TU WIEN Informatics

Vereinfachung von BIM Geometrie für den Einsatz in Gebäudeautomationssystemen

Ein automatisierter Ansatz

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Ing. Christoph Stampfel, BSc
Matrikelnummer 1125580

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Ao.Univ.Prof. Dipl.-Arch. Dr.phil. Georg Suter

Wien, 30. April 2021

Christoph Stampfel	Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

BIM geometry simplification for building management systems:

an automated approach

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering/Internet Computing

by

Ing. Christoph Stampfel, BSc
Registration Number 1125580

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Ao.Univ.Prof. Dipl.-Arch. Dr.phil. Georg Suter

Vienna, 30th April, 2021

Christoph Stampfel	Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Ing. Christoph Stampfel, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. April 2021

Christoph Stampfel

Danksagung

Ich möchte mich an dieser Stelle bei all denjenigen bedanken, die mich während meiner Arbeiten für diese Diplomarbeit unterstützt und motiviert haben. Ein ganz besonderer Dank geht an Wolfgang Kastner, der mir das Vertrauen geschenkt und mir erneut die Möglichkeit geboten hat, eine wissenschaftliche Arbeit am Institut für Computer Engineering zu schreiben. Besonderer Dank gebührt auch Georg Suter, der mir mit seinem herausragenden Fachwissen immer zur Seite stand und dadurch diese Arbeit erst ermöglichte. Ganz besonders bedanken möchte ich mich bei meiner Frau Katharina, die während den Arbeiten an dieser Diplomarbeit auf sehr viel gemeinsame Zeit verzichten musste und jederzeit hinter mir gestanden ist. Ohne diese großartige Unterstützung wäre diese Arbeit nicht möglich gewesen. Zu guter Letzt möchte ich auch meiner Familie und meinen Freunden danken, da sie mir im Laufe meines Studiums immer tatkräftig zur Seite gestanden sind und mich emotional unterstützt haben. Einen ganz besonderen Dank möchte ich an meinen verstorbenen Papa richten. Obwohl er diese großartige Zeit nicht mehr miterleben kann, wird er immer bei mir sein. Sein Stolz und seine Anerkennung haben mir den Weg zu diesem Abschluss geebnet. Leider ist er viel zu früh von uns gegangen.

Kurzfassung

In der heutigen Zeit ist nahezu jedes moderne Bürogebäude mit einem *Gebäudeautomationssystem* (GA-System) ausgestattet, welches sich um die Steuerung, Regelung und Kontrolle von Heizung, Lüftung und Klimaanlage kümmert. Moderne Systeme sind jedoch nicht auf diese Funktionen beschränkt, sondern verwalten zusehends immer mehr Bereiche im Gebäude. Aufgrund gegenwärtiger Umweltprobleme und der aktuellen Klimakrise wird das Energiesparpotential moderner GA-Systeme immer wichtiger. Um dieses Ziel zu erreichen, benötigt ein modernes Gebäude eine große Menge an Informationen, welche von einer Vielzahl von Sensoren bereitgestellt werden. Diese Sensoren liefern eine sehr große Menge an Daten, welche entsprechend aufbereitet werden müssen, um daraus Erkenntnisse erzielen zu können. Mittlerweile bedient man sich moderner Methoden aus dem Forschungsbereich *Visual Analytics*, um die Daten im Kontext des Gebäudes zu visualisieren. Grundlage dafür bildet das BIM-Modell des Gebäudes, welches bereits die wichtigsten Daten für die Verwendung in GA-Systemen bereitstellt. Die Geometrie des Gebäudemodells ist aber oft zu komplex und lenkt den Benutzer von den wesentlichen Informationen ab. Im Rahmen dieser Arbeit wurde ein Algorithmus entwickelt, welcher die Geometrie eines BIM-Modells vereinfacht, um sie in einem GA-System verwenden zu können. Ausgangspunkt des Algorithmus ist eine IFC-Datei, welche das Gebäudemodell enthält. Die im Modell enthaltene Geometrie wird vereinfacht und in Form von *Brep Solid Geometry* exportiert. Im Zuge dieser Arbeit wurde ein Prototyp entwickelt, welcher anhand ausgewählter Gebäudemodelle evaluiert und getestet wurde.

Abstract

Nowadays, nearly every modern office building is equipped with a *Building Management System* (BMS) which is responsible for monitoring and controlling of Heating, Ventilation and Air Conditioning (HVAC) in an intelligent way. Moreover, such systems are not only limited to HVAC, but are also able to control more and more other equipments in the building. Considering our environment, it becomes increasingly important to operate such buildings in an energy efficient way. To be able to fulfill these requirements, a modern building needs a large number of sensors. These sensors are generating a large amount of data which is extremely valuable for optimizing the energy efficiency of the installed systems e.g. by visual data analysis methods. By visualizing the building together with the installed sensors and their measured values, building maintenance becomes much easier. The aim of this master thesis is to present an automated approach which allows the simplification of the geometrical representation of the spatial structure and the technical equipment of a building in such a way that the geometry can be used in a BMS. To reach this goal, a prototype implementation will be developed, which is able to read the geometric information of a BIM in the industry standard format *Industry Foundation Classes* (IFC) and transforms the geometry into a simplified form. The simplified geometry then is exported in the form of *Brep Solid Geometry* to be used in a BMS. To verify the results of the prototype implementation, a set of buildings models has been selected.

Inhaltsverzeichnis

KAPITEL 1

Einführung

1.1 Motivation und Problemstellung

Heutzutage ist nahezu jedes moderne Bürogebäude mit einem GA-System ausgestattet, welches sich um die Steuerung, Regelung und Kontrolle von Heizung, Lüftung und Klimaanlage kümmert. Solche Systeme sind jedoch nicht nur auf diese Funktionen beschränkt, sondern verwalten zusehends auch andere Bereiche im Gebäude, wie z.B. Beleuchtung und Abschattung. Betrachtet man die gegenwärtigen Umweltprobleme und die damit verbundene Klimakrise, wird es immer wichtiger entsprechende Gegenmaßnahmen zu setzen. Ungefähr 40% des Energieverbrauchs innerhalb der Europäischen Union ist auf den Energieverbrauch von Gebäuden zurückzuführen. Die Hälfte davon, wird von den technischen Anlagen der Gebäudetechnik verbraucht [PLOP08, DCB17]. Aufgrund des großen Einsparungspotentials wird es immer wichtiger, diese Gebäude möglichst energieeffizient zu betreiben, ohne den Komfort für die Menschen spürbar zu beeinträchtigen. Um diese Anforderungen zu erfüllen, benötigt ein modernes Gebäude eine Menge an Informationen, welche durch den Einsatz von Sensoren zur Messung verschiedener Parameter wie z.B. Temperatur, Luftfeuchtigkeit und Luftgüte zur Verfügung gestellt werden. Diese Sensoren produzieren eine sehr große Menge an Daten, welche zur Optimierung des Energieverbrauchs benötigt werden. Damit aus diesen großen Datenmengen Erkenntnisse erzielt werden können, bedient man sich moderner Methoden aus dem Forschungsbereich *Visual Analytics*. Dabei werden die Sensordaten im Kontext des Gebäudes, z.B. in Form einer Zeitleiste, visualisiert, um Ineffizienzen beim Energieverbrauch intuitiv erkennen zu können. Ein Beispiel wäre die räumliche Darstellung der Sensoren in einem 3D-Modell des Gebäudes, um Probleme an der thermischen Hülle zu erkennen.

In den vergangenen Jahrzehnten wurde *Computer Aided Design* (CAD) eingesetzt, um den Entwurf, die Errichtung und den Betrieb von Gebäuden aller Art zu unterstützen. Moderne Bauprojekte werden jedoch immer komplexer, vor allem deshalb, weil häufig sehr viele verschiedene Unternehmen daran beteiligt sind. Eine Untersuchung hat ergeben, dass

über 90% der Arbeitskräfte im Baugewerbe, in Unternehmen arbeiten, die aus weniger als 10 Mitarbeiter bestehen [Eur05]. In den letzten Jahren wurde ein neues Konzept mit dem Namen *Building Information Modeling* (BIM) immer populärer, welches viele dieser Probleme lösen soll. BIM ist ein neuer Ansatz, um mithilfe eines gemeinsamen Datenmodells und dazu passender Software verschiedene Phasen eines Bauprojektes zu unterstützen. Ausgehend vom 3D-Modell eines Gebäudes werden zusätzlich semantische Informationen gespeichert. Diese Informationen umfassen die räumliche Struktur, Prozess-relevante Dokumente (z.B. Angebote, Rechnungen von beteiligten Unternehmen), Lage und Art der technischen Anlagen, sowie deren Funktionsweise. Da alle Projektbeteiligten an einem gemeinsamen Modell arbeiten, erhalten alle eine einheitliche Sicht auf den Projektfortschritt. In Zeiten von CAD wurden meist mehrere, unterschiedliche Modelle erstellt, um unterschiedliche Sichten wie z.B. Architektur und Gebäudetechnik, abzubilden. Diese Vorgehensweise führte oft zu Problemen und Inkonsistenzen, da nachträgliche Änderungen in mehreren Modellen gleichzeitig eingearbeitet werden mussten und die Software-Komponenten untereinander nicht kompatibel waren.

Damit BIM und die Idee des gemeinsamen Datenmodells überhaupt realisiert werden konnten, war es notwendig ein standardisiertes Modell zu entwickeln. In den vergangenen Jahren haben sich die IFC zum de-facto Standard rund um BIM herauskristallisiert. IFC ist unter ISO 16739 international standardisiert und erlaubt das Speichern von semantischen Informationen zu den einzelnen Strukturen und Anlagen in einem Gebäude, welche um verschiedene Arten von geometrischen Darstellungen ergänzt werden können.

Aus Sicht eines GA-System ist es sinnvoll, die statischen Daten aus einem BIM-Modell wiederzuverwenden, um das Gebäude zusammen mit der räumlichen Struktur und den technischen Anlagen zu visualisieren. Die semantische Information aus dem BIM-Modell erleichtert die Arbeit des *Facility Managements*, wenn sie im Kontext des Gebäudes dargestellt wird [PMB+18]. Durch die Wiederverwendung der vorhandenen Daten können Kosten gespart werden, da der Aufwand für das Neu-Erstellen des Modells wegfällt und dadurch weniger Fehler auftreten.

Die geometrische Information in BIM-Modellen ist sehr komplex, da die Geometrie von Wänden, Fenstern, Türen, etc. in vollem Detailgrad gespeichert wird. Diese Details sind bei Verwendung in GA-Systemen jedoch nicht notwendig, da eine vereinfachte Sicht völlig ausreichend ist, um die Benutzbarkeit für den Endanwender zu verbessern, z.B. in Form von orientierten Bounding Boxes. Für die Darstellung der räumlichen Struktur eines Gebäudes ist es z.B. nicht notwendig, die Tiefe von Wänden zu visualisieren. Wichtiger ist die Darstellung der Lage von technischen Anlagen innerhalb der räumlichen Struktur und die Nachbarschaftsbeziehungen von Räumen im Gebäude.

Ein weiteres Problem in BIM ist die oft fehlende, semantische Information zu den technischen Anlagen. Die geometrische Information der Anlagen ist in vielen Software-Produkten in Form von entsprechenden Komponenten-Katalogen vorhanden, jedoch werden wichtige technische Informationen, wie z.B. Gerätetyp, Seriennummern, Wartungsintervalle, uvm. nicht exportiert. Diese Informationen sind jedoch für den Betrieb eines Gebäudes sehr wichtig, um z.B. die Wartung der technischen Anlagen planen zu können.

1.2 Zielsetzung

Im Rahmen dieser Diplomarbeit wird ein automatisierter Ansatz vorgestellt, um die geometrische Darstellung der räumlichen Strukturen und der technischen Anlagen eines Gebäudes aus einem BIM-Modell zu exportieren. Die Geometrien werden nicht nur originalgetreu exportiert, sondern auch vereinfacht, damit sie z.B. in einem GA-System geeignet weiterverwendet werden können. Um dieses Ziel zu erreichen, wurde ein Prototyp entwickelt, welcher die geometrischen Informationen aus einem BIM-Modell extrahiert und anschließend vereinfacht. Ausgangspunkt für die Transformation ist ein Gebäudemodell, welches im standardisierten IFC-Format vorliegt. Die vereinfachte Geometrie wird anschließend in einem standardisierten Format exportiert, welches z.B. von einem modernen *Modeling Kernel* verarbeitet werden kann.

1.3 Methodische Vorgehensweise

Die methodische Vorgehensweise dieser Arbeit gliedert sich in folgende Punkte:

Literaturrecherche Um die erwarteten Ergebnisse zu erreichen, wurde eine umfassende Literaturrecherche durchgeführt. Ausgewählte Fachartikel werden im Rahmen der vorliegenden Arbeit diskutiert. Diese Vorgehensweise ist notwendig, um den aktuellen Stand der Technik zu den Themen BIM und GA-Systemen verstehen zu können. Im ersten Teil der Arbeit wird die Entstehungsgeschichte von BIM vorgestellt und wie sich diese neue Methode aus den ursprünglich verwendeten CAD-Modellen heraus entwickelt hat. Anschließend werden unterschiedliche Standards vorgestellt, welche sich im Laufe der vergangenen Jahre im Bereich BIM entwickelt haben. IFC gilt mittlerweile als de-facto Standard, um Gebäudemodelle zwischen unterschiedlichen Softwaresystemen austauschen zu können. Der Prototyp, welcher im Zuge dieser Arbeit entwickelt wurde, verwendet IFC als Ausgangspunkt. Aus diesem Grund werden die Grundlagen von IFC erklärt, wobei auf Aspekte, die im Prototypen verwendet werden, etwas detaillierter eingegangen wird. Das Ziel dieser Arbeit ist die Wiederverwendung der statischen Daten in einem BIM-Modell zur Verwendung in GA-Systemen, deshalb werden unterschiedliche Ansätze aus diesem Bereich, basierend auf den Artikeln der Literaturrecherche, diskutiert.

Prototyp Basierend auf den Erkenntnissen der Literaturrecherche wird ein Prototyp vorgestellt, welcher die in einem BIM-Modell vorhandenen Geometrien exportiert und anschließend vereinfacht, sodass diese in einem GA-System wiederverwendet werden können. Dieser Prototyp arbeitet vollständig automatisiert, sodass es mithilfe dieser Lösung möglich ist, ein BIM-Modell regelmäßig, auch während der Entwurfsphase im Zusammenspiel mit einem GA-System zu analysieren und validieren. Durch diese Vorgehensweise kann das Gebäude, schon während der Entwurfsphase, in Bezug auf weitere Aspekte, wie z.B. Energieeffizienz, optimiert werden. Der Algorithmus zur Vereinfachung der Geometrie lehnt sich an bestehenden Arbeiten an und wurde

entsprechend für den Einsatz in GAS adaptiert. Jeder Schritt des Algorithmus wird anhand von vereinfachten Geometrien auch in grafischer Form vorgestellt. Das soll sicherstellen, dass der Leser sowohl die einzelnen Schritte als auch die Zwischenergebnisse im Detail versteht.

Implementierung/Evaluierung Nach der Vorstellung des Prototyps werden die Ergebnisse evaluiert und ungelöste Probleme bzw. Sonderfälle, die vom Algorithmus nicht abgedeckt werden, diskutiert. Ein reales Bürogebäude wird als Beispiel für den Evaluierungsprozess herangezogen. Dieses Gebäude wurde bereits in anderen Arbeiten in den Bereichen BIM und GA-Systemen verwendet und ist deshalb auch für unseren Zweck gut geeignet. Am Ende des praktischen Teils dieser Arbeit wird die Implementierung des Prototyps im Detail vorgestellt. Die eingesetzten Werkzeuge und die Architektur des Prototyps werden diskutiert.

Zusammenfassung/Reflexion Am Ende dieser Arbeit werden die Ergebnisse und die gewonnenen Erkenntnisse zusammengefasst und die wichtigsten Aspekte hervorgehoben, um dem Leser ein Verständnis für die wesentlichen Teile dieses Forschungsbereichs näher zubringen. Um auf den aktuellen Stand der Technik aufzubauen, werden die Ergebnisse dieser Arbeit mit jenen anderer Arbeiten gegenübergestellt. Am Ende werden die offenen Punkte der vorliegenden Arbeit zusammengefasst, um den Leser zu motivieren, sich ausführlicher mit diesem Thema zu befassen.

1.4 Struktur der Arbeit

In Kapitel 2 wird der aktuelle Stand der Technik in den Bereichen BIM und GA-Systemen dargestellt. Dabei wird auf die geschichtliche Entwicklung von CAD eingegangen und wie sich daraus BIM entwickelt hat. Es werden verschiedene Standards in diesem Bereich vorgestellt, die auch im Prototyp dieser Arbeit Verwendung finden. Anschließend wird die aktuelle Bedeutung von GA-Systemen vorgestellt und wie die Arbeit des Facility Managements mit modernen Methoden aus dem Forschungsbereich Visual Analytics unterstützt werden kann. Anschließend werden verschiedene Methoden vorgestellt, um die statischen Daten aus einem BIM-Modell mit den dynamischen Daten eines GA-System verknüpfen zu können. In Kapitel 3 wird ein Algorithmus vorgestellt, der es ermöglicht, die Geometrien aus einem BIM-Modell zu exportieren und so zu vereinfachen, dass diese effizient in einem GA-System verwendet werden können. Dabei werden bestehende Ansätze in diesem Bereich herangezogen und entsprechend adaptiert. In Kapitel 4 wird ein Prototyp vorgestellt, der diesen Algorithmus implementiert. Dabei wird sowohl auf die verwendeten Werkzeuge, als auch auf die Architektur der Implementierung eingegangen und diskutiert. Der vorgestellte Algorithmus und die konkrete Implementierung des Prototypen werden in Kapitel 5 anhand von ausgewählten Gebäudemodellen evaluiert. Abschließend werden die wichtigsten Punkte dieser Arbeit nochmals zusammengefasst und reflektiert. Probleme und offene Punkte dieser Arbeit werden anschließend aufgezeigt und diskutiert, um den Leser zu motivieren, sich tiefgehender mit diesem Thema auseinanderzusetzen.

KAPITEL 2

Stand der Technik

2.1 Computer Aided Design (CAD)

Unter dem Begriff CAD versteht man die Unterstützung von konstruktiven Aufgaben mittels elektronischer Datenverarbeitung, vor allem zur Herstellung von Produkten wie z.B. Autos, Flugzeugen oder Gebäuden. Die visuelle Darstellung von Objekten zur Unterstützung des Konstruktionsprozesses ist vorwiegend ein Hilfsmittel für die initiale Entwicklung des Entwurfs durch den Designer und die spätere Umsetzung des Projektes anhand des erstellten Modells. Viele Jahrzehnte lang wurden diese Aufgaben manuell ausgeführt und die einzigen Hilfsmittel, die verwendet wurden, waren Papier, Tinte und Rechenbrett [CP14]. Aufgrund der gestiegenen Anforderungen im Laufe der Zeit waren die traditionellen Methoden nicht mehr effizient genug und es wurde deshalb notwendig, modernere Methoden zu entwickeln, die den Konstruktionsprozess besser unterstützen [Asa20].

Ivan Sutherland gilt als Begründer von CAD, da er durch die Entwicklung von Sketchpad im Jahr 1963, die damals innovative Möglichkeit schuf, Daten über einen Stift am Computer einzugeben, um damit 2D Modelle am Computer anzufertigen. Bis dahin war es lediglich üblich, Programme im Batch-Modus unter Verwendung von Lochkarten und Magnetbändern zu betreiben [TF10]. In den 1970er Jahren wurden die ersten kommerziellen Produkte auf den Markt gebracht und damit begann auch die großflächige Verbreitung von CAD. Am Beginn unterstützten die Programme lediglich 2D Abbildungen von 3D Objekten, wie es auch bei traditionellen Methoden üblich war. Wenige Jahre später wurde aber bereits damit begonnen, Programme zu entwickeln, die es auch erlauben 3D Modelle zu erstellen [TF10]. Im Laufe der Jahre wurde CAD intensiv weiterentwickelt und es entstanden verschiedene Modellierungstechniken, die auch heute noch weit verbreitet sind. In der Literatur [TF10] werden die verschiedenen Entwicklungsstufen von CAD häufig in folgende 4 Generationen eingeteilt:

2D In der ersten Generation werden die modellierten Objekte als Projektion auf eine Ebene in 2D dargestellt.

3D Darstellung der Objekte im 3-dimensionalen Raum und die damit geschaffene Möglichkeit beliebige 2D Ansichten des Modells anzufertigen.

B-Rep In dieser Generation werden die Objekte durch ihre begrenzenden Oberflächen beschrieben. Dieses Modell nennt man Begrenzungsflächenmodell (engl. Boundary Representation, kurz B-Rep).

CSG In der 4. Generation dominiert die Erstellung der Objekte in Form von konstruktiver Festkörpergeometrie (engl. Constructive Solid Geometry, kurz CSG). Bei dieser Form werden Körper durch die Anwendung der booleschen Operationen Vereinigung, Durchschnitt und Differenz aus primitiven Grundkörpern konstruiert. Dabei wird ein Baum als Datenstruktur verwendet, der die gesamte Historie der Operationen, ausgehend vom primitiven Grundkörper bis zum Zielkörper abbildet.

Mittlerweile ist CAD weltweit verbreitet und hat sich durch seine Fähigkeiten etabliert. In der Baubranche ist CAD nicht mehr wegzudenken, da es vermutlich nicht mehr möglich wäre, die komplexen Bauvorhaben der heutigen Zeit mit klassischen Methoden abzuwickeln. Durch die gestiegene Komplexität bei modernen Bauprojekten sind mitunter viele verschiedene Unternehmen und Personen beteiligt. Die hohe Anzahl an verfügbaren, meist nicht miteinander kompatiblen CAD Systemen erschwert die Zusammenarbeit in größeren Projekten erheblich. Um dieses Problem zu lösen, wurden in der Vergangenheit mehrere Standards für den Datenaustausch zwischen CAD Systemen entwickelt. Ein Vertreter ist die Initial Graphics Exchange Specification (IGES), welche im Jahr 1979 veröffentlicht wurde [TF10]. Dabei handelt es sich um ein Datenformat, welches zum Austausch von 2D und 3D Modellen verwendet wird. Die Weiterentwicklung von IGES ist das STEP Format (Standard for the Exchange of Product model data), welches mittels ISO 10303 standardisiert wurde. Leider ist der Austausch von Modellinformationen über ein neutrales Format wie IGES oder STEP immer mit einem Datenverlust verbunden, da der Austausch lediglich auf die Geometrie beschränkt ist. Die Geometrien stellen jedoch nur das Endprodukt des Konstruktionsprozesses dar und spiegeln nicht die Intuition des Designers wider [KPIS08]. Um die Zusammenarbeit in einem größeren Projekt zu verbessern, wurde BIM entwickelt, was im nächsten Abschnitt genauer behandelt wird.

2.2 Building Information Modeling (BIM

BIM gilt als direkter Nachfolger von CAD, da durch den Einsatz von BIM, Aspekte berücksichtigt werden, die in CAD vernachlässigt wurden. Dadurch wird die Planung, Errichtung und Verwaltung von Gebäuden noch tiefgreifender unterstützt [SELT18].

Der Einsatz von CAD, als Unterstützung von konstruktiven Aufgaben, hat sehr viele Vorteile mit sich gebracht, jedoch blieben einige Aspekte, wie z.B. die Koordination

der verschiedenen Projektbeteiligten weitestgehend unberücksichtigt. CAD bietet keine Möglichkeit, um verschiedene Fachrichtungen, wie z.B. Architekten, Gebäudetechniker, o.a. beim Planungsprozess zu unterstützen. Ohne digitale Unterstützung ist der Koordinationsaufwand in einem Bauprojekt sehr hoch, da im Zuge des Projektes sehr häufig mehrere Modelle eines Gebäudes erstellt werden. Jedes Gewerk benötigt andere Informationen zur Ausführung der gewünschten Tätigkeiten, deshalb stellt jedes Modell eine unterschiedliche Sicht auf das Gebäude dar. Der Architekt benötigt andere Informationen in seinem Plan als ein Gebäudetechniker, der hauptsächlich an den einzelnen Anlagen der Gebäudetechnik interessiert ist. Die unterschiedlichen Gewerke können nicht unabhängig voneinander arbeiten. Benötigt z.B. die Gebäudetechnik einen zusätzlichen Schacht für ein Belüftungssystem, dann betrifft diese Änderung u.U. auch die Architektur des Gebäudes. In aufwändigen Koordinationsbesprechungen werden die einzelnen Modelle ohne Computerunterstützung untereinander abgestimmt, um einerseits Fehler zu erkennen und andererseits notwendige Änderungen einzuarbeiten. Es muss natürlich auch beachtet werden, dass sich nahezu bei jeder Änderung in den Modellen, auch der Preis ändert, wenn z.B. zusätzliche Bauteile eingebaut werden müssen. Dementsprechend müssen die einzelnen Gewerke regelmäßig auf Änderungen reagieren, um entsprechende Korrekturen in der Projektplanung und Fakturierung rechtzeitig vornehmen zu können [CP14].

BIM zielt darauf ab, die genannten Nachteile durch technischen Lösungen zu verbessern um somit die Koordination in einem Bauprojekt zu verbessern. In einem typischen Bauprojekt sind häufig sehr viele, kleinere Unternehmen involviert, die im Rahmen von BIM koordiniert werden müssen. Eine Untersuchung aus dem Jahr 2005, durchgeführt von der European Construction Technology Platform (ECTP), kam zu der Erkenntnis, dass über 90% der Arbeitskräfte im Baugewerbe in Unternehmen arbeiten, die aus weniger als 10 Mitarbeitern bestehen [Eur05]. Die höhere Anzahl kleinerer Akteure in einem Projekt erhöht den Koordinationsaufwand noch weiter. Abgesehen von diesem komplexen Projektumfeld kommen noch sehr hohe Kosten dazu, die durch die Inkompatibilität unterschiedlicher Softwaresysteme entstehen [GCR04].

Die theoretischen Grundlagen zu BIM wurden bereits Ende der 1970er von Studenten am Georgia Institute of Technology entwickelt [PBC13]. Der Begriff selbst wurde jedoch erst im Jahr 2002 das erste Mal in der Literatur verwendet [J.10]. BIM konnte am Beginn nur sehr eingeschränkt verwendet werden, da die technischen Möglichkeiten Ende der 1970er limitiert waren, um die damals recht aufwändige Software zu betreiben [ALBF14].

Die Definition von BIM hat sich im Laufe der Zeit immer wieder verändert. Mittlerweile haben sich aber die folgenden Definitionen bzw. Sichtweisen zu BIM durchgesetzt [Wat10]:

Software BIM wird häufig als Kategorie von Softwareprodukten verstanden. Dabei steht im Gegensatz zu CAD nicht die Geometrie im Vordergrund, sondern es werden die Daten des gesamten Gebäudes in strukturierter Form, ähnlich einer Datenbank, abgespeichert. Abhängig vom jeweiligen Anwendungsfall werden alle benötigten Informationen abgebildet, wie z.B. Bauteile, Geräte, projektbezogene Daten, u.v.m. Die verschiedenen Ansichten und Pläne werden aus den strukturierten

Daten exportiert und stellen nicht, wie bei CAD, die einzige Informationsquelle dar. Der große Vorteil bei diesem Ansatz ist, dass sich die Ansichten und Pläne bei jeder Änderung automatisch aktualisieren, um immer am aktuellen Stand zu bleiben.

Prozess Da sich die Art und Weise, wie Bauprojekte in Zukunft geplant und umgesetzt werden, durch den Einsatz von BIM enorm ändern wird, kann BIM auch als Prozess betrachtet werden. Die Tatsache, dass eine Software selten den Lebenszyklus eines Gebäudes überlebt, unterstützt diese Sichtweise zusätzlich. Der zentrale Bestandteil des Prozesses ist das Gebäudemodell, welches von allen Beteiligten gemeinsam verwendet werden soll, um die Zusammenarbeit in größeren Projekten maßgeblich zu unterstützen. Dadurch entsteht eine einheitliche Sicht auf das Projekt und Fehler bzw. Missverständnisse werden reduziert und damit die Qualität erhöht.

Der Kern von BIM ist das gemeinsame Modell, welches ein Gebäude über den gesamten Lebenszyklus begleiten soll. Es handelt sich bei diesem Modell somit um eine Art "Lebendes Dokument", welches das Gebäude von der Planung ausgehend, über die Errichtung, bis hin zum Abriss begleitet. Dabei ist es wesentlich, dass alle Beteiligten jederzeit Zugriff auf das gemeinsame Modell haben und somit immer der aktuelle Planungsstand zur Verfügung steht. Alle benötigten Daten, beginnend von den Bauteilen über die Projektdokumente bis hin zu den technischen Einrichtungen der Gebäudeautomation, sollen abgebildet werden. Diese Denkweise ändert die bisherige Arbeitsweise so enorm, dass die Sichtweise, es handle sich bei BIM lediglich um eine neue Kategorie von Softwareprodukten, etwas kurzsichtig erscheinen lässt.

Die verschiedenen Entwicklungsstufen von BIM werden in der Literatur sehr häufig in folgende Dimensionen eingeteilt [CP14, MSM19, CAE18]:

3D 3D Modelle gelten als direkte Weiterentwicklung von 2D Modellen mit dem großen Vorteil, dass daraus beliebige 2D Ansichten erstellt werden können.

4D Zusätzlich zum 3D Modell wird der Faktor Zeit in die Modellierung aufgenommen. Das Projekt kann durch diese Erweiterung in Projektphasen unterteilt werden. Eine Visualisierung und Simulation der Zeitpläne ermöglicht eine verbesserte und dadurch genauere Projektplanung.

5D Ergänzend zum Faktor Zeit werden durch die 5. Dimension die Kosten ins Modell aufgenommen. Die beiden Faktoren Kosten und Zeit zusammen ermöglichen eine vereinfachte Kostenschätzung und Optimierung der beiden Variablen, da verschiedene Alternativen simuliert werden können.

6D Die 6. Dimension widmet sich dem Thema Nachhaltigkeit und ermöglicht die Optimierung in Richtung Umweltschutz und Energieeffizienz.

7D Bis zur 7. Dimension wird im Wesentlichen die Planung und Errichtung von Gebäuden unterstützt, jedoch der anschließende Betrieb außer Acht gelassen. Die 7.

Dimension widmet sich daher dem Thema Gebäudemanagement. Die Verwaltung der Gebäude erfordert zusätzliche semantische Informationen zur Gebäudestruktur und den technischen Anlagen, wie z.B. Raumnummern, Wartungsintervalle, Garantiebedingungen, u.v.m.

Das wichtigste Ziel von BIM ist die Verbesserung der Koordination der einzelnen Beteiligten in einem Projekt und die dadurch erzielte Fehlerreduzierung, unabhängig davon, ob BIM als Softwarekategorie oder Prozess verstanden wird [Jon14]. BIM definiert jedoch nicht, wie dieses Ziel genau erreicht werden kann oder welche Informationen im Gebäudemodell genau enthalten sein sollen. Möchte man die erstellten Modelle zwischen verschiedenen Software-Komponenten austauschen, treten ähnliche Probleme auf, wie sie auch bei CAD auftreten. Die Hersteller verwenden in ihrer Software proprietäre Formate, welche herstellerübergreifend nicht kompatibel sind. Aus diesem Grund werden im nächsten Abschnitt die Industry Foundation Classes vorgestellt, welche einen offenen Standard zur digitalen Beschreibung von Gebäudemodellen darstellen.

2.3 Industry Foundation Classes (IFC)

2.3.1 Geschichte

Die IFC sind ein offener und internationaler Standard zur digitalen Beschreibung von Gebäudemodellen [Ifcc]. Der Standard ist unter ISO 16739 registriert und wird durch buildingSMART International [Bui], ehemals Industrieallianz für Interoperabilität (IAI), definiert und laufend weiterentwickelt. In Abbildung 2.1 sind einige der bisher veröffentlichten Versionen von IFC dargestellt, wobei aus Gründen der Übersichtlichkeit auf ausgewählte kleinere IFC Versionen nicht eingegangen wird.

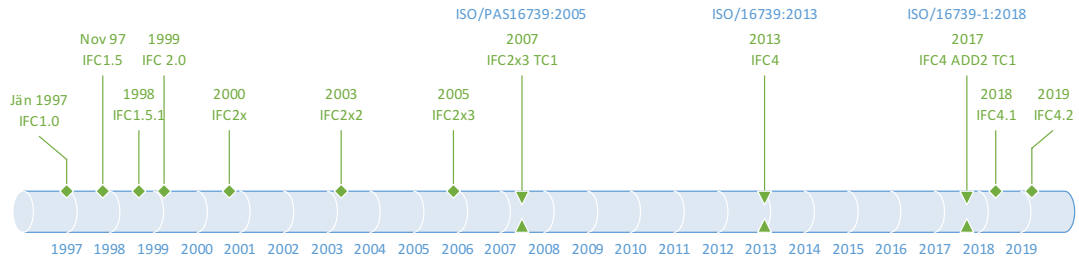

Abbildung 2.1: Veröffentlichte IFC Versionen [Ifcc, Kiv99, Lie10, LK12]

Mit der Entwicklung von IFC wurde im Jahr 1995 begonnen und bereits 2 Jahre später die Version 1.0 veröffentlicht. Der Funktionsumfang der ersten Version war recht klein und im Wesentlichen auf architekturelle Aspekte beschränkt [LK12]. IFC1.0 wurde lediglich zur Entwicklung von Prototypen verwendet, um den Standard auf seine Praxistauglichkeit zu testen. Insgesamt 17 Firmen haben sich dabei beteiligt und lieferten entsprechendes

Feedback, um die nächste Version von IFC zu verbessern, welche noch im selben Jahr unter der Versionsnummer 1.5 veröffentlicht wurde [Kiv99]. Diese Version bereitete in der Praxis jedoch Probleme, welche durch die Veröffentlichung von IFC1.5.1. behoben wurden [LK12].

Mit der Version 2.0 wurde IFC um zusätzliche Funktionen für Kostenschätzung, Planung und das Bauwesen erweitert, welche im Jahr 1999 veröffentlicht und durch die Version IFC2x im Jahr 2000 weiter verbessert wurde [Lie10]. Die Kern-Funktionalität von IFC 2x erreichte im Jahr 2005 den ISO Status *Public Available Specification* unter der Nummer *ISO/PAS16739* [LK12]. Im Jahr 2003 wurde IFC2x2 veröffentlicht, welche den bisher größten Funktionszuwachs verzeichnen konnte [Amo15]. Neben Erweiterungen, wie z.B. die strukturelle Analyse und die Unterstützung des Gebäudemanagements, war es erstmals möglich, 2D Modellgeometrien abzubilden. Da im Zuge der Entwicklung von IFC2x2 zu wenig Zeit blieb, um die Neuerungen auf ihre Praxistauglichkeit zu überprüfen, war es notwendig in den darauffolgenden Jahren den Fokus auf Stabilisierung der bestehenden Funktionalität zu setzen. Im Jahr 2006 wurde die verbesserte Version IFC2x3 veröffentlicht, welche die bis dahin erste, kommerziell erfolgreiche Version in der Geschichte von IFC wurde [Lie10].

Die nächste größere Version von IFC wurde im Jahr 2013 unter der Versionsnummer IFC4 veröffentlicht und umfasst neue Funktionen wie z.B. die Standortplanung und 5D Modellierung. Zusätzlich setzt diese Version vermehrt auf Nachhaltigkeit und bietet Möglichkeiten, um Umwelteinflüsse in seinen Planungen miteinbeziehen zu können [CDDA14]. IFC4 wurde unter der Nummer ISO/16739:2013 standardisiert und durch die Version IFC4 ADD2 TC1 mit der Nummer ISO/16739-1:2018 weiter verbessert. Die nächsten Versionen 4.1 und 4.2 enthalten kleinere Erweiterungen des Kerns im Bereich Brückenbau und eine verbesserte Basis zur Entwicklung von Erweiterungen in den Bereichen Schienen, Straßen, Tunnel, Häfen und Wasserstraßen [Ifcc].

