügt.

Darüber hinaus greift der CityGML-Parser auf einige Funktionen aus dem Core System zu, um grundlegende Funktionen für den Umgang mit dem Dateisystem nutzen zu können. Beispielsweise werden einige Funktionalitäten aus Klassen des Core Moduls Misc verwendet. Dazu gehört die Klasse `MessageDialog`, um Dialogfenster anzuzeigen, die den Benutzer über Fehler oder Ausgaben informieren. `FileHelper`, welche das Öffnen, Lesen und Schreiben von Dateien ermöglicht, und `Paths`, welche Methoden bereitstellt, um mit Dateipfaden zu arbeiten.

Um CityGML Dateien zu importieren und zu verarbeiten, verwendet der CityGML-Parser einen nativen XMLParser der Unreal Engine. Mit den Klassen FXmlFile und FXmlParser, welche zum Parsen von CityGML-Dateien verwendet werden, ermöglicht der XMLParser das Einlesen von XML-Daten und deren Überführung in eine hierarchische "FXmlNode" Struktur. Damit wird anschließend innerhalb des CityGML-Parsers weitergearbeitet.

Die importierten CityGML Daten werden durch den CityGML-Parser in die Game World geladen. Dazu wird ein `StaticMeshActor` erstellt, welcher als Behälter für die `ProceduralMeshComponents` dient. Der `StaticMeshActor` wird direkt in der `UWorld` platziert, während die `ProceduralMeshComponents` die Aufgabe übernehmen, Geometrie, die sie aus dem CityGML-Parser bekommen, in Gebäude der Game World abzubilden. Die `ProceduralMeshComponents` ermöglichen es zur Laufzeit Meshes zu generieren und zu manipulieren.

4.2 CityGML Datenlage

Die Struktur der CityGML-Dateien ist vorgegeben und daher weitestgehend einheitlich. Jede CityGML Datei, unabhängig von der CityGML Version und dem Level of Detail (LoD), besteht mindestens aus dem `<core:CityModel>` und einem `<gml:Envelope>` Element. `<CityModel>` ist das zentrale Container Objekt. Es bildet die höchste Ebene des Dokuments. Es beinhaltet Metadaten über das Dokument, wie beispielsweise die CityGML Version und das LoD. Das `<Envelope>` Element ist ein (verschachteltes) Kindelement von `<CityModel>` und enthält die beiden Eckpunkte für den zu modellierenden Bereich sowie das verwendete Koordinatensystem. Alle weiteren Informationen hängen sich unter dem `<CityModel>` in die Struktur ein und werden validiert, wenn sie wie im CityGML Schema definiert beschrieben sind. Relevant sind für diese Arbeit vor allem Gebäude, welche in einer Liste `cityObjectMembers` definiert werden. Diese Liste muss vorhanden sein, wenn Gebäude in CityGML beschrieben werden. Die Gebäude haben generell mindestens eine ID und bei den Hamburger Daten auch immer Koordinaten für die Wände, Dächer und Böden, eine genaue Beschreibung, wo das Gebäude in den Boden übergeht, und optional eine Reihe an Metadaten für die Gebäude.

Wie die Gebäude und deren Metadaten beschrieben werden, unterscheidet sich je nach CityGML Version und LoD. Für die Hamburger Dateien steht CityGML entweder in den CityGML Versionen 1 oder 2 zur Verfügung. Außerdem sind für Hamburg die LoDs 1 bis 3 definiert. LoD 1 und 2 sind sich sehr ähnlich und liegen beide in CityGML 1 vor, während LoD 3 schon in CityGML 2 vorliegt.

Die Unterschiede zwischen den CityGML Versionen 1 und 2 sind relativ gering und betreffen vor allem Erweiterungen und allgemeine Verfeinerungen der vorherigen Version. Es wurden ein paar Features hinzugefügt, unter anderem um spezifischere städtische Objekte wie Brücken, Tunnel oder Wasserflächen mit einer eigenen Klasse modellieren zu können. An vorhandenen Objekten oder Strukturen wurde jedoch nur wenig verändert, damit die Versionen kompatibel bleiben. Deshalb hat der Versionswechsel auch keine großen Auswirkungen auf die Struktur der Daten, welche für Hamburg zur Verfügung stehen. Spezifisch für die Daten in Hamburg haben sich nur geringfügige Änderungen bei den XML-Namen bemerkbar gemacht. So wurde `core:CityObject` beispielsweise in CityGML 2 zu `CityObject` umbenannt. [22]

Unabhängig von den CityGML Versionen gibt es zwischen den verschiedenen LoDs strukturelle Differenzen. Einerseits bestehen abhängig vom LoD verschiedene Möglichkeiten,

die Flächen der Gebäude zu beschreiben. Andererseits bestehen bei steigendem LoD mehr Optionen, Zusatzinformationen hinzuzufügen. Der größte Unterschied der verschiedenen LoDs sind jedoch unterschiedlich viele Attribute, die pro Gebäude beschrieben werden. Dieser Unterschied ergibt sich daraus, dass LoD 1 und 2 dieselbe Datengrundlage haben, LoD 3 jedoch eine andere.

Zur Erzeugung von LoD 2 werden die Grundrisse aus der amtlichen digitalen Liegenschaftskarte entnommen. Danach erfolgt ein Verschnitt der Gebäude mit dem beim Landesbetrieb vorgehaltenen digitalen Geländemodell, d. h., dass die Koordinaten, an denen das Gebäude in das Gelände übergeht, genau in CityGML beschrieben werden. Die Dachformen werden mithilfe von Punktwolken (Airborne Laserscanning oder Photogrammetrie) erfasst und vollautomatisiert in die standardisierten Dachformen eingeordnet. LoD 1 wird rückwärts aus LoD 2 berechnet, indem die Grundriss- und Landschaftsdaten übernommen werden und das Dach als flache Fläche in der Mitte des LoD 2 Daches abgebildet wird. Bei LoD 1 und 2 beträgt die Genauigkeit der Gebäudehöhe +/- 5 Meter. Größere Abweichungen entstehen nur, wenn die Dächer in LoD 2 komplex waren und nun, da auf LoD 1 reduziert wird, keine passende Dachmitte gefunden werden kann. Das passiert beispielsweise bei extrem schrägen Dächern.

Im Gegensatz dazu wird LoD 3 erzeugt, indem Nadir- (Senkrecht von oben) und Schrägflugbilder photogrammetrisch ausgewertet und dreidimensional modelliert werden. Diese Auswertung beschränkt sich nicht nur auf das amtliche Kataster, sondern bezieht sämtliche Gebäude, die zum Zeitpunkt der Datenerfassung existierten, mit ein. Infolgedessen sind in den LoD 3 Datensätzen mehr Gebäude als in den vorherigen LoDs enthalten, obwohl gleich große Flächen beschrieben werden. Die Dachlandschaft ist in LoD 3 außerdem deutlich genauer und den Gebäuden werden auch Texturkoordinaten und Texturen im JPEG Format mitgegeben. Die JPEGs haben eine Auflösung von 150 Dots per Inch (Dpi), also 150 Druckpunkte, welche auf dem Papier pro Zoll platziert würden. Diese Auflösung entspricht einem Bild eines Flyers.

Die verschiedenen LoDs können dafür genutzt werden, gleichzeitig in der Unreal Engine geladen zu werden und in festgelegter Entfernung zum Nutzer gegen niedrigere LoDs getauscht zu werden. Ein Gebäude, das direkt vor dem Nutzer ist, wird so in LoD 3 dargestellt. Doch ab einer gewissen Entfernung wird LoD 2 oder niedriger genutzt, da der Unterschied für den Nutzer kaum sichtbar ist, aber einiges an Rechenleistung eingespart werden kann. Diese Technik wird oft von Game Engines genutzt, um Rechenlast dynamisch zu reduzieren, ohne dabei die visuelle Qualität zu sehr zu beeinträchtigen.

Tabelle 4.1 stellt abhängig vom LoD dar, welche Metadaten zur Verfügung stehen und ob diese Informationen für das Projekt und damit für den Parser relevant sind.

Attribut	**LoD1**	**LoD2**	**LoD3**	**Relevant**
Attribute pro Datei				
Envelope	✓	✓	✓	✓
Attribute pro Gebäude				
ID	✓	✓	✓	✓
creationDate	✓	✓	✓	×
appearence	×	×	✓	✓
DatenquelleLage	✓	✓	×	×
DatenquelleBodenhoehe	✓	✓	×	×
Gemeindeschluessel	✓	✓	×	×
DatenquelleDachhoehe	✓	✓	×	×
DatenquelleGeschossanzahl	✓	✓	×	×
Geometrietyp2DReferenz	✓	✓	×	×
Grundrissaktualitaet	✓	✓	×	×
BezugspunktDach	✓	×	×	×
function	✓	✓	×	×
roofType	×	✓	×	×
measuredHeight	✓	✓	×	×
storeysAboveGround	✓	✓	×	×
lod[2|3]Solid	×	✓	✓	×
Address	✓	✓	×	✓
citygridUnitID	×	×	✓	×
Grundfläche 2D	×	×	✓	×
Grundhöhe NN	×	×	✓	×

Tabelle 4.1: Übersicht der Attribute pro LoD

In der Tabelle ist zu sehen, dass die Metadaten für LoD 1 und 2 mit Ausnahme von Attributen, die nur im jeweiligen LoD Sinn machen, nahezu identisch sind. Das ergibt sich daraus, dass LoD 1 direkt aus LoD 2 berechnet wird. Daher sind die Metadaten auch mit Ausnahme von `BezugspunktDach`, `Rooftype` und `LoD2Solid` dieselben. `BezugspunktDach` macht nur in LoD 1 Sinn, da bei der Berechnung von LoD 1 aus LoD 2 das Dach vereinfacht werden muss. Der Bezugspunkt bestimmt die Mitte des Daches und wird in LoD 2 nicht benötigt. Andersherum wäre es nicht sinnvoll, die Attribute `Rooftype` und `LoD2Solid` in LoD 1 zu definieren, da ein Gebäude in LoD 1 kein erkennbares Dach hat.