IFC wurde im Laufe der letzten Jahrzehnte enorm weiterentwickelt. Die aktuellen Versionen enthalten weit mehr als 700 Entitäten, über 1.400 Attribute und knapp 1.800 Property Sets [Amo15]. Auch das Ziel von IFC hat sich im Laufe der Jahre verändert. Am Beginn sollte das Format die Interoperabilität in den Bereichen Architektur, Ingenieurwesen, Bauwesen und Gebäudemanagement sicherstellen. Mittlerweile ist IFC nicht mehr auf diese Bereiche beschränkt, sondern hat sich zum offenen Standard für den Austausch von Daten im Rahmen von BIM in den unterschiedlichsten Bereichen weiterentwickelt [Amo15]. Aufgrund der gestiegenen Komplexität wurde es für die Industrie immer schwieriger, den Standard in die eigenen Produkte zu integrieren, um entsprechende Zertifizierungen zu erhalten. Aus diesem Grund wurden in der Vergangenheit Maßnahmen getroffen, die diesen Prozess vereinfachen.

Eine dieser Maßnahmen war die Einführung von *Model View Definitions (MVD)*, welche lediglich eine Teilmenge des IFC-Modells verwenden, um gewisse Anwendungsfälle in einer Domäne abzubilden. Durch diese Erweiterung ist es in konkreten Softwarelösungen nicht mehr notwendig, das gesamte IFC-Modell zu implementieren, sondern lediglich die MVDs,

die für die eigenen Anwendungsfälle benötigt werden. Zertifizierungen beschränken sich seitdem auf MVDs [LK12].

Information Delivery Manuals (IDM) waren die zweite Maßnahme, um der gestiegenen Komplexität entgegenzuwirken. Es handelt sich dabei um eine von buildingSMART standardisierte Vorgehensweise, um Anforderungen zur Unterstützung von Konstruktionsprozessen zu beschreiben. Die Anforderungen beschränken sich dabei im Wesentlichen auf technische Implementierungsdetails und auf Rollen-basierte Prozess-Workflows für die Endanwender. Dabei wird darauf eingegangen, wann ein Prozess angewendet werden soll, welche Akteure beteiligt sind, welche Informationen ausgetauscht werden und wie der Prozess durch Softwarelösungen unterstützt werden soll [LK12].

2.3.2 Architektur/Datenmodell

Das Datenmodell hinter IFC ist objektorientiert und verwendet das Konzept der Vererbung. Dabei stellen Typen, die sich weiter oben in der Vererbungshierarchie befinden, allgemeinere Konzepte dar. Typen, die sich weiter unten in der Hierarchie befinden, stehen für spezielle Konzepte, die oft nur in einer gewissen Anwendungsdomäne Verwendung finden. Das IFC-Modell wird mithilfe der Sprache EXPRESS definiert, welche für die Modellierung von Produktdaten entwickelt wurde. EXPRESS ist Teil des STEP Standards, welcher unter ISO-10303-11 2004 standardisiert wurde.

IFC wird in vielen unterschiedlichen Bereichen eingesetzt, wodurch die Erweiterbarkeit des Modells eine wichtige Eigenschaft darstellt. Zur Unterstützung dieser Eigenschaft und Verbesserung der Wartbarkeit wurde das Modell in 4 verschiedene Schichten strukturiert [And18]. Die Schichten sind in Abbildung 2.2 dargestellt. In der Architektur von IFC gibt es eine wichtige Grundregel zur Reduktion der Komplexität. Objekte einer darunterliegenden Schicht dürfen niemals Objekte einer darüber liegenden Schicht referenzieren. Umgekehrt nutzen Objekte darüber liegender Schichten, Funktionen und Eigenschaften von darunterliegenden Schichten. Diese Regel ist, durch die nach unten zeigenden Pfeile, in der Abbildung dargestellt. Die allgemeinen Konzepte des Modells sind in den tiefer liegenden Schichten angesiedelt, wohingegen die spezialisierten Konzepte in den höheren Schichten zu finden sind [LK12].

Resource Layer Die unterste Schicht wird Resource Layer genannt und enthält allgemeine Datenstrukturen, die im gesamten IFC Datenmodell verwendet werden. Beispiele hierfür sind Datentypen für Koordinaten, Vektoren, Datumsangaben, Kosten u.v.m. Die verschiedenen Typen werden im Datenmodell weiter in s.g. *Resourcen* strukturiert, die aus Gründen der Lesbarkeit in der Abbildung weggelassen wurden. Da es sich bei der Resource Layer um die unterste Schicht handelt, sind die Konzepte in dieser Schicht von anderen Konzepten unabhängig.

Core Layer Die Elemente in der Core Layer bilden, aufbauend auf den allgemeinen Datenstrukturen des Resource Layers, die Grundlagen des Datenmodells. In dieser Schicht befindet sich der Kern (Kernel) des Modells, welcher wichtige Basisklassen,

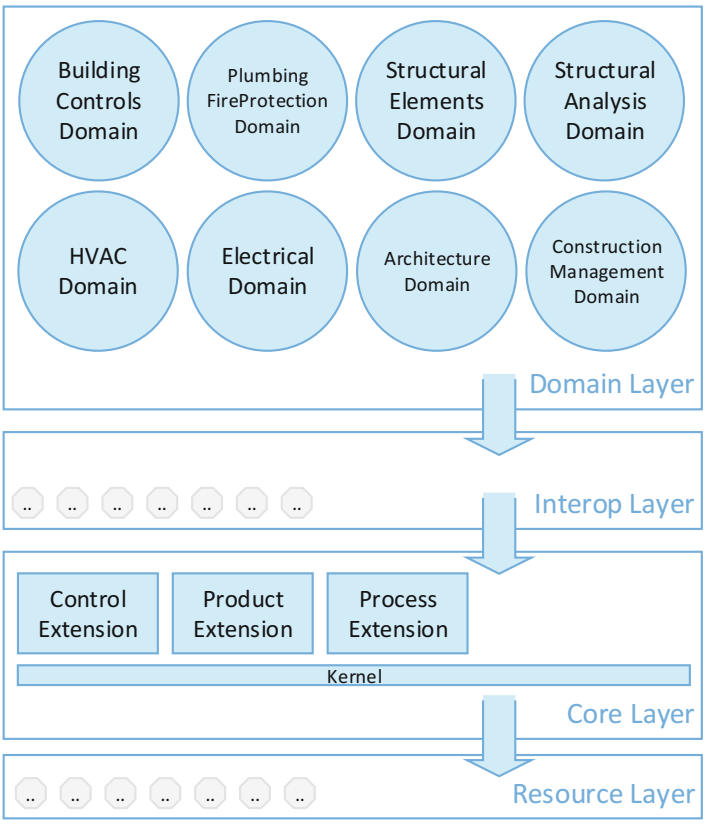

Abbildung 2.2: Schichtenarchitektur des IFC Datenmodells [Ifcc]

der oberen Vererbungshierarchien, beinhaltet, z.B. *IfcRoot*, *IfcObject*, *IfcProduct*, *IfcRelationship*, u.v.m. Der Kern bildet auch die Basis für die Schema-Erweiterungen *Control Extension*, *Product Extension* und *Process Extension*. Die wichtigste Erweiterung für unsere Zwecke bildet die *Product Extension*. Sie beschreibt physische und räumliche Objekte eines Gebäudes und deren Beziehungen untereinander. Die *Process Extension* bildet die Basis für die Abbildung von Prozessen, Abläufen, Arbeitsplanung und Aufgaben. Mithilfe der *Control Extension* werden Steuerungsfunktionen, basierend auf den Konzepten im Modell, abgebildet.

Interoperability Layer Die Interoperability Layer stellt die Zwischenschicht zwischen der Core Layer und der Domain Layer dar und verknüpft somit die spezialisierten Konzepte der darüber liegenden Schicht mit den allgemeinen Konzepten der darunterliegenden Schicht. Diese Schicht enthält Klassen, die auch in verschiedenen Anwendungsdomänen verwendet werden können, wie z.B. *IfcWall*, *IfcWindow*, *IfcDoor*, u.v.m.

Domain Layer In der obersten Schicht befinden sich alle Klassen, die auf eine Anwendungsdomäne spezialisiert sind. IFC unterstützt folgende Anwendungsdomänen: Architektur, Gebäudesteuerung, Baumanagement, Elektrik, Heizung-Lüftung-Klimatechnik (HLK), Installation, Brandschutz und Gebäudestruktur.

IFC unterscheidet im Wesentlichen zwischen Objekten, Beziehungen und Eigenschaften.

Objekte sind direkt von der Klasse *IfcObject* abgeleitet und werden für die Abbildung verschiedener Elemente im Raum, in der Zeit oder anderen Dimensionen verwendet. Dabei kann es sich z.B. um Prozesse für Planung oder Betrieb, Ressourcen im Sinne des Projektmanagements oder um physische Objekte im Gebäude handeln. Die für uns wichtigste Klasse ist *IfcProduct*, welche Objekte im räumlichen Kontext beschreibt. Aus diesem Grund können diese Objekte mit einer oder mehreren geometrischen Repräsentationen bzw. einer Position im Raum verknüpft sein. Elemente, die von *IfcProduct* abgeleitet werden, sind z.B. *IfcWall*, *IfcWindow* oder *IfcDoor*, welche Bauelemente im Gebäude darstellen. Die Gebäudestruktur lässt sich mit Objekten vom Typ *IfcSpatialElement* darstellen. Die wichtigsten Vertreter sind *IfcSite*, *IfcBuilding*, *IfcBuildingStorey* und *IfcSpace*.

Ein wichtiger Bestandteil von IFC sind Beziehungen, da mit ihnen die Semantik im Modell definiert wird. Beziehungen sind in IFC als Objekte mit Eigenschaften (engl. Objectified relationships) modelliert. Diese Art der Modellierung hat den Vorteil, dass alle relevanten Eigenschaften einer Beziehung direkt im jeweiligen Objekt abgebildet sind. Objekte stehen durch Beziehungen nicht alleine da, sondern werden miteinander verknüpft. Beispielsweise kann eine Tür als Teil eines Raumes und ein Raum als Teil eines Geschosses modelliert werden. Aneinandergrenzende Wände können so semantisch verknüpft werden, sodass diese Eigenschaft aufgrund der Daten im Modell und nicht nur über die Geometrie ausgedrückt werden kann. IFC kennt auch das Konzept von inversen Beziehungen, sodass in einem Objektbaum in beide Richtungen navigiert werden kann. Z.B. ist ein Raum nicht nur Teil eines Geschosses, sondern es kann auch ausgehend vom Geschoss zu allen Räumen navigiert werden, die sich in diesem Geschoss befinden.

Da IFC in sehr vielen Anwendungsbereichen eingesetzt werden kann, ist es nicht möglich, alle benötigten Attribute in den dazugehörigen Klassen direkt abzubilden. Diese anwendungsspezifischen Eigenschaften können in Form von *Property Sets* definiert und Objekten über eine spezielle Beziehung oder einen benutzerdefinierten Objekttyp zugeordnet werden. Dadurch lässt sich das Datenmodell beliebig erweitern, jedoch muss darauf geachtet werden, dass *Property Sets* möglichst wiederverwendet werden, um die Interoperabilität zu gewährleisten.

Ein großer Vorteil des IFC-Modells ist, dass neben der semantischen Bedeutung aller Objekte eines Gebäudes auch die dazugehörige geometrische Darstellung abgebildet werden kann. Diese Möglichkeit wurde vom STEP Standard übernommen, auf dem das IFC Format basiert. Die Basis hierfür bildet die Klasse *IfcProductRepresentation*, die mit der jeweiligen Instanz von *IfcProduct* verknüpft wird. In IFC ist es möglich, dasselbe

Objekt mit mehreren unterschiedlichen geometrische Darstellungen zu verknüpfen. Die Speicherung verschiedener Darstellungen ist notwendig, damit verschiedene Implementierungen, die für sie am besten passende geometrische Darstellung verwenden können. IFC erlaubt u.a. folgende Darstellungsformen für die Geometrien der Objekte: Triangulated Surface, Bounding Box, Boundary Representation und Constructive Solid Geometry.

2.4 BIM in der Gebäudeautomation

Der Begriff *Gebäudeautomation* umfasst alle technischen Einrichtungen, die sich um die Steuerung, Regelung und Überwachung bestimmter Anlagen im Gebäude kümmern. Üblicherweise sind damit Anlagen in den Bereichen Heizung, Lüftung, Klimatechnik, Licht, Brandschutz und Sicherheit gemeint [WS97]. Das Ziel von GA-Systemen ist die Steigerung der Energieeffizienz von Gebäuden ohne den Komfort für den Menschen spürbar zu beeinträchtigen. Untersuchungen haben ergeben, dass 40% des Energieverbrauchs innerhalb der Europäischen Union auf den Energieverbrauch von Gebäuden zurückzuführen ist. Die Hälfte davon wird von den technischen Anlagen der Gebäudetechnik verbraucht [PLOP08, DCB17]. Vor allem in Zeiten der Klimakrise wird der Umweltschutz und die damit einhergehende Reduktion des Energieverbrauchs und der Emissionen immer wichtiger. Durch den enormen Anteil der Gebäudetechnik an der Energiebilanz ergibt sich in diesem Bereich ein enormes Einsparungspotential. Aus diesem Grund ist es wichtig, weitere Optimierungen, einerseits an den technischen Anlagen und andererseits an den zugrundeliegenden Prozessen vorzunehmen.

Aufgrund von Entwicklungen wie dem *Internet of Things* und *Big Data* ist die Menge an verbauten Sensoren in Gebäuden enorm gestiegen. Dadurch werden in modernen GA-Systemen sehr viele Daten verarbeitet und gespeichert. Bei diesen Daten handelt es sich in den meisten Fällen um Messwerte von Sensoren, welche in kurzen und regelmäßigen Abständen, Werte wie z.B. Temperatur, Luftfeuchtigkeit, Luftgüte, u.v.m., an das GA-System liefern. Diese Daten müssen dem Facility Management in geeigneter Form visualisiert werden, um das Gebäude überwachen und Ineffizienzen im Energieverbrauch erkennen zu können. Die Visualisierung der Daten gestaltet sich als sehr schwierig, da vor allem auch die Historie von enormer Wichtigkeit ist, um z.B. die Temperatur im Verlauf eines Tages verfolgen zu können, um etwaige Probleme aufzudecken. Derzeit werden diese enormen Datenmengen von den meisten Systemen noch nicht geeignet visualisiert, wodurch das volle Potential der GA-Systeme nicht ausgeschöpft werden kann [YE14]. Die Einbindung verschiedener Datenquellen und die kombinierte Darstellung dieser Daten schafft ein besseres Verständnis über das Gebäude und die Funktionsweise der technischen Anlagen [PMB+18].

Es hat sich gezeigt, dass die Visualisierung der technischen Anlagen und Sensoren im Kontext des Gebäudes, in Form einer 2D- oder 3D-Darstellung für den Benutzer viel sinnvoller ist, als z.B. eine tabellarische Auflistung der Sensorwerte [SFR09, SLL11]. [PMB+18] hat im Rahmen seiner Arbeit einen Prototypen entwickelt, wo die Daten aus einem GA-System einer Universität mit Positionsdaten der Studenten verknüpft und

zusammen visualisiert wurden. Ergänzend dazu konnten die Studenten Feedback zum Komfort, in Form einer Bewertung über eine App, übermitteln. Dieses Feedback wurde mit den Positionsdaten der Studenten ergänzt, um dem Facility Management geeignete Informationen zur Verfügung zu stellen. Die Daten wurden in 2-dimensionaler Form visualisiert, wodurch der Kontext der einzelnen Messwerte jederzeit ersichtlich ist. Die Geometrie des Gebäudes wurde dabei sehr vereinfacht dargestellt, da eine realitätsgetreue Darstellung für diesen Zweck nicht notwendig ist. Ergänzend zur 2D Darstellung wurde ein Dashboard entwickelt, welches die historische Entwicklung der einzelnen Messwerte und das subjektive Feedback der Studenten visualisiert. Mithilfe des Dashboards war es nun einfach möglich, Änderungen des Energieverbrauchs zu erkennen. Durch die Visualisierung der Daten im Verlauf der Zeit war es für das Facility Management nun einfacher, festzustellen, ob getroffene Maßnahmen zur Energieeinsparung erfolgreich waren oder nicht. Interessanterweise zeigte sich auch, dass die Motivation zum Energiesparen steigt, wenn die Visualisierung des Energieverbrauchs auch von den Bewohnern, oder in diesem Fall den Studenten, eingesehen und damit interagiert werden kann.

Die Darstellung der dynamischen Daten eines GA-System in Form eines 2D- oder 3D-Modells verbessert die Nutzbarkeit im Bereich des Facility Management, da die Daten nicht isoliert, sondern im Kontext des Gebäudes visualisiert werden. Auch Probleme bei technischen Anlagen lassen sich leichter erkennen, da die Lage des betroffenen Geräts direkt aus der Visualisierung abgelesen werden kann. Ein Temperatursensor, der im Winter in der Nähe eines Fensters eine niedrige Temperatur anzeigt, könnte z.B. auf ein geöffnetes Fenster hinweisen [PMB+18, SFR09, YE14].

Zur Visualisierung der dynamischen Daten des GA-System ist es naheliegend, die statischen Daten aus dem BIM-Modell des Gebäudes wiederzuverwenden, damit diese nicht aufwändig neu erstellt werden müssen. Idealerweise enthält das BIM-Modell bereits alle Informationen zu den technischen Anlagen, inklusive der geometrischen Darstellung und der Lage im Gebäude. Modelle im Zuge von BIM werden üblicherweise mit entsprechender Software (z.B. Autodesk Revit) erstellt und entweder als Datei gespeichert oder in einem Repository auf einem Server abgelegt. Die Speicherung in einem Repository hat den Vorteil, dass das Modell über das Internet abrufbar ist und mehrere Personen gemeinsam daran arbeiten können. Bei den dynamischen Daten eines GA-System, v.a. bei Sensordaten, steht die zeitliche Reihenfolge der Daten im Vordergrund. Diese Daten werden häufig in relationalen Datenbanken oder auch in speziellen *Time Series Datenbanksystemen*, die speziell auf die Speicherung von Zeitserien optimiert sind, abgespeichert.

[TSE+19] beschreibt in seiner Arbeit verschiedene Ansätze, wie Daten aus einem BIM-Modell mit Sensordaten aus dem IoT-Umfeld miteinander verknüpft werden können. Für die Verwaltung der Anlagen im Gebäude sind ähnliche Lösungen denkbar. Die Ansätze werden nachfolgend zusammengefasst und den Vor- und Nachteilen gegenübergestellt:

BIM Software Schnittstellen + relationale Datenbank Bei diesem Ansatz werden den Schnittstellen der verwendeten BIM-Software genutzt, um die Daten des BIM-Modells in eine relationale Datenbank zu exportieren. Die Sensordaten werden

ebenfalls in einer relationalen Datenbank gespeichert, sodass die Daten einfach über einen gemeinsamen Schlüssel verknüpft werden können. Ein weiterer Vorteil dieser Vorgehensweise ist, dass durch die Verwendung integrierter Schnittstellen nahezu keine Kenntnisse über IFC und die Softwareentwicklung notwendig sind. Dieser Ansatz ist aber nur bei einfachen BIM-Modellen sinnvoll, da für die Darstellung der Sensordaten entsprechende virtuelle Objekte im BIM-Modell definiert werden müssen, um die Daten visualisieren zu können.

Relationale Datenbank Die statischen und dynamischen Daten werden, analog dem vorigen Ansatz, in einer gemeinsamen Datenbank gespeichert. Es wird jedoch keine Schnittstelle der BIM-Software verwendet, sondern der Export wird, abhängig von den benötigten Daten, selbst entwickelt und ein Datenbank-Schema erstellt. Das Erstellen des Schemas kann je nach Umfang sehr aufwändig sein, jedoch vereinfacht sich die Erstellung der virtuellen Objekte zur Darstellung der Sensordaten im BIM-Modell, da sie auf den jeweiligen Anwendungsfall reduziert werden können.

Neue Abfragesprache Die BIM-Daten werden bei diesem Ansatz nicht in eine Datenbank exportiert, sondern über eine eigene domänenspezifische Abfragesprache direkt aus dem BIM- oder IFC-Modell ausgelesen. Über eine weitere Abfragesprache können die Sensordaten aus dem jeweiligen System abgefragt und mit den statischen BIM-Daten verknüpft werden. Bei Verwendung einer ausdrucksstarken, auf den jeweiligen Anwendungsfall optimierten Sprache können die Daten effizient abgefragt werden. Die Entwicklung einer neuen Abfragesprache und der dazugehörigen Werkzeuge stellt jedoch einen hohen Aufwand dar, sodass die Verwendung etablierter Abfragesprachen wie z.B. SQL oder SPARQL vermutlich zielführender ist.

Semantic Web Bei diesem Ansatz werden sowohl die statischen BIM-Daten als auch die dynamischen Sensordaten in Form von Ontologien abgebildet und miteinander verknüpft. Diese Vorgehensweise folgt dem *Semantic Web* Paradigma und ermöglicht die Verknüpfung von Daten aus unterschiedlichsten Quellen. Zur Abfrage der Daten kann die Abfragesprache SPARQL verwendet werden, die sich in diesem Bereich mittlerweile etabliert hat und dafür viele Werkzeuge existieren. Ein Nachteil dieser Vorgehensweise ist, dass sowohl die BIM- als auch die Sensordaten neu modelliert werden müssen. Die Sensordaten können nicht so effizient gespeichert werden wie in einer relationalen Datenbank, sodass mehr Speicherplatz benötigt wird, da Daten auch redundant abgelegt werden müssen.

Hybridansatz Der Hybridansatz verbindet das *Semantic Web* Paradigma mit den Vorteilen einer relationalen Datenbank. Die statischen BIM-Daten werden in Form einer Ontologie abgebildet und die Sensordaten verbleiben in der relationalen Datenbank, sodass sowohl die statischen als auch die dynamischen Daten möglichst effizient gespeichert werden können. Verknüpft werden die Daten in Form einer Referenz-ID, welche die Daten in der Ontologie mit den Sensordaten in der Datenbank verknüpft.

Der Hybridansatz ist nach derzeitigem Stand der erfolgversprechendste Ansatz, um die statischen Informationen zum Gebäude und den technischen Anlagen aus dem BIM-Modell mit den dynamischen Daten aus einem GA-System zu verbinden [TSE+19].

Montazer et al. haben im Rahmen ihrer Arbeit [MSP+19] ein Workflow-Modell entwickelt, um die Erstellung und die Anpassung eines integrierten Gebäudemodells, im Rahmen des Lebenszyklus eines Gebäudes, zu unterstützen. Das integrierte Gebäudemodell verknüpft die statischen Daten aus dem BIM-Modell mit den dynamischen Daten aus einem GA-System. Im Zuge der Entwicklung wurden unterschiedliche Akteure identifiziert, welche unterschiedliche Verantwortlichkeiten im Workflow besitzen. Der *BIM Modeler* ist zuständig für die Erstellung bzw. Weiterentwicklung des BIM-Modells und ist mit dem IFC-Datenmodell vertraut. Für die Speicherung und Verwaltung der dynamischen Daten des GA-System ist der *BEMS Modeler* zuständig. Der *System Integrator* kümmert sich um die Verknüpfung der statischen und dynamischen Daten und entwickelt das integrierte Gebäudemodell, welches die Grundlage für verschiedene Softwarelösungen bildet, die den *Facility Manager* als Endbenutzer unterstützen sollen. Zur Verknüpfung der BIM- und BEMS-Daten wurde ein Hybridansatz (vgl. [TSE+19]) gewählt. Die BIM-Daten wurden automatisiert in passende Ontologien exportiert, wohingegen die BEMS-Daten in einer *Time Series* Datenbank gespeichert wurden. Als Ontologien wurden u.a. *ifcOWL*, *Brick* und *BOT* verwendet, welche untereinander durch entsprechende Mappings verknüpft wurden. *ifcOWL* ist die Repräsentation des IFC-Datenmodells in Form einer Ontologie. Die Ontologie wird dabei automatisiert aus dem IFC-Schema exportiert und ist somit immer am selben Stand wie das EXPRESS Schema [Ifcb]. *ifcOWL* ermöglicht die einfache Verknüpfung von BIM-Daten, basierend auf dem IFC-Datenmodell, mit anderen Ontologien, ohne ein neues Schema definieren zu müssen. Die *Building Topology Ontology (BOT)* ermöglicht die Darstellung der Gebäudestruktur und enthält Klassen zur Repräsentation von Standorten, Gebäuden, Stockwerken, Räumen und Bauelementen. Diese Objekte können durch verschiedene Beziehungen miteinander verknüpft werden, um die Gebäudestruktur zu bilden [RPL+17]. Die *Brick* Ontologie ermöglicht die Abbildung der Gebäudetechnik auf semantischer Ebene, indem verschiedene Arten von Geräten definiert und miteinander verknüpft werden können. Es ist mithilfe von Brick z.B. möglich, den Wasserfluss vom Speicher bis hin zur Armatur zu modellieren [BBF+16]. Durch die Verknüpfung dieser Ontologien werden die statischen BIM-Daten mit den dynamischen BMS-/BEMS-Daten zusammengeführt, um übergreifende Abfragen zu ermöglichen. Diese Abfragen können z.B. effizient über *SPARQL* durchgeführt werden. Die Sensordaten werden aus Effizienzgründen dennoch in einer eigens dafür optimierten Datenbank gespeichert und über eine gemeinsame ID miteinander verknüpft. Beim Exportieren der BIM-Daten wurde festgestellt, dass die Nachbarschaftsbeziehungen zwischen Räumen und andere räumliche Abhängigkeiten nicht ins IFC-Modell übernommen wurden. Diese semantischen Informationen mussten durch eine eigene Software auf Basis der Geometrie rekonstruiert werden, sodass sie in die Ontologie übernommen werden konnten.

Wie bereits erwähnt, ist es von Vorteil, wenn die Daten des GA-System im Kontext des Gebäudes, in Form einer 2D- oder 3D-Darstellung, visualisiert werden. Bisher wurde

diese Art der Darstellung, in den meisten Fällen, per Hand erstellt, wobei das CAD- bzw. BIM-Modell als Vorlage dient. Diese Vorgehensweise verursacht jedoch höhere Kosten, einerseits durch den Zusatzaufwand, der durch die mehrfache Abbildung des Gebäudes entsteht und andererseits durch Fehler, welche durch die Neu-Modellierung auftreten [RB13]. Wünschenswert wäre, wenn die Geometrien direkt aus der IFC-Datei des BIM-Modells exportiert werden könnten. Dieser Ansatz würde eine iterative Herangehensweise beim Modellieren des Gebäudes unterstützen, da die Daten jederzeit exportiert und in verschiedenen Werkzeugen analysiert und weiterverarbeitet werden könnten. Es wäre z.B. möglich, die Auswirkungen auf den Energieverbrauch des Gebäudes in regelmäßigen Abständen zu überprüfen, indem das BIM-Modell in ein entsprechendes Format exportiert und in einem Energieanalyse-Programm analysiert wird. Für den Einsatz in GA-Systemen sind die Geometrien im IFC-Modell häufig zu komplex, da sie in erster Linie aus Sicht der Architektur erstellt werden und dadurch einen hohen Detailgrad besitzen. Vor allem Fenster und Türen besitzen oft komplexe Geometrien. Bei Fenstern werden häufig sämtliche Fenstersprossen modelliert, wodurch in der Geometrie viele Flächen entstehen [RB13]. Aus diesem Grund ist es notwendig, die Geometrien im Rahmen des Exports zu vereinfachen, damit die Darstellung für den Endbenutzer einfach und verständlich bleibt.

In verschiedenen Arbeiten [RB13, LBB+16, LBK+16] wurden Ansätze vorgestellt, wie ausgehend von einem CAD- bzw. BIM-Modell, vereinfachte Geometrien für die Verwendung in Energieanalyse-Werkzeugen erstellt werden können. Bei diesen Verfahren werden die s.g. *Space Boundaries* [Baz10] in Form von Polygonen exportiert. Für die Verwendung in GA-Systemen ist es jedoch praktikabler, wenn die Geometrie in Form von *Brep Solid Geometry* vorliegt, um boolesche Operationen wie Vereinigung, Durchschnitt und Differenz zu ermöglichen. Diese sind notwendig, um z.B. Zonen aus mehreren Räumen zu bilden, welche ähnliche thermische Eigenschaften aufweisen und vom GA-System gemeinsam gesteuert werden. Im nächsten Kapitel wird ein adaptierter Ansatz vorgestellt, um die Geometrien aus dem IFC-Modell in vereinfachter Form als *Brep Solid Geometry* zu exportieren. Dieser Ansatz baut auf die Arbeit von [RB13] und [LBB+16, LBK+16] auf.

KAPITEL 3

Vereinfachung der Geometrie

In diesem Kapitel wird ein Ansatz vorgestellt, um die Geometrien eines Gebäudes so zu exportieren, dass sie in einem GA-System verwendet werden können. Dieser Ansatz baut auf dem GINGER-Algorithmus aus den Arbeiten von [LBB+16, LBK+16] auf und enthält einige Adaptierungen, die für die Verwendung in GA-Systemen notwendig sind. Der wesentlichste Unterschied zum GINGER-Algorithmus ist, dass die Geometrie als *Brep Solid Geometry*, anstatt als *Polygon-Mesh* exportiert wird.

3.1 Ziel

Die Geometrie eines Gebäudes ist sehr komplex und enthält viele Details. Vor allem Fenster und Türen besitzen eine komplexe Struktur und werden deshalb oft sehr detailliert dargestellt, sodass bei größeren Gebäuden erhebliche Datenmengen entstehen können. In modernen GA-Systemen werden oft sehr viele Sensordaten im Kontext der Gebäudegeometrie visualisiert. Aus diesem Grund ist es notwendig, dass die Geometrie aller Bauelemente möglichst vereinfacht dargestellt wird, um den Benutzer nicht zu überfordern bzw. den Blick aufs Wesentliche zu lenken. Bei Fenstern und Türen ist die Darstellung als einfaches Rechteck im Raum völlig ausreichend. Auch bei Wänden, Böden und Decken ist es nicht notwendig, die exakte Dicke der einzelnen Bauelemente darzustellen. Die Darstellung als 2-dimensionale Fläche im Raum ist vollkommen ausreichend.

Die Geometrien zur Weiterverwendung in Energieanalyse-Werkzeugen werden meist in Form von Polygon-Meshes exportiert, wo das Volumen der einzelnen Elemente nur implizit modelliert ist. Im Gegensatz dazu ist bei Verwendung in GA-Systemen die Darstellung in Form von *Brep Solid Geometry* besser geeignet, um Operationen wie z.B. die Vereinigung zweier Objekte zu unterstützen. Das ist notwendig, um z.B. zwei benachbarte Räume über die Vereinigungs-Operation zu einer Zone zusammenfassen zu können. Eine grundlegende Voraussetzung für den Algorithmus ist, dass kein Volumen existiert, welches nicht von

einem Bauelement oder einem Raum eingenommen wird. Es dürfen daher keine Lücken im Modell existieren, die nicht zumindest einem Raum zugeordnet sind.

In Abbildung 3.1 ist die Geometrie eines einfachen Gebäudes dargestellt. Auf der linken Seite sieht man die unveränderte, komplexe Geometrie. Die einzelnen Bauelemente werden in ihren vollen Details dargestellt. Die Decke und einzelne Wände werden in dieser Abbildung transparent dargestellt, damit die Räume und die darin befindlichen Elemente besser sichtbar sind. Auf der rechten Seite ist die vereinfachte Geometrie dargestellt, welche durch den Algorithmus exportiert wurde, der im Rahmen der vorliegenden Arbeit entwickelt wurde. In der vereinfachten Geometrie werden alle Elemente als 2-dimensionale Flächen im Raum dargestellt. Fenster und Türen sind in der rechten Darstellung nicht vorhanden, da diese getrennt exportiert werden, damit diese bei Verwendung in einem GA-System bei Bedarf ein- oder ausgeblendet werden können.

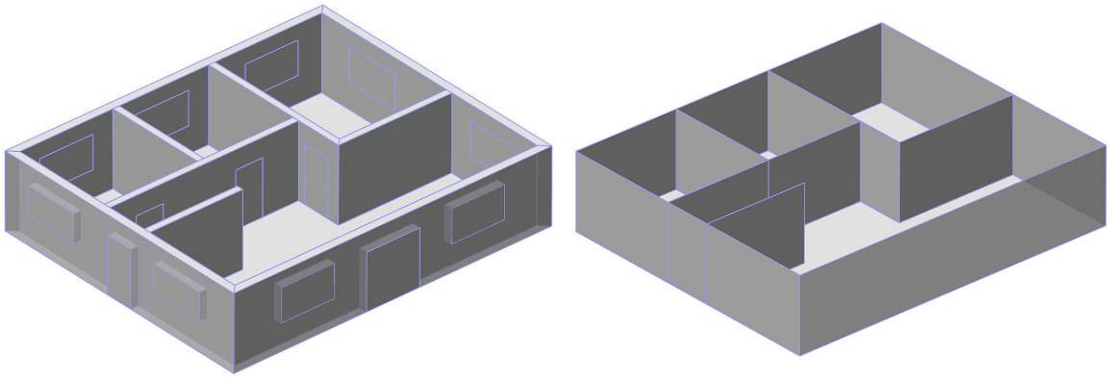

Abbildung 3.1: Komplexe und vereinfachte Geometrie eines Gebäudes

3.2 Voraussetzungen

Es existieren verschiedene Ansätze, um die Flächen und das Volumen der Räume zu berechnen. Revit unterstützt die folgenden 4 Arten:

At wall finish Bei dieser Variante ist die Raumgeometrie durch die innen liegende Seite der Wände begrenzt und die Volumina unterschiedlicher Räume berühren sich nicht.

At wall center Die Raumgrenze befindet sich bei dieser Variante in der Mitte der Wände und die Volumina unterschiedlicher Räume berühren sich, sodass dazwischen keine Lücken bestehen.

At wall core layer Eine Wand kann aus unterschiedlichen Materialschichten bestehen, jedoch ist immer eine dieser Schichten als Kernschicht (z.B. eine tragende Betonwand) definiert. Die Raumgrenze befindet sich bei dieser Variante an der Innen- oder Außenseite der Kernschicht.

At wall core center Bei dieser Variante befindet sich die Raumgrenze in der Mitte der Kernschicht und die Volumina unterschiedlicher Räume berühren sich.

Abbildung 3.2: Auf der linken Seite ist das Raumvolumen mit der Berechnungsmethode *At wall finish* und auf der rechten Seite mit *At wall center* dargestellt.

Der Algorithmus geht davon aus, dass die Volumina der Räume mit dem Ansatz *At wall finish* berechnet wurden. Ein Beispiel ist in Abbildung 3.2 auf der linken Seite dargestellt. Bei diesem Ansatz berühren sich die Volumina unterschiedlicher Räume nicht, deshalb können die Raumgeometrien nicht direkt als vereinfachte Geometrie verwendet werden. Die Geometrie muss dementsprechend noch vereinfacht werden, damit keine Lücken zwischen den Räumen bestehen. Wurden die Volumina der Räume mit dem Ansatz *At wall center* oder *At wall core center* berechnet, dann berühren sich die Volumina unterschiedlicher Räume. Die Methode *At wall center* ist in Abbildung 3.2 auf der rechten Seite schematisch dargestellt. Bei diesen Varianten kann die Raumgeometrie direkt als vereinfachte Darstellung der Gebäudegeometrie verwendet werden und dadurch entfallen einige Schritte des Algorithmus. In diesem Fall müssen nur die Schritte 3.5.1, 3.5.2, 3.5.9, 3.5.10 und 3.5.11 ausgeführt werden und die übrigen fallen weg. Beim Ausrichten der Fenster- und Tür-Geometrien an den Raumgrenzen ist zu beachten, dass in diesem Fall die Öffnungen der Wände nicht herangezogen werden können. Stattdessen müssen die Operationen rein geometrisch durchgeführt werden, indem anhand der Geometrie ermittelt wird, welche Fenster- und Tür-Geometrien an welchen Raumgeometrien ausgerichtet werden müssen. Die semantischen Informationen der Öffnungen sind in diesem Fall nicht hilfreich, da sie sich nur auf die Wände beziehen.

Als Eingabe benötigt der Algorithmus eine IFC-Datei, welche die vollständige Geometrie von Wänden, Fenstern, Türen und Räumen enthält. Jedes Volumen im Modell muss entweder von einem Bauelement oder einem Raum eingenommen werden und es dürfen keine Lücken zwischen den Räumen existieren. Bauelemente und Räume dürfen sich nicht überlappen, da ansonsten die Erkennung der *Space Boundaries* nicht korrekt funktioniert. Geschosse werden getrennt exportiert, d.h., die Bauelemente müssen entsprechend ihrer Geschoss-Zugehörigkeit gefiltert werden. Sind diese Informationen in der IFC-Datei nicht oder nur zum Teil enthalten, dann können die Geometrien über entsprechende z-Koordinaten Intervalle geometrisch herausgefiltert werden. Die Space Boundaries werden

ausgehend von der Raumgeometrie berechnet. Der Algorithmus betrachtet jeden Raum für sich, als gäbe es nur diesen Raum. Es ist deshalb bei der Betrachtung eines Raums irrelevant, wie das Gebäude außerhalb dieses Raums strukturiert ist. Das entspricht in Bezug auf die Raumgeometrie dem Space Boundary Level 1 [Spaa].

Wie bereits in Abschnitt 2.3 erwähnt, unterstützt IFC verschiedene geometrische Darstellungen. Die Geometrie muss so vorliegen, dass sie mittels Triangulation zu einem Dreiecksnetz konvertiert werden kann. Dadurch wird der Algorithmus erheblich vereinfacht, da er nur eine geometrische Darstellungsform unterstützen muss. Durch die Triangulation werden grundsätzlich auch gebogene Bauelemente berücksichtigt, da diese durch planare Flächen angenähert werden. Die Geometrie eines IFC-Elements bezieht sich immer auf ein lokales Objekt-Koordinatensystem, welches relativ zu einem übergeordneten Objekt-Koordinatensystem oder dem globalen Projekt-Koordinatensystem positioniert ist. Die relativen Koordinaten müssen für den Algorithmus in globale Koordinaten umgerechnet werden.

3.3 Allgemeines

Wie bereits erwähnt, baut der im Folgenden beschriebene Algorithmus auf dem GINGER-Algorithmus aus den Arbeiten [Lad14, LBB+16, LBK+16] auf. Der Algorithmus unterscheidet im Wesentlichen zwischen Bauelementen und Räumen. Bauelemente setzen sich aus Wänden, Böden, Decken, Türen, Fenstern, u.v.m. zusammen. In unserer Arbeit werden ausschließlich die Bauelemente Wände (*IfcWall*), Türen (*IfcDoor*) und Fenster (*IfcWindow*) aus der IFC-Datei verwendet. Im Gegensatz zum GINGER-Algorithmus werden Böden und Decken (*IfcSlab*) in unserem Algorithmus nicht verwendet, da die horizontalen Raumgrenzen auf Basis der Raumgeometrie berechnet werden. Bei Dachgeschossen kann es vorkommen, dass die gesamte äußere Abgrenzung des Stockwerks nahezu ausschließlich mittels *IfcSlab*-Elementen modelliert ist. In solchen Fällen müssten diese Elemente ebenfalls als Bauelemente berücksichtigt werden, da ansonsten keine Außenwände zur Verfügung stehen. GA-Objekte zählen im Algorithmus nicht zu den Bauelementen. Diese werden in einem eigenen Schritt im Algorithmus behandelt. Für die Vereinfachung dieser Objekte wird die Bounding Box Darstellung aus der IFC Datei herangezogen.

Die interne geometrische Darstellung aller Elemente erfolgt durch s.g. *Polygon-Soups*, d.h., Elemente werden durch ein oder mehrere Polygone dargestellt, deren Orientierung als Normalvektor gespeichert wird. Erst am Ende des Algorithmus werden die Polygone eines Raums in Solids konvertiert. Durch die Triangulation am Beginn entstehen ausschließlich planare Flächen, die sich jeweils in einer Ebene im Raum befinden. Die Entfernung dieser Ebene zum Ursprung wird abgespeichert, da die Information immer wieder benötigt wird. Zusätzlich zu den geometrischen Informationen wird auch eine Referenz zum ursprünglichen IFC-Element aufbewahrt, damit die Information nicht verloren geht, welche Geometrie zu welchem Bauelement im Gebäude gehört.

Ein zentraler Bestandteil des GINGER-Algorithmus ist das Konzept des *Boundary Parts*.

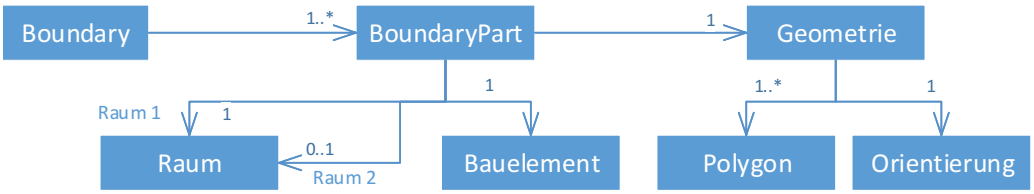

Abbildung 3.3: Boundary Part als zentrales Konzept des GINGER-Algorithmus [Lad14]

Der Algorithmus berechnet auf Basis der Geometrien vereinfachte Begrenzungen (engl. Boundaries), welche die Räume voneinander trennen. Eine Boundary besteht aus einer oder mehreren *Boundary Parts*. Im einfachsten Fall besteht eine Boundary aus genau einem *Boundary Part*. Bei einer geraden, durchgängigen Wand wird die Boundary durch ein Polygon mit einem Normalvektor dargestellt. Bei Elementen, welche unterschiedliche Orientierungen in der Geometrie aufweisen, wird die Boundary auf mehrere *Boundary Parts*, gruppiert nach Orientierung, aufgesplittet. Der Zusammenhang zwischen Boundary und *Boundary Part* wird in Abbildung 3.3 anschaulich dargestellt. Jeder *Boundary Part* referenziert genau ein Bauelement, dessen Geometrie er abbildet. Wie bereits in Abschnitt 3.2 erwähnt, muss sich jedes Bauelement in zumindest einem Raum befinden. Wände können einen Raum nicht nur abgrenzen, sondern auch zwei Räume voneinander trennen. Aus diesem Grund referenziert ein *Boundary Part* entweder einen oder zwei Räume. Wie bereits erwähnt, wird die Geometrie eines *Boundary Parts* durch ein Polygon und einen Normalvektor für die Orientierung abgebildet.

Analog zum GINGER-Algorithms [Lad14] wird ein Großteil der geometrischen Operationen in 2D durchgeführt, da diese in der Ebene effizienter ausgeführt werden können. Dabei werden die Koordinaten in die Ebene projiziert, die jeweilige geometrische Operation ausgeführt und anschließend das Resultat wieder in den Raum zurück projiziert. Wie bereits erwähnt, wird die Geometrie aller Elemente am Beginn durch Triangulation vereinheitlicht. Das hat zur Folge, dass die gesamte Geometrie aus planaren Flächen besteht, deren Koordinaten jeweils in einer Ebene angeordnet sind. Bei der Projektion wird diese Ebene, als xy-Ebene herangezogen, wie auch in Abbildung 3.4 dargestellt ist. Die x-Achse bildet einen Orthonormalvektor zum Normalvektor und die y-Achse ist das Kreuzprodukt aus diesem Orthonormalvektor und dem Normalvektor.

3.4 Überblick

Als Eingabe benötigt der Algorithmus eine IFC-Datei, welche die vollständige Geometrie aller benötigten Bauelemente und Räume enthält. Das Ergebnis ist die vereinfachte Geometrie der Raumgrenzen als *Brep Solid Geometry*. Der Algorithmus setzt sich aus folgenden Schritten zusammen:

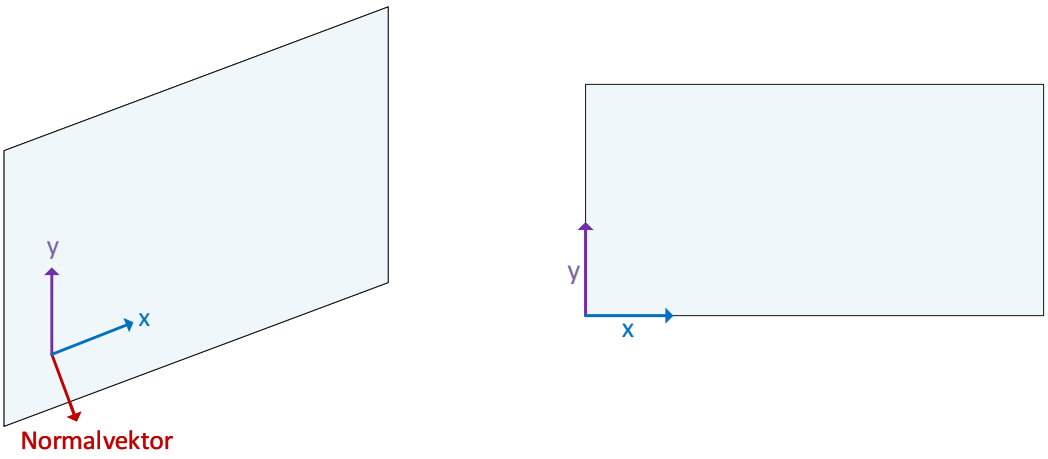

Abbildung 3.4: 3D Polygone werden für geometrische Operationen auf die Ebene projiziert. Angelehnt an die Arbeit von [Lad14].

Triangulation Am Beginn des Algorithmus wird die Geometrie aus der IFC-Datei extrahiert und trianguliert, sodass nur mehr planare Flächen vorhanden sind. Nicht planare Flächen werden durch die Triangulation approximiert. Der Algorithmus muss somit nur mehr mit einer Darstellungsform für die Geometrie umgehen können. Im Zuge der Triangulation werden lokale Koordinaten in globale Koordinaten umgerechnet, damit alle Objekte für die nachfolgenden Schritte korrekt positioniert sind. Am Ende des ersten Schrittes wird die triangulierte Geometrie vereinfacht, sodass möglichst wenige Flächen entstehen.

Vereinfachung von Fenstern und Türen Die vielen geometrischen Details von Fenstern und Türen sind für die Verwendung in GA-Systemen nicht notwendig und würden die nachfolgenden Schritte des Algorithmus eher behindern als unterstützen. In diesem Schritt werden die komplexen Fenster- und Tür-Geometrien ersetzt und stattdessen in Form von orientierten Bounding Boxes dargestellt.

Entfernen der Öffnungen aus den Wänden Die Öffnungen von Fenstern und Türen werden aus den Wänden entfernt, damit diese getrennt exportiert werden können.

Erkennung der Boundaries Dieser Schritt ist auch Teil des GINGER-Algorithmus [LBB+16] und dient der Erkennung der Raumgrenzen (Space Boundaries), welche für die nachfolgenden Schritte wesentlich sind. Die Raumgrenzen werden im Normalfall aus den Wandelementen gebildet, welche den Raum von den anderen Räumen abgrenzen. Sie werden entsprechend ihrer Orientierung, wie in Abbildung 3.3 ersichtlich, in mehrere *Boundary Parts* aufgeteilt.

Berechnung der horizontalen Boundaries Da die vereinfachte Geometrie in Form von Solids exportiert wird, dürfen die einzelnen Raumgeometrien keine Öffnungen und Lücken aufweisen. Die horizontalen Boundaries werden auf Basis der in den vorigen Schritten erkannten Boundaries und der Raumgeometrien berechnet.

Zusammenfassen der inneren Boundaries Bei Innenwänden zwischen zwei Räumen entstehen durch die Art und Weise, wie die Boundary Erkennung durchgeführt wird, je zwei *Boundary Parts*. Diese zwei Boundary Parts können zu einem Boundary Part zusammengefasst werden, um die Geometrie entsprechend zu vereinfachen. Diese Vorgehensweise folgt analog dem GINGER-Algorithmus [LBB+16], wo dieser Schritt ebenfalls notwendig ist.

Füllen der entstandenen Lücken Durch die Art und Weise, wie der GINGER-Algorithmus die Boundary-Erkennung durchführt, entstehen zwischen den verschiedenen Boundaries Lücken, die gefüllt werden müssen, um am Ende abgeschlossene Solids zu erhalten.

Geometrische Nachbearbeitungen Um die Geometrie weiter zu vereinfachen und die Robustheit des Algorithmus zu erhöhen, sind einige Nachbearbeitungen notwendig. Horizontale Boundaries werden mit den vertikalen Boundaries verbunden, falls noch Lücken bestehen, welche im Zuge der vorherigen Schritte nicht aufgelöst werden konnten. Die zuvor zusammengefassten Boundary Parts müssen wieder aufgesplittet werden, damit jeder Raum eine vollständige Boundary besitzt. Geometrien, die sich überlappen, werden vereinfacht, damit keine Kollisionen auftreten. Am Ende der Nachbearbeitungen werden noch vorhandene Lücken für jeden Raum gefüllt und Kanten vereinfacht.

Export von Fenstern und Türen In den vorherigen Schritten wurden die Fenster- und Tür-Geometrien bereits durch orientierte *Bounding Boxes* ersetzt und die verbleibenden Lücken in den Wänden gefüllt. Diese Geometrien werden nun weiter vereinfacht, sodass Fenster und Türen nur mehr in Form von Rechtecken dargestellt werden, welche an den entsprechenden *Boundary Parts* der Wände ausgerichtet sind.

Export der GA-Objekte Ähnlich wie Fenster und Türen sollen die GA-Objekte vereinfacht dargestellt werden. In diesem Schritt wird die Geometrie von GA-Objekten, ausgehend von der Bounding Box Darstellung im IFC-Modell, zu Rechtecken im Raum vereinfacht.

Berechnung der Solids Im letzten Schritt werden die *Boundary Parts* der Räume miteinander verbunden. Dabei werden Flächen, die geometrisch zusammenfallen, mittels s.g. *Sewing* topologisch zusammengefasst. Anschließend wird daraus die vereinfachte Geometrie in Form von Solids exportiert, wie auf der rechten Seite in Abbildung 3.1 dargestellt ist.

3.5 Algorithmus

3.5.1 Triangulation

Abbildung 3.5: Triangulierte und vereinfachte Geometrie

Die Geometrie aller Gebäudeelemente kann in einem IFC-Modell auf viele verschiedene Arten gespeichert sein. Unter anderem erlaubt IFC die Darstellungsformen *Triangulated Surface*, *Bounding Box*, *Boundary Representation* und *Constructive Solid Geometry* [And18]. Damit der nachfolgend beschriebene Algorithmus nicht mit allen von IFC unterstützten Darstellungsformen umgehen können muss, ist es sehr wichtig, dass die Geometrie aller Elemente vereinheitlicht wird. Die Vereinheitlichung erfolgt mittels Triangulation, sodass ein komplexes Dreiecks-Netz aller Bauelemente und Räume entsteht.

Abgesehen von der Triangulation ist es wichtig, dass die Koordinaten aus dem IFC-Modell in globale Koordinaten umgerechnet werden. In IFC wird die Geometrie eines Elements immer in Bezug auf ein lokales Koordinatensystem dargestellt, welches in Beziehung zu einem weiteren lokalen Koordinatensystem steht. Die Geometrie einer Wand wird z.B. nicht aus globalen Koordinaten gebildet, sondern die Koordinaten beziehen sich auf das lokale Koordinatensystem der Wand. Das Koordinatensystem der Wand bezieht sich wiederum auf das Koordinatensystem des Geschosses und dieses bezieht sich wiederum auf das Koordinatensystem des Gebäudes. Das letzte Koordinatensystem in dieser Kette ist das Projekt-Koordinatensystem [And18]. Damit die Geometrie vom Algorithmus verwendet werden kann, müssen die lokalen Koordinaten in globale Koordinaten umgerechnet werden, damit die Elemente in der resultierenden Geometrie richtig positioniert werden.

Wie bereits erwähnt, ist die Triangulation sehr wichtig zur Vereinheitlichung der Geometrie, damit der Algorithmus nur eine Darstellungsform unterstützen muss. Abgesehen davon ist die triangulierte Geometrie notwendig, um gebogene Elemente durch planare Flächen anzunähern, da der Algorithmus nur mit planaren Flächen umgehen kann. Durch die Triangulation entstehen üblicherweise sehr viele Dreiecke, wodurch die Anzahl der Flächen enorm steigt. Es werden jedoch auch ohnehin planare Flächen in mehrere Dreiecke aufgesplittet. Aus diesem Grund wird die Geometrie nach der Triangulation vereinfacht, indem alle Flächen eines Elements, die in derselben Ebene liegen und damit dieselbe Orientierung aufweisen, zusammengefasst werden. Die Vereinfachung ist in Abbildung 3.5 anschaulich dargestellt. In der linken Grafik ist eine Wand in ihrer triangulierten

Form abgebildet. Man kann sehr gut erkennen, dass die planare Fläche der Wand in 6 Dreiecke aufgeteilt wurde. Auf der rechten Seite der Abbildung ist die Geometrie nach der Vereinfachung dargestellt. Hier wurden die 6 Dreiecke durch eine rechteckige Fläche ersetzt.

Algorithmus 3.1: VereinfacheTriangulierteGeometrie

Eingabe: Triangulierte Geometrie als Liste von Dreiecken D mit einem Polygon p, einem Normalvektor n und der Distanz der Ebene zum Ursprung d

Ausgabe: Vereinfachte Geometrie als Liste E von Polygonen

```
1 E ← []                                          // Initialisierung der Ergebnismenge
2 für alle Dreiecke in derselben Ebene D_ebene ⊆ D tue
3 |   G ← ∅                                       // leere Geometrie
4 |   für jedes d ∈ D_ebene tue
5 |   |   G ← G ∪ d.p     // Vereinigung d. Ergebnismenge mit Polygon p von d
6 |   Ende
7 |   FügeHinzu(E, G)                  // Füge G zur Ergebnismenge E hinzu
8 Ende
9 return E
```

Der Algorithmus zur Vereinfachung der triangulierten Geometrie ist in Listing 3.1 dargestellt. Als Eingabe wird die Geometrie eines IFC-Elements in Form von Dreiecken benötigt. Die geometrischen Informationen zu einem Dreieck beinhalten das Polygon selbst, den Normalvektor und die Distanz vom Ursprung zur Ebene, in der sich das Dreieck befindet. Als Ergebnis liefert der Algorithmus eine Menge von Polygonen, welche die vereinfachte Geometrie des IFC-Elements darstellen. Die Dreiecke werden nach Normalvektor und Distanz der Ebene zum Ursprung gruppiert, um alle Dreiecke zusammenzufassen, welche sich in derselben Ebene befinden. Bei Verwendung von *Floating Point Numbers* in der konkreten Implementierung des Algorithmus können bei einzelnen Berechnungen Rundungsfehler entstehen, wodurch gewisse Vergleiche im Algorithmus mit einer gewissen Ungenauigkeit durchgeführt werden müssen. Dieses Thema wird in Abschnitt 5.3 genauer behandelt. Aus diesem Grund ist es in diesem Schritt u.U. notwendig die Gruppierung der Dreiecke mit einer gewissen Ungenauigkeit durchzuführen, um die Robustheit des Algorithmus zu erhöhen. In Zeile 2 werden alle Dreiecke derselben Ebene Schritt für Schritt iteriert. Ziel des Algorithmus ist es, alle Dreiecke einer Ebene durch Vereinigung zu einem Polygon zusammenzufassen. Das Polygon für die jeweilige Ebene wird in Zeile 3 mit einer leeren Geometrie initialisiert und Schritt für Schritt in Zeile 5 mittels Vereinigung erweitert. Die Vereinigung erfolgt dabei in 2D, indem beide Geometrien auf dieselbe Ebene in 2D projiziert werden. Anschließend wird das Ergebnis wieder in den Raum zurück projiziert. Die einzelnen Polygone werden in Zeile 7 zur Ergebnismenge

hinzugefügt.

3.5.2 Vereinfachung von Fenstern und Türen

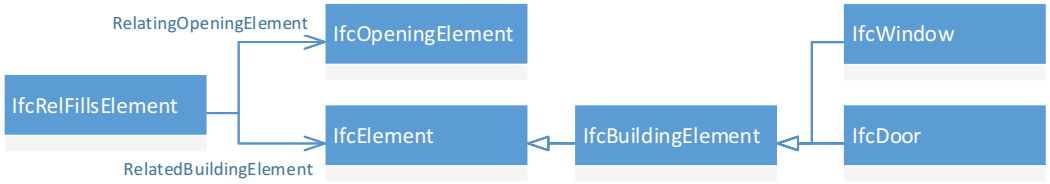

Abbildung 3.6: Beziehung *IfcRelFillsElement* als Verknüpfung zwischen Fenster/Türen und Öffnungen

Nachdem die Geometrie der IFC-Elemente mittels Triangulation vereinheitlicht und anschließend vereinfacht wurde, wird im 2. Schritt die Geometrie aller Fenster und Türen des Gebäudes vereinfacht. Fenster und Türen sind häufig sehr komplex, da diese Bauelemente oft aus sehr vielen kleinen Details bestehen. Bei Fenstern wird z.B. oft jede Sprosse sehr detailliert abgebildet, wodurch in der Geometrie sehr viele einzelne Flächen entstehen. Dieser hohe Detailgrad ist für die Anwendung in GA-Systemen nicht notwendig, sodass eine Vereinfachung als sinnvoll erscheint. Ein Fenster oder eine Tür befindet sich im Normalfall immer in einer entsprechenden Öffnung einer Wand. Diese Öffnung wird in IFC durch die Entität *IfcOpeningElement* abgebildet und ist über die Beziehung *IfcRelFillsElement* mit dem entsprechenden Bauelement, also dem Fenster oder der Tür, verknüpft. Die Beziehung zwischen den Entitäten *IfcOpening* und *IfcBuildingElement* ist in Abbildung 3.6 dargestellt. Fenster und Türen werden in IFC durch die Entitäten *IfcWindow* und *IfcDoor* abgebildet und sind von *IfcBuildingElement* abgeleitet. *IfcBuildingElement* ist wiederum von *IfcElement* abgeleitet. Die Verknüpfung der Fenster und Türen mit der entsprechenden Öffnung in der Wand erfolgt über die Eigenschaften *RelatingOpeningElement* und *RelatedBuildingElement* der Beziehung *IfcRelFillsElement*. Aus Gründen der Übersichtlichkeit sind in der Abbildung nur ausgewählte Eigenschaften und Beziehungen dargestellt. Die Geometrie des *IfcOpeningElement* ist typischerweise viel einfacher als die Geometrie des Bauelements selbst und gleicht in vielen Fällen der Geometrie einer *orientierten Bounding Box*. [RB13] empfiehlt im Rahmen seiner Arbeit ebenfalls die Verwendung der vereinfachten Geometrie der Öffnungen, anstatt der

komplexeren Geometrie der Fenster und Türen, die diese Öffnungen füllen.

Algorithmus 3.2: VereinfacheFensterTueren

Eingabe: IFC-Element *ifcElement* des Fensters (IfcWindow) oder der Tür (IfcDoor), die vereinfacht werden soll, als Teil des IFC-Objektbaums

Ausgabe: Vereinfachte Geometrie als Liste *vereinfacht* von Polygonen

```
1 opening ← ErmittleIfcOpeningElement(ifcElement)
2 trianguliert ← TrianguliereGeometrie(opening)
3 vereinfacht ← VereinfacheTriangulierteGeometrie(trianguliert)
4 return vereinfacht
```

Der Algorithmus zur Vereinfachung der Geometrie von Fenster und Türen ist in Listing 3.2 dargestellt. Als Eingabe benötigt der Algorithmus das IFC-Element des Fensters oder der Tür, das vereinfacht werden soll. Das *IfcWindow*- oder *IfcDoor*-Element wird als Teil des IFC-Objektbaums benötigt, sodass eine Navigation zum jeweiligen *IfcOpeningElement* möglich ist. In Zeile 1 des Algorithmus wird das *IfcOpeningElement* über die Beziehung *IfcRelFillsElement* ermittelt, wie sie auch in Abbildung 3.6 dargestellt ist. Üblicherweise entspricht die Geometrie dieser Öffnung bereits der orientierten Bounding Box des Fensters und ist für unsere Zwecke direkt verwendbar. In Zeile 2 wird die Geometrie trianguliert und anschließend in Zeile 3 mit dem Algorithmus aus Abschnitt 3.5.1 vereinfacht.

Das Ergebnis des Algorithmus ist in Abbildung 3.7 der ursprünglichen Geometrie eines Fensters gegenübergestellt. Auf der linken Seite sieht man ein Fenster mit seiner Original-Geometrie, wo alle Fenstersprossen sehr detailliert dargestellt werden. Die vereinfachte Geometrie des Fensters wird auf der rechten Seite der Abbildung dargestellt.

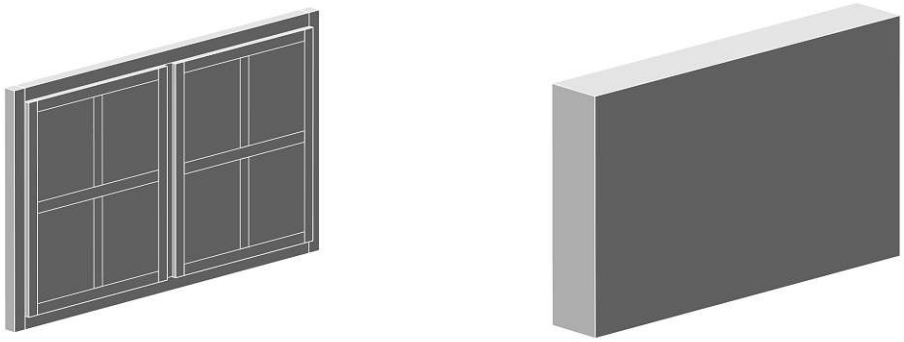

Abbildung 3.7: Komplexe und vereinfachte Geometrie eines Fensters

3.5.3 Entfernen der Öffnungen aus den Wänden

Nachdem im vorigen Schritt die Geometrien der Fenster und Türen vereinfacht wurden, werden in diesem Schritt die Öffnungen aus den Wänden entfernt, da diese für unseren Zweck nicht benötigt werden. Die Öffnungen einer Wand können in IFC über die Eigenschaft *HasOpenings* des Elements *IfcWall* ermittelt werden. Diese Eigenschaft referenziert die Beziehung *IfcRelVoidsElement*, welche wiederum über die Eigenschaft *RelatedOpeningElement* zur jeweiligen Öffnung verweist. Die Öffnung wird durch das Element *IfcFeatureElementSubtraction* dargestellt. Diese Beziehung ist in Abbildung 3.8 dargestellt, wobei aus Gründen der Einfachheit nur ausgewählte Eigenschaften und Beziehungen visualisiert werden.

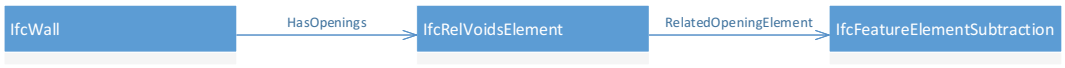

Abbildung 3.8: Beziehung *IfcRelVoidsElement* als Verknüpfung zwischen Wänden und Öffnungen

Der Algorithmus zum Entfernen der Öffnungen aus den Wänden ist in Listing 3.3 dargestellt. Als Eingabe benötigt der Algorithmus das *IfcWall*-Element der Wand und die Geometrie in Form einer Liste von Teilgeometrien, die jeweils das Polygon selbst, den Normalvektor und die Distanz der Ebene zum Ursprung enthält. In Zeile 1 wird das Ergebnis mit einer leeren Liste initialisiert, welche anschließend Schritt für Schritt erweitert wird. Die Funktion *ErmittleOpenings* in Zeile 2 liefert alle Öffnungen, die von der Wand über die Beziehung *IfcRelVoidsElement* referenziert werden. Die Geometrien der Öffnungen werden analog zu Abschnitt 3.5.1 zuerst trianguliert und anschließend vereinfacht, um die Anzahl der Flächen möglichst gering zu halten. In Zeile 3 werden die Teilgeometrien der Wand iteriert. Die Lücken der Teilgeometrie werden schrittweise gefüllt und das jeweilige Zwischenergebnis in $g_{ergebnis}$ gespeichert. In Zeile 4 wird die Variable $g_{ergebnis}$ mit der Geometrie g_i initialisiert. Für jede Teilgeometrie werden in Zeile 5 alle Öffnungen ermittelt, die sich in derselben Ebene befinden, wie die Teilgeometrie selbst. Eine Öffnung befindet sich in derselben Ebene, wenn der Normalvektor und die Distanz der Ebene zum Ursprung mit denen der Teilgeometrie übereinstimmen. Dabei ist zu beachten, dass die Genauigkeit der Berechnungen durch die Verwendung von Gleitkommazahlen beschränkt ist. Dadurch ist es notwendig, dass Vergleiche nicht exakt durchgeführt werden, sondern mit entsprechenden Rundungsfehlern zu rechnen ist. Die Ermittlung aller Öffnungen, die sich in derselben Ebene befinden, muss dementsprechend mit einer gewissen Ungenauigkeit umgehen können. In Zeile 7 wird die Teilgeometrie aus $g_{ergebnis}$ mit der aktuellen Öffnung, die sich in derselben Ebene wie die Teilgeometrie befindet, gebildet. Durch Ungenauigkeiten bei diversen Berechnungen, vor allem bei Projektionen und Rück-Projektionen zwischen Ebene und Raum, entstehen Punkte, die in der ursprünglichen Geometrie zusammenfallen, jedoch aufgrund von Rundungsfehlern etwas auseinanderliegen. Im hier beschriebenen Anwendungsfall lässt sich das Problem

lösen, indem beim Bilden der Vereinigungsmenge jede Koordinate der Öffnung mit jener Koordinate des Wandelements ersetzt wird, die ihr am nächsten liegt. Zusätzlich sollte eine obere Schranke für die Entfernung der Koordinaten definiert werden, um auszuschließen, dass sich das Wandelement und die Öffnung zu weit voneinander entfernt befinden. Durch diese Methode ist garantiert, dass nach der Vereinigung der beiden Geometrien in Zeile 7 keine Lücken in der Wandgeometrie entstehen, die in weiterer Folge Probleme im Algorithmus auslösen können. Besteht die Vereinigung der beiden Geometrien aus genau einem Polygon, dann bedeutet das, dass das Füllen der Lücke in der Teilgeometrie der Wand erfolgreich war. Das Ergebnis der Vereinigung wird in diesem Fall in Zeile 9 als Zwischenergebnis in $g_{ergebnis}$ gespeichert. Besteht die Vereinigung der beiden Geometrien jedoch aus mehr als einem Polygon, dann war das Füllen der Lücke nicht erfolgreich, da sich die Geometrien in diesem Fall nicht überschneiden.

Algorithmus 3.3: EntferneÖffnungenAusWänden

Eingabe: IfcWall-Element ifcWall der Wand, als Teil des Objektbaums, sowie die Geometrie der Wand als Liste G mit je Polygon p, Normalvektor n und der Distanz der Ebene zum Ursprung d als Daten

Ausgabe: Geometrie der Wand ohne Öffnungen als Liste E

```
1  E ← []
2  O ← ErmittleOpenings(ifcWall)
3  für jedes g_i ∈ G tue
4      g_ergebnis ← g_i                          // Initialisiere Ergebnisgeometrie für g_i
5      O_ebene ← {o ∈ O | n = n_i ∧ d = d_i}     // Öffnungen derselben Ebene
6      für jedes o_j ∈ O_ebene tue
7          g_{o_j g_i} ← o_j ∪ g_ergebnis
8          wenn |g_{o_j g_i}| = 1 dann
9              g_ergebnis ← g_{o_j g_i}          // Füllen der Öffnung
10         Ende
11     Ende
12     FügeHinzu(E, g_ergebnis)
13 Ende
14 return E
```

Das Ergebnis des Algorithmus ist in Abbildung 3.9 ersichtlich. Auf der linken Seite der Grafik ist eine Wand, mit einem Fenster und einer Tür als Öffnung, vor Anwendung des Algorithmus dargestellt. Nach Anwendung des Algorithmus sind die Lücken, die durch Fenster und Tür entstanden sind, gefüllt. Das Ergebnis ist auf der rechten Seite der

Abbildung zu sehen.

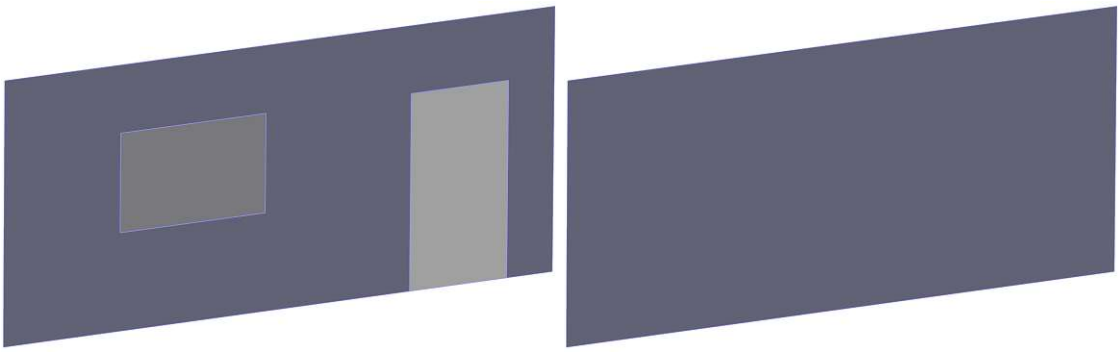

Abbildung 3.9: Geometrie einer Wand mit und ohne Öffnungen

3.5.4 Erkennung der Space Boundaries

Algorithmus 3.4: BoundaryErkennung

Eingabe: Räume und Bauelemente als Liste von Teilgeometrien G_{raum} bzw. G_{bau} mit einer Referenz zum jeweiligen IFC-Element i, dem Polygon p, Normalvektor n und der Distanz der Ebene zum Ursprung d als Daten Maximale Entfernung zwischen Raumgeometrie und Geometrie des Bauelements als Schwellwert $E_{max_{boundary}}$

Ausgabe: Boundary Parts als Liste von Teilgeometrien sowie Referenzen zum IFC-Element des Raums und des Bauelements

```
1  E ← []                                   // Ergebnisliste - Erkannte Boundary Parts
2  für jedes g_raum ∈ G_raum tue            // Räume
       // Elemente von G_adj nach Entfernung der Geometrien g_bau und g_raum
       aufsteigend sortiert
3      G_adj ← {g_bau ∈ G_bau | n = n_raum ∧ d − d_raum > 0 ∧ entf(p,
       p_raum) < E_max_boundary}
4      g_check ← ∅                          // leere Geometrie
5      für jedes g_adj ∈ G_adj tue          // Benachbarte Bauelemente
6          wenn g_adj ∩ g_raum ≠ ∅ dann
7              FügeHinzu(E, (g_adj ∩ g_raum) \ g_check)   // Boundary Part
8              g_check ←g_check ∪ (g_adj ∩ g_raum)
9          Ende
10     Ende
11 Ende
12 return E
```

Die bisherigen Schritte des Algorithmus können als Vorbereitung für die Anwendung des GINGER-Algorithmus betrachtet werden. In unserem Fall werden die Geometrien der Fenster und Türen nicht als Ausschnitte in den Wänden benötigt und wurden deshalb im vorigen Schritt entfernt. Im nächsten Schritt wird der erste Teil des GINGER-Algorithmus auf die vorbereiteten Geometrien angewandt. Ausgehend von den Geometrien der Räume und Bauelemente werden die *Boundary Parts* der Räume berechnet. Die *Boundary Parts* werden dabei von den äußeren Teilgeometrien der Bauelemente gebildet, die den Raum abgrenzen. Jeder Raum wird dabei für sich betrachtet, als gäbe es keine anderen Räume im Gebäude. Dementsprechend werden alle Wände des Raumes so behandelt, als wären sie Außenwände des Gebäudes [LBB+16].

Der erste Teil des GINGER-Algorithmus ist in Listing 3.4 dargestellt. Beim GINGER-Algorithmus werden Wände, Böden, Decken, Fenster und Türen zu den Bauelementen gezählt. In der vorliegenden Arbeit werden nur Wände als solche verwendet, jedoch der Begriff Bauelement beibehalten, um unsere Adaptierung des Algorithmus besser mit dem ursprünglichen GINGER-Algorithmus vergleichen zu können. Wie bereits erwähnt, werden die Fenster und Türen für die Verwendung in GA-Systemen nicht benötigt und deshalb als Bauelemente weggelassen. Böden und Decken sind in dieser Arbeit nicht von Bedeutung, da die horizontalen Raumgeometrien anders berechnet werden. Der Algorithmus benötigt als Eingabe die Geometrien der Räume und Bauelemente als Polygon mit der Orientierung als Normalvektor und der Distanz der Ebene zum Ursprung. Zusätzlich wird die Referenz zum IFC-Element benötigt, um den Ursprung der Geometrie über die verschiedenen Schritte des Algorithmus hinweg feststellen zu können.