Auch die Metadaten zwischen LoD 1/2 zu LoD 3 unterscheiden sich. Beispielsweise fehlen die Adressdaten und viele Informationen über die Gebäude in LoD 3 vollständig. Jedoch

ist LoD 3 das einzige LoD, welches das Attribut `appearence` definiert. `appearence` beschreibt die Texturkoordinaten und welches JPEG welchem Gebäude zuzuordnen ist.

Allgemein sind die meisten dieser Metadaten für das Projekt "Sound Reality" und damit auch für den CityGML-Parser nicht relevant. Relevant ist hingegen das `<Envelope>` Node, da es den geographischen Raum, den die Datei beschreibt, abgrenzt und das Koordinatensystem daraus auszulesen ist. Außerdem ist die `ID` relevant. Die ID soll ein Gebäude klar definieren, was hier jedoch nur in LoD 1 und 2 gegeben ist. Bei LoD 3 kann es dazu kommen, dass die ID eines Gebäudes von der in LoD 1 und 2 abweicht.

Erwähnenswert sind die Attribute `appearence` und `address`, die pro Gebäude mitgegeben werden können. Beide würden einen Mehrwert im fortlaufenden Projekt bringen, müssen hier aber noch nicht berücksichtigt werden. Metadaten wie `function` oder `Gemeindeschlüssel` sind auch nicht zwingend erforderlich, aber potentiell durchaus nützlich. So kann beispielsweise die Funktion eines Gebäudes, die in `function` beschrieben wird, Hinweise über die Umgebung und die damit verbundenen Geräusche geben. Dies kann die Erzeugung der Klanglandschaft unterstützen. Wenn die Funktion eines Gebäudes "Botanisches Gewächshaus" ist, können passende Geräusche abgespielt werden, was zur höheren Realitätsnähe beitragen könnte.

Ein weiterer signifikanter Unterschied ist in der Auflösung zu sehen. In LoD 1 und 2 werden jeweils 384.626 Gebäude modelliert, in LoD 3 sind es mehr als 531.000. Der wesentliche Unterschied ist aber erst durch die Anzahl der Dreiecke, die pro Gebäude durchschnittlich berechnet werden, zu sehen. In LoD 1 werden im Durchschnitt 27,7 Dreiecke pro Gebäude berechnet, in LoD 2 sind es 33,7 und in LoD 3 94,3. Das entspricht nahezu dem Dreifachen. Durch die höhere Anzahl an Gebäuden, der größeren Anzahl an Dreiecken pro Gebäude sowie der Texturkoordinaten sind die Daten in LoD 3 um circa das 8-fache (ohne die Textur Daten) größer als die für LoD 1 und 2.

4.3 Funktionsweise des Plugins

Das Sequenzdiagramm in Abbildung 4.3 beschreibt die Funktionsweise des CityGML-Parsers.

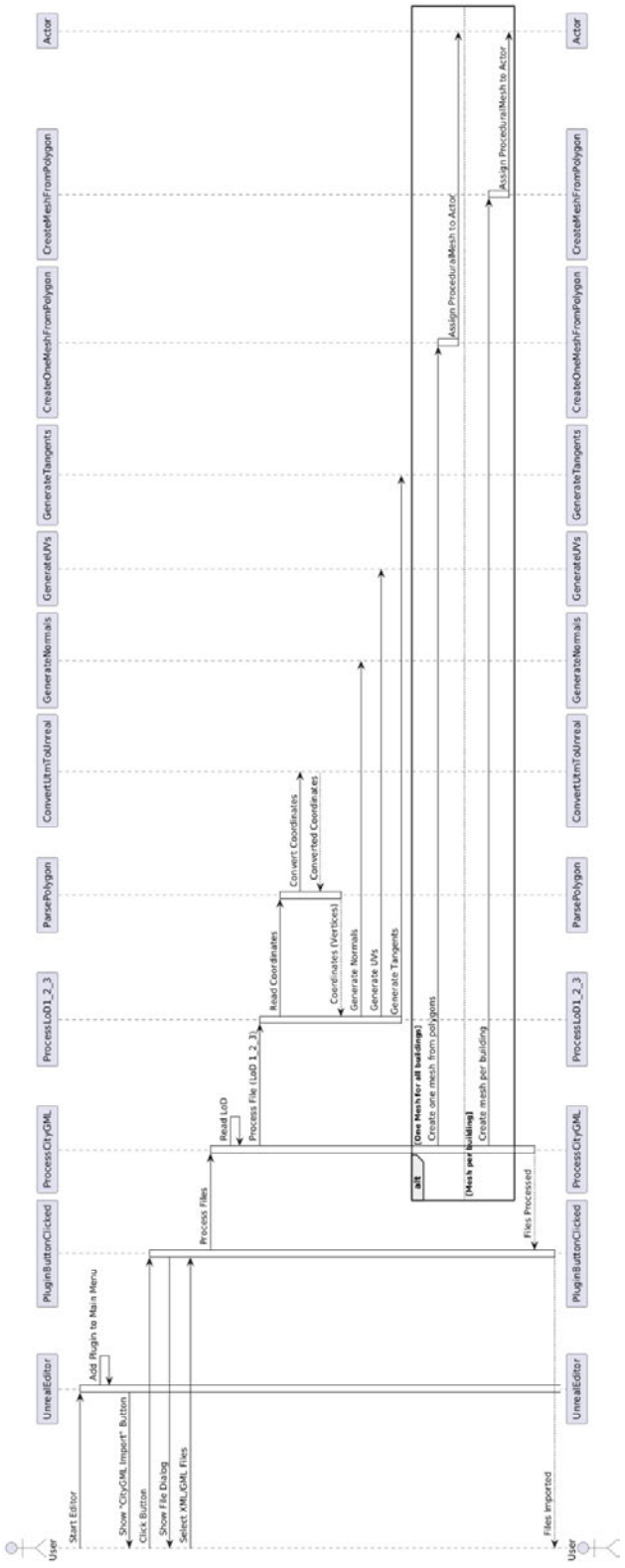

Abb. 4.3: Funktionsweise des CityGML-Parsers

Beim Starten wird das Plugin in das Hauptmenü des Unreal Editors integriert und der Menüpunkt/Button für den Import der CityGML Dateien hinzugefügt. Diesen Button findet man im Unreal Engine Editor oben rechts im "file menu" unter "project". Der Button ruft die Methode *PluginButtonClicked()* auf, wenn er angeklickt wird. Dadurch kann der Benutzer über ein Dialogfenster eine oder mehrere XML/GML Dateien auswählen, die anschließend nacheinander durch die Methode *ProcessCityGML()* verarbeitet werden.

Zuerst wird das Level of Detail ausgelesen. Es bestimmt, durch welche der drei Methoden *ProcessLoD1-3()* die Datei weiterverarbeitet wird. Diese drei Methoden sind wegen der strukturellen Unterschiede der verschiedenen LoDs nötig (vgl. Kap. 4.2). Sie bilden die Grundlage zum Auslesen der Koordinaten, indem sie an den passenden Stellen Hilfsfunktionen aufrufen. In CityGML Dateien bestehen die Gebäude aus mehreren Flächen. Diese Flächen (auch Polygone genannt) sind als Listen von Punkten dargestellt. Die Punkte stellen die Koordinaten dar und werden durch die Methode *ParsePolygon()* ausgelesen und in Arrays abgespeichert. Die Punkte müssen noch konvertiert, skaliert und verschoben werden, um in einem für Unreal Engine passendem Format vorzuliegen, wofür *ConvertUtmToUnreal()* verantwortlich ist.

In der Computergrafik werden Polygone (bzw. Flächen) durch Dreiecke dargestellt. Die Zerlegung eines Polygons in Dreiecke (auch Triangulation genannt) ist nötig, um in der Unreal Engine Geometrie darzustellen. Die Listen an ausgelesenen Punkten reichen dafür nicht aus. Die Triangulation wird hier von einer Hilfsmethode aus dem Plugin SpatialGeometryTools übernommen. Um die geometrischen Formen aus den CityGML Dateien in der Unreal Engine darzustellen, sind die Punkte und definierten Dreiecke genug. Die vorberechneten Daten können über ProceduralMeshes dargestellt werden, welche wiederum einem Actor Objekt übergeben werden. Entweder übernimmt das die Methode *CreateOneMeshFromPolygon()* oder *CreateMeshFromPolygon. CreateOneMeshFromPolygon()* erstellt genau ein *ProceduralMesh* pro Import. Wenn jedoch pro Gebäude ein Mesh erzeugt werden soll, kann die Methode *CreateMeshFromPolygon* genutzt werden. Diese erstellt ein Mesh pro Gebäude, was es ermöglicht, jedem Gebäude klare Eigenschaften zuzuordnen, was aber mit erheblichen Leistungseinbußen einhergeht.