Wie bereits erwähnt, werden die *Boundary Parts* pro Raum getrennt berechnet. Aus diesem Grund werden die Räume in Zeile 2 iteriert. In Zeile 3 werden alle Bauelemente ermittelt, welche die selbe Orientierung besitzen und sich in Richtung des Normalvektors vom Bauelement entfernt befinden. Bei Geometrien, die sich nicht in einer Ebene befinden, welche parallel zu einer der drei Koordinatenebenen ist, können die Normalvektoren geringfügig voneinander abweichen, auch wenn die Orientierung dieselbe ist. Daher sollten die Normalvektoren mit einer gewissen Toleranz verglichen werden, um die Robustheit des Algorithmus zu erhöhen. Für die Ermittlung benachbarter Bauelemente benötigt der Algorithmus eine Entfernung $E_{max_{boundary}}$ als Parameter, um die maximale Entfernung zwischen den zwei Geometrien zu beschränken. Das soll verhindern, dass fälschlicherweise Bauelemente erkannt werden, die nicht zum aktuellen Raum gehören. Diese Vorgehensweise ermöglicht die Berechnung der *Boundary Parts* ohne auf die semantische Information in IFC zurückgreifen zu müssen, wodurch die Anforderungen an die semantische Korrektheit des IFC-Modells reduziert werden. Eine Verwendung der semantischen Informationen würde den Algorithmus natürlich entsprechend beschleunigen, vor allem, wenn das zu verarbeitende Gebäude komplex ist und somit viele Geometrien enthält. Die ermittelten Bauelemente werden aufsteigend nach der Entfernung zur Raumgeometrie nacheinander in Zeile 5 abgearbeitet. Im nächsten Schritt wird in Zeile 6 überprüft, ob sich die Raumgeometrie und die Geometrie des Bauelements schneiden und die Schnittmenge im Erfolgsfall in Zeile 7 der Ergebnismenge hinzugefügt. In der Variable g_{check} wird die bisher erkannte Geometrie gespeichert und in jedem Schritt entsprechend erweitert (Zeile 8). Die bereits erkannte Geometrie wird von der Schnittmenge der aktuellen Iteration abgezogen, damit dieselbe Geometrie nicht mehrfach erkannt wird. Das soll verhindern, dass Geometrien, die sich in größerer Entfernung zur Raumgeometrie befinden, nicht fälschlicherweise erkannt werden. Die geometrischen Operationen werden in 2D durchgeführt, indem die Raumgeometrie auf die Ebene des Bauelements projiziert und das Ergebnis anschließend wieder zurück projiziert wird. Zusätzlich zur Geometrie der *Boundary Parts* werden die Referenzen zum IFC-Element des Raums und des Bauelements gespeichert. Diese Information ist für die weiteren Schritte des Algorithmus notwendig, damit der Ursprung der vereinfachten Geometrie nachvollziehbar ist.

Die Funktionsweise der *Boundary Part*-Erkennung ist in Abbildung 3.10 schematisch dargestellt. Es handelt sich dabei um dasselbe Gebäude, welches auch in Abbildung 3.1 verwendet wurde. Zusammengehörige Räume und *Boundary Parts* werden in derselben Farbe dargestellt. Die schraffierten Flächen kennzeichnen die Raumgeometrie des entsprechenden Raumes. *Boundary Parts* werden als Pfeile mit Spitzen an jedem Ende dargestellt. Pfeile mit nur einer Spitze kennzeichnen die Normalvektoren der *Boundary Parts* und der danebenliegenden Wandgeometrie. In der Abbildung ist leicht zu erkennen, dass die *Boundary Parts*, ausgehend von der Raumgeometrie, aus der gegenüberliegenden Geometrie des nächstgelegenen Bauelements gebildet werden. D.h., es werden die Geometrien als *Boundary Part* erkannt, die der Raumgeometrie in Richtung des Normalvektors am nächsten liegen und die maximale Entfernung $E_{max_{boundary}}$ nicht überschreiten. Sowohl der Paramter $E_{max_{boundary}}$ als auch die Überprüfung mit der Variable g_{check}, welche die bisher erkannte Geometrie enthält, verhindern, dass Geometrien, die von der Raumgeometrie zu weit entfernt liegen, fälschlicherweise als *Boundary Part* erkannt werden. Ohne diese Mechanismen würde z.B. ein Teil von *Wand 3* als *Boundary Part* von Raum A und B erkannt werden.

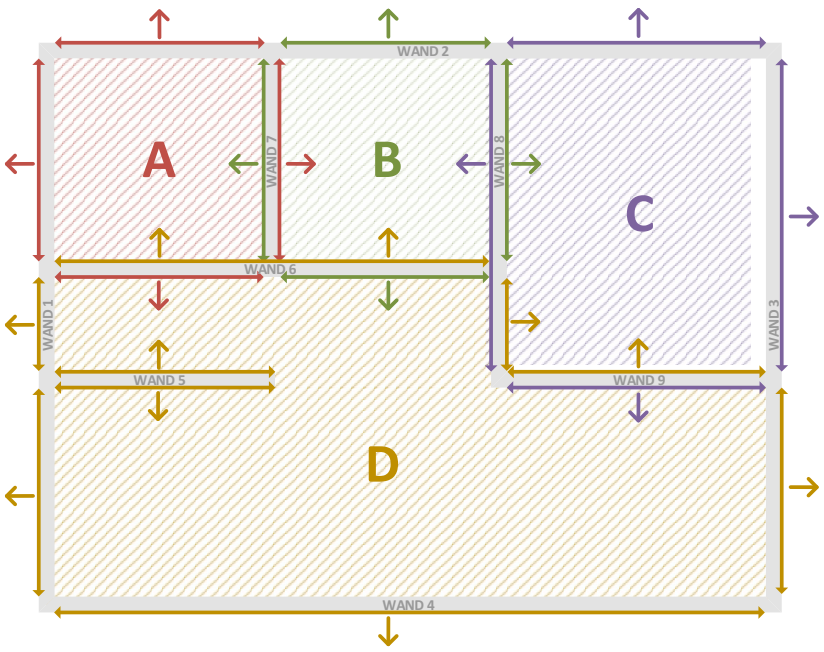

Abbildung 3.10: Schematische Darstellung der Space Boundaries nach dem ersten Schritt des GINGER Algorithmus. Angelehnt an die Arbeit von [Lad14].

In Abbildung 3.11 ist das Ergebnis des aktuellen Schrittes, analog zur schematischen Darstellung, in 3D dargestellt. Die Farben der Räume korrespondieren mit den Farben der schematischen Darstellung.

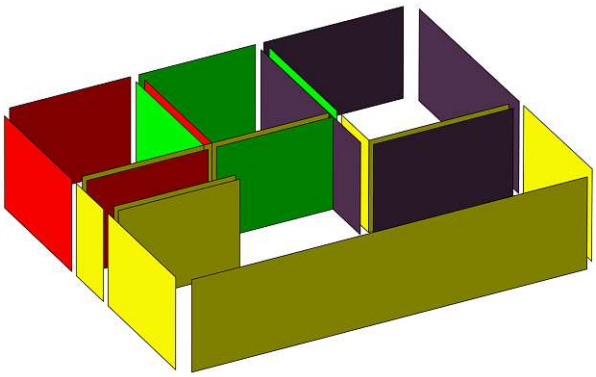

Abbildung 3.11: 3D Darstellung der Space Boundaries nach dem ersten Schritt des GINGER Algorithmus

3.5.5 Berechnung der horizontalen Boundaries

Nachdem im vorigen Schritt die vertikalen Boundaries ermittelt wurden, werden im nächsten Schritt die horizontalen Boundaries, ausgehend von den Raumgeometrien, berechnet. Das Ziel des Algorithmus ist der Export der vereinfachten Geometrie als *Brep Solid Geometry*, daher darf die resultierende Geometrie keine Lücken aufweisen. Der Algorithmus zur Berechnung der horizontalen Boundaries ist in Listing 3.5 dargestellt.

Algorithmus 3.5: BerechnungHorizontalerBoundaries

Eingabe: Liste R aller Räume mit Bauelement e, vereinfachter Geometrie g mit Polygon p und Normalvektor n
Liste B aller vertikalen Boundary Parts mit Bauelement e, Raum r und der Geometrie g mit Polygon p und Normalvektor n

Ausgabe: Liste B von Boundary Parts erweitert um horizontale Boundaries

1 **für jedes** $r_i \in R$ **tue**
2 $B_{raum} \leftarrow \{b \in B \mid r = e_i\}$ // Boundary Parts für Raum r_i
3 $B_{horizontal} \leftarrow \{r \in R \mid \nexists\, b_{raum} \in B_{raum} : n_{raum} = n\}$
4 $b_{max} \leftarrow$ Boundary Part $b \in B_{horizontal}$ mit maximaler Fläche
5 $b_{opposite} \leftarrow$ Boundary Part $b \in B_{horizontal}$ mit $\vec{n} = -n_{\vec{b_{max}}}$ und maximaler Fläche
6 Erstelle neue horizontale Boundary Parts für b_{max} und $b_{opposite}$
7 **Ende**
8 **return** B

Als Eingabe benötigt der Algorithmus die vertikalen *Boundary Parts* aus dem vorherigen

Schritt und zusätzlich eine Liste aller Räume, inklusive der dazugehörigen Geometrie. Die Berechnung der horizontalen Boundaries erfolgt pro Raum, deshalb wird in Zeile 1 über alle Räume iteriert. Für jeden Raum werden in Zeile 2 die *Boundary Parts* ermittelt, welche zum jeweiligen Raum gehören, d.h., wo die Referenz r dem aktuellen Raum entspricht. In Abschnitt 3.5.4 ist beschrieben, wie die *Boundary Parts*, ausgehend von der Raumgeometrie berechnet werden. In diesem Schritt wurden die horizontalen Flächen der Raumgeometrie vernachlässigt. In Zeile 3 werden diese horizontalen Flächen aus der Raumgeometrie ermittelt. Dabei kommen alle Flächen in Frage, deren Orientierungen in den bisher ermittelten *Boundary Parts* des Raums nicht vorkommen. Die fehlenden Flächen werden also anhand der Normalvektoren ermittelt, indem überprüft wird, welche Normalvektoren aus der Raumgeometrie noch nicht in den *Boundary Parts* vorkommen. In diesem Schritt wird also angenommen, dass die tatsächlichen horizontalen Raumgrenzen einen Normalvektor besitzen, der in den bisher berechneten Boundary Parts nicht vorkommt. Aus diesen Kandidaten wird in Zeile 4 die Geometrie mit der größten Fläche ermittelt und stellt die erste ermittelte horizontale Boundary dar. Diese Heuristik geht davon aus, dass die horizontalen Boundaries nach den vertikalen Boundaries die meiste Fläche einnehmen. Die zweite horizontale Boundary wird in Zeile 5 ermittelt, indem die Geometrie mit der größten Fläche ermittelt wird, die der ersten horizontalen Boundary gegenüberliegt, also einen entgegengesetzten Normalvektor besitzt. Für die beiden berechneten horizontalen Boundaries wird abschließend ein neuer Boundary Part erstellt und zu den restlichen Boundary Parts hinzugefügt. Die ermittelten horizontalen Boundaries sind als farbige Flächen in Abbildung 3.12 schematisch und in Abbildung 3.13 in 3D dargestellt.

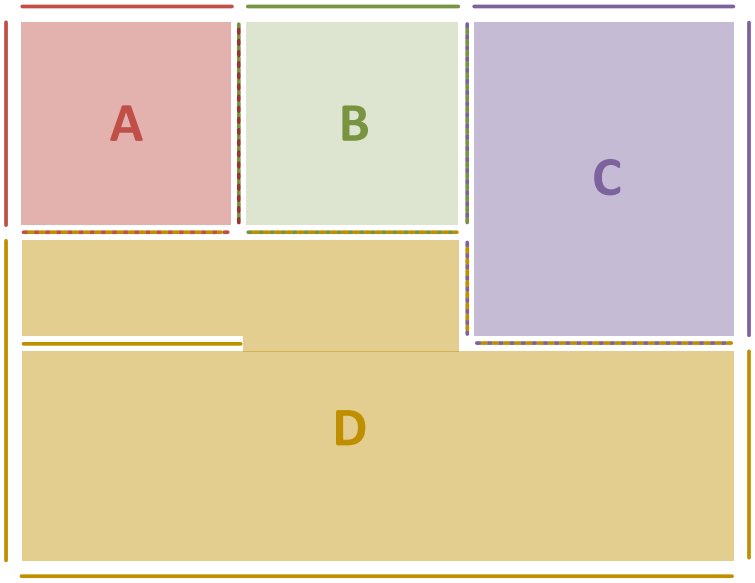

Abbildung 3.12: Schematische Darstellung der horizontalen Boundaries

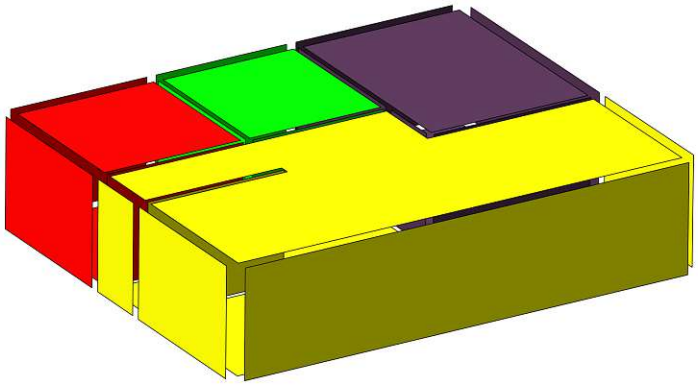

Abbildung 3.13: 3D Darstellung der horizontalen Boundaries

3.5.6 Zusammenfassen der inneren Boundaries

Algorithmus 3.6: InnereBoundariesZusammenfassen

Eingabe: Boundary Parts B mit Bauelement e, Raum r_1, Raum r_2 und der Geometrie g mit Polygon p und Normalvektor n
Maximale Entfernung zweier Boundary Parts als Schwellwert $E_{max_{inner}}$

Ausgabe: Liste B von Boundary Parts mit zusammengefassten inneren Boundaries

1 **für jedes** $(b_i, b_j) \in B$ **tue**
2 **wenn** $n_i = -n_j \wedge$ `entf` $(g_i,g_j) < E_{max_{inner}}$ **dann**
3 $g_{schnitt} \leftarrow g_i \cap g_j$
4 **wenn** $g_{schnitt} \neq \emptyset$ **dann**
5 Erstelle Boundary Part b_{ij}
6 $e_{ij} \leftarrow e_i$ `// Bauelement`
7 $r1_{ij} \leftarrow r_1$ `// Raum 1`
8 $r2_{ij} \leftarrow r_2$ `// Raum 2`
9 $p_{ij} \leftarrow$`trans(`$g_{schnitt}$, n_i `* (entf(`g_i,g_j`)/2))` `// Geometrie`
10 $n_{ij} \leftarrow n_i$ `// Orientierung`
11 Markiere b_i und b_j zum Entfernen
12 **Ende**
13 **Ende**
14 **Ende**
15 Entferne markierte Boundary Parts
16 **return** B

In den vorigen beiden Schritten des Algorithmus wurden die Raumgrenzen in Form von *Boundary Parts* berechnet. Bei Innenwänden, welche zwei Räume voneinander trennen, entstanden dadurch je zwei *Boundary Parts* an den Außenflächen der entsprechenden Bauelemente. Ziel unseres Algorithmus ist es, dass die Geomtrie für die Darstellung in GA-Systemen vereinfacht wird, dabei nimmt die Exaktheit der Abbildung einen geringeren Stellenwert ein. Die *Boundary Parts* der Innenwände sollen deshalb zusammengefasst werden, um die Geometrie weiter zu vereinfachen. Der Algorithmus für das Zusammenfassen der inneren Boundaries ist in Listing 3.6 dargestellt. Es handelt sich dabei im Wesentlichen um den 2. Schritt des GINGER-Algorithmus [LBK+16] mit einer geringfügigen Adaptierung, auf die wir später genauer eingehen werden.

Der Algorithmus benötigt als Eingabe die zuvor ermittelten *Boundary Parts*. Zu jedem *Boundary Part* wird die Information benötigt, zu welchem Bauelement er gehört und in welchem Raum sich das Bauelement befindet. Im Zuge des Algorithmus werden jeweils zwei *Boundary Parts* zweier angrenzender Räume zu einem zusammengeführt. Dadurch ist es notwendig, dass ein zweiter Raum zusätzlich zum *Boundary Part* gespeichert werden kann. Am Beginn des Algorithmus wird der Verweis auf den zweiten Raum mit *Null* initialisiert. Zusätzlich zu diesen Informationen wird als Eingabe noch das Polygon und die Orientierung der Geometrie benötigt.

In Zeile 1 wird über alle Paare der *Boundary Parts* iteriert und in Zeile 2 alle Kandidaten ermittelt, welche potentiell zusammengefasst werden könnten. Dabei kommen alle *Boundary Parts* infrage, deren Normalvektoren in die entgegengesetzte Richtung zeigen und die beiden Polygone eine maximale Entfernung $E_{max_{inner}}$ nicht überschreiten. Die maximale Entfernung darf dabei nicht zu groß gewählt werden, da ansonsten *Boundary Parts* zusammengefasst werden würden, die nicht zur selben Innenwand gehören. In Zeile 3 werden die Geometrien zweier potentieller Kandidaten miteinander geschnitten. Der Schnitt der beiden Geometrien erfolgt dabei in 2D, indem b_j auf die Ebene von b_i projiziert und das Ergebnis anschließend zurück projiziert wird. Eine leere Schnittmenge bedeutet, dass die beiden *Boundary Parts* nicht zusammengehören und somit nicht zusammengefasst werden können. Ist die Schnittmenge nicht leer, dann wird eine neues *Boundary Part* mit der Schnittmenge als Geometrie erzeugt und zur Liste der *Boundary Parts* hinzugefügt. Dabei müssen die Daten der beiden ursprünglichen *Boundary Parts* ebenfalls zusammengefasst und im neuen *Boundary Part* hinterlegt werden. Als Bauelement wird das Bauelement des ersten Boundary Parts übernommen. In den meisten Fällen verweisen beide Kandidaten ohnehin auf dasselbe Bauelement, nur in speziellen Fällen sind diese unterschiedlich. Auf diese Sonderfälle wird am Ende des Abschnitts noch genauer eingegangen. Nachfolgende Schritte des Algorithmus benötigen die Information, welche *Boundary Parts* zu welchen Räumen gehören, deshalb werden beide Räume dem neu erstellten *Boundary Part* zugeordnet (Zeile 7 und 8). Die Geometrie des neuen *Boundary Parts* soll sich genau zwischen den beiden Bauelementen befinden. Deshalb wird die Geometrie der Schnittmenge in Richtung des Normalvektors, um die halbe Entfernung zwischen beider Geometrien, verschoben. Als Normalvektor wird der Vektor des ersten *Boundary Parts* b_i zugeordnet. Welcher Vektor verwendet wird, ist nicht von Bedeutung, jedoch sollte eine einheitliche Vorgehensweise gewählt werden. Die beiden *Boundary Parts* werden nach dem Zusammenfassen aber nicht sofort aus der Liste entfernt, da *Boundary Parts* unter Umständen zu unterschiedlichen *Boundary Parts* zusammengefasst werden können. Das Ergebnis des aktuellen Schritts ist in Abbildung 3.14 schematisch und in Abbildung 3.15 in 3D dargestellt.

Wie bereits erwähnt, gibt es bei der Zusammenfassung der inneren Boundaries kleinere Unterschiede zum GINGER-Algorithmus. Beim GINGER-Algorithmus werden ausschließlich *Boundary Parts* zusammengefasst, welche aus demselben Bauelement entstanden sind, aber eine Raumgrenze für unterschiedliche Räume darstellen. Diese Änderung betrifft ausschließlich die Zeile 2 in Listing 3.6. Die erste Bedingung kann bei Räumen

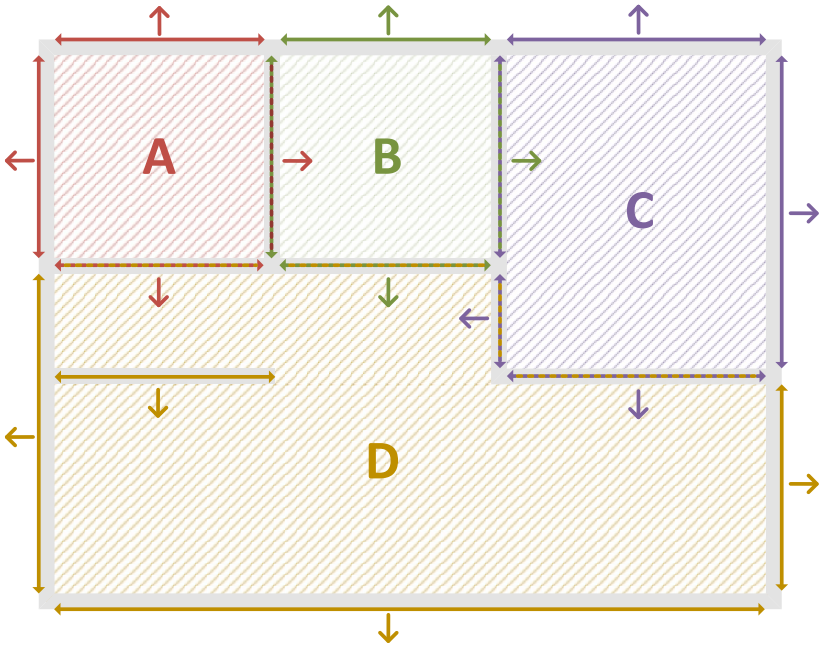

Abbildung 3.14: Schematische Darstellung der Space Boundaries nach dem Zusammenfassen der inneren Boundaries. Angelehnt an die Arbeit von [Lad14].

mit schrägen Wänden zu Problemen führen. Ein Beispiel hierfür ist in Abbildung 3.16 dargestellt, wo ein Teil eines Raumes, nach Anwendung der *Boundary Part*-Erkennung dargestellt ist und die inneren Boundaries noch nicht zusammengefasst wurden. Die unterschiedlichen Bauelemente sind hier in unterschiedlichen Farben dargestellt. Nach dem ersten Schritt des GINGER-Algorithmus wird ein Teil der roten und blauen Wand als Boundary Part abgebildet, welcher sich in derselben Ebene befindet, wie die graue Wand. Damit auch solche Raumgrenzen zusammengefasst werden können, die sich aus *Boundary Parts* unterschiedlicher Bauelemente zusammensetzen, wird die Bedingung im Algorithmus 3.6 aufgeweicht und auch auf unterschiedliche Bauelemente angewandt.

Manche Innenwände fungieren als Raumteiler und befinden sich gesamtheitlich im selben Raum. Eine solche Wand ist in Abbildung 3.10 als *Wand 5* dargestellt. Damit solche Innenwände ebenfalls zusammengefasst werden können, muss die 2. Bedingung, dass die Kandidaten aus unterschiedlichen Räumen stammen müssen, ebenfalls aufgeweicht werden. Damit durch die Aufweichung der beiden Bedingungen keine *Boundary Parts* zusammengefasst werden, die nicht zusammengehören, muss die maximale Entfernung der beiden *Boundary Parts* limitiert werden. Dies erfolgt über den Parameter $E_{max_{inner}}$ und muss auf die Gegebenheiten des jeweiligen Gebäudes abgestimmt sein.

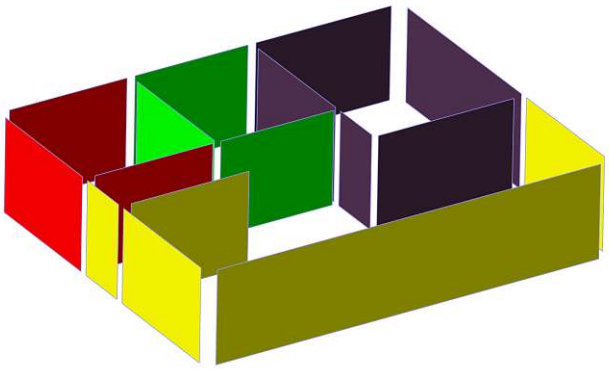

Abbildung 3.15: 3D Darstellung der Space Boundaries nach dem Zusammenfassen der inneren Boundaries

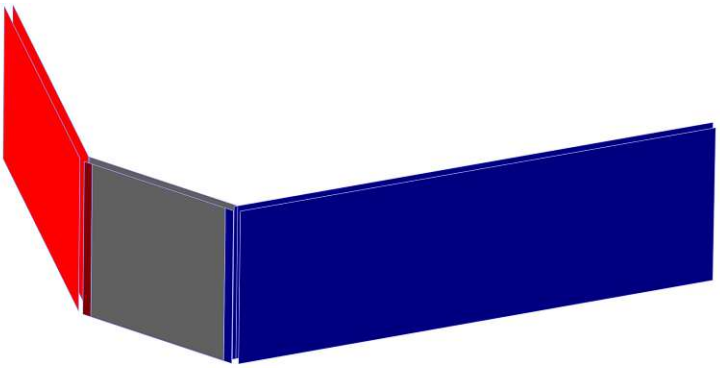

Abbildung 3.16: Raum mit schräger Wand

3.5.7 Füllen der entstandenen Lücken

Algorithmus 3.7: LückenFüllen

Eingabe: Liste B aller Boundary Parts mit Bauelement e, Raum r_1, Raum r_2 und der Geometrie g mit Polygon p und Normalvektor n
$E_{max_{lücken}}$ als obere Grenze für die minimale Entfernung zweier Geometrien, deren Lücken gefüllt werden sollen

Ausgabe: Liste B von Boundary Parts mit gefüllten Lücken

```
1  für jedes b_i ∈ B tue
2      B_adj ← {b ∈ B | b ≠ b_i ∧ selberRaum(b,b_i) ∧ minEntf(p,p_i) < E_max_lücken}
3      für jedes b_j ∈ B_adj tue
4          für jedes kante_k ∈ p_i tue
5              punkt_schnitt ← Schnittpunkt der Linie von Kante kante_k mit der
                Ebene, in der p_j liegt
6              wenn punkt_schnitt ≠ ∅ dann
7                  Ersetze den Punkt der Kante kante_k mit punkt_schnitt, welcher
                    näher zur Ebene von p_j liegt
8              Ende
9          Ende
10     Ende
11 Ende
12 return B
```

Durch die Art und Weise wie der Algorithmus die *Boundary Parts* berechnet, entstehen Lücken in der Geometrie, wie in Abbildung 3.13 gut ersichtlich ist. Die Lücken entstehen vorwiegend in den Ecken von Räumen, dort wo verschiedene Bauelemente aufeinandertreffen. Das liegt daran, dass in der ursprünglichen Raumgeometrie die Flächen der Bauelemente, welche den Raum abgrenzen, nicht enthalten sind. In diesem Schritt geht es nun darum, diese Lücken weitestgehend zu schließen. Es handelt sich dabei im Wesentlichen um den 3. Schritt des GINGER-Algorithmus. Im Gegensatz zum GINGER-Algorithmus werden auch Lücken zwischen den vertikalen und den zuvor berechneten horizontalen Boundaries geschlossen. Der Algorithmus für das Schließen der Lücken ist in Listing 3.7 dargestellt.

Als Eingabe benötigt der Algorithmus die bisher ermittelten *Boundary Parts*. Im ersten Schritt wird für einen *Boundary Part*, ein benachbarter *Boundary Part* ermittelt, wie in Zeile 2 ersichtlich ist. Die Bedingung $b \neq b_i$ schließt aus, dass Lücken zwischen *Boundary Parts* desselben Bauelements gefüllt werden. In diesem Schritt des Algorithmus sollen nur Lücken zwischen *Boundary Parts* unterschiedlicher Bauelemente gefüllt werden. Be-

nachbarte *Boundary Parts* müssen an denselben Raum angrenzen. Das bedeutet, dass einer der beiden Räume r_1 oder r_2 der beiden *Boundary Parts* übereinstimmen muss. Im Algorithmus wird diese Bedingung durch die Funktion *selberRaum* mit den Parametern b und b_i dargestellt. Um *Boundary Parts* auszuschließen, die zu weit voneinander entfernt sind, wird die minimale Entfernung der beiden Geometrien durch den Parameter $E_{max_{lücken}}$ begrenzt. Die Liste der ermittelten, angrenzenden *Boundary Parts* wird nach Entfernung von b_i zur Ebene des angrenzenden *Boundary Parts* sortiert. In Algorithmus 3.7 werden benachbarte *Boundary Parts* nur auf Basis der Geometrie ermittelt. Der Verbund zweier Elemente, wie z.B. aneinandergrenzende Wände, kann im IFC-Modell über die Beziehung *IfcRelConnectsElements* dargestellt werden. Benachbarte *Boundary Parts* können daher auch über diese Beziehung ermittelt werden, anstatt den Umweg über die Geometrie zu gehen. Häufig sind Verbindungen zwischen zwei Elementen in IFC nicht richtig abgebildet, sodass der einzige Weg zur Bestimmung benachbarter *Boundary Parts* über die Geometrie führt.

Wurde ein benachbarter *Boundary Part* gefunden, dann wird im nächsten Schritt versucht, die Lücken zwischen den zwei *Boundary Parts* zu schließen. Dabei wird für jede Kante des 1. Polygons versucht, einen Schnittpunkt mit der Ebene, in der das 2. Polygon liegt, zu finden. Wurde ein Schnittpunkt gefunden, dann wird jener Punkt der Kante, der näher zum Schnittpunkt liegt, durch den Schnittpunkt ersetzt und damit die Lücke geschlossen. In Abbildung 3.17 ist schematisch dargestellt, wie das Füllen der Lücken mit zwei einfachen Geometrien funktioniert. Die graue Geometrie stellt ein Wandelement mit einem offenen Durchgang dar und soll mit dem gelben Wandelement verbunden werden. Auf der linken Seite sind die beiden *Boundary Parts* vor und auf der rechten Seite nach dem Füllen der Lücken dargestellt. Die Linien der Kanten 1 und 2 werden mit der Ebene, in der die gelbe Wand liegt, geschnitten und der Schnittpunkt ersetzt jenen Endpunkt der Kante, der näher zum Schnittpunkt liegt. Die Kante 3 in der Abbildung stellt einen Sonderfall dar. In diesem Fall darf der Endpunkt der Kante nicht durch den Schnittpunkt ersetzt werden, da dadurch eine komplett andere Geometrie entstehen würde. Um diesen Sonderfall zu erkennen, muss überprüft werden, ob sich die Linie der Kante mit einer anderen Kante desselben Polygons schneidet. Ist das der Fall, dann darf der Endpunkt der Kante nicht durch den Schnittpunkt mit der Ebene ersetzt werden.

Bei der Ermittlung benachbarter *Boundary Parts* ist die minimale Entfernung beider Polygone nach oben hin über den Parameter $E_{max_{lücken}}$ begrenzt. Diese Schranke ist aber sehr ungenau und dient lediglich dazu, um den Algorithmus zu beschleunigen und *Boundary Parts*, die zu weit voneinander entfernt liegen, bereits im 1. Schritt auszusortieren. Um zu vermeiden, dass Lücken zwischen *Boundary Parts* gefüllt werden, die im ursprünglichen Gebäudemodell nicht miteinander verbunden sind, ist eine weitere Schranke notwendig. Diese Schranke definiert die maximale Entfernung zwischen dem Endpunkt der Kante und dem Schnittpunkt mit der Ebene. Die Entfernung dieser zwei Punkte ist in Abbildung 3.17 durch die grünen Linien dargestellt. Ein guter Wert für diese Schranke in unserem Beispiel ist, wenn die Entfernung mit der Dicke der gelben Wand beschränkt wird. Zusätzlich sollte eine Toleranz dazugerechnet werden, da z.B. die

Dicke der Wand als obere Schranke bei sehr schrägen Wänden nicht ausreicht. Je nach Gebäudemodell kann es notwendig sein, unterschiedliche Heuristiken für solche Schranken zu verwenden. Zusätzlich sollte auch die minimale Entfernung des berechneten Punktes mit der zu verbindenden Geometrie beschränkt werden. Der Schnittpunkt der Ebene kann sehr weit von der tatsächlichen Geometrie entfernt sein. Dieser Umstand muss entsprechend berücksichtigt werden. In Abbildung 3.17 entspricht das der Entfernung zwischen dem berechneten Punkt auf der Ebene der gelben Geometrie und der gelben Geometrie.

Sollte bei der verwendeten Heuristik für die oberen Schranken die Dicke von Wänden benötigt werden, dann kann diese bei Wänden gleichbleibender Dicke direkt aus dem IFC-Modell über die Entität *IfcWallStandardCase* ermittelt werden. Ist das nicht der Fall, dann kann die maximale Dicke der Wand auf Basis der Geometrie berechnet werden, indem z.B. die maximale Entfernung der beiden größten Flächen einer Wand herangezogen wird. Bei dieser Art der Berechnung handelt es sich jedoch um eine Heuristik, die unter Umständen nicht in jedem Fall funktioniert. Ist es nicht möglich, die Dicke der Wand zu ermitteln, dann sollte die maximale Entfernung über einen Parameter limitiert und für das jeweilige Gebäude passend gewählt werden.

Das Ergebnis nach dem Füllen der Lücken ist in Abbildung 3.18 schematisch dargestellt. In Abbildung 3.19 ist das Ergebnis in 3D auf der linken Seite ohne und auf der rechten Seite mit transparenten Wänden dargestellt. Durch die Transparenz der Wände erhält man einen besseren Einblick in die Innenräume des Gebäudes nach dem Füllen der Lücken. In der 3D Darstellung kann man erkennen, dass nicht alle Lücken in diesem Schritt geschlossen werden konnten. Vor allem Lücken zwischen vertikalen und horizontalen Boundaries konnten nicht gänzlich gefüllt werden. Die verbleibenden Lücken werden später genauer behandelt.

3.5.8 Geometrische Nachbearbeitungen

Nach dem Ausführen des GINGER-Algorithmus und der Berechnung der horizontalen Boundaries ist es notwendig, gewisse Nachbearbeitungen und Vereinfachungen in der Geometrie durchzuführen. Lücken zwischen vertikalen und horizontalen Boundaries konnten bisher noch nicht zur Gänze gefüllt werden. Die Raumgeometrie ist durch die innen liegende Seite der Bauelemente begrenzt und die berechneten *Boundary Parts* befinden sich auf der Außenseite der Bauelemente. Dadurch kann, vorwiegend in Ecken von Räumen, durch den Algorithmus aus Abschnitt 3.5.7 kein Schnittpunkt zwischen zwei *Boundary Parts* gefunden werden, da diese leicht versetzt zueinander stehen. Dieses Problem wird in diesem Abschnitt genauer behandelt. Der Algorithmus aus Abschnitt 3.5.6 fasst *Boundary Parts*, die aus Zwischenwänden entstanden sind, zu einem *Boundary Part* zusammen. Am Ende des Algorithmus wird jedoch pro Raum ein Solid exportiert, deshalb müssen die zusammengefassten *Boundary Parts* wieder aufgesplittet werden, bevor mit den weiteren Schritten fortgefahren wird. Dadurch entstehen für jeden Raum wieder separate *Boundary Parts*. Befinden sich unterschiedliche Wandelemente nah nebeneinander, dann kann es durch den Algorithmus aus Abschnitt 3.5.7 vorkommen, dass

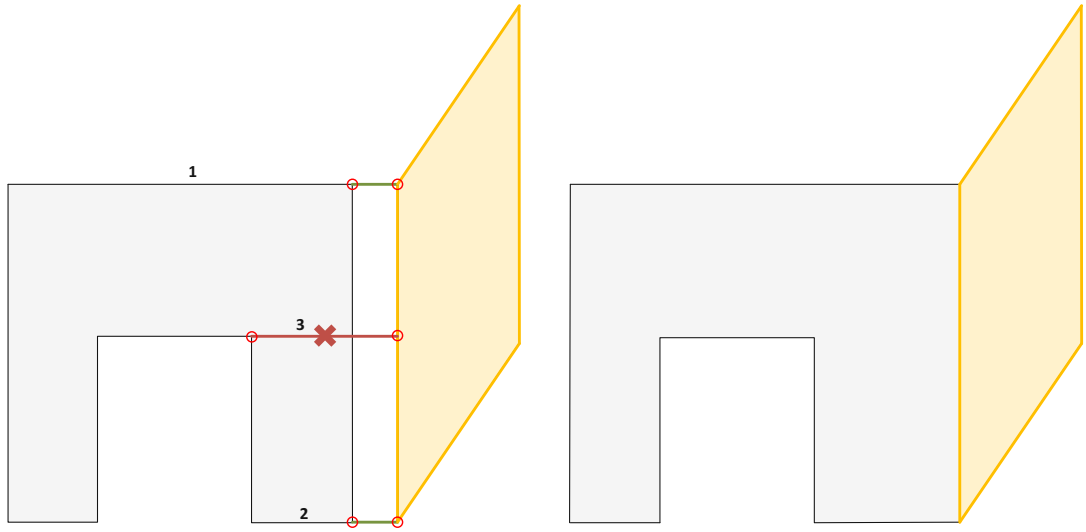

Abbildung 3.17: Schematische Darstellung zweier Wandelemente. Das graue Wandelement stellt einen offenen Durchgang dar. Es wird gezeigt, wie Lücken zwischen zwei Boundary Parts gefüllt werden.

diese Lücken mehrfach bei nebeneinander liegenden *Boundary Parts* gefüllt werden und es entstehen Flächen, die sich überlappen. Die überlappenden Flächen können in weiterer Folge zu Kollisionen führen, deshalb werden sie entsprechend vereinfacht. Bisher wurden die Lücken Raum-übergreifend gefüllt, sofern es sich um benachbarte Räume handelt. Die vereinfachte Geometrie wird jedoch auf Raumebene exportiert, deshalb müssen die Lücken nach dem Aufsplitten der *Boundary Parts* mit dem Algorithmus aus Abschnitt 3.5.7 nochmal auf Raumebene gefüllt werden. Am Ende der Nachbearbeitungen werden Kanten vereinfacht, welche durch die Art der konkreten Implementierung aufgesplittet sein könnten, damit in weiter Folge beim Berechnen der Solids keine Probleme auftreten. Ziel dieser Nachbearbeitungen ist in erster Linie die Erhöhung der Robustheit des Algorithmus und die zusätzliche Vereinfachung der Geometrie, sodass am Ende eine möglichst geringe Anzahl an Flächen, Kanten und Punkten vorhanden ist.

3.5.8.1 Verbinden von horizontalen Boundaries

Bisher konnten noch nicht alle Lücken zwischen horizontalen und vertikalen Boundaries geschlossen werden. Diese Lücken treten vor allem in Ecken zwischen zwei Räumen auf, da diese durch den Algorithmus aus Abschnitt 3.5.7 noch nicht berücksichtigt werden konnten. In Abbildung 3.20 sind die verbleibenden Lücken aus unserem Beispiel dargestellt. Die Lücke auf der linken Seite entsteht durch die innere Wand, die als Raumteiler in diesem Raum fungiert. Die Fläche dieser Wand ist in der Raumgeometrie nicht enthalten, aus der die horizontalen Boundaries berechnet werden. Im Algorithmus aus Abschnitt 3.5.7 werden immer die Kanten der Polygone betrachtet und die Lücken

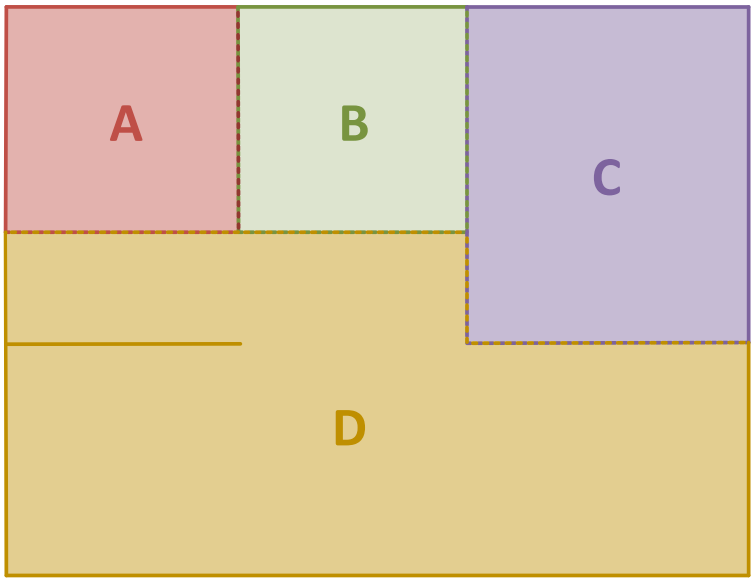

Abbildung 3.18: Schematische Darstellung nach dem Füllen der Lücken

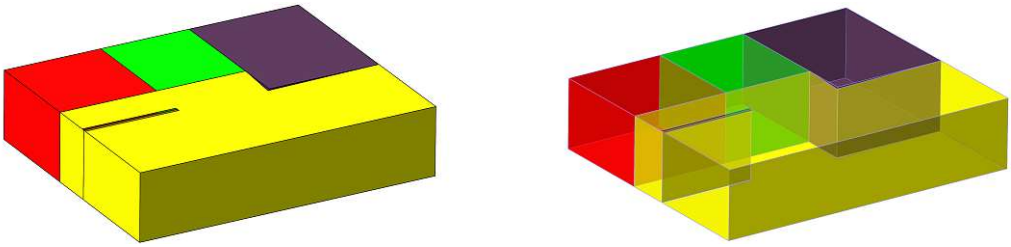

Abbildung 3.19: 3D Darstellung nach dem Füllen der Lücken

durch Verlängerung dieser Kanten geschlossen. Dabei wird ein Endpunkt einer Kante durch den Schnittpunkt mit der Ebene des benachbarten *Boundary Parts* ersetzt. Die rechte Lücke in Abbildung 3.20 konnte nicht geschlossen werden, da keine Kante im Polygon der horizontalen Boundary existiert, die so verlängert werden könnte, sodass diese Lücke geschlossen wird. Aus diesem Grund werden in nachfolgendem Schritt nicht die Kanten des Polygons betrachtet, sondern die Punkte selbst. In einem Raum existieren immer nur zwei *Boundary Parts* für die horizontalen Raumgrenzen, einer für die Decke und einer für den Boden. Jeder Punkt der horizontalen Boundaries wird mit dem Punkt der vertikalen Boundaries ersetzt, der ihm am nächsten liegt. Wichtig hierbei ist, dass die

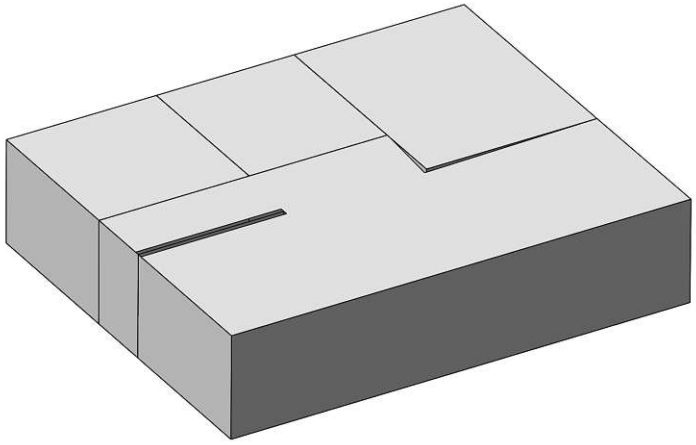

Abbildung 3.20: 3D Darstellung vor dem Verbinden der horizontalen mit den vertikalen Boundaries

vertikalen und horizontalen Boundaries immer nur für einen Raum betrachtet werden.

Algorithmus 3.8: VerbindeHorizontaleBoundaries

Eingabe: Liste R aller Räume mit Bauelement e
Liste B aller Boundary Parts mit Bauelement e, Raum r_1, Raum r_2 und der Geometrie g mit Polygon p

Ausgabe: Liste B von Boundary Parts ohne Überlappungen

für jedes $r_i \in R$ **tue**
  // Vertikale Boundary Parts für Raum r_i
  $B_{vertikal} \leftarrow \{b \in B \mid (r_1 = e_i \vee r_2 = e_i) \wedge \texttt{istVertikal}(b)\}$
  // Horizontale Boundary Parts für Raum r_i
  $B_{horizontal} \leftarrow \{b \in B \mid (r_1 = e_i \vee r_2 = e_i) \wedge \texttt{istHorizontal}(b)\}$
  für jedes $b_h \in B_{horizontal}$ **tue**
    für jedes $c_j \in p_h$ **tue** // Für jede Koordinate des Polygons p_h von b_h
      $c_{vert_{min}} \leftarrow$Koordinate aus den vertikalen Boundary Parts $B_{vertikal}$ mit minimaler Distanz zu c_j
      Ersetze c_j durch $c_{vert_{min}}$
    Ende
  Ende
Ende
return B

Der Algorithmus zum Verbinden der horizontalen mit den vertikalen Boundaries ist in Listing 3.8 dargestellt. Als Eingabe benötigt der Algorithmus alle *Boundary Parts* und zusätzlich eine Liste aller Räume. Die horizontalen Boundaries werden in diesem Schritt pro Raum betrachtet, deshalb wird in Zeile 1 über alle Räume iteriert. In Zeile 2 werden die vertikalen *Boundary Parts* für den Raum der aktuellen Iteration ermittelt. Die Information, ob ein *Boundary Part* eine vertikale oder horizontale Raumgrenze darstellt, wird idealerweise beim Erkennen der jeweiligen Boundaries aus den vorherigen Schritten des Algorithmus zum *Boundary Part* dazu gespeichert. Die horizontalen *Boundary Parts* für den aktuellen Raum werden in Zeile 3 ermittelt. Für jede Koordinate der horizontalen Boundary wird in Zeile 6 jene Koordinate aus den vertikalen Boundaries desselben Raums ermittelt, welche der Koordinate am nächsten liegt. In Zeile 7 wird die Koordinate der horizontalen Boundary mit der ermittelten Koordinate ersetzt und somit eine Lücke zwischen zwei *Boundary Parts* geschlossen.

3.5.8.2 Splitten zusammengefasster Boundary Parts

In Abschnitt 3.5.6 wurden *Boundary Parts* von Zwischenwänden zu einem *Boundary Part* zusammengefasst, damit das anschließende Füllen der Lücken und das Verbinden der vertikalen mit den horizontalen Boundaries funktioniert. Die nachfolgenden Schritte betrachten die Geometrien aber auf Raumebene, deshalb müssen die zusammengefassten *Boundary Parts* wieder aufgesplittet werden, damit eine vollständige Boundary für jeden Raum existiert. Dabei wird für jeden *Boundary Part*, wo sowohl r_1 als auch r_2 gesetzt ist, zwei separate *Boundary Parts* erstellt. Dem ersten *Boundary Part* wird der Raum r_1 zugeordnet und dem zweiten *Boundary Part* r_2. Beim zweiten *Boundary Part* ist zu beachten, dass der inverse Normalvektor verwendet und die Koordinaten des Polygons invertiert werden.

3.5.8.3 Zusammenfassen überlappender Boundary Parts

Durch das Füllen der Lücken, wie es in Abschnitt 3.5.7 beschrieben ist, kann es vorkommen, dass sich *Boundary Parts* überlappen. Als Beispiel dient uns die schräge Wand aus Abbildung 3.16. Die *Boundary Parts* der roten und blauen Wand, die sich in derselben Ebene befinden, wie die graue Wand, bleiben auch nach dem Zusammenfassen der inneren Boundaries bestehen. Die Lücke zwischen den Wänden wird in diesem Fall mehrmals gefüllt, da sowohl die graue Wand, als auch die roten und blauen Teile der anderen Wände miteinander verbunden werden. Nach dem Füllen der Lücken überlappen sich also die graue Wand mit den beiden kleineren Teilen der roten und blauen Wand. Dieser Umstand ist in Abbildung 3.21 dargestellt. Auf der linken Seite sieht man die Wände vor dem Füllen der Lücken, wie sie auch in Abbildung 3.16 vor dem Zusammenfassen der inneren Boundaries dargestellt sind. Auf der rechten Seite ist das Ergebnis nach dem Füllen der Lücken dargestellt und die gepunkteten Flächen deuten an, dass sich die graue Wand und die beiden kleineren Teile der anderen Wände überlappen.

In diesem Schritt des Algorithmus werden überlappende *Boundary Parts* vereinfacht, indem jene *Boundary Parts* entfernt werden, deren Geometrie bereits Teil eines anderen

Boundary Parts ist. Im Beispiel in Abbildung 3.21 werden durch diesen Schritt die beiden roten und blauen *Boundary Parts* entfernt, die sich mit dem *Boundary Part* der grauen Wand überlappen. Der Algorithmus ist in Listing 3.9 dargestellt. Er benötigt als Eingabe eine Liste der *Boundary Parts* inklusive der Geometrie, den Normalvektor und die Distanz der Ebene, in der die Geometrie liegt, zum Ursprung. Überlappende *Boundary Parts* die entfernt werden können, werden in der Liste B_{entf} gespeichert. Die Liste wird in Zeile 1 initialisiert. Anschließend wird in Zeile 2 über alle *Boundary Parts* iteriert. Damit überlappende Geometrien nicht mehrfach entfernt werden, wird in Zeile 3 überprüft, ob sich der *Boundary Part* in der Liste der markierten *Boundary Parts* befindet. Ist das der Fall, dann wird die Vereinfachung für diesen Schritt übersprungen, da ansonsten Lücken in der Geometrie entstehen würden. In Zeile 4 werden jene *Boundary Parts* ermittelt, die sich im selben Raum und in derselben Ebene befinden. Durch die Bedingung $b \neq b_i$ wird ausgeschlossen, dass ein *Boundary Part* mit sich selbst verglichen wird. Die Vereinfachung darf immer nur innerhalb eines Raumes erfolgen, da ansonsten Lücken in Räumen entstehen könnten, wenn übergreifende Überlappungen entfernt werden. Der Vergleich der Normalvektoren und die Distanzen zum Ursprung sollten mit einer gewissen Ungenauigkeit durchgeführt werden, da Rundungsfehler auftreten können, wodurch exakte Vergleiche nicht in jedem Fall funktionieren. In Zeile 6 wird überprüft, ob die Geometrie des *Boundary Part* b_j bereits durch die Geometrie von *Boundary Part* b_i abgedeckt ist. Die Überprüfung erfolgt dabei durch die Differenz von g_j mit g_i. Ist die Differenz leer, dann bedeutet das, dass die Geometrie g_j bereits vollständig von g_i abgebildet wird und somit eine Teilmenge von g_i darstellt. In diesem Fall wird der *Boundary Part* b_j in Zeile 7 markiert, um später in Zeile 12 entfernt zu werden. Die Differenz der beiden Geometrien wird dabei in 2D durchgeführt, indem beide Geometrien auf die Ebene von g_i projiziert werden, bevor die Operation durchgeführt wird.

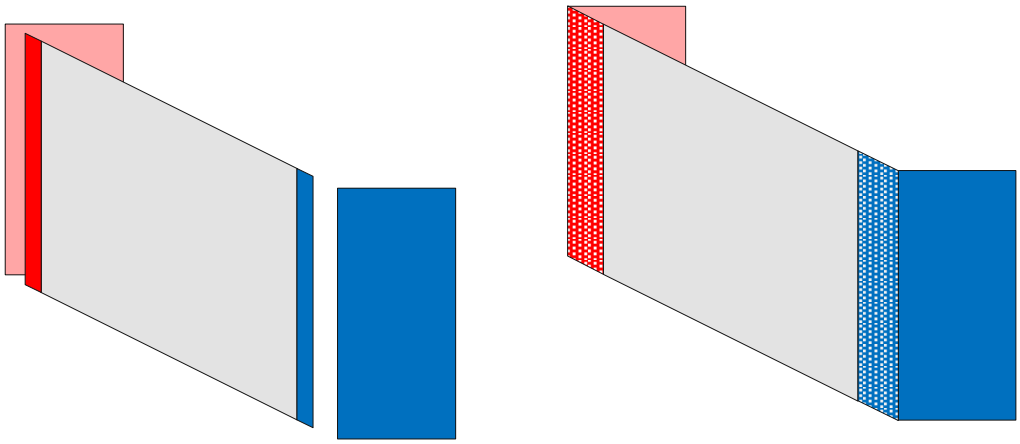

Abbildung 3.21: 3D Darstellung vor und nach dem Füllen der Lücken. Die gepunkteten Flächen stellen die überlappenden Wandelemente dar.

Algorithmus 3.9: VereinfacheUeberlappendeBoundaryParts

Eingabe: Liste B aller Boundary Parts mit Bauelement e, Raum r und der Geometrie g mit Polygon p, Normalvektor n und der Distanz der Ebene zum Ursprung d

Ausgabe: Liste B von Boundary Parts ohne Überlappungen

```
1  B_entf ← ∅
2  für jedes b_i ∈ B tue
3      wenn b_i ∉ B_entf dann
4          B_ebene ← {b ∈ B | b ≠ b_i ∧ r = r_i ∧ n = n_i ∧ d = d_i}
5          für jedes b_j ∈ B_ebene tue
6              wenn g_j \ g_i = ∅ dann
7                  FügeHinzu(B_entf, b_j)        // Markiere b_j zum Entfernen
8              Ende
9          Ende
10     Ende
11 Ende
12 Entferne markierte Boundary Parts B_entf von den Boundary Parts
13 return B
```

3.5.8.4 Verbinden noch freistehender Flächen

In Abschnitt 3.5.8.2 wurden *Boundary Parts* von Zwischenwänden wieder aufgesplittet, damit jeder Raum eine vollständige Boundary besitzt. In diesem Schritt sollen noch etwaige Lücken auf Raumebene geschlossen werden, damit am Ende Solids pro Raum erstellt werden können. In Abbildung 3.22 ist ein Beispiel zu sehen, wo nach Anwendung der bisherigen Schritte des Algorithmus noch Lücken bestehen. Auf der linken Seite der Abbildung sieht man die 3D Darstellung einer schrägen Wand, die mit den umliegenden Wänden verbunden werden soll. Die Lücke auf der linken Seite der roten Wand konnte bisher noch nicht gefüllt werden. Warum das so ist, ist auf der rechten Seite der Abbildung schematisch dargestellt. Vor dem Füllen der Lücken, wie in Abschnitt 3.5.7 beschrieben, sind die beiden Wände 2 und 3 miteinander verbunden und müssten eigentlich nicht weiter behandelt werden. Durch den Schritt aus Abschnitt 3.5.7 werden jedoch die Wände 1 und 2 miteinander verbunden, indem die Endpunkte der Kanten mit den Schnittpunkten der beiden Ebenen ersetzt werden. Die Kanten von Wand 3, welche die Verbindung zur Wand 2 darstellen, werden jedoch nicht verlängert, sondern verkürzt und durch die Schnittpunkte mit der Ebene von Wand 1 ersetzt. Dadurch wird die Wand 3 zu einer

freistehenden Fläche, welche nun mit den Wänden 1 und 2 verbunden werden muss. Durch nochmalige Ausführung des Algorithmus aus Listing 3.7 werden diese zwei Wände miteinander verbunden und damit sollten auch die restlichen Lücken in der Geometrie geschlossen sein.

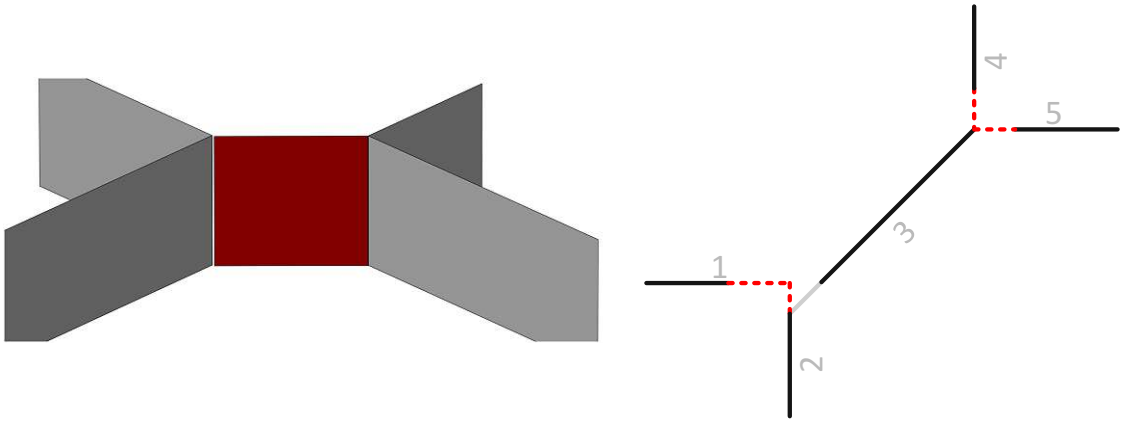

Abbildung 3.22: Beispiel einer freistehenden Fläche, die bisher nicht verbunden werden konnte

3.5.8.5 Vereinfachung der Kanten

In den bisherigen Schritten des Algorithmus wurden eine Reihe von geometrischen Operationen durchgeführt, um die Geometrie des Gebäudes schrittweise zu vereinfachen. Je nachdem wie diese geometrischen Operationen implementiert sind, kann es vorkommen, dass Kanten eines Polygons auf mehrere Kanten aufgesplittet werden, obwohl sie in dieselbe Richtung zeigen und aneinander grenzen. Im Polygon können auch dieselben Punkte mehrfach hintereinander vorkommen. Diese lassen sich vereinfachen, indem die redundanten Punkte aus der Geometrie entfernt werden. Dieser Schritt ist notwendig, um die Geometrie weiter zu vereinfachen und die Anzahl der Kanten und Punkte zu verringern. Zusätzlich soll damit die Robustheit des Algorithmus erhöht werden, sodass das anschließende *Sewing* der Geometrie und die Berechnung der Solids fehlerfrei ausgeführt werden kann. Je nach konkreter Implementierung kann es notwendig sein, diesen Schritt vor dem Füllen der Lücken (siehe Abschnitt 3.5.7) durchzuführen, damit nicht fälschlicherweise Punkte miteinander verbunden werden, die vom gegenüberliegenden Boundary Part weiter entfernt liegen, als der gewünschte Verbindungspunkt.

Der Algorithmus ist in Listing 3.10 dargestellt. Innerhalb der Schleife ab Zeile 2 werden die Koordinaten des *Boundary Parts* einzeln überprüft. Dabei wird die vorherige und nachfolgende Koordinate des Polygons in die Überprüfung miteinbezogen. In den nachfolgenden Zeilen 4 und 5 werden aus den drei Koordinaten zwei Kanten gebildet. Die 1. Kante wird aus der vorherigen und der aktuellen Koordinate gebildet und die 2. Kante aus der aktuellen und der nachfolgenden Koordinate. Anschließend wird in Zeile 6 u.a. überprüft, ob sich die aktuelle Koordinate von der nachfolgenden unterscheidet.

Ist das nicht der Fall, dann wird die aktuelle Koordinate nicht in die Ergebnismenge aufgenommen, da ansonsten zwei Koordinaten hintereinander denselben Wert aufweisen würden. Sind die beiden Koordinaten unterschiedlich, dann wird zusätzlich überprüft, ob die beiden Kanten parallel zueinander sind. Ist das der Fall, dann handelt es sich um eine längere Kante, die auf zumindest zwei kleinere Kanten aufgeteilt wurde, die jedoch in dieselbe Richtung zeigen. Die aktuelle Koordinate wird in diesem Fall nicht in die Ergebnismenge aufgenommen und damit übersprungen. Dadurch bleibt in der nächsten Iteration die Koordinate k_{vorher} gleich und es ändern sich nur die aktuelle und nachfolgende Koordinate, sodass aus zwei aufgesplittete Kanten eine einzelne Kante gebildet wird. In Zeile 11 wird die erste Koordinate der Ergebnismenge auch als letzte Koordinate eingefügt, damit ein geschlossener Ring entsteht. Theoretisch könnte auch $k_{nachher}$ als letzte Koordinate hinzugefügt werden, diese könnte sich aber aufgrund von Rechenfehlern geringfügig von der ersten Koordinate unterscheiden und in weiterer Folge zu Fehlern führen. Am Ende des Algorithmus in Zeile 12 wird aus den neuen Koordinaten $k_{ergebnis}$ ein vereinfachter *Boundary Part* erzeugt und als Ergebnis vom Algorithmus zurückgeliefert.

Algorithmus 3.10: VereinfacheKanten

Eingabe: Boundary Part mit Koordinaten k des Polygons

Ausgabe: Boundary Part mit vereinfachten Kanten

1 $K_{ergebnis} \leftarrow []$; $k_{vorher} \leftarrow k[0]$; $k_{nachher} \leftarrow k[1]$

2 **für** $i \leftarrow 0$ **bis** `size(`k`) - 2` **tue**

3 $k_{aktuell} \leftarrow k[i]$; $k_{nachher} \leftarrow k[i+1]$

4 $kante_{vorher} \leftarrow$`kante(`$k_{vorher}$, $k_{aktuell}$`)`

5 $kante_{nachher} \leftarrow$`kante(`$k_{aktuell}$,$k_{nachher}$`)`

6 **wenn** $k_{aktuell} \neq k_{nachher} \wedge kante_{vorher} \nparallel kante_{nachher}$ **dann**

7 `FügeHinzu(`$K_{ergebnis}$, $k_{aktuell}$`)`

8 $k_{vorher} \leftarrow k_{aktuell}$

9 **Ende**

10 **Ende**

11 `FügeHinzu(`$K_{ergebnis}$, $K_{ergebnis}[0]$`)`

12 **return** Boundary Part mit den neuen Koordinaten $k_{ergebnis}$

3.5.9 Export von Fenster/Türen

In Abschnitt 3.5.2 wurde bereits erwähnt, wie die Geometrie von Fenster und Türen vereinfacht werden kann, sodass sie in Form von orientierten Bounding Boxes dargestellt

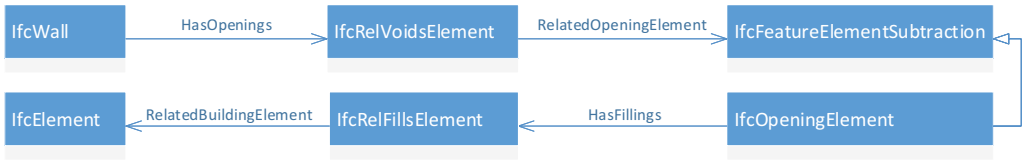

Abbildung 3.23: Verknüpfung zwischen *IfcWall* und den Bauelementen der Öffnungen

wird. Diese Darstellung lässt sich für unseren Zweck noch weiter vereinfachen, indem nicht die gesamte Bounding Box herangezogen wird, sondern lediglich eine Fläche davon. Dabei wird jene Fläche für die Vereinfachung ausgewählt, welche denselben Normalvektor besitzt, wie der *Boundary Part* der dazugehörigen Wand. Diese Fläche muss jedoch so verschoben werden, damit sie sich auf derselben Ebene wie das dazugehörige Wandelement befindet.

Algorithmus 3.11: FensterTürenAusrichten

Eingabe: Liste B aller Boundary Parts mit Bauelement e und der Geometrie g mit Polygon p
Liste FT aller vereinfachten Fenster- und Türgeometrien mit Bauelement e und der Geometrie g mit Polygon p und Normalvektor n
Maximale Entfernung zwischen Fenster- bzw. Tür-Geometrie und der Wand-Geometrie als Schwellwert $E_{max_{ft}}$

Ausgabe: Liste E der ausgerichteten Fenster- und Türgeometrien

1 $E \leftarrow []$
2 **für** *alle Wandgeometrien* w_i *in* B **tue**
3 $O \leftarrow$ `ErmittleOpeningElemente(`w_i`)`
4 **für jedes** $o_j \in O$ **tue**
5 $b_{ft} \leftarrow \{b \in FT \mid b = o_j \ \wedge \ n = n_i \wedge$ `entf` $(g, g_i) < E_{max_{ft}}\}$
6 **wenn** $g_{ft} \cap g_i \neq \emptyset$ **dann**
7 Füge b_{ft} mit Geometrie g_{ft}, projiziert auf die Ebene von w_i, der Ergebnismenge E hinzu
8 **Ende**
9 **Ende**
10 **Ende**
11 **return** E

Der Algorithmus für die weitere Vereinfachung der Fenster und Türen ist in Listing 3.11

dargestellt. Als Eingabe benötigt der Algorithmus einerseits eine Liste aller berechneten Boundary Parts und andererseits die vereinfachte Geometrie der Fenster und Türen in Form von orientierten Bounding Boxes. Damit Fenster und Türen nicht doppelt exportiert werden, müssen die noch nicht aufgesplitteten *Boundary Parts* (Abschnitt 3.5.8.2) für den Algorithmus herangezogen werden. Die Geometrien werden auf die berechneten Boundary Parts der Wände ausgerichtet, deshalb werden in Zeile 2 alle Wände iteriert. Anschließend werden in Zeile 3 für alle Öffnungen der jeweiligen Wand die entsprechenden Bauelemente ermittelt, welche diese Öffnungen füllen. Die Verknüpfung der Wände mit den entsprechenden Bauelementen ist in Abbildung 3.23 dargestellt. Ausgehend vom *IfcWall*-Element wird analog zur Abbildung 3.8 über die Beziehungen *HasOpenings* und *RelatedOpeningElement* das entsprechende *IfcOpeningElement* ermittelt. Anschließend kann über die Beziehungen *HasFillings* und *RelatedBuildingElement* zum entsprechenden Bauelement navigiert werden. Handelt es sich beim gefundenen Bauelement um ein Fenster oder eine Tür, kann die Vereinfachung durchgeführt werden. Für jedes gefundene Bauelement wird in Zeile 5 aus der Bounding Box des Fensters oder Tür jene Fläche ermittelt, welche denselben Normalvektor wie das Wand-Element aufweist. Zur Erhöhung der Robustheit sollte eine maximale Entfernung $E_{max_{ft}}$ angegeben werden, damit zu weit entfernte Elemente nicht fälschlicherweise miteinbezogen werden. In Zeile 6 wird überprüft, ob das Fenster bzw. die Tür zur Wandgeometrie passt. Dazu wird die Fenster- bzw. Tür-Geomtrie auf die Ebene der Wand projiziert und die Geometrien anschließend miteinander geschnitten. Ist die Schnittmenge nicht leer, dann kann die projizierte Fenster- bzw. Tür-Geometrie in die Ergebnismenge aufgenommen werden.

Das Ergebnis des Algorithmus ist in Abbildung 3.24 dargestellt. Auf der linken Seite ist die ursprüngliche komplexe Geometrie von Fenster und Türen dargestellt. Auf der rechten Seite ist die vereinfachte Geometrie ersichtlich. Fenster und Türen werden nach der Vereinfachung nur mehr als Rechtecke im Raum dargestellt, welche jeweils auf die Außenseite der Wand ausgerichtet ist.

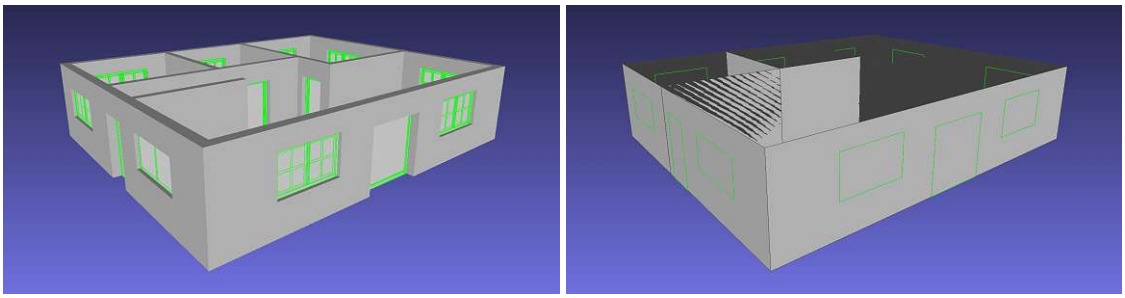

Abbildung 3.24: Komplexe und vereinfachte Geometrie der Fenster und Türen

3.5.10 Export der GA-Objekte

Eine wichtige Eigenschaft von GA-Systemen ist die Darstellung verschiedener *Objekte der Gebäudeautomation* (GA-Objekte), eingebettet in die Gebäude-Geometrie, um den Status

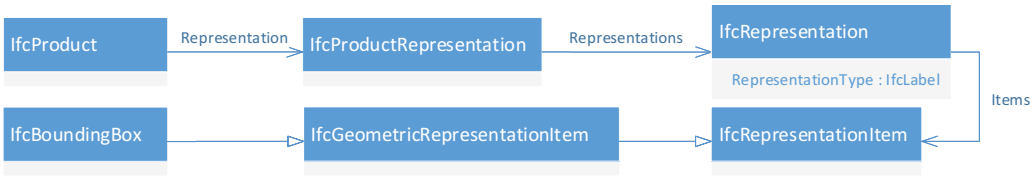

Abbildung 3.25: Verknüpfung zwischen *IfcProduct* und der vereinfachten geometrischen Darstellung *IfcBoundingBox*

der Geräte in Echtzeit zu visualisieren. Zu den GA-Objekten zählen z.B. Heizkörper, Lüftungsschächte, Luftauslässe, Temperatursensoren, Luftgütesensoren u.v.m.. In IFC werden diese Objekte derzeit nur unzureichend unterstützt und es stehen daher nur für die gängigsten Geräte entsprechende IFC-Elemente zur Verfügung. Für Elemente die derzeit noch nicht direkt abgebildet werden können, bietet IFC das Element *IfcBuildingElement-Proxy* an, welches von *IfcBuildingElement* abgeleitet ist und wie ein Bauelement verwendet werden kann. Im Zuge der Implementierung des Prototypen wurden die IFC-Elemente *IfcFlowTerminal*, *IfcDistributionControlElement* und *IfcBuildingElementProxy* verwendet. Der nachfolgend beschriebene Algorithmus unterstützt jedoch sämtliche IFC-Elemente die von *IfcProduct* abgeleitet sind, für die eine *Bounding Box* Darstellung im Modell existiert.

Ähnlich wie bei Fenster und Türen sollen auch die GA-Objekte möglichst einfach dargestellt werden, um die Benutzbarkeit der Software zu verbessern. Die exakte Darstellung der oft komplexen Geometrien ist für viele Anwendungsfälle im Bereich der Gebäudeautomation nicht notwendig. Nach der Ausführung des nachfolgenden Algorithmus werden die GA-Objekte als einfaches Rechteck im Raum dargestellt, welches sich genau in der Mitte der beiden größten, gegenüberliegenden Flächen der orientierten *Bounding Box* befindet. Aus diesem Grund ist die Vorbedingung für die Anwendung des Algorithmus, dass für jedes zu exportierende GA-Objekt eine *Bounding Box* Darstellung im IFC-Modell existiert. In Revit muss beim Export in eine IFC-Datei extra angegeben werden, ob die *Bounding Box* Darstellung für die Objekte exportiert werden soll.