Die Methoden *GenerateNormals()*, *GenerateUVs()* und *GenerateTangents()* berechnen die Normalen, Tangenten und Texturkoordinaten der Flächen, damit die Gebäude beleuchtet und texturiert werden können. Dadurch sehen die Gebäude realistischer aus (vgl. Kap. 4.4.1/4.4.2).

4.4 Meshgenerierung für die Unreal Engine

Wie im vorherigen Kapitel beschrieben, müssen im Plugin nicht nur die Punkte bzw. Vertizen ausgelesen werden, sondern auch passende Dreiecke trianguliert werden. Um die Dreiecke zu bestimmen, existieren verschiedene Triangulationsalgorithmen. Besprochen werden hier der FAN- und der Earcut Algorithmus. Weitere Triangulationsalgorithmen wären der Sweep Line Algorithmus oder die Delaunay Triangulation. Beide Algorithmen haben eine gute Laufzeit und sollten die Polygone korrekt zerlegen können. Jedoch wurden beide nicht implementiert, weil der Earcut Algorithmus die Polygone bereits korrekt zerlegt. Auch wenn Earcut etwas langsamer ist, reicht das in der Praxis für diesen Anwendungsfall vollkommen aus.

4.4.1 FAN-Algorithmus

Der FAN- Algorithmus wird oft in der Computergrafik verwendet, da er einfach umzusetzen ist. Die Dreiecke werden von einem Pivot Punkt aufgespannt, der in diesem Fall der erste Punkt aus der Punkteliste ist. Beispielsweise wird das erste Dreieck durch den ersten, zweiten und dritten Punkt aufgespannt, während das zweite Dreieck aus dem ersten, dritten und vierten Punkt gebildet wird. Nach dem Muster wird weiter verfahren, bis alle Punkte des Polygons verarbeitet wurden. Der Algorithmus ist leicht zu implementieren, hat eine gute Laufzeit, ist aber nur für konvexe und lochfreie Polygone anwendbar. Abbildung 4.4 zeigt beispielhaft ein konvexes, lochfreies Polygon, welches durch den FAN Algorithmus zerlegt wird.

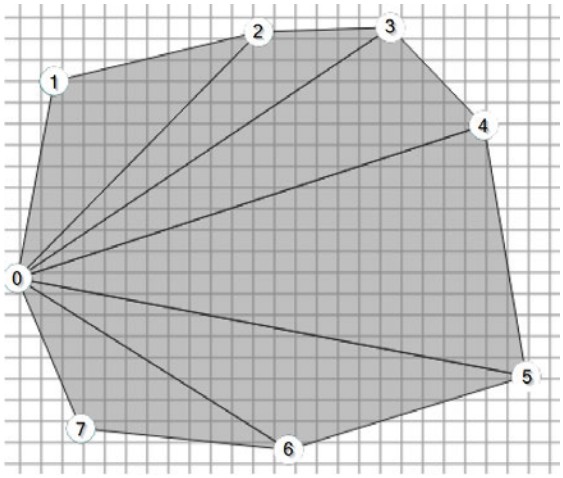

Abb. 4.4: Durch den FAN Algorithmus zerlegtes Polygon

Durch seine Einfachheit stößt er aber auch recht schnell an seine Grenzen. Für konkave Flächen oder Flächen mit Löchern ist der Algorithmus nicht geeignet. Die erzeugten Dreiecke überlappen sich, sodass Löcher nicht dargestellt werden und sich einige Flächen doppeln. Die Überlappungen sind bei Gebäuden ohne Beleuchtung und ohne Texturen oft unauffällig. Sobald aber Beleuchtungseffekte berücksichtigt werden, treten optische Fehler auf (vgl. Abb. 4.5 in Kap. 4.4.2). Daher ist es nötig, einen anderen Algorithmus zu nutzen, insbesondere wenn die Beleuchtung mitberechnet wird oder die Gebäudestrukturen komplexer werden, wie es ab LoD 2 und 3 der Fall ist.

4.4.2 Earcut-Algorithmus

Um auch die etwas komplexeren Polygone korrekt darstellen zu können, wurde anstatt des FAN Algorithmus der Earcut Algorithmus benutzt. Der Earcut Algorithmus startet mit einer geordneten Liste an Eckpunkten des Polygons, die die äußere Form und Struktur beschreiben, und durchläuft die Liste iterativ, um bei jeder Iteration ein Dreieck zu bilden. Bei jeder Iteration wird ein "Ohr" identifiziert. Ein Ohr ist ein Dreieck, welches zwei Bedingungen erfüllt. Erstens muss es konvex sein, d. h. dass der Winkel zwischen den zwei Kanten nicht über 180 Grad betragen darf. Zweitens darf sich das Dreieck nicht mit anderen Punkten überschneiden, also muss es vollständig im Polygon liegen und darf keine anderen Punkte innerhalb der Fläche beinhalten.

Sobald der Algorithmus ein solches Ohr identifiziert hat, wird das Dreieck aus dem Polygon geschnitten und zu den bereits definierten Dreiecken hinzugefügt. Das restliche Polygon wird kleiner und der Algorithmus iteriert so oft über das restliche Polygon, bis nur noch drei Eckpunkte übrig sind, welche das letzte Dreieck bilden.

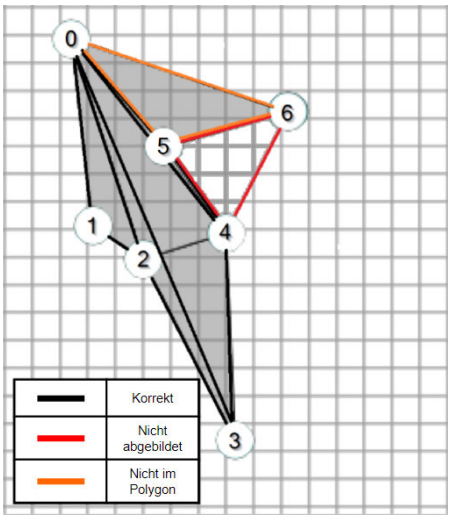

Abb. 4.5: Polygon FAN-Algorithmus Fehler

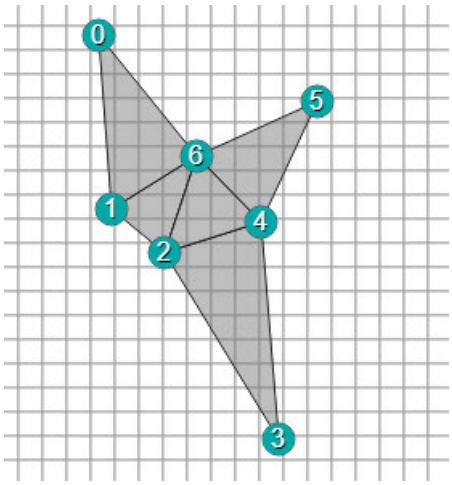

Abb. 4.6: Earcut zerlegt konkaves Polygon

Abbildung 4.5 zeigt, welche Fehler durch die Anwendung des FAN Algorithmus mit einem konkaven Polygon entstehen. Die schwarz umrandeten Dreiecke sind korrekt, doch das letzte Dreieck, welches orange umrandet ist, bildet eine Fläche, welche nicht im Polygon liegt. Zusätzlich wird hier das rote Dreieck nicht abgebildet, obwohl es im Polygon sein sollte. Auf der rechten Seite in Abbildung 4.6 ist zu sehen, wie der Earcut Algorithmus mit solchen Dreiecken umgeht. Die Punkte 0,3 und 5 sind klare Beispiele für Eckpunkte, die Ohren bilden. Nachdem diese entfernt wurden, kann jeder der anderen Punkte auch ein Ohr bilden und die restlichen 3 Punkte bilden dann das abschließende Dreieck.

Der Earcut Algorithmus ist komplexer als der FAN Algorithmus und hat im schlechtesten Fall eine Zeitkomplexität von (O(n^2)), wobei n die Zahl an Eckpunkten des Polygons ist. Das bedeutet, dass die maximale Zeit, die der Algorithmus zur Triangulation benötigt, quadratisch mit der Anzahl der Punkte steigt. Diese hohe Laufzeit tritt dann auf, wenn das Dreieck viele konkave Punkte hat, also wie ein Zick Zack aufgebaut ist. Im besten Fall hat der Algorithmus eine Zeitkomplexität von (O(n)). Das ist der Fall, wenn ein einfaches konvexes Polygon vorliegt, weil bei jedem Schritt leicht ein Ohr identifiziert werden kann. Durchschnittlich ist die Laufzeit zwar zwischen (O(n^2)) und O(n), aber

näher an der linearen O(n) Laufzeit, da auch moderat komplexe Formen eher zu O(n) tendieren. Außerdem ist der schlechteste Fall in der Praxis recht selten.

Trotz der theoretisch recht hohen Laufzeit des Algorithmus im schlechtesten Fall wird Earcut häufig in der Computergrafik eingesetzt. Das liegt daran, dass sowohl konvexe als auch konkave Dreiecke, wenn sie keine Überschneidungen haben, trianguliert werden können. Zudem ist die praktische Laufzeit wie beschrieben eher linear als quadratisch und damit gut. Ein weiterer Vorteil ist auch in diesem Fall, dass sich der Algorithmus einfach umsetzen lässt.