Der Algorithmus ist in Listing 3.12 dargestellt und benötigt als Eingabe eine Liste aller zu exportierenden GA-Objekte. Die GA-Objekte müssen als IFC-Element vorliegen, welches von *IfcProduct* abgeleitet ist. In Zeile 2 werden alle GA-Objekte iteriert und in Zeile 3 für das aktuelle GA-Objekt die *Bounding Box* Darstellung aus dem IFC-Modell ermittelt. In Abbildung 3.25 ist dargestellt, wie ausgehend vom jeweiligen GA-Objekt als Element vom Typ *IfcProduct* zur *Bounding Box* Darstellung navigiert werden kann. Ausgehend vom *IfcProduct*-Element kann über die Beziehung *Representation* zum Element *IfcProductRepresentation* navigiert werden, welches die verschiedenen geometrischen Darstellungsformen des IFC-Elements repräsentiert. Eine geometrische Darstellung wird über das Element *IfcRepresentation* abgebildet und kann von *IfcProductRepresentation* über die Beziehung *Representations* ermittelt werden. Jedes *IfcRepresentation*-Element

besitzt die Eigenschaft *RepresentationType*, welches den Typ der Darstellung in Form eines *IfcLabel* repräsentiert. In unserem Fall ist nur der Typ *BoundingBox* interessant. Ist kein *IfcRepresentation*-Element mit dem Typ *BoundingBox* vorhanden, dann ist die Vorbedingung für den Algorithmus nicht erfüllt. Im Normalfall besitzt das Element *IfcRepresentaion* vom Typ *BoundingBox* nur ein Element *IfcRepresentationItem*, welches über die Beziehung *Items* ermittelt werden kann. Die geometrische Repräsentation besitzt üblicherweise den konkreten Typ *IfcBoundingBox* welcher indirekt über *IfcGeometricRepresentationItem* von *IfcRepresentationItem* abgeleitet ist. Das Element *IfcBoundingBox* enthält alle Eigenschaften, um daraus 6 Polygone zu berechnen, welche die Flächen der *Bounding Box* darstellen. In Zeile 5 wird aus den 6 Polygonen der *Bounding Box* das flächengrößte ermittelt. Anschließend wird in Zeile 6 die größte Fläche ermittelt, welche der zuvor ermittelten Fläche gegenüberliegt, d.h., den inversen Normalvektor dieser Fläche besitzt. In Zeile 7 wird die größte Fläche um die halbe Entfernung in Richtung der gegenüberliegenden Fläche verschoben und anschließend in Zeile 8 zur Ergebnismenge hinzugefügt. Die resultierende Geometrie befindet sich nun genau in der Mitte zwischen den beiden größten, gegenüberliegenden Flächen.

Algorithmus 3.12: ExportiereGAObjekte

Eingabe: GA-Objekte als Liste von IfcProduct-Elementen GA

Ausgabe: Liste der vereinfachten Geometrien der GA-Objekte

1 $E \leftarrow []$

2 **für** *alle GA-Objekte* ga_i *in* GA **tue**

3 $bb \leftarrow$ `ErmittleIfcBoundingBox(`ga_i`)`

4 $P_{bb} \leftarrow$ `ErmittlePolygone(`bb`)`

5 $p_{max} \leftarrow$ Polygon $p \in P_{bb}$ mit maximaler Fläche

6 $p_{opposite} \leftarrow$ Polygon $p \in P_{bb}$ mit $\vec{n} = -n_{\vec{p_{max}}}$ und maximaler Fläche

7 $p_{erg} \leftarrow$ `Translate(`p_{max}, $\frac{n_{\vec{opposite}}}{2}$ $*$ `entf(`p_{max},$p_{opposite}$`))`

8 Füge p_{erg} der Ergebnismenge E hinzu

9 **return** E

10 **Ende**

Das Ergebnis des Algorithmus ist in Abbildung 3.26 dargestellt. Auf der linken Seite ist die ursprüngliche komplexe Geometrie der GA-Objekte dargestellt. Auf der rechten Seite sieht man die vereinfachte Darstellung in Form von Rechtecken im Raum. Im Gegensatz zu Fenster und Türen befinden sich GA-Objekte meist inmitten eines Raumes und können deshalb nicht an anderen Elementen ausgerichtet werden. Die zentrierte Darstellung in Form eines Rechtecks im Raum scheint daher bei GA-Objekten sinnvoller zu sein.

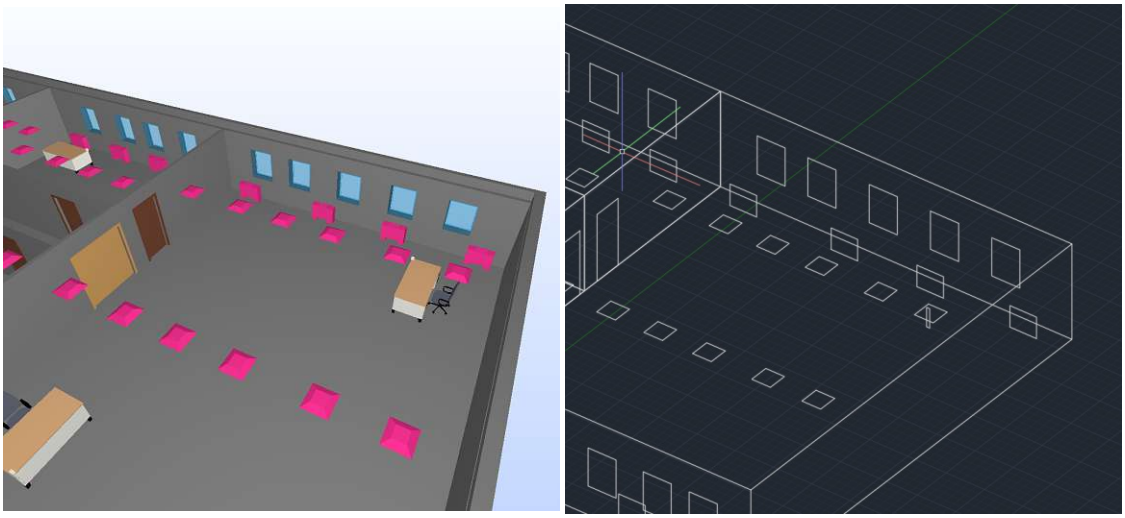

Abbildung 3.26: Komplexe und vereinfachte Geometrie der GA-Objekte

3.5.11 Berechnung der Solids

In den bisherigen Schritten des Algorithmus wurden die *Space Boundaries* der Räume berechnet, die daraus resultierende Geometrie vereinfacht und Lücken zwischen *Boundary Parts* geschlossen. Im Vergleich zur triangulierten Geometrie wurde die Anzahl der Flächen, Kanten und Punkte erheblich reduziert, was eine Weiterverwendung der Geometrie wesentlich vereinfacht. Das Ziel des Algorithmus ist der Export der Geometrie als *Brep Solid Geometry*, damit diese in einem GA-System verwendet werden kann, um die Systeme der Gebäudeautomation im Kontext des Gebäudes visualisieren zu können. Die *Boundary Parts* eines Raumes werden in diesem Schritt zu einem Solid zusammengefügt, sodass jeder Raum als ein eigener Solid repräsentiert wird. Später können Zonen aus mehreren Räumen gebildet werden, indem mittels Booleschen Operationen die Solids der Räume zusammengefasst werden. Als Ergebnis der bisherigen Schritte des Algorithmus liegt die Geometrie der *Boundary Parts* in Form von *Polygon-Meshes* vor. Zusätzlich zur Geometrie steht für jeden Boundary Part die Information zur Verfügung, aus welchem Bauelement im IFC Modell er entstanden ist und zu welchem Raum er gehört. Damit aus der Raumgeometrie Solids berechnet werden können, müssen zuerst die einzelnen Flächen der unterschiedlichen *Boundary Parts* miteinander *vernäht* werden. Dieser Schritt wird *Sewing* genannt. Ziel vom *Sewing* ist es, Flächen die geometrisch aneinandergrenzen, auch topologisch zusammenzufassen. Dadurch werden Punkte und Kanten, die geometrisch zusammenfallen, miteinander verschmolzen. Das *Vernähen* der Geometrie muss mit einer gewissen Toleranz durchgeführt werden, da aneinandergrenzende Flächen, aufgrund von Ungenauigkeiten bei der Berechnung, nicht exakt zusammenfallen. Dadurch wird die Robustheit des Algorithmus weiter erhöht. Nachdem die Flächen eines Raumes mittels

Sewing vernäht wurden, kann aus der geschlossenen Hülle ein Solid berechnet werden.

Algorithmus 3.13: BerechneSolids

Eingabe: Liste R aller Räume mit Bauelement e
Liste B aller Boundary Parts mit Bauelement e, Raum r und der Geometrie g mit Polygon p und den Koordinaten k des Polygons

Ausgabe: Brep Solid Geometry für jeden Raum

```
1  Solids ← []
2  für jedes r_i ∈ R tue
3      B_raum ← {b ∈ B | r = e_i}
4      Faces ← []
5      für jedes b_raum ∈ B_raum tue
6          Wire ← []
7          für j ← 0 bis size(k_raum) - 1 tue
8              Edge ←ErstelleEdge(k_raum[j], k_raum[j + 1])
9              FügeHinzu(Wire, Edge)
10         Ende
11         Face ←ErstelleFace(Wire)
12         FügeHinzu(Faces, Face)
13     Ende
14     Sewed ←Sewing(Faces)
15     Solid ←ErstelleSolid(Sewed)
16     FügeHinzu(Solids,Solid)
17 Ende
18 return Solids
```

Der Algorithmus ist in Listing 3.13 dargestellt. Als Eingabe benötigt der Algorithmus alle Räume mit den dazugehörigen Bauelementen und die Liste aller Boundary Parts inklusive Geometrie. Für jeden Raum wird ein Solid berechnet, deshalb werden in Zeile 3 alle *Boundary Parts* für den Raum der aktuellen Iteration ermittelt. Aus der Geometrie eines *Boundary Parts* wird ein s.g. *Face* erstellt, welches die Fläche des Polygons darstellt. Um ein *Face* zu erstellen, werden die Punkte des Polygons zu *Edges*, also Kanten zusammengefügt und daraus ein *Wire* erstellt. Ein *Wire* stellt in unserem Fall eine geschlossene Kette von Kanten dar, welche anschließend zu einem *Face* zusammengefasst wird. Ziel dieser Schritte ist es, die Topologie der Geometrie von Grund auf neu zu erstellen, um anschließend alle *Faces* eines Raumes in Zeile 14 mittels *Sewing* zu vernähen.

Abschließend wird in Zeile 15 aus der geschlossenen Hülle ein Solid erstellt und in die Ergebnismenge hinzugefügt.

Das Ergebnis des Algorithmus ist in Abbildung 3.27 dargestellt. Es stellt die exportierten Solids des Gebäudes aus Abbildung 3.1 in AutoCAD dar.

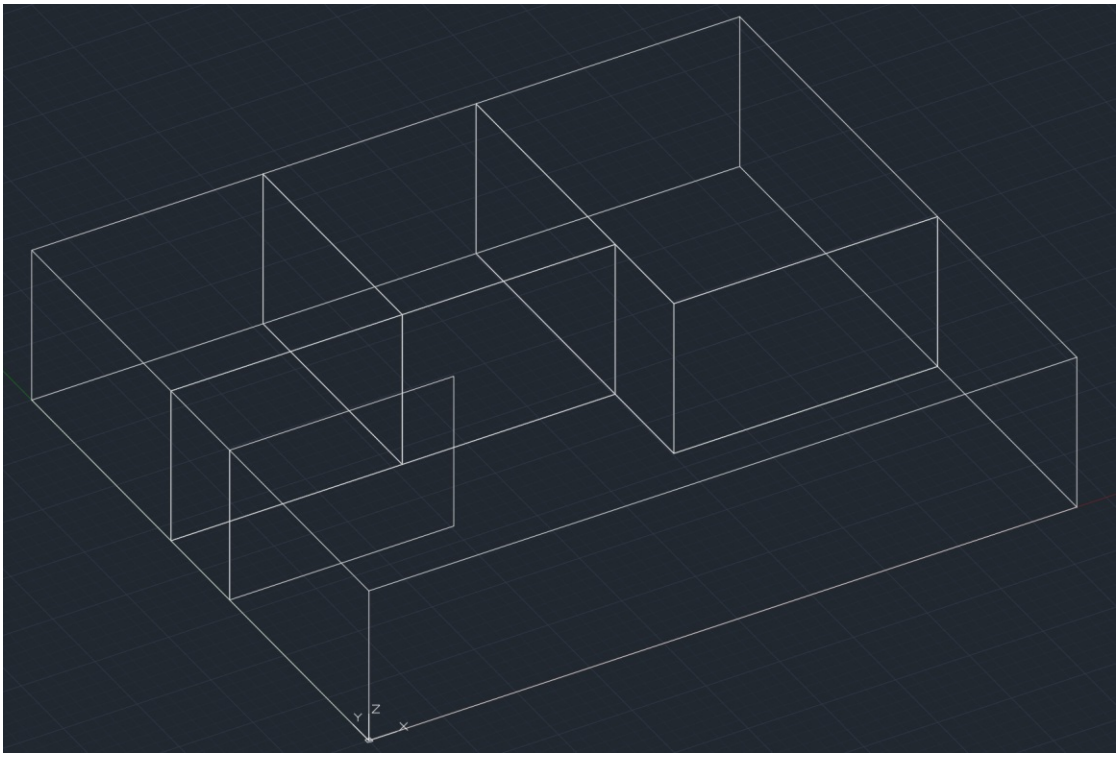

Abbildung 3.27: Ergebnis des Solid Exports dargestellt in AutoCAD 2019

KAPITEL 4

Prototyp

Im Rahmen der vorliegenden Arbeit wurde ein Prototyp entwickelt, welcher die in Abschnitt 3 vorgestellten Algorithmen implementiert, um deren Funktionsweise anhand von realen Daten zu überprüfen. Der Prototyp benötigt als Eingabe eine IFC-Datei mit der Geometrie des Gebäudes, welche anschließend vereinfacht und in Form von IGES- und STEP-Dateien exportiert wird. Das Ergebnis kann anschließend zur Weiterverarbeitung in einem GA-System verwendet werden. Ein Überblick über die Architektur des Prototypen wird in Abbildung 4.1 gezeigt. Die Funktionalität des Prototypen ist in zwei Teile aufgeteilt. Der erste Teil ist in der Abbildung als *Geometrie Exporter* dargestellt und implementiert die Algorithmen aus Abschnitt 3. Lediglich der Solid Export der Geometrie wird getrennt behandelt und ist in der Abbildung als *Solid Geometrie Exporter* dargestellt. Das Ergebnis des *Geometrie Exporter* ist die vereinfachte Geometrie in Form von Polygon-Meshes. Der *Solid Geometrie Exporter* wandelt dieses Zwischenergebnis um, sodass die Geometrie am Ende in Form von *Brep Solid Geometry* vorliegt.

Der erste Teil des Prototypen wurde als Java-Client für die Open Source Software *BIMserver* [BIMa] konzipiert, welcher das IFC Modell und die triangulierte Geometrie für die übergebene IFC Datei zur Verfügung stellt. Damit der Prototyp einfach in verschiedenen Umgebungen verwendet werden kann, wurde für den *BIMserver* ein Docker-Image erstellt, wodurch die Konfiguration der Software vereinheitlicht wird. Docker ermöglicht die Isolierung von Anwendungen mittels Containervirtualisierung und vereinfacht dadurch die Bereitstellung von Software-Komponenten in Form von Images. Die interne Darstellung der Geometrie und die Durchführung geometrischer Operationen erfolgen mithilfe der Bibliothek *Java Topology Suite* (JTS), welche für die Erstellung und Manipulation von Vektorgeometrie konzipiert wurde. Der zweite Teil des Prototypen wurde in C++ implementiert und wird vom *Geometrie Exporter* über das *Java Native Interface* (JNI) aufgerufen. Der Export der Geometrie in Form von Solids wird mithilfe der Bibliothek *Open Cascade* durchgeführt, welche für die Programmiersprachen C++ und Python zur Verfügung steht. Auf die verwendeten Technologien und Bibliotheken

wird in den nachfolgenden Abschnitten noch genauer eingegangen. *BIMserver* und *JTS* wurden bereits in der Arbeit von Ladenhauf [Lad14] verwendet. Die Vorteile dieser beiden Softwarekomponenten treffen auch auf unseren Prototypen zu, sodass diese in einer aktuelleren Version auch in der vorliegenden Arbeit eingesetzt wurden.

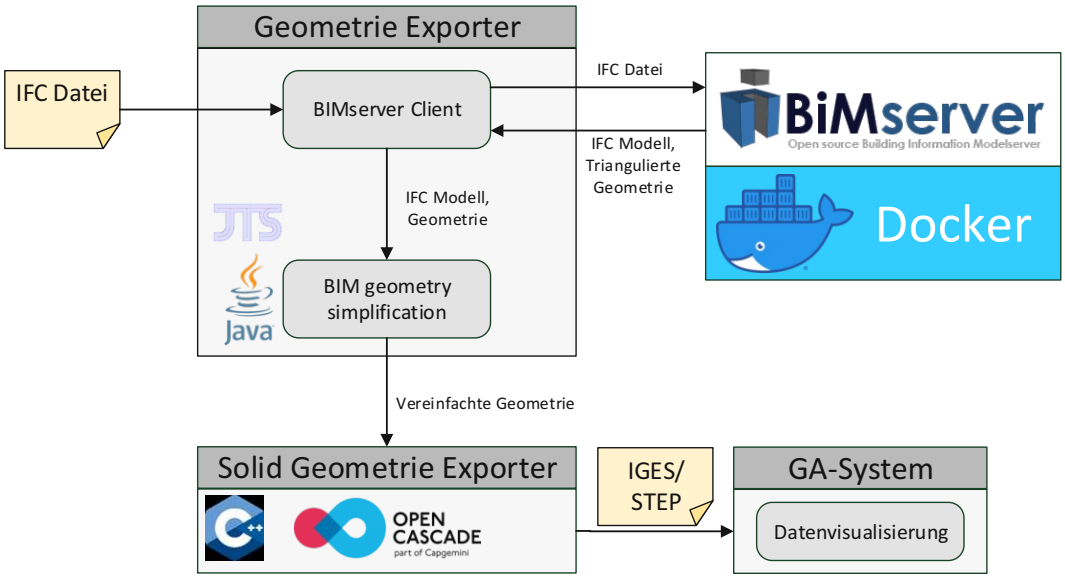

Abbildung 4.1: Architektur des Prototyps

4.1 Technologien

4.1.1 BIMserver

Wie bereits in der Einleitung erwähnt, wurde der Prototyp als Client für die Open Source Software *BIMserver* [BIMa] konzipiert. Es handelt sich dabei um eine Plattform zur Entwicklung von Software im Bereich BIM. Der *BIMserver* ist kein fertiges Endprodukt, sondern bildet vielmehr eine Basiskomponente, um aufbauend darauf Software für Endanwender zu entwickeln. Die Kernfunktionalität des Servers ist die zentrale Speicherung von IFC-Modellen in Form eines Repository. Diese Funktion vereinfacht das gemeinsame Arbeiten an einem Gebäudemodell, da der aktuelle Zustand des Modells für jedermann, zu jeder Zeit zugänglich ist und man bei Aktualisierungen entsprechend informiert werden kann.

Der *BIMserver* wurde in der Programmiersprache Java entwickelt und folgt dem Ansatz einer modellgetriebenen Architektur. Die Basis dieser Architektur bildet das IFC-Metamodell, welches seit der ersten Version von IFC als EXPRESS Schema in Form einer STEP-Datei veröffentlicht wird. IFC-Modelle sind typischerweise objektorientiert aufgebaut. Der *BIMserver* stellt die Entitäten des IFC-Metamodells in Form von typi-

sierten Java-Klassen zur Verfügung. Intern wird das *Eclipse Modeling Framework* (EMF) verwendet, um Java-Klassen aus dem IFC-Schema zu generieren. Die Klassen werden dabei nicht direkt aus dem IFC-Schema generiert, sondern das Schema muss zuvor in ein entsprechendes EMF Schema konvertiert werden. Eine weitere wichtige Eigenschaft des *BIMserver* ist die Möglichkeit, die Funktionalität über Plugins zu erweitern, welche recht einfach über entsprechende Schnittstellen installiert werden können. Die Standardinstallation des Servers enthält nur wenige Funktionen, da selbst die Basiskomponenten über Plugins zur Verfügung gestellt werden, damit die Installation flexibel gestaltet und gezielt an die jeweiligen Anforderungen angepasst werden kann. Die Interaktion mit dem *BIMserver* erfolgt über Schnittstellen, welche plattformübergreifend verwendet werden können. Dadurch ist das Anwendungsgebiet für die Software sehr weitreichend, da keine Einschränkungen bezüglich Programmiersprache und Betriebssystem getroffen werden müssen. Derzeit stehen folgende Kommunikationsmöglichkeiten mit dem *BIMserver* zur Verfügung [BIMb]: SOAP, REST und Protocol Buffers. Es werden dabei verschiedene Services zur Verfügung gestellt, um mit dem Server und den gespeicherten Modellen zu interagieren. Wichtige Funktionen sind z.B. das Hoch- bzw. Herunterladen von IFC-Modellen, das Ausführen von Queries, der Zugriff auf den gesamten IFC-Objektbaum und die triangulierte Geometrie. Zusätzlich zu diesen Funktionen wird auch ein Service zur Administration des Servers zur Verfügung gestellt. Dieses Service ermöglicht u.a. das Anlegen von Benutzern und das Installieren von Plugins. Für die Programmiersprache Java steht ein Client zur Verfügung, welcher den Zugriff auf die SOAP-Schnittstelle des *BIMserver* für die gängigsten Services ermöglicht.

Der Algorithmus aus Abschnitt 3 benötigt als Eingabe die triangulierte Geometrie des Gebäudes, um anschließend die Geometrie vereinfachen zu können. Durch die Verwendung des *BIMserver* muss dieser Schritt nicht selber implementiert werden, da die Triangulation der Geometrie bereits unterstützt wird. Intern verwendet der Server für diese Funktion die Bibliothek *IfcOpenShell* [Ifca]. *IfcOpenShell* wurde in der Programmiersprache C++ implementiert und ist dadurch plattformabhängig. Es werden jedoch entsprechende Bibliotheken für die gängigsten Systeme mitgeliefert, sodass das Endprodukt trotzdem weitestgehend plattformunabhängig verwendbar ist. Zusätzlich zur triangulierten Geometrie stellt der *BIMserver* das IFC Modell als Objektbaum zur Verfügung. Der im Prototypen eingesetzte Java-Client verwendet Java-Klassen, um ein IFC-Modell im Programm darzustellen. Dadurch erspart man sich den Aufwand, um das Objektmodell manuell in eine passende Darstellung für die jeweilige Programmiersprache zu konvertieren.

4.1.2 Docker

BIMserver ist eine Serverkomponente und wird vom *Geometrie Exporter* verwendet. Um die Auslieferung dieser Komponente zu vereinfachen, wird die Virtualisierungslösung *Docker* verwendet. Docker wird primär für die Isolation von Anwendungen eingesetzt und nutzt dafür das Konzept der Containervirtualisierung [Doc]. Container sind im Vergleich zu virtuellen Maschinen effizienter und leichtgewichtiger, da sie sich den Kernel

mit dem Betriebssystem des Host-Systems teilen. Sie beinhalten das Betriebssystem, den Programmcode und evt. zusätzliche Dateien, die für das Ausführen der Software erforderlich sind. Container werden in Form von Images ausgeliefert. Man spricht erst nach dem Starten des Images von einem Container. Images können u.a. mithilfe von s.g. *Dockerfiles* definiert werden. Es handelt sich dabei um eine einfache Textdatei, in der definiert ist, wie die Software auf dem verwendeten Betriebssystem des Basis-Images installiert wird. Jedes Image, außer Basis-Images, werden von anderen Images abgeleitet und erweitern diese. Basis-Images sind üblicherweise Images von Betriebssystemen, welche von keinem anderen Image ableiten und grundlegende Betriebssystemfunktionen zur Verfügung stellen. Abgeleitete Images können zusätzliche Software, Dateien, u.v.m. beinhalten und erweitern damit die Basis-Images. Damit zusätzliche Software installiert bzw. Dateien in das Image kopiert werden können, stehen verschiedene Befehle zur Verfügung, die im *Dockerfile* angegeben und beim Erstellen des Images ausgeführt werden. Es lassen sich damit unter anderem auch gewöhnliche Shell-Befehle ausführen.

Das Docker-Image für den *BIMserver* verwendet als Basis-Image *tomcat:9.0.27-jdk11-openjdk* und beinhaltet einen Tomcat-Server und ein *Java Development Kit* (JDK), welches die Basis für den *BIMserver* bildet. Das Software-Paket des *BIMserver* wird beim Erstellen des Images heruntergeladen und dem Tomcat-Server zur Verfügung gestellt. Das *Dockerfile* für die *BIMserver*-Installation des Prototyps ist in Listing 4.1 dargestellt. Das Basis-Image wird in der ersten Zeile mit dem Schlüsselwort *FROM* definiert. Anschließend werden zur Verbesserung der Lesbarkeit eine Reihe von Environment-Variablen definiert, welche diverse Versionsnummern und Pfade im Dateisystem repräsentieren. Nach der Definition der Variablen folgen die Befehle zum Downloaden und Entpacken des *BIMserver*. Danach wird das Plugin *ifcplugins* installiert, welches die Serializer und Deserializer für verschiedene IFC-Versionen enthält und damit die grundlegenden Funktionen für den Umgang mit IFC-Dateien zur Verfügung stellt. Am Ende des Dockerfiles wird ein s.g. *HEALTHCHECK* definiert, welcher verwendet wird, um zu überprüfen, ob der *BIMserver* erfolgreich hochgefahren ist, sobald das Image gestartet wurde. Im Zuge des Healthchecks wird überprüft, ob die REST-Schnittstelle des *BIMserver* erreichbar ist. Ist das der Fall, dann funktioniert der Server einwandfrei, ansonsten wird beim Starten des Docker-Containers eine Fehlermeldung von Docker ausgegeben.

Listing 4.1: Dockerfile der *BIMserver*-Installation des Prototyps

```
FROM tomcat:9.0.27-jdk11-openjdk

ENV BIMSERVER_VERSION 1.5.162
ENV IFC_PLUGIN_VERSION 0.0.90
ENV TOMCAT_DIRECTORY /usr/local/tomcat
ENV WEBAPPS_DIRECTORY $TOMCAT_DIRECTORY/webapps
ENV LOGS_DIRECTORY $TOMCAT_DIRECTORY/logs
ENV PLUGINS_DIRECTORY $WEBAPPS_DIRECTORY/ROOT/WEB-INF/plugins

RUN rm -rf $WEBAPPS_DIRECTORY/*
RUN curl -L https://github.com/opensourceBIM/BIMserver/releases
   /download/v$BIMSERVER_VERSION/bimserverwar-
   $BIMSERVER_VERSION.war --output $WEBAPPS_DIRECTORY/ROOT.war
RUN unzip $WEBAPPS_DIRECTORY/ROOT.war -d $WEBAPPS_DIRECTORY/
   ROOT
RUN rm -rf $WEBAPPS_DIRECTORY/ROOT.war

# Installing Plugins
RUN mkdir -p $PLUGINS_DIRECTORY

RUN curl -L https://github.com/opensourceBIM/IfcPlugins/archive
   /ifcplugins-$IFC_PLUGIN_VERSION.tar.gz --output
   $PLUGINS_DIRECTORY/ifcplugins-$IFC_PLUGIN_VERSION.tar.gz
RUN tar -xf $PLUGINS_DIRECTORY/ifcplugins-$IFC_PLUGIN_VERSION.
   tar.gz
RUN rm $PLUGINS_DIRECTORY/ifcplugins-$IFC_PLUGIN_VERSION.tar.gz

# Redirect BIM server logs to stdout
RUN ln -sf /dev/stdout $LOGS_DIRECTORY/bimserver.log

HEALTHCHECK --interval=5s --timeout=3s --retries=20 \
    CMD curl --header "Content-Type: application/json" --
       request POST --data '{"request":{"interface":"
       AdminInterface","method":"getVersion"}}' http://
       localhost:8080/json
```

Abgesehen vom *BIMserver* und dem Plugin *ifcplugins* werden noch weitere Einstellungen und Plugins für den Betrieb des Prototypen benötigt. Die Konfiguration des Servers wird von einem zweiten Docker-Container, einem *Provisioner* übernommen. Der *Provisioner* wartet, bis der *BIMserver* erfolgreich gestartet wurde und installiert anschließend über die REST-Schnittstelle weitere Plugins und nimmt verschiedene Konfigurationen vor, u.a. das Einrichten eines Administrator-Kontos, welches für den Aufruf der REST-

Schnittstelle und die Weboberfläche benötigt wird. Für die Orchestrierung mehrerer Docker-Container stellt Docker das Werkzeug *docker-compose* zur Verfügung und erlaubt die Definition von Abhängigkeiten zwischen Containern in Form einer YAML-Datei. Dadurch können mehrere Container mit nur einem Befehl in der richtigen Reihenfolge gestartet und verwaltet werden. Die *docker-compose* Konfiguration für den *BIMserver* und den *Provisioner* ist in Listing 4.2 dargestellt. Im Abschnitt *services* sind die zu startenden Docker-Container angegeben. Die Abhängigkeit zwischen *Provisioner* und *BIMserver* wird über die Einstellung *depends_on* definiert. Der *Provisioner* wird von *docker-compose* erst gestartet, sobald der Healthcheck des *BIMserver* erfolgreich durchgeführt werden konnte. Sobald der *BIMserver* erfolgreich hochgefahren ist, nimmt der *Provisioner* die entsprechenden Konfigurationen vor und beendet sich anschließend wieder. Danach kann der *BIMserver* vom Prototypen verwendet werden.

Listing 4.2: docker-compose.yml der *BIMserver*-Installation des Prototyps

```
version: '3.7'
services:
  open-bimserver:
    image: open-bimserver:rc
    ports:
      - "8080:8080"
    deploy:
      resources:
        limits:
          memory: 4G
  open-bimserver-provisioner:
    image: open-bimserver-provisioner:rc
    depends_on:
      - open-bimserver
```

4.1.3 Spring Boot

Als Basis für den *Geometrie Exporter* wird *Spring Boot* [Spra] verwendet, welches grundlegende Funktionen eines Programms über einfache Konfigurationsmöglichkeiten zur Verfügung stellt. Zu diesen Funktionen gehören z.B. Logging, Configuration Management, Dependency Injection. Dadurch wird die Entwicklung von Programmen wesentlich vereinfacht, da Basisfunktionen einfach eingebunden werden können. Durch den Einsatz von Spring Boot werden auch Versionskonflikte bei Standardbibliotheken weitestgehend verhindert, da kompatible Versionen über Spring verwaltet werden. Zusätzlich bietet das *Spring Framework* [Sprb] einen Dependency-Injection Mechanismus, wodurch Abhängigkeiten zwischen Klassen besser verwaltet werden können und dadurch eine losere Kopplung ermöglicht. Im Gegensatz zum klassischen Ansatz werden Objekte nicht über ihren Konstruktor und der Verwendung des Schlüsselwort *new* direkt erstellt, sondern die Instantiierung der Klassen wird vom *Spring Framework* übernommen. Dadurch wird die Lesbarkeit des Quellcodes und damit die Wartbarkeit wesentlich erhöht.

4.1.4 Java Topology Suite (JTS)

Die Implementierung des Algorithmus zur Vereinfachung der Gebäudegeometrie erfordert die Durchführung von geometrischen Operationen in der Ebene und die interne Speicherung der Geometrie in Form von Polygon-Meshes. Für diese Aufgaben verwendet der Prototyp die Java-Bibliothek JTS [JTS]. JTS ist ein Projekt der Arbeitsgruppe *LocationTech* [Loc] der Eclipse Foundation und frei verfügbar.

JTS stellt u.a. Funktionen für die Erstellung und die Manipulation von Vektorgeometrie zur Verfügung. Wichtige geometrische Operationen, wie z.B. Vereinigung, Durchschnitt und Differenz sind ebenfalls verfügbar. Echte 3D Operationen werden von JTS nicht unterstützt, jedoch erlauben einige Klassen wie z.B. *Coordinate* und *Polyon* die Verwendung von z-Koordinaten, sodass mit 3D-Geometrien zumindest rudimentär gearbeitet werden kann. JTS verwendet für diese Art der 3D Unterstützung den Begriff „2.5D“. Die meisten geometrischen Operationen des Algorithmus werden ohnehin in 2D durchgeführt, sodass diese Einschränkung akzeptabel ist. Einfache 3D Operationen, die von JTS nicht unterstützt werden, werden stattdessen mit den Funktionen des Java Pakets *javax.vecmath* durchgeführt. Eine nützliche Funktion von JTS ist das Konzept des *PrecisionModel*, welches die Robustheit des Prototypen erheblich erhöht. Das *PrecisionModel* ermöglicht die Angabe der Genauigkeit mit der die einzelnen Berechnungen durchgeführt werden, um Probleme mit Rundungsfehlern zu vermeiden. Der Wert für die Genauigkeit kann im Prototypen mittels Parameter verändert werden, da er bei unterschiedlichen Gebäudemodellen u.U. angepasst werden muss. Ein guter Wert für ein gegebenes Modell sollte mittels Tests ermittelt werden.

4.1.5 Open CASCADE

Der *Solid Geometrie Exporter* ist für den Export der vereinfachten Geometrie in Form von *Brep Solid Geometry* verantwortlich und verwendet dafür die Bibliothek *Open CASCADE* [Ope]. Bei Open CASCADE handelt es sich um ein *Software Development Kit* (SDK) zur Entwicklung von Anwendungen in den Bereichen CAD, CAM und CAE. Das SDK stellt Funktionen für die geometrische Modellierung in 2D oder 3D, mit Hauptaugenmerk auf CAD Systeme, zur Verfügung. Dementsprechend stehen dem Entwickler eine Reihe von Datenstrukturen für *Punkte*, *Kanten*, *Wires*, *Faces*, *Solids* uvm. zur Verfügung. Der *Geometrie Exporter* stellt die vereinfachte Geometrie in Form von Polygon-Meshes dem *Solid Geometry Exporter* zur Verfügung. Open CASCADE kann mit Polygon-Meshes umgehen und stellt verschiedene Import- und Export-Funktionen für den Austausch von Geometrien zwischen verschiedenen Formaten zur Verfügung. Das Kernpaket von Open CASCADE unterstützt die herstellerunabhängigen Formate IGES und STEP. Weitere Formate wie z.B. ACIS, DXF oder Parasolid werden durch kostenpflichtige Erweiterungen unterstützt. Zur Umwandlung der Polygon-Meshes in Solids werden Flächen, die geometrisch zusammenfallen mittels *Sewing* auch topologisch zusammengefasst. Das SDK stellt einen entsprechenden Sewing-Algorithmus zur Verfügung und bietet die Möglichkeit, die Geometrien anschließend als Solid in die Formate IGES und STEP zu exportieren. Open CASCADE steht als Bibliothek für die Programmiersprache C++ zur Verfügung.

Aus diesem Grund wurde der Algorithmus für die Umwandlung der Polygon-Meshes in Solids als eigene Bibliothek ausgelagert, welche über das JNI vom *Geometrie Exporter* aufgerufen werden kann.

4.2 Entwicklungsumgebung

Als Entwicklungsumgebung für den Prototypen wurde IntelliJ IDEA 2019 für die Entwicklung des *Geometrie Exporter* und Visual Studio 2017 für den *Solid Geometrie Exporter* verwendet. Durch die Verwendung der beiden Programmiersprachen Java und C++ ist es sinnvoll, die passende Entwicklungsumgebung für die jeweilige Sprache zu verwenden. Der *Geometrie Exporter* verwendet den *BIMserver* für die Triangulation der Geometrie und den Zugriff auf das IFC-Modell. Dadurch ergeben sich eine Reihe von Vorteilen, die bereits in Abschnitt 4.1.1 erwähnt wurden. Der *BIMserver* wurde bereits in der Arbeit von [Lad14] erfolgreich für die Implementierung des GINGER-Algorithmus verwendet, was die Entscheidung zusätzlich unterstützt. Es wäre denkbar den Prototypen in Zukunft weiterzuentwickeln, um die Funktionalität in Form eines *BIMserver*-Plugins zur Verfügung zu stellen. Für die Kommunikation mit dem *BIMserver* über SOAP steht ein Java-Client zur Verfügung, sodass Java als Programmiersprache für den *Geometrie Exporter* als sinnvoll erscheint. Ein weiterer Vorteil von Java ist, dass sehr viele Bibliotheken und Werkzeuge zur Verfügung stehen, welche die Arbeit wesentlich erleichtern. Für die geometrischen Operationen der Algorithmen wird die Java-Bibliothek *JTS* verwendet, welche ebenfalls bereits bei der Implementierung des GINGER-Algorithmus [Lad14] erfolgreich verwendet wurde.