Nach Anwendung des Earcut Algorithmus werden die CityGML Daten in Unreal wie in Abbildung 4.7 bis 4.9 gerendert. Dabei sind die Gebäude selbst bereits erkennbar, jedoch fehlt noch die Texturierung, und die Beleuchtung der Gebäude ist noch nicht möglich.

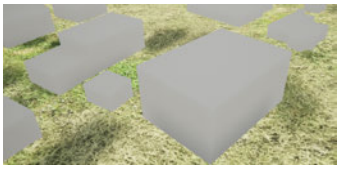

Abb. 4.7: LoD 1 Haus

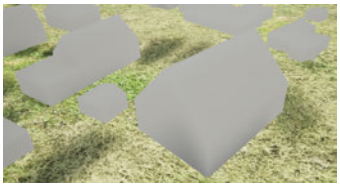

Abb. 4.8: LoD 2 Haus

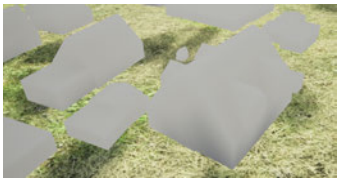

Abb. 4.9: LoD 3 Haus

4.4.3 Beleuchtung und Texturierung

In der Computergrafik wird Beleuchtung verwendet, um Oberflächen realistischer darzustellen, indem Lichtquellen simuliert werden, mit denen die Oberflächen interagieren. Die Texturierung ergänzt den Prozess, indem sie den Oberflächen weitere visuelle Details hinzufügt, die durch die Geometrie nicht erfasst werden können. Für die korrekte Beleuchtung und Texturierung müssen Normalen, Tangenten und Texturkoordinaten berechnet werden.

Normalvektoren sind für die Beleuchtungsberechnung essenziell, da sie die Ausrichtung einer Fläche beschreiben. Dafür wird im CityGML-Parser die Methode `GenerateNormals()` verwendet. Diese berechnet zwei Vektoren innerhalb des Polygons, indem jeweils zwei Punkte voneinander subtrahiert werden, und bildet die Flächennormale mithilfe des Kreuzprodukts der beiden Kanten. Der Normalenvektor wird auf die Einheitslänge normalisiert. Mithilfe dieses Normalenvektors kann der Winkel zwischen der Lichtquelle, welche von der Unreal Engine gestellt wird, und der Normalen der Oberfläche berechnet werden.

Um 2D Texturen korrekt auf die 3D Modelle abbilden zu können, werden Texturkoordinaten benötigt. Diese 2D Koordinaten legen fest, wie die 2D Textur auf die Punkte des 3D Modells projiziert wird. Die Texturkoordinaten werden von der Methode `GenerateUVs()` berechnet, indem jeder Vertex auf eine Ebene projiziert wird. Auf welche Ebene projiziert wird, hängt von der Lage des Vertex im Raum ab. Das Ziel ist es, Verzerrungen der Textur zu minimieren. Wenn zum Beispiel die Z-Koordinate des Vertex am größten ist (d. h. der Punkt am weitesten von der XY-Ebene entfernt ist), wird die Textur auf die Z-Ebene projiziert, sodass X und Y als 2D-Texturkoordinaten dienen.

Um die Texturen realistisch darstellen zu können, sind Tangenten entscheidend. Tangentenvektoren geben an, in welche Richtung die Textur auf die Oberfläche projiziert wird. Diese Aufgabe wird von der Methode `GenerateTangents()` übernommen. Der Tangentenvektor wird dabei pro Fläche über die geometrische Struktur der Fläche (den Kanten) und die Differenzen zwischen den Eckpunkten der Texturkoordinaten berechnet. Dieser zeigt die Richtung der Textur auf der Oberfläche an. In Kombination mit den Normalen und den Texturkoordinaten ermöglicht der Tangentenvektor eine korrekte Texturprojektion und realistische Lichtberechnung.

Nach Berechnung der Normalen, Tangenten und Texturkoordinaten sehen die Häuser aus Abbildung 4.7 bis 4.9 wie in Abbildung 4.10 bis 4.12 aus. Die Defaulttextur und Beleuchtung der Gebäude tragen so bereits zu einer verbesserten räumlichen Darstellung bei.

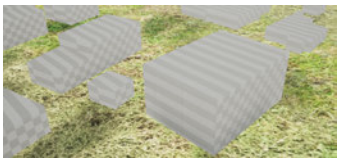

Abb. 4.10: LoD 1 texturiertes Haus

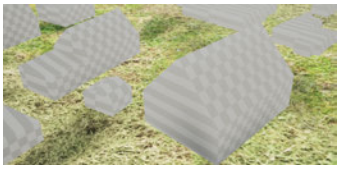

Abb. 4.11: LoD 2 texturiertes Haus

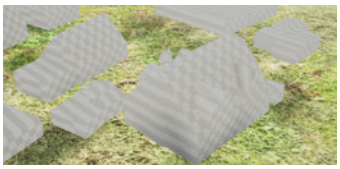

Abb. 4.12: LoD 3 texturiertes Haus

4.5 Ergebnisse

Der CityGML-Parser ermöglicht es, Gebäude in der Unreal Engine zu importieren und abzubilden. In den vorherigen Abschnitten wurden nur vereinzelt Gebäude gezeigt, dieses Plugin wurde jedoch entwickelt, um ganze Stadtteile Hamburgs darzustellen. Daher wird in Kapitel 4.5.1 die Hafencity in unterschiedlichen LoDs und Texturen präsentiert

und in Kapitel 4.5.2 wird dann auf die gesamte Stadt Hamburg skaliert. Dabei wird auch auf die Performance beim Laden und während der Laufzeit eingegangen. In Kapitel 4.5.3 wird danach vorgestellt, wie der CityGML-Parser mit Daten außerhalb Hamburgs zurechtkommt.

4.5.1 Darstellungsvarianten der Hafencity

Besonders gut wiederzuerkennen ist die Hafencity, die durch Gebäude wie die Elbphilharmonie und die Platzierung der Gebäude, die maßgeblich von der Elbe bestimmt werden, geprägt ist. Abbildung 4.13 zeigt, wie die Hafencity in der Unreal Engine anhand von LoD 3 Daten visualisiert wird und daneben, in Abbildung 4.14, sieht man denselben Abschnitt in LoD 2.

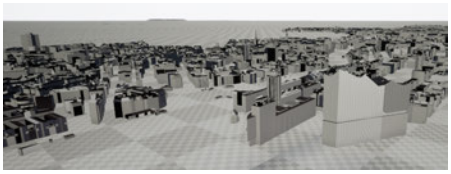

Abb. 4.13: Hafencity LoD3

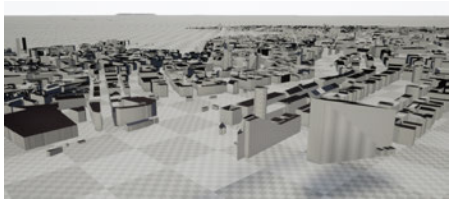

Abb. 4.14: Hafencity LoD2

Bei dem Vergleich der beiden LoDs sieht man, dass nicht nur die Dachformen von LoD 2 zu LoD 3 detaillierter sind, sondern die ganze Form des Gebäudes. Auf der linken Seite sind außerdem ein paar mehr Gebäude zu sehen und bei spitzen Gebäuden ist hier erst ab LoD 3 ein Dach sichtbar. Anhand von LoD 2 ist die Hafencity trotzdem schon klar zu erkennen. Eine Texturierung der Gebäude ist hier auch bereits durch die berechneten Texturkoordinaten möglich. In Abbildung 4.15 wird die Hafencity in LoD 2 gezeigt, wobei die Gebäude eine der Standardmaterialien aus der Unreal Engine zugewiesen bekommen haben. Die CityGML eigenen Texturen, welche ab LoD 3 vorhanden sind, wurden hier nicht verwendet.

Abb. 4.15: Hafencity texturiert in LoD 2

4.5.2 Skalierung auf die gesamte Stadt

Darüber hinaus ist auch der Import der gesamten Stadt Hamburg möglich. Abbildung 4.16 zeigt eine Darstellung der Stadt aus der Vogelperspektive in LoD 2 und Abbildung 4.17 stellt Hamburg nördlich der Elbe dar. Dabei kann man durch die Platzierung der Gebäude klar die Alster, den Stadtpark oder den Ohlsdorfer Friedhof erkennen. Für ganz Hamburg werden in LoD 2 384.629 Gebäude durch etwa 13 Millionen Dreiecke visualisiert. Zum Vergleich wären es in LoD 1 etwa 10 Millionen Dreiecke bei gleich vielen Gebäuden und 50 Millionen Dreiecke in LoD 3 mit ca. 531.000 Gebäuden.

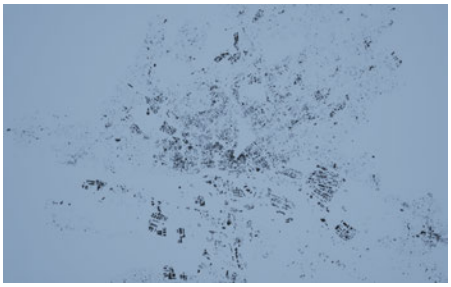

Abb. 4.16: Hamburg Vogelperspektive in LoD 2

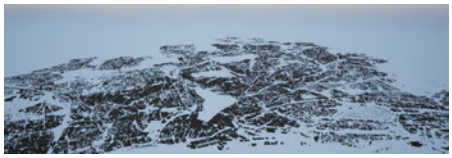

Abb. 4.17: Hamburg nördlich der Elbe in LoD 2

Der CityGML-Parser ist in der Lage, in LoD 2 durchschnittlich 1280 Gebäude pro Sekunde zu verarbeiten, was bedeutet, dass der Import der gesamten Stadt Hamburg rund fünf Minuten dauert. Der Importvorgang ist für LoD 1 ein wenig schneller, während er für LoD 3 fast fünfmal so lange benötigt. Um die ganze Stadt Hamburg untexturiert, in LoD 3 und in einem einzigen Mesh importieren zu können, würden hochgerechnet 25 Minuten gebraucht werden. Da der Importvorgang der Stadt oder kleinere Teile der

Stadt in der Regel einmalig durchgeführt wird und die Daten anschließend in der Engine genutzt werden, ist die Ladezeit akzeptabel.

Im Gegensatz zum Ladevorgang, bei dem Minuten akzeptabel sind, ist die Performance während der Laufzeit permanent relevant. Damit diese als flüssig wahrgenommen wird, darf das rendern nur ein Bruchteil (1/60 i. d. R.) einer Sekunde dauern. In der Unreal Engine müssen mehrere Millionen Dreiecke gerendert werden, nachdem die gesamte Stadt geladen wurde. Dass die Performance dabei stabil bleibt, liegt hauptsächlich daran, dass hier alles in einem einzigen Mesh gespeichert wird. Daher muss nur dieses eine Mesh pro Frame gerendert werden. Würden alle Gebäude durch ein separates Mesh dargestellt werden, wie es mit der Methode *CreateMeshFromPolygon* der Fall wäre, würde dies zu erheblichen Performanceproblemen führen. Diese würden sich durch das Einfrieren des Editors oder durch eine stark reduzierte Bildrate bemerkbar machen.

Die Tatsache, dass mehrere Tausend Meshes benötigt würden, wenn jedes Gebäude durch ein eigenes Mesh abgebildet werden soll, führt zu denselben Polygon- und Dreiecksanzahlen wie bei einem großen gemeinsamen Mesh. Wieso die Performance also bei 13 Millionen Dreiecken flüssig bleibt, die in einem gemeinsamen Mesh vorliegen, obwohl auch diese alle gezeichnet werden müssen, liegt an der Art und Weise, wie die Unreal Engine mit Meshes und der Rendering Pipeline umgeht. Jedes ProceduralMesh erzeugt einen eigenen Draw Call, welcher dafür sorgt, dass es in jedem Frame neu gezeichnet wird. Ein großes ProceduralMesh erzeugt nur einen Draw Call, welcher von der Rendering Pipeline effizient verarbeitet werden kann. Bei mehreren Tausend kleinen ProceduralMeshes werden zwar dieselbe Anzahl an Dreiecken gerendert, es werden jedoch tausende Draw Calls pro Frame erzeugt. Das führt zu einer erhöhten CPU Belastung, da sie jedes Mesh verwalten muss. Selbst wenn die GPU in der Lage ist, die Millionen Dreiecke zu rendern, wird die CPU durch die Verwaltung der vielen Meshes überfordert, was letztlich zu den erheblichen Performanceeinbußen führt.

4.5.3 Verarbeitung von CityGML Daten außerhalb Hamburgs

CityGML ist, wie in Kap. 2 besprochen, ein weltweit etabliertes Format. Um die Flexibilität und Kompatibilität des CityGML-Parsers zu testen, wurden daher zusätzlich CityGML-Daten aus anderen Bundesländern sowie aus verschiedenen Ländern getestet. Die meisten Bundesländer nutzen die CityGML Versionen 1 und 2. Auch die bereitgestellten LoDs gehen meist nicht über LoD 3 hinaus. Die Datengrundlage ist daher

oftmals dieselbe. In vielen Fällen konnten die Daten dadurch problemlos geladen werden. Abbildung 4.18-4.20 zeigen verschiedene erfolgreiche Importe aus verschiedenen Städten innerhalb Deutschlands.

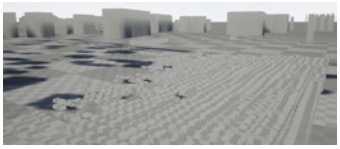

Abb. 4.18: Berlin, Jüdisches Denkmal LoD 2

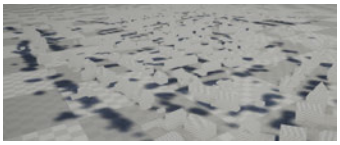

Abb. 4.19: Dortmund Schnarhorst LoD 2

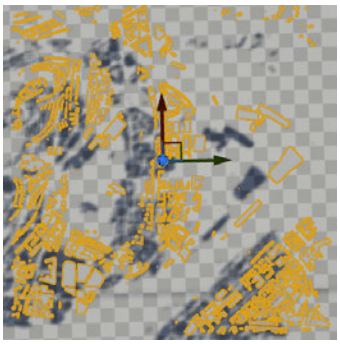

Abb. 4.20: Stuttgart Hbf. von oben LoD 2

In Abbildung 4.18 ist beispielsweise das jüdische Denkmal in Berlin mit Gebäuden im Hintergrund zu sehen, wobei selbst die einzelnen Steine modelliert wurden. Die anderen beiden Abbildungen zeigen Gebäude bzw. Strukturen aus Dortmund und Stuttgart. Die Daten konnten wie in Hamburg erfolgreich geladen und texturiert werden.

Das ist jedoch nicht überall der Fall. In einigen Bundesländern, wie beispielsweise Bayern oder Sachsen, können die Gebäude nicht dargestellt werden. Das liegt an der Kombination der CityGML Version und dem LoD. Der CityGML-Parser wurde für die Hamburger Daten entwickelt, bei denen CityGML Version 2 ausschließlich in Verbindung mit LoD 3 genutzt wird. In Bayern und Sachsen kommen jedoch andere Kombinationen von CityGML-Versionen und LoDs zum Einsatz, was zu der Inkompatibilität führt. Diese Inkompatibilität könnte man durch wenige Anpassungen im Plugin beheben.

Auch in anderen Ländern schlug der Import von CityGML Dateien fehl, jedoch aus anderen Gründen. Innerhalb Deutschlands wird für CityGML durchgehend das Koordinatensystem ETRS89 /UTM32 Nord verwendet. In anderen Ländern hingegen kommen andere Koordinatensysteme zum Einsatz, mit denen der CityGML-Parser derzeit nicht umgehen kann. Umgekehrt wäre es für Länder, welche vollständig in Europa liegen, möglich, UTM 32 Nord zu nutzen, und damit könnten auch diese geladen werden. Jedoch wäre auch eine Umrechnung der Koordinaten entweder nach ETRS89 /UTM32 oder direkt in das Koordinatensystem der Unreal Engine möglich.

4.6 Probleme bei der Umsetzung und Erfahrungsbericht

Während der Entwicklung sind Probleme aufgetreten, welche dazu geführt haben, dass alternative Lösungen und neue Ansätze erforderlich wurden. Diese werden im folgenden kurz vorgestellt:

- Performance zur Laufzeit bei vielen Meshes

  Im Projekt Sound Reality, in dessen Rahmen der CityGML-Parser geschrieben wurde, sollen die Gebäude jeweils durch eigene Meshes dargestellt werden. So können in der Unreal Engine individuelle Informationen und Materialien für jedes Gebäude zugewiesen werden. Jedoch führt eine große Anzahl von Procedural Meshes, die hier genutzt werden, zu einer hohen CPU-Auslastung, was die Performance erheblich beeinträchtigt. Das gilt insbesondere, wenn große Stadtteile mit Tausenden Gebäuden visualisiert werden sollen. Um dieses Problem vorübergehend zu umgehen, wurde im CityGML-Parser die Option implementiert, alle Gebäude in einem einzigen großen Mesh zusammenzufassen. Dadurch können auch umfangreiche Datensätze, wie etwa die gesamte Stadt Hamburg mit rund 384.000 Gebäuden, flüssig visualisiert werden, ohne dass der Editor einfriert. Diese Lösung adressiert das Problem nur temporär, da die Procedural Meshes nicht ausgetauscht wurden und die Gebäude im Projekt jeweils von einem Mesh dargestellt werden sollen. Die umfassende technische Behebung liegt jedoch außerhalb des Umfangs dieser Bachelorarbeit und wird bereits von anderen Projektteilnehmern im übergeordneten Projekt behoben (vgl. die Diskussion zu statischen Meshes in Kap. 5.1).

- Beleuchtung

  Bei der Beleuchtung der Gebäude traten mit dem FAN-Algorithmus Probleme auf, da sich viele Flächen überlappten oder an nicht korrekten Positionen dargestellt wurden. Durch die Umstellung auf den Earcut Algorithmus konnte dieses Problem behoben werden und die Flächen wurden korrekt beleuchtet. Jedoch trat ein neues Problem auf: Einige Flächen erschienen von außen durchsichtig. Dies liegt daran, dass die Rückseite von Flächen nicht gerendert wird, was auf eine Optimierung namens "Backface culling" zurückzuführen ist. Die Unreal Engine spart dabei Rechenleistung, indem sie die Rückseiten von Flächen standardmäßig nicht darstellt.