Als Build-Management Werkzeug wurde *Apache Maven 3* [Mav] verwendet. Dadurch lässt sich das Projekt in allen Entwicklungsumgebungen verwenden, welche Maven unterstützen. Sämtliche Projekteinstellungen und Abhängigkeiten können in einer einzigen XML-Datei, unabhängig von der Entwicklungsumgebung, vorgenommen werden. Die getroffenen Einstellungen werden durch die Maven-Integration der gewählten Entwicklungsumgebung automatisch vorgenommen, sodass die individuelle Konfiguration der konkreten Umgebung wegfällt. Ein weiterer Vorteil ist, dass Maven die Bibliotheken, von denen der *Geometrie Exporter* abhängig ist, automatisch aus dem Internet herunterlädt.

Im Gegensatz zum *Geometrie Exporter* wurde der *Solid Geometrie Exporter* in C++ implementiert, da dieser für die Bewältigung seiner Aufgaben das SDK von *Open CASCADE* verwendet, welches ausschließlich für die Programmiersprachen C++ und Python zur Verfügung steht. Die Anbindung des *Solid Geometrie Exporter* an den *Geometrie Exporter* erfolgt über das JNI. JNI ermöglicht den Aufruf von nativen Funktionen in Java. Diese Funktionen müssen entweder in den Programmiersprachen C oder C++ implementiert sein. Java ermöglicht den nativen Funktionen den Zugriff auf die Java Laufzeitumgebung und damit den Zugriff auf entsprechenden Java-Objekte im Speicher. Die Polygon-Meshes des *Geometrie Exporter* können auf diesem Weg dem *Solid Geometrie Exporter* übermittelt werden, welcher anschließend die Java-Objekte in entsprechende C++ Strukturen umwandelt und den Solid Export über *Open CASCADE* durchführt.

Während der Implementierung des Prototypen war es immer wieder notwendig, die einzelnen Zwischenergebnisse zu analysieren, um die Geometrie auf ihre Richtigkeit überprüfen zu können. Die Polygon-Meshes werden deshalb nach jedem Zwischenschritt exportiert. Aufgrund seiner Einfachheit wurde das Format *Wavefront OBJ* gewählt. Es handelt sich dabei um ein offenes Dateiformat, welches für das Speichern von geometrischen Formen im Textformat verwendet wird. Die Ergebnisse wurden anschließend mit dem Programm *MeshLab* analysiert, welches sich für die Darstellung der Polygon-Meshes sehr gut eignet. Für weitere Analysen und die Erstellung der geometrischen Abbildungen dieser Arbeit wurde das Programm *Maple* verwendet.

KAPITEL 5

Evaluierung

Im Zuge dieser Arbeit wurde ein Prototyp entwickelt, welcher den Ansatz zur Vereinfachung der Gebäudegeometrie implementiert. Dieser Ansatz wurde anhand von ausgewählten Gebäudemodellen evaluiert, um zu zeigen, dass der Algorithmus auch unter realen Bedingungen funktioniert. Am Beginn dieses Kapitels wird ein Gebäudemodell vorgestellt, welches die Implementierung des Prototypen von Anfang an begleitet hat. Die Vereinfachung der Geometrie dieses Gebäudes war das primäre Ziel dieser Arbeit. Anschließend wurden ausgehend von diesem Gebäudemodell weitere Modelle abgeleitet und für die Evaluierung des Algorithmus herangezogen, um zu überprüfen, ob der Ansatz auch bei anderen Modellen funktioniert. Zusätzlich zu diesen Modellen wurden auch öffentlich zugängliche Modelle aus dem Internet analysiert, um einerseits weitere Modelle zum Testen zur Verfügung zu haben und andererseits, um gängige Probleme in der Modellierung aufzuzeigen. Dadurch ist es möglich, die Grenzen des Algorithmus aufzuzeigen und Voraussetzungen zu definieren, welche die verwendeten Gebäudemodelle erfüllen müssen. Am Ende des Kapitels werden noch einige Punkte zur Robustheit des Algorithmus zusammengefasst, die sich im Zuge der Evaluierung ergeben haben.

5.1 Gebäudemodelle

Im Rahmen der Arbeit von [MSP+19] wurde ein Workflow-Modell entwickelt, um ausgehend von einem BIM-Modell ein integriertes Gebäudemodell zu entwickeln, welches die statischen Daten des Gebäudes mit den dynamischen Daten eines GA-Systems vereint. Zur Evaluierung dieses Workflow-Modells wurde ein reales Bürogebäude herangezogen, welches sich in einem Vorort von Wien befindet. Wie bereits erwähnt, ist es in GA-Systemen vorteilhaft, die Geometrie des Gebäudes zu vereinfachen, um die Benutzbarkeit für den Endanwender zu erhöhen. In der Arbeit von [MSP+19] wurde die Geometrie manuell vereinfacht, indem das integrierte Gebäudemodell von Grund auf neu entworfen wurde. Das primäre Ziel der vorliegenden Arbeit war es, diesen Schritt zu automatisie-

ren. Deshalb wurde im Zuge der Entwicklung des Prototypen dasselbe Gebäudemodell herangezogen, um die einzelnen Implementierungsschritte laufend zu überprüfen. Es handelt sich dabei um ein L-förmiges, 5-stöckiges Bürogebäude, welches sich im Süden von Wien befindet. Das Gebäude ist ein typischer Vertreter für die Bauweise von Bürogebäuden in den 70er und 80er Jahren in Westeuropa. Für die Evaluierung des Algorithmus wurde das 2. Obergeschoss des Gebäudes verwendet, da dieses Geschoss im Zuge der Arbeit [MSP+19] vollständig in Form eines BIM-Modells rekonstruiert wurde. Das 2. Obergeschoss des Gebäudes ist in Abbildung 5.1 dargestellt. Das Geschoss besteht aus insgesamt 26 Räumen, welche über 3 verschiedene Gänge erreichbar sind. Das Stiegenhaus wurde aus Gründen der Einfachheit weggelassen und befindet sich im Anschluss zur Doppelflügeltüre, welche sich in der schrägen Wand befindet und in Abbildung 5.1 in braun dargestellt ist. Neben Fenstern und Türen befinden sich auch GA-Objekte im Gebäude, welche in Abbildung 5.1 in blau dargestellt sind.

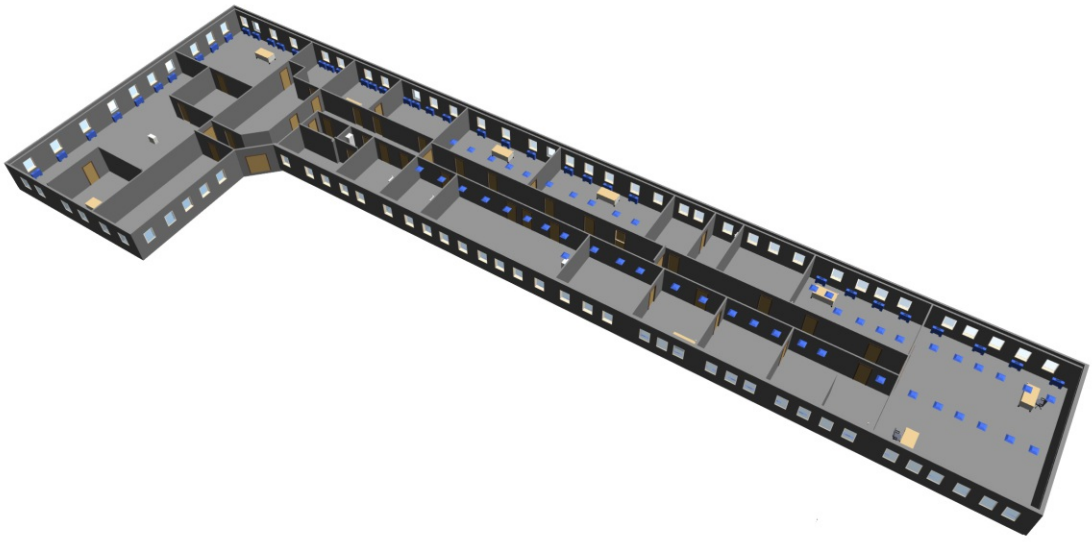

Abbildung 5.1: 2. Obergeschoss des verwendeten Gebäudemodells

Zusätzlich zu diesem Gebäudemodell wurden weitere Varianten des Modells angefertigt, um einerseits verschiedene Konstellationen von Räumen zu testen und andererseits die Robustheit des Algorithmus zu überprüfen. Die drei Varianten des Modells sind in den Abbildungen 5.2, 5.3 und 5.4 dargestellt. Bei diesen Varianten wurden vermehrt schräge Wände verwendet, um speziell die Robustheit des Prototypen zu testen. Schräge Wände verursachen häufig Rechenfehler, da die Normalvektoren dieser Wände meist viele Nachkommastellen besitzen, wodurch bei den einzelnen Operationen oft Rundungsfehler entstehen, die sich über die verschiedenen Schritte des Algorithmus fortpflanzen. Im Modell 5.3 wurden zusätzlich Innenwände mit unterschiedlichen Wandstärken verwendet, was in der Realität sehr häufig vorkommt. Das Modell 5.4 enthält Räume, die von anderen Räumen umschlossen sind. Dieser Fall kommt häufig in Großraumbüros vor,

wo verschiedene allgemeine Räumlichkeiten im Kern eines Stockwerks angesiedelt sind. Zusätzlich zu diesen Modellen wurden aus dem Internet zwei weitere, frei zugängliche Modelle analysiert. Diese Modelle enthalten jedoch einige Schwächen in der Modellierung, welche vom Algorithmus nicht unterstützt werden. Auf diese Einschränkungen wird in Abschnitt 5.2 im Detail eingegangen. Die beiden Modelle sind in Abbildung 5.5 und 5.6 dargestellt. Beim ersten Modell handelt es sich um ein L-förmiges, 4-stöckiges Bürogebäude. Für die Analyse des Algorithmus wurde das Erdgeschoss herangezogen. Eine Besonderheit dieses Modells ist die komplexe Struktur der Außen- und Innenwände, die für den Algorithmus eine besondere Herausforderung darstellt. Viele Wände in diesem Modell haben Ausbuchtungen wie in den Abbildungen 5.12, 5.13 und 5.14 ersichtlich ist. Manche Wände setzen sich aus mehreren einfachen *IfcWall*-Elementen zusammen, welche eine unterschiedliche Dicke besitzen. Andere setzen sich nur aus einem *IfcWall*-Element zusammen, besitzen aber dennoch eine unterschiedliche Dicke an verschiedenen Stellen entlang der Wand. Das Modell enthält einige Features, die für die Verwendung des GINGER-Algorithmus eine Herausforderung sind. Die Vereinfachung der Geometrie war daher bei diesem Modell leider nicht möglich. Beim zweiten Modell handelt es sich um ein 2-stöckiges Gebäude, welches vermutlich als Klinik genutzt wird. Die Besonderheiten dieses Modells sind die Verwendung von *IfcCurtainWall*-Elementen und die materielle Struktur von Innenwänden, welche gesondert behandelt werden müssen, damit der GINGER-Algorithmus angewandt werden kann.

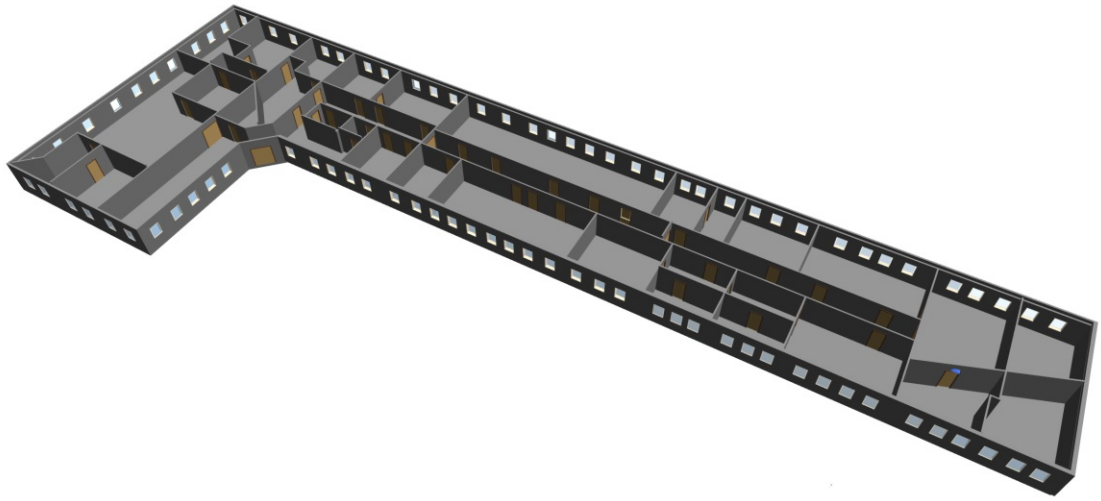

Abbildung 5.2: Variante 2 des Gebäudemodells

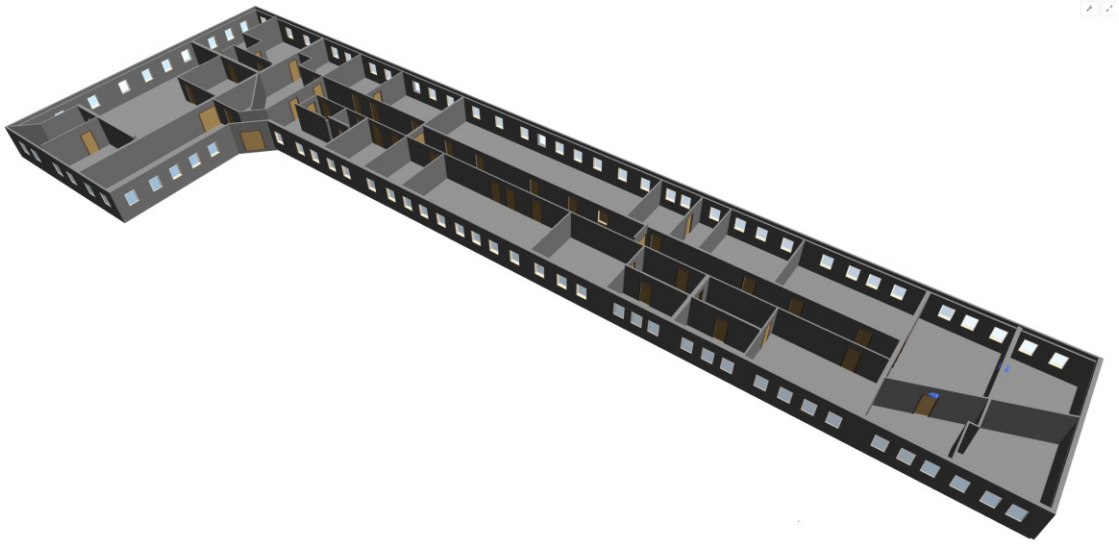

Abbildung 5.3: Variante 3 des Gebäudemodells

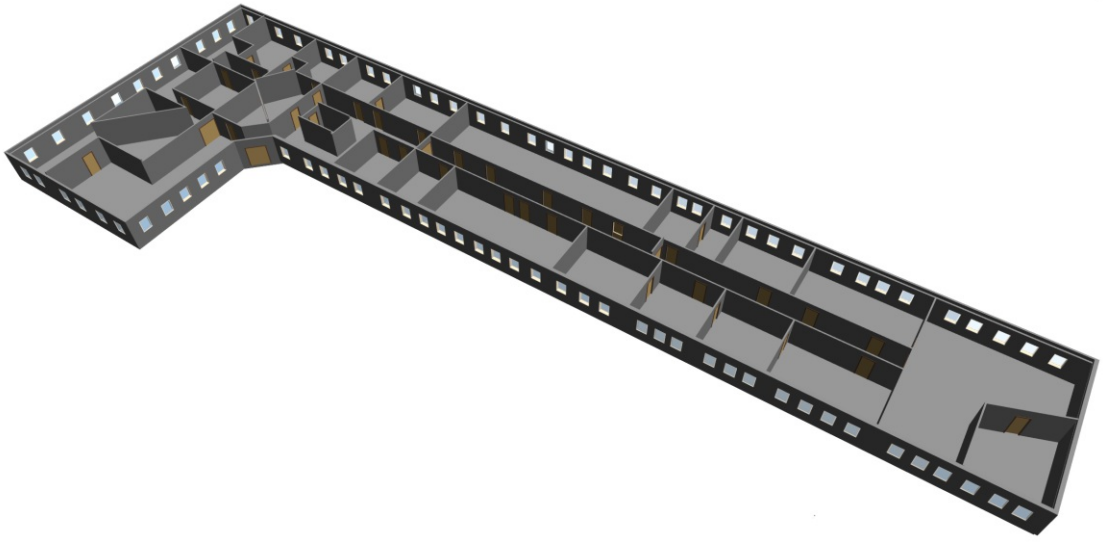

Abbildung 5.4: Variante 4 des Gebäudemodells

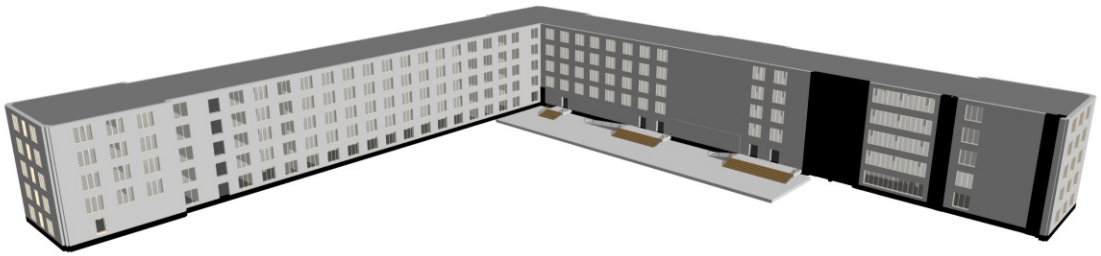

Abbildung 5.5: Gebäudemodell *Office Building* [Off]

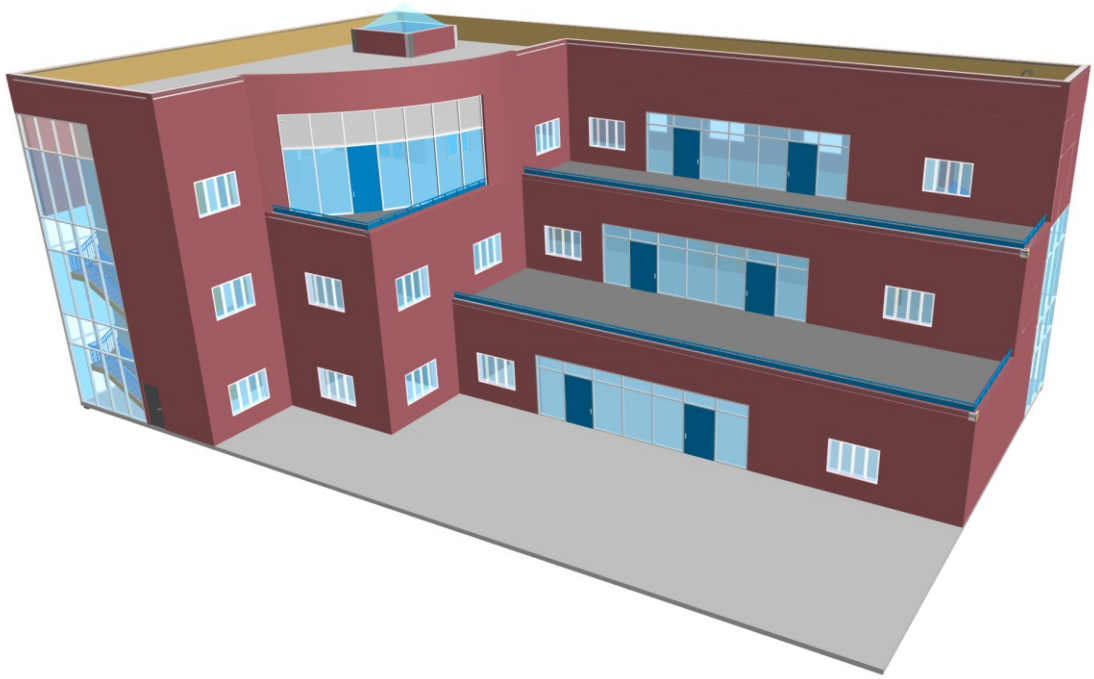

Abbildung 5.6: Gebäudemodell *RNK Architectural* [RNK]

5.2 Evaluierung der Testmodelle

Im vorigen Abschnitt wurden mehrere Gebäudemodelle vorgestellt, welche für die Evaluierung des Algorithmus herangezogen wurden. In erster Linie wurde das Modell aus Abbildung 5.1 für die Implementierung des Prototypen verwendet. Im Zuge der Evaluierung wurde die vereinfachte Geometrie visuell beurteilt. Dazu wurde die exportierte Geometrie im STEP bzw. IGES Format in AutoCAD importiert und die Genauigkeit durch Verwendung der Operation *Solid Union* überprüft. Anschließend wurde die vereinfachte Geometrie als Input für den Space Modeler [Spab] verwendet, welcher bereits in der Arbeit von [MSP+19] eingesetzt wurde. Der Space Modeler bietet gewisse Funktionen eines GA-Systems, u.a. ermöglicht er die Berechnung und Darstellung von räumlichen Beziehungen des Gebäudes und die Visualisierung der Temperatursensoren zusammen mit den gemessenen Werten im Kontext des Gebäudes. Im Zuge der Arbeit von [MSP+19] wurde der Space Modeler erweitert und erlaubt mittlerweile die Darstellung von Sensorwerten der Gebäudeautomation in Echtzeit, direkt im Gebäudemodell. Damit die Darstellung dieser Werte funktioniert, muss das vereinfachte Gebäudemodell mit semantischen Informationen zu den GA-Objekten angereichert werden. Dieser Teil der Integration wurde bisher noch nicht automatisiert und ist derzeit nur manuell möglich.

Einige Teile des Algorithmus erfordern die Definition einer Schranke, um die maximale Entfernung zweier Objekte zueinander zu begrenzen. Die Wahl dieser Schranke stellt eine gewisse Herausforderung dar. Sie darf einerseits nicht zu groß gewählt werden, da ansonsten z.B. beim Füllen der Lücken, Wände miteinander verbunden werden würden, die in der Realität nicht aneinandergrenzen. Ist die Schranke jedoch zu klein gewählt, dann würden diese Wände nicht verbunden werden und der Export als *Brep Solid Geometry* schlägt fehl. Aus diesem Grund macht es Sinn, dass man das Gebäudemodell im Vorfeld analysiert, um gute Werte für die Parameter des Algorithmus zu finden. Manche Programme ermöglichen es, die Abstände zwischen Objekten in einem IFC-Modell zu messen. Das erleichtert das Herausfinden der richtigen Parameter für das jeweilige Gebäude. Abgesehen von den vorgesehen Schranken, welche über Parameter des Prototypen angegeben werden können, muss der Algorithmus auch mit Ungenauigkeiten umgehen können, die aufgrund der Numerik entstehen. Wie bereits in Abschnitt 4.1.4 erwähnt, wird ein *PrecisionModel* verwendet, um die Genauigkeit der Geometrien zu reduzieren und die Genauigkeit der Berechnungen zu steuern. Abgesehen vom *PrecisionModel* müssen manche Vergleiche mit einer gewissen Ungenauigkeit durchgeführt werden. Ein Beispiel hierfür ist der Vergleich zweier Normalvektoren, die aufgrund von Rundungsfehlern unterschiedlich sein können, obwohl sie im ursprünglichen Modell dieselbe Orientierung aufweisen.

Die vereinfachte Geometrie des Geschosses aus Abbildung 5.1 ist in Abbildung 5.7 dargestellt. Die Räume werden in Form von *Brep Solid Geometry* exportiert und sind in der Abbildung transparent in gelb dargestellt. Fenster und Türen werden im Zuge des Algorithmus so vereinfacht, dass sie nur mehr aus Rechtecken im Raum bestehen. Diese Rechtecke werden an den berechneten Boundaries des Raums ausgerichtet und sind in der Abbildung in weiß dargestellt. Die GA-Objekte werden ebenfalls als einfache Rechtecke exportiert und sind in der Abbildung dunkelgelb gekennzeichnet. In Abbildung 5.8 ist

dasselbe Geschoss im Space Modeler dargestellt. Zusätzlich werden die Daten der Temperatursensoren visualisiert. Dieses Modell stellte bezüglich Robustheit des Algorithmus die geringste Herausforderung dar, da die meisten Wände entlang einer Koordinatenachse angeordnet sind und dadurch nur mit wenigen Rundungsfehlern gerechnet werden muss.

In Abbildung 5.9 ist die vereinfachte Geometrie der Varianten 2 und 3 des Gebäudemodells dargestellt. Wie bereits erwähnt unterscheiden sich die beiden Varianten durch unterschiedliche Wandstärken der Innenwände und die häufigere Verwendung von schrägen Wänden, wodurch die Robustheit des Algorithmus getestet werden konnte. Bei schrägen Wänden treten, vorwiegend durch die Projektion der Koordinaten in 2D und die anschließende Rück-Projektion in 3D, Rundungsfehler auf, da die Normalvektoren bei solchen Geometrien sehr viele Nachkommastellen besitzen, obwohl die Genauigkeit der Geometrien am Beginn über das *PrecisionModel* reduziert wird. Diese Probleme lassen sich lösen, indem bei diversen Vergleichen im Algorithmus eine gewisse Toleranz berücksichtigt wird. Der Umgang mit ungenauen Berechnungen bei Verwendung von Gleitkommazahlen wird im Abschnitt 5.3 genauer behandelt.

Die vereinfachte Geometrie der Variante 4 ist in Abbildung 5.10 dargestellt. In Abbildung 5.4 ist die ursprüngliche Geometrie abgebildet. Es ist deutlich zu erkennen, dass die Geometrie zweier Räume nicht vereinfacht werden konnte. Das Problem besteht bei Räumen, die andere Räume vollständig einschließen. Dadurch entstehen in der Geometrie Polygone, welche nicht nur aus einer Hülle (engl. Shell), sondern auch aus Löchern (engl. Holes) bestehen. Aus Gründen der Einfachheit werden Löcher in Polygonen derzeit vom *Solid Geometry Exporter* nicht unterstützt. Um solche Geometrien vereinfachen zu können, müssen Löcher lediglich aus den Solids, die aus der Polygon-Hülle erstellt werden, subtrahiert werden. JTS unterstützt Operationen mit Polygonen, welche sowohl eine Hülle, als auch Löcher besitzen, bereits vollständig.

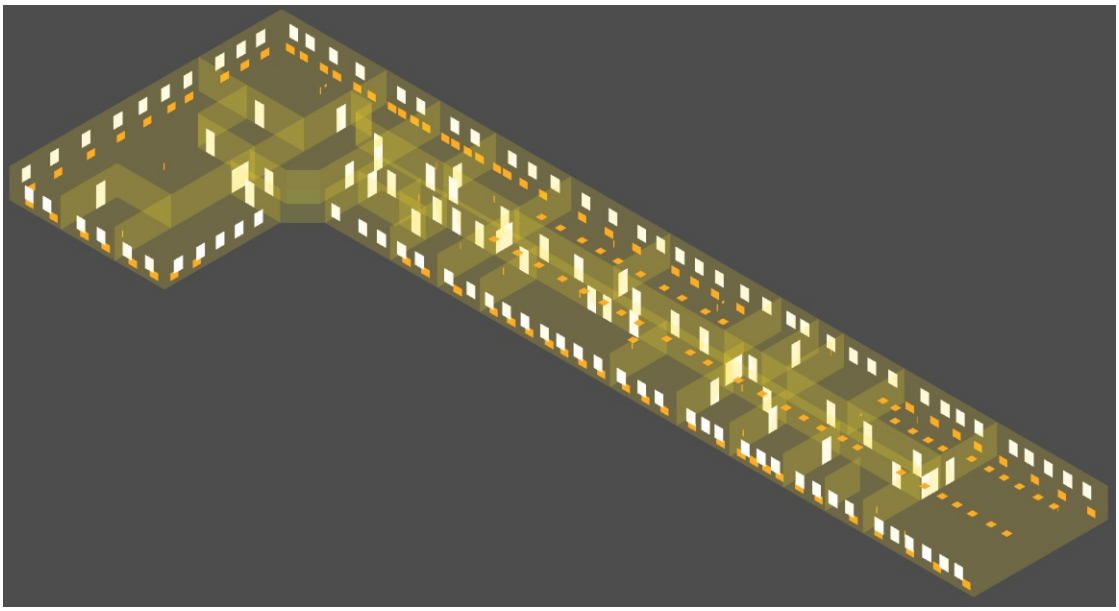

Abbildung 5.7: Vereinfachte Geometrie des 2. Obergeschosses

Abbildung 5.8: Visualisierung von Temperatursensor-Daten im Space Modeler

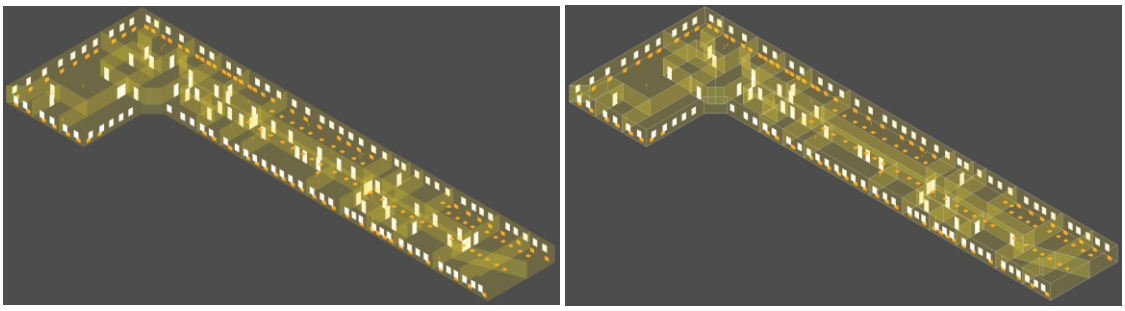

Abbildung 5.9: Vereinfachte Geometrie der Varianten 2 und 3 des Gebäudemodells

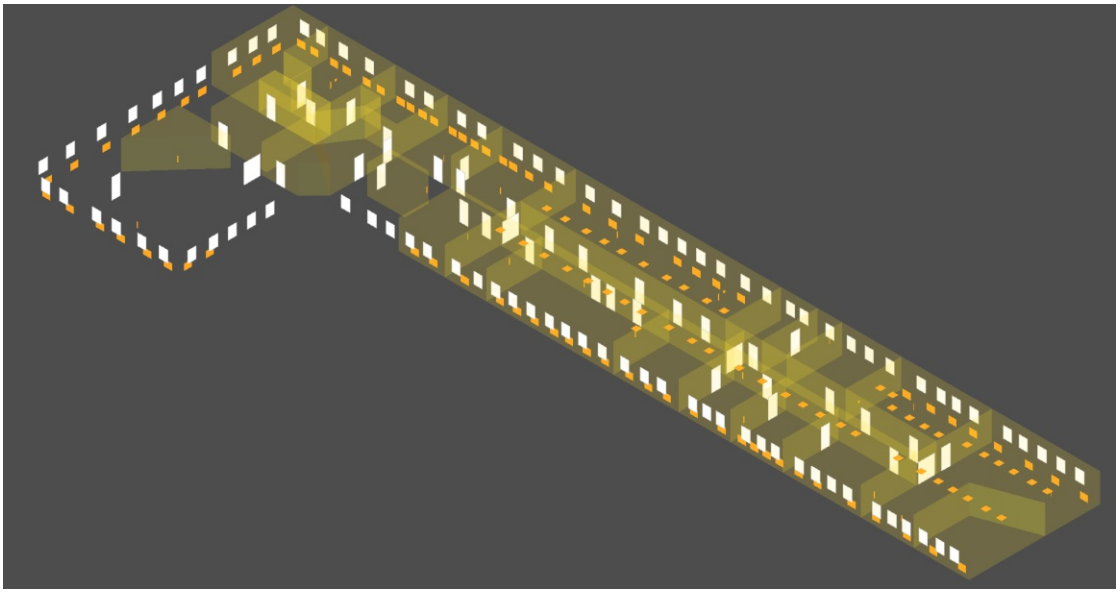

Abbildung 5.10: Variante 4 des Gebäudemodells

5.2.1 Office Building

Zusätzlich zu den bisher genannten Varianten des Gebäudemodells wurden auch zwei weitere IFC-Modelle aus dem Internet analysiert. Eines dieser Modelle ist in Abbildung 5.5 dargestellt. Dieses Modell bereitet einige Probleme, da einerseits Schwächen in der Modellierung vorhanden sind, wodurch gewisse Vorbedingungen des Algorithmus nicht erfüllt werden und andererseits enthält das Modell komplexe Wand-Strukturen, welche die Verarbeitung zusätzlich erschweren. Durch die komplexe Struktur der Wände ist es schwierig, geeignete Schranken für die max. Entfernung der zu verbindenden Wände zu definieren, da diese teilweise aufgrund ihrer Struktur weit auseinanderliegen. Eine Vorbedingung des GINGER-Algorithmus ist, dass sämtliches Volumen des Gebäudemodells entweder von einem Bauelement oder einem Raum eingenommen werden muss. Es darf also kein Volumen existieren, welches keinem IFC-Element vom Typ *IfcSpace* oder einem der unterstützten Bauelemente zugeordnet werden kann. Auf der linken Seite in Abbildung 5.11 ist ein nicht genutzter Raum dargestellt, in dem sich eine Säule befindet. Dieser ungenutzte Raum ist aber keinem *IfcSpace*-Element zugeordnet, wodurch die bereits erwähnte Vorbedingung des GINGER-Algorithmus nicht erfüllt ist. Dadurch können die *Space Boundaries* nicht richtig berechnet werden, da diese immer ausgehend von der Raumgeometrie ermittelt werden. Auf der rechten Seite der Abbildung ist eine Toilette dargestellt. Der gesamte Raum inklusive Kabinen wurde nur mit einem *IfcSpace*-Element modelliert und schließt somit auch die Trennwände der Kabinen mit ein. Korrekterweise müsste für jede Kabine und den Waschraum davor, je ein eigener Raum definiert werden, damit die Berechnung der *Space Boundaries* richtig funktioniert.

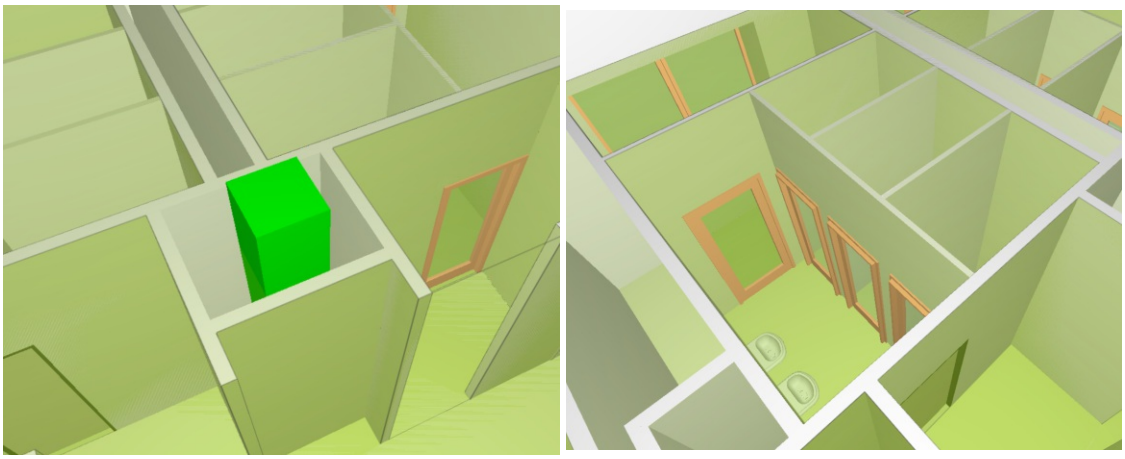

Abbildung 5.11: Fehlende oder fehlerhafte *IfcSpace* Definitionen im Modell *Office Building*

In Abbildung 5.12 ist eine komplexe Außenwand dargestellt, welche auf mehrere *IfcWall*-Elemente aufgesplittet ist. Auf der rechten Seite ist die Wand schematisch dargestellt. Die einzelnen *IfcWall*-Elemente sind durchnummeriert und durch die dunkelgrauen Linien begrenzt. Die komplexe Struktur der Außenwand kann beim Erkennen der *Boundary Parts* bzw. in weiterer Folge beim Füllen der Lücken zu Problemen führen. Die Geometrie des

Raums ist teilweise sehr weit vom jeweiligen Wandelement entfernt. In der schematischen Darstellung ist dieser Umstand durch den punktierten roten Pfeil dargestellt und stellt die Entfernung zwischen Raumgeometrie und Wandelement dar. Die Entfernung entspricht in der Realität knapp einem Meter und ist deshalb zu hoch, um als obere Schranke für den Algorithmus verwendet zu werden, da bei so einem hohen Wert u.U. zu viele *Boundary Parts* ermittelt werden würden, welche in der Realität nicht zum jeweiligen Raum gehören. Durch die große Entfernung fehlt nach dem ersten Schritt des GINGER-Algorithmus der rot markierte *Boundary Part*. In weiterer Folge steigt dadurch die Distanz der *Boundary Parts*, welche beim Füllen der Lücken verbunden werden sollten. Auch hier sollte keine zu hohe Schranke für das Füllen der Lücken verwendet werden. Wie bereits in Abschnitt 3.5.7 erwähnt, könnten die benachbarten Wände aus dem IFC-Modell ermittelt werden, sofern die Daten korrekt exportiert wurden. Leider führt diese Vorgehensweise in unserem Beispiel auch nicht zum Ziel, da die beiden Wände 2 und 5 in der Realität nur indirekt über die Wände 3 und 4 miteinander verbunden sind. Aufgrund der Struktur der gesplitteten Wandelemente wird aber für die Wände 3 und 4 kein einziger *Boundary Part* erzeugt. Dieses Beispiel zeigt die Grenzen des Algorithmus sehr gut auf und verdeutlicht die Wichtigkeit gut gewählter Schranken als Parameter für den Algorithmus. Aufgrund der Komplexität mancher Geometrien reicht ein einfache Schranke u.U. nicht aus. Es wäre jedoch denkbar in Zukunft weitere Heuristiken zu entwickeln, welche bei komplexeren Geometrien bessere Ergebnisse liefern.