  Ob eine Fläche Vorder- oder Rückseite ist, wird durch den Normalenvektor bestimmt, dessen Richtung aus der Reihenfolge der Punkte in den Punktlisten be-

stimmt wird. Die Punktlisten sind in CityGML Dateien größtenteils rechtsdrehend (im Uhrzeigersinn), was zum rechtsdrehenden Koordinatensystem der Unreal Engine passt. Jedoch sind in den CityGML Dateien auch in einigen Fällen linksdrehende Punktelisten (gegen den Uhrzeigersinn) vorhanden. Diese fehlerhaft definierten Polygone führen dazu, dass die Normalenvektoren in die falsche Richtung zeigen, wodurch die Rückseite der Fläche als Vorderseite interpretiert wird und bei Gebäuden nach außen zeigt.

Zwar konnte auch dieses Problem zum Teil gelöst werden, indem ein zweiseitiges Material verwendet wird, welches sowohl die Vorder- als auch die Rückseite der Flächen rendert. Allerdings bleiben die Normalenvektoren bei dieser Lösung weiterhin falsch ausgerichtet. Dadurch sind die vorher durchsichtigen Flächen zwar nun sichtbar, erscheinen jedoch komplett schwarz. Der Grund dafür ist, dass der Normalenvektor in das Innere des Gebäudes zeigt, wo kein Licht einfällt, was zu einer fehlerhaften Berechnung der Beleuchtung führt. Dies ist in Abbildung 4.21 an einer Wand des Gebäudes deutlich zu sehen.

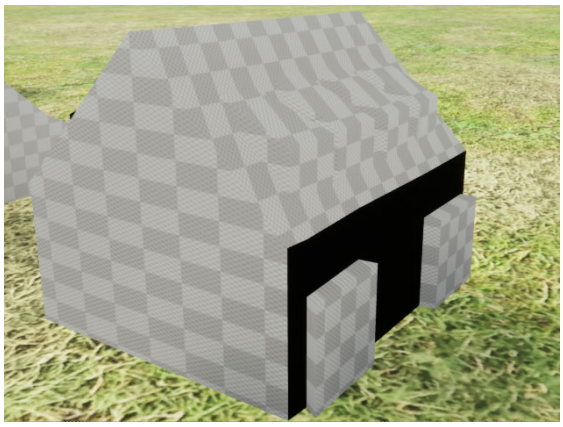

Abb. 4.21: Haus mit einer fehlerhaften Wand LoD 3

- Kompatibilität mit GDAL

  GDAL ist eine Open Source Bibliothek für die Verarbeitung und Umwandlung von Geodaten. Da CityGML eine spezielle Form von GML ist, wäre GDAL eine logische Wahl gewesen, um CityGML Dateien in eine einfach zu verarbeitende Datenstruktur zu überführen, die robust verschiedene CityGML Versionen unterstützt. Jedoch benötigt GDAL einen XML Parser wie Xerxes oder Expat, um GML Dateien einlesen zu können. Damit Xerxes oder Expat in das Projekt integriert werden können,

wäre eine vollständige Integration von GDAL in die bestehende Unreal Code Basis des Projekts nötig. Das würde eine komplexe und sehr arbeitsaufwändige Rekompilierung der gesamten Codebasis erfordern. Daher wurde dieser Ansatz verworfen und stattdessen entschieden, einen nativen Parser der Unreal Engine zu nutzen, welcher die CityGML Dateien umwandelt.

Neben den vorgestellten Problemen gibt es mehrere potentielle Erweiterungen für den CityGML-Parser:

- Generellere Verarbeitung von LoD und CityGML-Version

  Derzeit wird das LoD aus dem XML-Attribut „Name“ ausgelesen und die CityGML Version daraus abgeleitet. Bei variierenden Kombinationen aus CityGML Version und LoD führt dies zu Fehlern, wie in Kapitel 4.5.3 beschrieben wurde. Diese Informationen stehen auch im Header des `<core:CityModel>` Nodes zu Verfügung. Durch das Auslesen dieser Informationen könnte der CityGML-Parser die `ProcessLoD1-3` Methoden so anpassen, dass er flexibel mit verschiedenen Kombinationen von LoD und CityGML-Versionen umgehen kann. Mit dieser Änderung könnten beispielsweise die Daten für Bayern und Sachsen verarbeitet werden. Diese Änderung wäre leicht zu implementieren und würde es dem CityGML-Parser ermöglichen, aus vermutlich allen Bundesländern CityGML Dateien zu importieren.

- Unterstützung verschiedener Koordinatensysteme

  Um noch mehr CityGML Dateien verarbeiten zu können, sollte der CityGML-Parser in der Lage sein, mit verschiedenen gängigen Koordinatensystemen umzugehen. Hauptsächlich müssten die richtigen Offsets gefunden werden, welche aus den jeweiligen Dateien auslesbar sein sollten, und in einigen Fällen müssten die Achsen vertauscht werden. Auch diese Änderung ist recht unkompliziert und würde den CityGML-Parser in die Lage versetzen, auch CityGML Dateien aus dem Ausland verarbeiten zu können.

- Unterstützung verschiedener CityGML *Themes*

  CityGML ist in 11 *Themes* unterteilt. Darunter ist das *Theme Building*, das momentan vom CityGML-Parser verarbeitet wird, um Gebäude darzustellen. Andere *Themes* wie *Bridge* oder *Waterbody* könnten relevant werden, um Brücken oder Wassermassen zu modellieren. Obwohl die grundlegende Struktur zwischen den

Themes ähnlich ist, sucht der CityGML-Parser momentan explizit nach den XML-Attributen, die speziell für das *Building Theme* definiert sind. Dadurch ist es nicht möglich, CityGML Dateien mit anderen *Themes* durch den CityGML-Parser verarbeiten zu lassen, ohne diesen vorher anzupassen. Für die nötigen Anpassungen gibt es im CityGML-Parser nur wenige hilfreiche Strukturen. Es müsste sich komplett neu durch die spezifische XML Struktur der *Themes* gearbeitet werden. Eine solche Anpassung wäre umfangreich und der CityGML-Parser würde nur als Schablone für neue Parser dienen können.

4.7 Zusammenfassung

In diesem Kapitel wurden der Entwurf und die Umsetzung vorgestellt. Zunächst beschreibt Kapitel 4.1 die Architektur der Unreal Engine, die im Rahmen des Projekts Sound Reality und insbesondere für den CityGML-Parser relevant ist. Anschließend wurde gezeigt, wie sich der CityGML-Parser in die Architektur der Unreal Engine integriert. Kapitel 4.2 beinhaltet eine detaillierte Analyse des CityGML Formats für unterschiedliche LoDs und Versionen, bei der die geometrischen Repräsentationen erläutert sowie die relevanten Metadaten identifiziert wurden. In Kapitel 4.3 erfolgte eine Erläuterung der Funktionsweise des CityGML-Parsers anhand eines Sequenzdiagramms. Anschließend hat Kapitel 4.4 den für die Meshgenerierung verantwortlichen Teil des CityGML-Parsers vertieft. Dafür wurden der FAN und Earcut Algorithmus miteinander verglichen, wobei sich herausstellte, dass der Earcut Algorithmus qualitativ gute und effiziente Triangulationen liefert, während der FAN Algorithmus anfällig für Fehler ist. Für die korrekte Beleuchtung und Texturierung der Gebäude wurde daraufhin beschrieben, wie der Parser die Normalen, Tangenten und Texturkoordinaten berechnet. In Kapitel 4.5 wurden die Ergebnisse vorgestellt, indem zuerst verschiedene Darstellungsvarianten der Hafencity und danach der gesamten Stadt Hamburg präsentiert wurden. Dabei wurde auch festgehalten, dass die Importgeschwindigkeit zufriedenstellend ist, es jedoch anfangs zu Performance Probleme während der Laufzeit gekommen ist. Diese konnten durch die Zusammenführung aller Gebäude in ein einziges Mesh behoben werden. Anhand von exemplarischen Tests wurde zudem gezeigt, dass deutschlandweit Städte importiert werden können, sofern die Kombination aus LoD, Version und Koordinatensystem stimmt. Im letzten Kapitel (4.6) wurden Probleme, die bei der Umsetzung auftraten, sowie alternative Lösungsansätze vorgestellt. Außerdem wurden Erweiterungsmöglichkeiten des

CityGML-Parsers in Bezug auf deren Sinnhaftigkeit und dem erforderlichen Implementierungsaufwand diskutiert. Abschließend lässt sich feststellen, dass alle in Kapitel 3 definierten Anforderungen erfüllt wurden. Der Quellcode des entwickelten CityGML-Parsers ist auf Github verfügbar, wie in der Quelle [31] angegeben.

5 Fazit

In dieser Arbeit wurde ein Plugin für die Unreal Engine entwickelt, mit dem CityGML Dateien von unterschiedlichem LoD und Version in die Unreal Engine importiert werden können. Am Beispiel der Stadt Hamburg wurde für unterschiedliche Ausdehnungen gezeigt, dass CityGML Daten des Stadtgebiets für die LoDs 1-3 in der Unreal Engine dargestellt werden können. Exemplarische Tests anderer Städte haben zudem gezeigt, dass bei gleichen Voraussetzungen von LoD, CityGML Version und Koordinatensystem der Import für diese Gebiete problemlos möglich ist. Dabei konnten aber auch Konfigurationen der CityGML Dateien identifiziert werden, die ohne Änderung am CityGML-Parser nicht verarbeitet werden konnten. Die Triangulation zur Aufbereitung der Polygondarstellung in CityGML für das Rendering in Unreal wurde anhand zweier Triangulationsalgorithmen durchgeführt, wobei die jeweiligen Probleme der Algorithmen diskutiert wurden. Neben der Aufbereitung der Geometriedaten berechnet der Parser Normalen, Texturkoordinaten und Tangenten, um Gebäude korrekt zu beleuchten und zu texturieren. Im Laufe der Entwicklung hat sich herausgestellt, dass ein Mesh pro Gebäude zu Performance Problemen zur Laufzeit führen kann, da jedes dieser Meshes eigene Draw Calls erzeugt und dadurch die CPU überlastet. Daher wurde in dieser Arbeit eine Variante entwickelt, die es ermöglicht, ein Mesh pro Import zu erstellen, und damit die vielen Draw Calls umgeht.

In Kapitel 2 wurde sich zunächst mit dem Format CityGML und dessen Eignung für akustische Simulationen auseinandergesetzt. Dabei konnten relevante *Themes* identifiziert und die verschiedenen LoDs analysiert werden. Es zeigt sich, dass CityGML durch die Verfügbarkeit von LoDs und seine Erweiterbarkeit eine geeignete Grundlage für akustische Simulation in städtischen Umgebungen bietet. Die wissenschaftliche Untersuchung hat gezeigt, dass CityGML in der Stadtmodellierung bereits weit verbreitet ist und erhebliches Potential für die Stadtentwicklung bietet, sofern die Stadtmodellierung ausreichend groß angelegt ist und mit genügend konkreten Anwendungen einhergeht. Darüber hinaus wurden verschiedene Anwendungen der akustischen Simulation im urbanen Raum

vorgestellt, wobei CityGML in keiner der Echtzeit Anwendungen der akustischen Simulation verwendet wurde. Im vorgestellten Projekt SONORUS wurden akustische Echtzeit Simulationen bereits erfolgreich mit Virtual Reality (VR) kombiniert. Dabei konnte festgestellt werden, dass VR positiv zur immersiven Wahrnehmung der akustischen Umgebung beiträgt. Zur Hörbarmachung (Auralisation) der akustischen Simulationen wurden Techniken wie die geometrische Akustik eingesetzt, welche die Echtzeitberechnung von Umgebungsgeräuschen ermöglicht. Diese Berechnungen sind rechenintensiv, da sie komplexe Schallausbreitungen simulieren und auf ein Headset runterrechnen müssen. Die Kombination mit VR-Umgebungen, in denen zusätzlich auch die Visualisierung in Echtzeit erfolgt, lässt den Rechenaufwand erheblich steigen. Daher wurden Game Engines, vor allem die Unreal Engine, als vielversprechendes Werkzeug zur Realisation dieses Anwendungsbereiches vorgestellt. Insbesondere aufgrund ihrer leistungsstarken Visualisierung und fortschrittlichen Sound Engine, die durch bereits vorhandene Plugins wie Steam Audio optimiert werden kann, bietet die Unreal Engine eine geeignete Plattform für akustische VR Simulationen. Während die 3D Akustik in VR bereits heute überzeugende Ergebnisse liefert, eröffnet die Integration der CityGML Daten neue Möglichkeiten. Die Kombination von realen städtischen Daten und akustischen Simulationen ermöglicht eine realitätsnahe und detailgetreue Simulation akustischer Phänomene im virtuellen urbanen Raum, die in nahezu Echtzeit erfahrbar wird.

In Kapitel 3 wurden dann die funktionalen und nicht funktionalen Anforderungen formuliert, und in Kapitel 4 wurde die Umsetzung dieser Anforderungen erklärt. Dabei wurde die Integration des CityGML-Parsers in die Architektur der Unreal Engine und der bestehenden Code Basis thematisiert. Danach wurde sich mit der Struktur des vorliegenden Datenbestands der CityGML Daten beschäftigt. Dazu wurden die geometrische Repräsentation der Gebäude (inkl. des Koordinatensystems) beschrieben, die relevanten Metadaten identifiziert und die Unterschiede zwischen den verschiedenen LoDs und Versionen erläutert. Mit dem Wissen, wie sich das Plugin in die Unreal Engine integriert, sowie einem Verständnis für den Aufbau der CityGML Dateien wurde daraufhin beschrieben, wie das entwickelte Plugin die Dateien erfolgreich in die Unreal Engine importiert. Da die Meshgenerierung ein wesentlicher Bestandteil des Importvorgangs ist und hier zur Darstellung von 3D Objekten benötigt wird, wurden die dafür relevanten Algorithmen und Methoden vorgestellt und diskutiert. Darunter ist der vom CityGML-Parser aktuell verwendete Earcut Algorithmus, der für die notwendige Geometrie Triangulation korrekte und qualitativ gute Ergebnisse liefern konnte. Zur Optimierung der visuellen Darstellung wurden auch Normalen, Tangenten und Texturkoordinaten berechnet und beschrieben.

Ein Funktionsnachweis des CityGML-Parsers wurde mittels umfangreichen Bildmaterials der erfolgreich importierten Hamburger CityGML Daten in unterschiedlichen LoDs erbracht. Um die Generalisierbarkeit des CityGML-Parsers zu prüfen, wurden Dateien aus anderen Städten und Ländern exemplarisch getestet. Dabei stellte sich heraus, dass der CityGML-Parser bundesweit bei gleicher Kombination aus LoD und CityGML Version anwendbar ist. Auf internationaler Ebene traten jedoch Probleme durch unterschiedliche Koordinatensysteme auf, weshalb die Dateien ohne Änderung am Parser nicht importiert werden konnten. Abschließend wurden Probleme und mögliche Erweiterungen des CityGML-Parser diskutiert, aus denen sich eine Reihe an Optimierungsmöglichkeiten ergeben hat (vgl. Kap. 5.1).

Diese Arbeit hebt sich insbesondere durch den Import von CityGML Daten, die als Grundlage für akustische VR Simulationen im urbanen Raum dienen, ab. Während andere Arbeiten den Fokus entweder auf die visuelle Darstellung von CityGML oder auf die akustische Simulation städtischer Umgebungen gelegt haben, kombiniert diese Arbeit erstmals den Import realer städtischer Daten im CityGML Format mit einer Game Engine, um diese in akustischen VR Umgebungen zu nutzen. Dadurch wird eine Schnittstelle zwischen CityGML und akustischen Echtzeitsimulationen geschaffen. Mit dieser neuen Herangehensweise erweitert diese Arbeit den Anwendungsbereich von CityGML und schafft eine Basis mit offiziellen Gebäudedaten, die innovative Möglichkeiten für immersive und realitätsnahe akustische Stadtplanung hinsichtlich Lärmsituationen eröffnen.

5.1 Optimierung und Ausblick

Die CityGML Daten werden durch `ProceduralMeshes` dargestellt, wobei ursprünglich jedem Gebäude ein eigenes Mesh zugeordnet werden sollte, um individuelle Informationen zuweisen zu können. Dies führte jedoch zu Performance Problemen während der Laufzeit, da `ProceduralMeshes` veränderbar sind und daher jedes Frame neu gerendert werden müssen. Dies überlastet schnell die CPU aufgrund der hohen Anzahl an Draw Calls, die von den `ProceduralMeshes` ausgehen. Dieses Problem kann behoben werden, indem statt `ProceduralMeshes` `StaticMeshes` benutzt werden. `StaticMeshes` sind nicht veränderbar, weswegen sie nicht jedes Frame neu gerendert werden müssen. Ein weiterer Vorteil ist die Möglichkeit, `StaticMeshes` dynamisch in die Szene zu laden, ohne dass ein erneuter Import erforderlicher wird. Durch den Einsatz von `StaticMeshes` kann die

Framerate auch bei vielen Meshes stabil gehalten werden, da sie die CPU entlasten. Dies wird bereits im Projekt Sound Reality von anderen Projektteilnehmern implementiert.

Eine weitere Optimierung wäre es, die verschiedenen LoDs, die bereits geladen werden können, nicht nur zum Vergleich zu nutzen, sondern auch zur Performance Verbesserung einzusetzen. Es müssen nur die Details gerendert werden, die auch auf dem Bildschirm sichtbar sind, und dafür ist es möglich, die LoDs zu nutzen, indem dynamisch zur Entfernung der Kamera die LoDs getauscht werden. Während diese Bachelor Arbeit geschrieben wurde, wurde die Version der Unreal Engine im Projekt von 4.27 auf 5.x erhöht. Mit der Unreal Engine 5 gibt es die Option, Nanite Meshes zu nutzen, welche viele Optimierungen an der Mesh Darstellung bereits implementieren.

Nanite Meshes sind statische Meshes, welche eine Vielzahl von Optimierungen automatisch anwenden. Sie ermöglichen es, sehr hohe Polygonanzahlen (und damit hohe Dreiecksanzahlen) in Echtzeit darzustellen, ohne die Laufzeit stark zu beeinflussen. Beispielsweise wird ein LoD Ansatz implementiert, bei dem entweder eigene LoDs in das Mesh geladen werden können oder Nanite eigenständig LoDs generiert, falls keine bereitgestellt werden. Eine weitere Optimierung ist das Streaming von Geometrie, das Daten auf Abruf streamt, sodass nur das in der jeweiligen Kameraperspektive Sichtbare abgerufen und gerendert wird. Außerdem läuft Nanite in einem eigenen Rendering Durchgang, wodurch die Draw Calls umgangen werden und damit ein Bottleneck, welches in dieser Arbeit zu erheblichen Performance Problemen geführt hat (vgl. Kap. 4.6), einfach wegfällt. Nanite bietet neben diesen Funktionen noch eine Reihe weiterer Optimierungen, welche in [7] ausführlich beschrieben werden. Kurzgefasst implementiert Nanite viele Optimierungsmöglichkeiten, welche die Performance der Simulation verbessern und den LoD Ansatz überflüssig machen.

Mit der INSPIRE (Infrastructure for Spatial Information in the European Community) Richtlinie der EU, die eine gemeinsame europäische Geodateninfrastruktur schaffen will, sollen insgesamt 34 Geodatenthemen für alle Mitgliedsstaaten schrittweise vereinheitlicht werden. Mit der Standardisierung wird das Koordinatensystem auf ETRS89 festgelegt werden, ein Koordinatensystem, das der Parser bereits unterstützt. Dies eröffnet die Möglichkeit, zukünftig europaweit Gebiete importieren zu können, ohne dass Anpassungen am Parser notwendig sind. Außerdem verpflichtet INSPIRE die EU Mitgliedsstaaten dazu, einige raumbezogene Daten sowie die dazugehörigen Metadaten öffentlich zugänglich zu machen und diese zu vereinheitlichen. Dadurch wird der Austausch von Geodaten

über Landesgrenzen hinaus erleichtert, was die Anwendbarkeit des CityGML-Parsers erweitert.

Literaturverzeichnis

[1] Andrade, A.: Game engines: a survey. In: *EAI Endorsed Transactions on Game-Based Learning* 2 (2015), 11, S. 150615

[2] Audiokinetic: *Wwise - Overview*. 2024. – URL https://www.audiokinetic.com/en/wwise/overview/. – Accessed: 2024-08-09

[3] Ciekanowska, Agata M. ; Gliński, Adam Krzysztof K. ; Dziedzic, Krzysztof: *Comparative analysis of Unity and Unreal Engine efficiency in creating virtual exhibitions of 3D scanned models Analiza porównawcza wydajności silników Unity i Unreal Engine w aspekcie tworzenia wirtualnych pokazów modeli pochodzących ze skanowania 3D*. 2021

[4] Czerwinski, A ; Sandmann, S ; Stöcker-Meier, E ; Plümer, L: *Sustainable SDI for EU noise mapping in NRW-best practice for INSPIRE*. 2007. – URL https://www.researchgate.net/publication/26495544_Sustainable_SDI_for_EU_noise_mapping_in_NRW_-_best_practice_for_INSPIRE

[5] DerStandart: *300 Milliarden Dollar: Gaming-Industrie offenbar mehr wert als Musik- und Filmindustrie kombiniert*. 2021. – URL https://www.derstandard.de/story/2000126339126/300-mrd-dollar-gaming-industrie-offenbar-mehr-wert-als-musik. – Zugriff am 29. Juli 2024

[6] DImitrov, H. ; Petrova-Antonova, D.: 3D city model as a first step towards digital twin of Sofia City. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives* Bd. 43, International Society for Photogrammetry and Remote Sensing, 6 2021, S. 23–30. – ISSN 16821750

[7] Engine, Unreal. – URL https://dev.epicgames.com/documentation/de-de/unreal-engine/nanite-virtualized-geometry-in-unreal-engine. – Accessed: 2024-10-21

[8] Estorff, Otto V.: *Numerische Berechnungen in der Akustik: Brauchen wir demnächst keine Messungen mehr?* 2006. – URL https://pub.dega-akustik.de/DAGA_1999-2008/data/articles/002721.pdf

[9] Europäische Kommission, Gemeinsame Forschungsstelle (.: *Development of a methodology for the assessment of the environmental impact of CO2-based fuels.* 2012. – URL https://publications.jrc.ec.europa.eu/repository/handle/JRC72550. – Zugriff am 7. August 2024

[10] Europäischen Union, Europäisches P. und Rat der: *Richtlinien des Europäischen Parlaments.* 2002. – URL https://eur-lex.europa.eu/eli/dir/2002/49/oj. – Zugriff am 29. Juli 2024

[11] Firat, Hasan B. ; Maffei, Luigi ; Masullo, Massimiliano: 3D sound spatialization with game engines: the virtual acoustics performance of a game engine and a middleware for interactive audio design. In: *Virtual Reality* 26 (2022), 6, S. 539–558. – ISSN 14349957

[12] Gröger, Gerhard ; Kolbe, Thomas H. ; Czerwinski, Angela ; Nagel, Claus: *OpenGIS ® City Geography Markup Language (CityGML) Encoding Standard.* 2008. – URL http://www.opengeospatial.org/legal/.

[13] Hamburg, Geoportal. – URL https://geoportal-hamburg.de. – Landesbetrieb Geoinformation und Vermessung (LGV) Hamburg

[14] Jiang, Like ; Masullo, Massimiliano ; Maffei, Luigi ; Meng, Fanyu ; Vorländer, Michael: A demonstrator tool of web-based virtual reality for participatory evaluation of urban sound environment. In: *Landscape and Urban Planning* 170 (2018), 2, S. 276–282. – ISSN 01692046

[15] Klanglandschaften: *Klanglandschaften: Kennenlernen.* 2024. – URL https://klanglandschaften.ch/kennenlernen/. – Accessed: 2024-07-29

[16] Kolbe, Thomas H. ; Gröger, Gerhard ; Plümer, Lutz: Representing and Exchanging 3D City Models with CityGML. In: *Proceedings of the International Conference on 3D Geoinformation*, 2009, S. 63–78

[17] Kropp, Wolfgang ; Forssen, Jens ; Estevez-Mauriz, Laura: *Urban sound planning : the SONORUS project.* Chalmers University of Technology, 2016. – 117 S. – ISBN 9789163918599

[18] Kuhlen, Torsten ; Sabina Jeschke, Alicia D. (Hrsg.): *Exploring Virtuality*. Springer Fachmedien Wiesbaden, 2014

[19] Lewis, Michael ; Jacobson, Jeffrey: GameAIp27-lewis. In: *COMMUNICATIONS OF THE ACM* (2002). – URL https://www.researchgate.net/publication/278232144_Game_engines_in_scientific_research

[20] Malhotra, Avichal ; Raming, Simon ; Frisch, Jérôme ; Treeck, Christoph van: Open-source tool for transforming citygml levels of detail. In: *Energies* 14 (2021), 12. – ISSN 19961073

[21] Neidhardt, Annika ; Ahrens, Jens ; Pörschmann, Christoph ; Aspöck, Lukas: *Anwendungen der Virtuellen Akustik*. 2024. – URL https://www.researchgate.net/publication/378263931

[22] Open Geospatial Consortium: *Open Geospatial Consortium (OGC)*. 2024. – URL https://www.ogc.org/. – Accessed: 2024-09-05

[23] Pind, Finnur ; Jeong, Cheol-Ho ; Llopis, Hermes S. ; Kosikowski, Kacper: *Acoustic Virtual Reality-Methods and challenges*. 2018. – URL https://www.researchgate.net/publication/333105778

[24] Seto, T. ; Furuhashi, T. ; Uchiyama, Y.: ROLE OF 3D CITY MODEL DATA AS OPEN DIGITAL COMMONS: A CASE STUDY OF OPENNESS IN JAPAN'S DIGITAL TWIN "PROJECT PLATEAU". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* Bd. 48, International Society for Photogrammetry and Remote Sensing, 6 2023, S. 201–208. – ISSN 16821750

[25] Technologies, Unity: *Unity Web Player*. 2015. – URL https://web.archive.org/web/20151201203401/https://unity3d.com/webplayer. – Zugriff am 6.August 2024

[26] Valve Corporation: *Steam Audio - Overview*. 2024. – URL https://valvesoftware.github.io/steam-audio/. – Accessed: 2024-08-09

[27] Vorländer, Michael ; Feistel, Stefan: *Raumakustische Simulation und Auralisation*. S. 1–21, Springer Berlin Heidelberg, 2023

[28] WebGL: *WebGL - A JavaScript API for Rendering Interactive 3D Graphics*. 2024. – URL https://get.webgl.org/. – Zugriff am 7. August 2024

[29] Wefers, Frank ; Pelzer, Sönke ; Bomhardt, Ramona ; Müller-Trapet, Markus ; Trapet, Markus-Müller ; Vorländer, Michael: *Audiotechnik des aixCAVE Virtual Reality-Systems*. 2015. – URL https://www.researchgate.net/publication/280979616

[30] Wickramasekera, Suranga ; Weblin, Garratt ; Pothier, Boris: *How Open Standards are making Game Engines more accessible for Serious Simulation applications-A use case based on Unreal Engine*. 2021. – URL https://www.researchgate.net/publication/368707317

[31] Wiedenfeld, Mirco. – URL https://github.com/Mirwie/CityGMLImporter. – Accessed: 2024-10-22

[32] Xia, Haishan ; Liu, Zishuo ; Efremochkina, Maria ; Liu, Xiaotong ; Lin, Chunxiang: Study on city digital twin technologies for sustainable smart city design: A review and bibliometric analysis of geographic information system and building information modeling integration. In: *Sustainable Cities and Society* 84 (2022), 9. – ISSN 22106707

Erklärung zur selbständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original