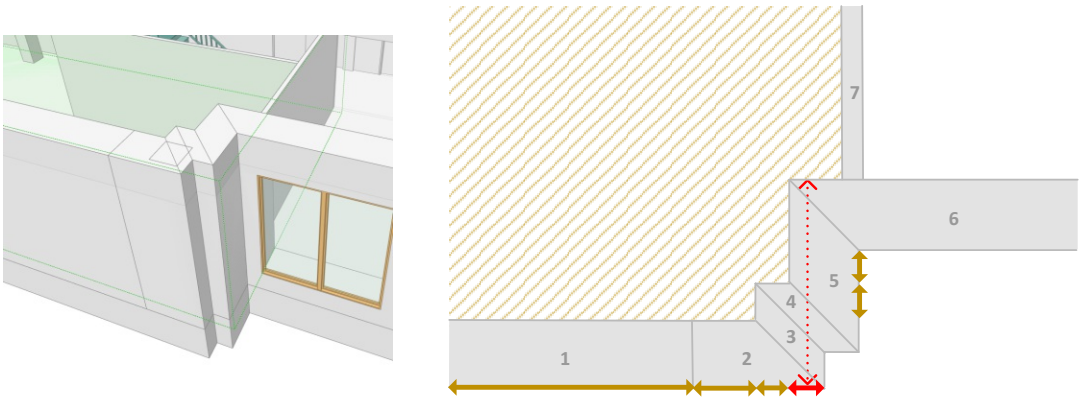

Abbildung 5.12: Komplexe Außenwand, die aus mehreren *IfcWall*-Elementen besteht

In Abbildung 5.13 ist eine Ausbuchtung einer Innenwand dargestellt. Auf der linken Seite ist die Original-Geometrie zu sehen und auf der rechten Seite die berechneten *Boundary Parts*. Da die Ausbuchtung der Innenwand auf mehrere *IfcWall*-Elemente aufgesplittet ist, wird der rot eingerahmte *Boundary Part* erkannt, welcher in weiterer Folge zu Problemen führt, da sich dort keine Raumbegrenzung befinden sollte. Um solche Fehler zu vermeiden, müsste die Erkennung der *Boundary Parts* angepasst werden, damit *Boundary Parts*

die direkt an eine weitere Wand grenzen, ignoriert werden. In diesem Beispiel ist es wichtig, keine zu großen Schranken für den Algorithmus zu wählen, da bei solchen feinen Strukturen ansonsten die falschen *Boundary Parts* miteinander verbunden werden.

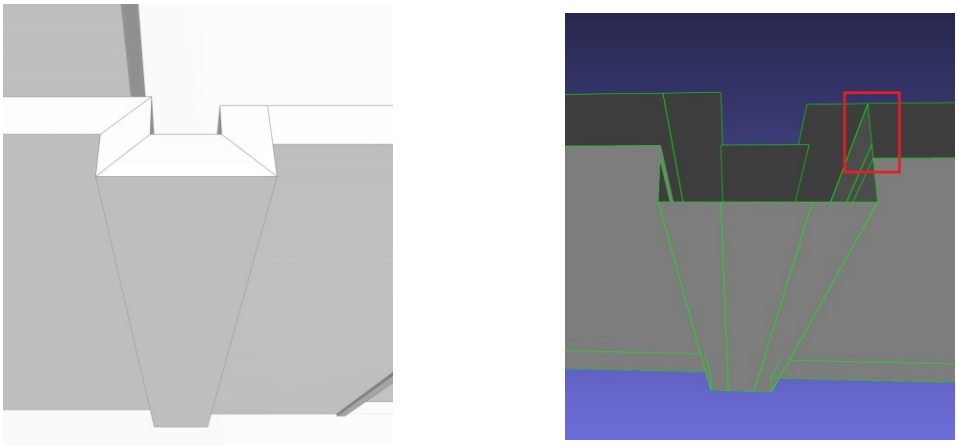

Abbildung 5.13: Ausbuchtung einer Innenwand. Links die Originalgeometrie im IFC-Modell. Rechts die vereinfachte Geometrie nach Anwendung des Algorithmus.

Abgesehen von komplexen Wänden, die auf mehrere *IfcWall*-Elemente aufgesplittet sind, beinhaltet des IFC-Modell auch komplexe Wände, die nur aus einem Element bestehen. In Abbildung 5.14 ist so eine Wand dargestellt. Diese Wandstrukturen führen beim Füllen der Lücken zu Problemen, da in solchen Fällen *Boundary Parts* vom selben Bauelement miteinander verbunden werden müssten, was derzeit vom GINGER-Algorithmus nicht vorgesehen ist.

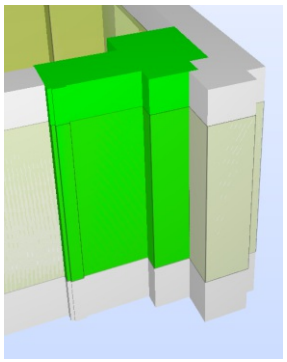

Abbildung 5.14: Komplexe Wandstruktur, die nur aus einem *IfcWall*-Element besteht

Das Modell *Office Building* weist einerseits Schwächen in der Modellierung auf und eignet sich andererseits sehr gut, um die Grenzen des Algorithmus, vor allem im Umgang mit komplexen Geometrien, aufzuzeigen. Die bisher genannten Probleme bei der Verarbeitung des Modells stellen lediglich einen Auszug dar, da die Analyse abseits der aufgelisteten Probleme nicht tiefer gehend verfolgt wurde.

5.2.2 RNK Architectural

In Abbildung 5.6 ist das Gebäudemodell *RNK Architectural* abgebildet. Dieses Gebäude ist das zweite Modell, welches für die Analyse des Prototypen aus dem Internet herangezogen wurde. Eine Besonderheit des Modells ist die Angabe der Längen in der Einheit Fuß. Es handelt sich dabei nicht um eine Standardeinheit in IFC, sondern um eine *IfcConversionBasedUnit*. Einige Schranken im Algorithmus werden in Zentimeter bzw. Millimeter angegeben, deshalb ist es notwendig entsprechende Umrechnungen durchzuführen. Der dazu nötige Umrechnungsfaktor kann über die Entität *IfcConversionBasedUnit* im IFC-Modell ermittelt werden. Wie im *Office Building* Modell sind auch in diesem Modell fehlerhafte Raum-Definitionen vorhanden. In Abbildung 5.15 sind Schächte dargestellt, wo die Raum-Definition fehlt, sodass die Vorbedingung des GINGER-Algorithmus nicht erfüllt ist. Ein Schacht, wie er auf der linken Seite der Abbildung zu sehen ist, befindet sich meist zwischen zwei Räumen im Modell. Beim Schacht auf der rechten Seite handelt es sich vermutlich um den Aufzugschacht im Stiegenhaus. Auch das Volumen innerhalb eines Schachts muss mittels *IfcSpace*-Element definiert sein, damit der Algorithmus angewandt werden kann.

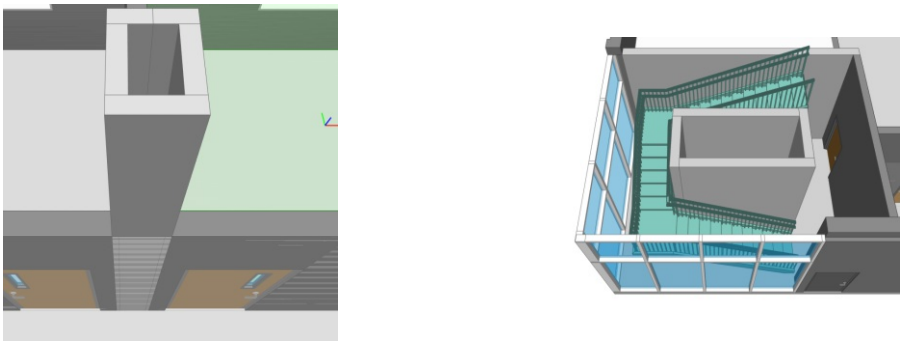

Abbildung 5.15: Schächte mit fehlenden Raum-Definitionen

Glasfronten werden im Modell in Form von *IfcCurtainWall*-Elementen dargestellt. In Abbildung 5.16 ist so eine Glasfront zu sehen. *IfcCurtainWall* wird für komplexe Außenwände verwendet, welche sich aus verschiedenen Komponenten zusammensetzen. Derzeit unterstützt der Algorithmus keine Außenwände, die als *IfcCurtainWall* definiert sind. Da dieses Element von *IfcBuildingElement* abgeleitet ist, sollte eine entsprechende Erweiterung des Algorithmus prinzipiell kein Problem darstellen.

Die Innenwände des Modells bestehen aus verschiedenen Materialschichten, welche sich auch in der triangulierten Geometrie widerspiegeln. Die Geometrie einer solchen Wand ist in Abbildung 5.17 dargestellt. Materialschichten bereiten dem Algorithmus einige Probleme, da der GINGER-Algorithmus nicht die äußersten Wandelemente als *Boundary Part* erkennt, sondern jene, die sich in Richtung des Normalvektors am nächsten zur Raum-Geometrie befinden und dieselbe Orientierung aufweisen. Aus diesem Grund müssen die Materialschichten vor Anwendung des Algorithmus entfernt werden. Das Entfernen der Schichten lässt sich so durchführen, indem ausgehend von einer Wandgeo-

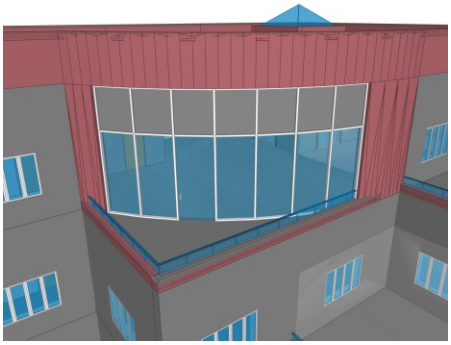

Abbildung 5.16: Glasfronten werden im Modell in Form von *IfcCurtainWall*-Elementen dargestellt

metrie alle Wandgeometrien geometrisch subtrahiert werden, welche sich in Richtung des Normalvektors innerhalb einer gewissen Entfernung davor befinden. Ist die Geometrie nach Subtraktion aller davor liegenden Geometrien leer, dann handelt es sich um eine Materialschicht die sich vollständig innerhalb der äußersten Schicht befindet und somit entfernt werden kann.

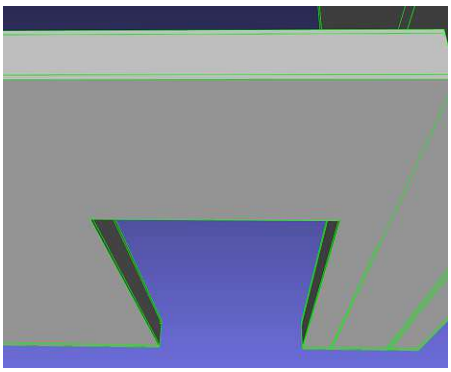

Abbildung 5.17: Materialschichten der Innenwände

Abgesehen von den bisher erwähnten Problemen wurde die Analyse nicht tiefer gehend verfolgt, da die notwendigen Erweiterungen des Algorithmus, um die Geometrie dieses Modells vereinfachen zu können, nicht Teil dieser Arbeit sind. Die Analyse der beiden Modelle hat gezeigt, dass eine tiefer gehende Evaluierung des Algorithmus mit weiteren realistischen Modellen sinnvoll wäre, um die Robustheit weiter zu erhöhen und eine größere Menge an möglichen Gebäudemodellen zu unterstützen. Die Vereinfachung der GA-Objekte wurde bisher lediglich mit dem Modell aus Abbildung 5.1 getestet, da im Zuge der Recherche keine öffentlich verfügbaren Modelle gefunden wurden, welche entsprechende Objekte beinhalten.

5.3 Robustheit

In der vorliegenden Arbeit wurde bereits öfter darauf hingewiesen, dass diverse Vergleiche im Algorithmus nur mit einer gewissen vorgegebenen Genauigkeit durchgeführt werden dürfen, da die Geometrien bei Verwendung von *Floating Point Numbers* nicht exakt abgebildet werden können. Dadurch entstehen bei geometrischen Operationen Rundungsfehler, welche sich durch die verschiedenen Schritte im Algorithmus fortpflanzen. Das Problem tritt vorwiegend bei schrägen Wänden auf, deren Ebene nicht entlang einer Koordinatenachse ausgerichtet ist. Diese Wände besitzen Normalvektoren mit vielen Nachkommastellen, welche in weiterer Folge zu Rundungsfehlern führen. Im Gebäudemodell aus Abbildung 5.1 sind 2 und in den Modellen aus den Abbildungen 5.2 und 5.3 5 schräge Wände enthalten. Das Modell aus Abbildung 5.4 enthält 6 schräge Wände. Durch die Verwendung von Vergleichen mit einer gewissen Ungenauigkeit konnten diese Geometrien verarbeitet werden. Der Prototyp exportiert die Geometrie nach jedem Schritt des Algorithmus im Format *Wavefront OBJ*. Diese Dateien können z.B. im Programm *MeshLab* geöffnet und die Zwischenergebnisse begutachtet werden, um mögliche Probleme mit Ungenauigkeiten zu erkennen und in weiterer Folge entsprechende Parameter anpassen zu können. Sollte bei der Verarbeitung ein Problem mit der Topologie einer Geometrie auftreten, dann wird eine entsprechende Fehlermeldung von *JTS* in der Logdatei ausgegeben.

Eine andere Herangehensweise ist die Durchführung exakter geometrischer Operationen wie es u.a. in der Arbeit von [MTX14] zusammengefasst wurde. In der Fachliteratur wird diese Methode *Exact Geometric Computation* (EGC) genannt. Bei dieser Vorgehensweise wird jede geometrische Operation exakt durchgeführt, sodass mit keinen Rundungsfehlern zu rechnen ist. Der Vorteil dieser Methode liegt auf der Hand und ermöglicht die Konzentration auf das Wesentliche, der Entwicklung von Algorithmen ohne auf die beschränkte Genauigkeit von *Floating Point Numbers* Rücksicht nehmen zu müssen. Ein großer Nachteil ist die langsamere Verarbeitung, da die exakte Darstellung der Zwischenergebnisse nicht direkt im Computer abgebildet werden kann, sondern nur simuliert wird. Es gibt mittlerweile einige Bibliotheken, welche die exakte Verarbeitung von Geometrien unterstützen wie z.B. CGAL [MTX14]. CGAL ermöglicht die Defintion von Algorithmen, ohne auf die Probleme mit inexakter Verarbeitung Rücksicht zu nehmen. Die Bibliothek kümmert sich im Hintergrund um die möglichst exakte Verarbeitung der definierten Algorithmen und erhöht bei Bedarf die Genauigkeit [CGA].

Wie bereits in Abschnitt 4 erwähnt, wurde ein großer Teil des Prototypen mithilfe der Bibliothek JTS umgesetzt. Diese Bibliothek unterstützt EGC leider nicht, wodurch die numerischen Probleme im Prototypen berücksichtigt werden mussten. JTS ermöglicht die Definition eines s.g. *PrecisionModel*, um einerseits die Genauigkeit der Geometrien und andererseits die Genauigkeit der durchgeführten Operationen zu beschränken. Dadurch kann die Anzahl der Rundungsfehler erheblich reduziert werden. Bei den Modellen aus den Abbildungen 5.1, 5.2, 5.3 und 5.4 wurde 0.01 mm für die Genauigkeit des *PrecisionModel* durch Ausprobieren festgelegt. Bei Geometrien, deren Ebene nicht entlang einer Koordinatenachse ausgerichtet ist, kann der Normalvektor trotz Reduzierung der Genauigkeit viele

Nachkommastellen aufweisen, welche in weiterer Folge zu ungenauen Berechnungen führen können. Es liegt nahe, die Genauigkeit der Normalvektoren analog zu den Koordinaten der Geometrien zu reduzieren. Diese Vorgehensweise würde bei der Projektion in 2D und der anschließenden Rück-Projektion in 3D zu größeren Rechenfehlern führen. Tests haben gezeigt, dass Geometrien teilweise an eine ganz andere Stelle im Modell zurück projiziert wurden. Aus diesem Grund wird im Prototypen ein zweites *PrecisionModel* definiert, welches für Vergleiche von Normalvektoren verwendet wird. Diese Vorgehensweise ermöglicht die Ermittlung von Normalvektoren die »nahezu« in dieselbe Richtung zeigen, was in unserem Algorithmus in den meisten Fällen ausreichend ist. Hierfür wurde beim Modell aus Abbildung 5.1 0.0001 mm durch Ausprobieren festgelegt. Bei den anderen Varianten des Modells wurde 0.1 mm verwendet, da diese Modelle viele schräge Wände enthalten, wodurch die Normalvektoren viele Nachkommastellen besitzen.

Abgesehen von der Verwendung von *PrecisionModels* mussten einige Vergleiche im Algorithmus mit reduzierter Genauigkeit durchgeführt werden. Die Genauigkeit dieser Vergleiche kann im Prototyp über entsprechende Parameter festgelegt werden. Ein Beispiel für einen ungenauen Vergleich im Algorithmus ist die Vereinfachung der triangulierten Geometrie, wo Dreiecke, die sich in derselben Ebene befinden, vereinfacht werden. Durch Ungenauigkeiten kommt es vor, dass die Ebenen nicht genau zusammenfallen und somit eine gewisse Toleranz eingerechnet werden muss. Diese Toleranz kann im Prototypen als Parameter angegeben werden. Abgesehen davon gibt es weitere Vergleiche, deren Toleranzen über Parameter im Prototypen festgelegt werden können. Trotz der bisher getroffenen Maßnahmen kann es vorkommen, dass durch die *Boundary Part* Erkennung kleinere Artefakte entstehen, welche meist durch Ungenauigkeiten bei der Projektion bzw. Rück-Projektion auftreten. Diese Artefakte können in weiterer Folge zu Problemen in den verschiedenen Schritten des Algorithmus führen. Das Problem lässt sich beheben, indem kleinere *Boundary Parts*, welche eine gewisse Fläche unterschreiten, nach der Erkennung wieder entfernt werden. In den verschiedenen Varianten des Modells wurden *Boundary Parts* mit einer Fläche kleiner als 0,04 m^2 entfernt, um Artefakte zu vermeiden.

Am Ende des Algorithmus wird aus den berechneten *Space Boundaries* für jeden Raum ein Solid berechnet. Es kann vorkommen, dass Punkte, welche in der ursprünglichen Geometrie zusammenfallen, am Ende des Algorithmus unterschiedliche Koordinaten besitzen. Die Koordinaten liegen dabei nicht sehr weit auseinander, das ist aber ausreichend, um Fehler bei der Solid Berechnung auszulösen. Dieses Problem lässt sich einfach lösen, indem die Genauigkeit der Koordinaten vor der Solid Generierung nochmal reduziert wird, sodass diese Punkte wieder zusammenfallen. Wird die Genauigkeit aber zu gering konfiguriert, dann fallen auch Punkte zusammen, welche in der ursprüngliche Geometrie auseinander liegen. Eine weitere Lösung für dieses Problem ist der Einsatz eines Snap Rounding Algorithmus [Her11]. In Abschnitt 3.5.3 wurde das Problem gezielt gelöst, indem jede Koordinate einer Öffnung mit jener Koordinate aus dem Wandelement ersetzt wurde, die ihr am nächsten liegt. Im Prinzip handelt es sich um dasselbe Problem und ließe sich ebenfalls über Snap Rounding lösen.

Im Zuge der Evaluierung des Algorithmus mit den in Abschnitt 5.1 erwähnten Gebäude-

modellen wurden einige Probleme bezüglich Robustheit erkannt, welche hauptsächlich der Numerik geschuldet sind. Diese Probleme lassen sich mit den bisher erwähnten Methoden für die analysierten Gebäudemodelle großteils lösen. Im Zuge der Evaluierung wurde der Algorithmus jedoch nur mit einer kleinen Auswahl an Modellen überprüft. Eine genauere Analyse mit weiteren Modellen erscheint daher sinnvoll und geht als offener Punkt aus dieser Arbeit hervor. Abgesehen davon wäre es interessant zu evaluieren, wie sich der Algorithmus unter Einsatz von EGC im Vergleich zur derzeitigen Lösung verhält. Dabei sollte einerseits die Robustheit und andererseits die Veränderung der Laufzeit durch den Einsatz von EGC analysiert werden. Auch ein Benchmark Vergleich unter Einsatz verschiedener Gebäudemodelle würde sich anbieten.

KAPITEL 6

Zusammenfassung und Reflexion

6.1 Zusammenfassung

Nahezu jedes moderne Bürogebäude ist heutzutage mit einem GA-System ausgestattet, welches für die Steuerung, Regelung und Kontrolle von Heizung, Lüftung und Klimaanlage eingesetzt wird. Auch andere Bereiche wie z.B. Beleuchtung und Abschattung werden mittlerweile von GA-Systemen gesteuert. In Zeiten von Umweltkrisen wird das enorme Energiesparpotential von GA-Systemen immer wichtiger, da ungefähr 40% des Energieverbrauchs innerhalb der Europäischen Union auf den Verbrauch von Gebäuden zurückzuführen ist [PLOP08, DCB17]. In der vorliegenden Arbeit wird die geschichtliche Entwicklung von BIM, ausgehend von CAD erläutert und auf verschiedene Standards rund um diese Technologien eingegangen. Eine dieser Technologien sind die IFC, welche sich als Standardformat von Gebäudemodellen in BIM etabliert haben. Aufgrund der Wichtigkeit dieses Standards wurde die geschichtliche Entwicklung der IFC erläutert und sowohl die Architektur als auch das Datenmodell im Detail erklärt. Anschließend wurde erwähnt, wie BIM-Modelle als Basis für Anwendungen im Bereich Gebäudeautomation verwendet werden können. Im Zuge dieser Arbeit wurde auf Basis des GINGER-Algorithmus [LBB+16, LBK+16] ein Algorithmus entwickelt, um die komplexe Geometrie in modernen Gebäudemodellen zu vereinfachen, damit sie in GA-Systemen Verwendung finden kann. In modernen GA-Systemen sind häufig sehr viele Sensoren verbaut, die überwacht werden müssen. Durch die Vereinfachung der Geometrie können die Sensorwerte in Echtzeit im Kontext der Gebäudegeometrie dargestellt werden, ohne den Benutzer vom Wesentlichen abzulenken. Im Rahmen dieser Arbeit wurde ein Prototyp entwickelt, welcher den beschriebenen Algorithmus implementiert. Die Architektur und die verwendeten Technologien werden im Detail vorgestellt. Der Algorithmus wurde anhand ausgewählter Gebäudemodelle evaluiert, um die Funktionsweise anhand realer Verhältnisse zu überprüfen. Die verwendeten Gebäudemodelle wurden vorgestellt und auf die Ergebnisse der Evaluierung im Detail eingegangen. Dabei wurden Schwächen des Algorithmus und

mögliche Verbesserungen für zukünftige Arbeiten aufgezeigt.

6.2 Vergleich mit anderen Arbeiten

BIM hat sich mittlerweile etabliert, um den Entwurf und die Konstruktion von Gebäuden zu unterstützen und die Kommunikation zwischen den Projektbeteiligten zu verbessern. Mittlerweile ist der Einsatz von BIM in öffentlichen Projekten innerhalb der EU in immer mehr Ländern verpflichtend [WKO]. Abgesehen von der Entwurfs- und Konstruktionsphase ist BIM im Facility Management derzeit noch nicht sehr weit verbreitet. In der Literatur lassen sich mittlerweile bereits verschiedene Ansätze finden, um BIM-Modelle als Basis für Anwendungen im Facility Management und in der Gebäudeautomation einzusetzen. In der Arbeit von [MSP+19] wird ein Workflow-Modell definiert, um ausgehend von einem BIM-Modell ein integriertes Gebäudemodell zu erstellen bzw. anzupassen, welches die statischen Daten aus dem BIM-Modell mit den dynamischen Daten eines GA-Systems miteinander vereint. [TSE+19] beschreibt in seiner Arbeit verschiedene Strategien, wie die statischen Daten von BIM mit den dynamischen Daten von GA-Systemen miteinander verknüpft werden können und stellt die Vorteile der einzelnen Ansätze den Nachteilen gegenüber. Es wird ein Hybridansatz empfohlen, wo die statischen BIM-Daten in einer Ontologie nach dem *Semantic Web* Paradigma und die dynamischen Daten des GA-Systems in einer relationalen Datenbank gespeichert werden. In der Arbeit von [MSP+19] wird dieser Hybridansatz verwendet. [PMB+18] untersucht in seiner Arbeit Ansätze aus dem Forschungsbereich *Visual Analytics*, um das Facility Management bei seiner Arbeit besser unterstützen zu können. Aus dieser Arbeit geht hervor, dass die Verwendung von 2D bzw. 3D Modellen für die Visualisierung der technischen Anlagen im Gebäude einen enormen Vorteil bringt, da die Daten im Kontext des Gebäudes visualisiert werden und dadurch die Benutzerfreundlichkeit der Software erheblich verbessert werden kann. In der Arbeit von [SLL11] wurde ein System evaluiert, welches das Facility Management bei der Wartung der technischen Anlagen im Gebäude unterstützen soll. Dabei wurde der Wartungszustand der einzelnen Geräte in vier verschiedenen Farben visualisiert, sodass ein Überblick über den gesamten Zustand der Anlagen im Gebäude zur Verfügung stand. Zusätzlich wurden Dokumente wie z.B. Reports zu den einzelnen Geräten direkt im Modell verlinkt und in einem externen System zur Verfügung gestellt. Die Mitarbeiter hatten durch das System einen besseren Überblick über den Zustand und die nächsten anstehenden Tätigkeiten als bei der Verwendung von 2D CAD Modellen oder Papierplänen. In der Arbeit von [PMB+18] wurde ein Prototyp entwickelt, wo die Daten aus einem GA-System einer Universität mit den Positionsdaten der Studenten verknüpft wurden. Die Studenten konnten über eine App Feedback zum gefühlten Komfort übermitteln, sodass das Facility Management darauf reagieren konnte. Die Daten wurden im Prototypen in 2-dimensionaler Form visualisiert. Die Geometrie des Gebäudes wurde stark vereinfacht, um den Blick auf das Wesentliche zu lenken und die Benutzbarkeit zu erhöhen. In den Arbeiten [PMB+18] und [MSP+19] wurde die Gebäudegeometrie für die Visualisierung im GA-System über einen manuellen Schritt vereinfacht. [RB13] beschreibt in seiner Arbeit einen Algorithmus, um ausgehend von

Level 1 Space Boundaries, Space Boundaries mit höherem Level zu berechnen. Space Boundaries mit höherem Level werden für die Energieanalyse benötigt, da es in diesem Bereich wichtig ist, was sich außerhalb des Raumes befindet und aus welchen Materialien sich die Raumgrenzen zusammensetzen. In der vorliegenden Arbeit wurde der GINGER-Algorithmus aus den Arbeiten von [Lad14, LBB+16, LBK+16] verwendet und erweitert. Der GINGER-Algorithmus ermöglicht die Vereinfachung der Geometrie eines IFC-Modells zur Verwendung in Energieanalyse-Werkzeugen und wurde im Rahmen des Forschungsprojekts GINGER entwickelt [Lad14]. Im Rahmen der Literaturrecherche wurden einige Arbeiten gefunden, wo die Daten eines BIM-Modells im Bereich des Facility Management oder in der Gebäudeautomation wiederverwendet werden, damit diese nicht neu modelliert werden müssen. Abgesehen davon gibt es derzeit wenig Forschung zur Vereinfachung der Gebäudegeometrie für die Verwendung in GA-Systemen.

6.3 Offene Punkte

In diesem Abschnitt wird auf offene Punkte eingegangen, welche durch die vorliegende Arbeit nicht oder nicht ausreichend behandelt wurden. Das Ziel dieser Arbeit war die Vereinfachung der Gebäudegeometrie eines BIM-Modells und der Export als *Brep Solid Geometry*, damit diese in einem GA-System verwendet werden kann. Die Implementierung des Prototypen wurde vom selben Gebäudemodell begleitet, welches bereits in der Arbeit von [MSP+19] verwendet wurde. In dieser Arbeit wurde die Vereinfachung der Geometrie noch manuell vorgenommen, indem das vereinfachte Modell von Grund auf neu entworfen wurde. Das Ziel der vorliegenden Arbeit war es diesen Schritt zu automatisieren. Im Zuge der Evaluierung wurden verschiedene Varianten dieses Gebäudemodells erstellt, um verschiedene Konstellationen von Räumen und die Robustheit des Algorithmus zu testen. Zusätzlich wurden zwei ausgewählte Modelle aus dem Internet analysiert, jedoch hat die Vereinfachung der Geometrie aufgrund diverser Mängel in der Modellierung nicht funktioniert. Leider konnten im Internet keine öffentlich zugänglichen Modelle gefunden werden, welche GA-Objekte beinhalten. Daher konnte die Vereinfachung von GA-Objekten bisher nur unzureichend evaluiert werden. Es sollte eine tiefer gehende Analyse mit weiteren realen Gebäudemodellen durchgeführt werden, um die Robustheit des Algorithmus weiter zu erhöhen. Diese Analyse geht als offener Punkt der vorliegenden Arbeit hervor. Wie bereits in Abschnitt 5.3 erwähnt, wäre auch eine Implementierung des Algorithmus unter Verwendung von EGC interessant und könnte dem in der vorliegenden Arbeit verwendeten Ansatz gegenübergestellt werden. Im Zuge dessen wäre es spannend, entsprechende Benchmarks durchzuführen, um die beiden Ansätze zu vergleichen und aufzuzeigen, welche Auswirkungen die Verwendung von EGC auf die Laufzeit des Algorithmus hat. Im Zuge dieser Arbeit wurde ausschließlich die Geometrie eines Gebäudemodells behandelt, um diese in einem GA-System verwenden zu können. Ein Großteil der semantischen Informationen von GA-Objekten gehen beim Export in das IFC-Format verloren, da diese bisher nur unzureichend unterstützt werden. Für die Verwendung in GA-Systemen sind diese Informationen jedoch von enormer Wichtigkeit, um zusätzlich zur geometrischen Darstellung auch semantische Daten visualisieren zu

können. Die Semantik von GA-Objekten ist ein großes Thema und konnte aufgrund des Umfangs im Zuge der vorliegenden Arbeit nicht behandelt werden.

Abbildungsverzeichnis

Liste der Algorithmen

Abkürzungsverzeichnis

BMS Building Management System

BIM Building Information Modeling

CAD Computer Aided Design

EGC Exact Geometric Computation

EMF Eclipse Modeling Framework

GA-System Gebäudeautomationssystem

GA-Objekt Objekt der Gebäudeautomation

IFC Industry Foundation Classes

JDK Java Development Kit

JNI Java Native Interface

JTS Java Topology Suite

SDK Software Development Kit

Literaturverzeichnis

[ALBF14] Aryani Ahmad Latiffi, Juliana Brahim, and Mohamad Syazli Fathi. The development of building information modeling (bim) definition. *Applied Mechanics and Materials*, Volume 567:625–630, 06 2014.

[Amo15] Robert Amor. Analysis of the evolving ifc schema. In *32nd CIB W78 Information Technology for Construction Conference (CIB W78 2015), Eindhoven, Netherlands*, 2015.

[And18] André Borrmann, Markus König, Christian Koch, Jakob Beetz. *Building Information Modeling: Technology Foundations and Industry Practice*. Springer, 10 2018.

[Asa20] Aleksander Asanowicz. Evolution of computer aided design: Three generations of cad. *Architectural Computing from Turing to 2000 [eCAADe Conference Proceedings / ISBN 0-9523687-5-7] Liverpool (UK) 15-17 September 1999, pp. 94-100*, 01 2020.

[Baz10] Vladimir Bazjanac. Space boundary requirements for modeling of building geometry for energy and other performance simulation. 2010.

[BBF+16] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, and et al. Brick: Towards a unified metadata schema for buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, BuildSys '16, page 41–50, New York, NY, USA, 2016. Association for Computing Machinery.

[BIMa] BIMserver. `http://bimserver.org/`.

[BIMb] BIMserver Wiki. `https://github.com/opensourceBIM/BIMserver/wiki`. Zuletzt aufgerufen am 15.04.2020.

[Bui] buildingSMART International. `https://www.buildingsmart.org/`. Zuletzt aufgerufen am 02.02.2020.

[CAE18] Rabia Charef, Hafiz Alaka, and Stephen Emmitt. Beyond the third dimension of bim: A systematic review of literature and assessment of professional views. *Journal of Building Engineering*, 19, 05 2018.

[CDDA14] Jack Cheng, Yichuan Deng, Moumita Das, and Chimay Anumba. Evaluation of ifc4 for the gis and green building domains. pages 2216–2223, 06 2014.

[CGA] CGAL - Exact Geometric Computation. `https://www.cgal.org/`. Zuletzt aufgerufen am 19.08.2020.

[CP14] Ireneusz Czmoch and Adam Pękala. Traditional design versus bim based design. *Procedia Engineering*, 91, 12 2014.

[DCB17] Delia D'Agostino, Barbara Cuniberti, and Paolo Bertoldi. Energy consumption and efficiency technology measures in european non-residential buildings. *Energy and Buildings*, 153:72 – 86, 2017.

[Doc] Docker. `https://www.docker.com`. Zuletzt aufgerufen am 20.04.2020.

[Eur05] European Construction Technology Platform (ECTP) . Challenging and Changing Europe's Built Environment - A vision for a sustainable and competitive construction sector by 2030. Technical report, 2005.

[GCR04] NIST GCR. Cost analysis of inadequate interoperability in the us capital facilities industry. *National Institute of Standards and Technology (NIST)*, pages 223–253, 2004.

[Her11] John Hershberger. Stable snap rounding. volume 46, pages 197–206, 01 2011.

[Ifca] IfcOpenShell für BIMserver. `http://ifcopenshell.org/bimserver`. Zuletzt aufgerufen am 15.04.2020.

[Ifcb] ifcOWL. `https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/`. Zuletzt aufgerufen am 13.02.2020.

[Ifcc] Industry Foundation Classes, buildingSMART International. `https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/`. Zuletzt aufgerufen am 02.02.2020.

[J.10] Harris J. ïntegration of bim and business strategy". Technical report, McCormick School of Engineering and Applied Science, Northwestern University, Evanston, IL, 2010.

[Jon14] Stephen A. Jones. Der unternehmerische vorteil von bim im bauwesen in den großen weltmärkten - building information modeling als weltweiter impulsgeber für innovationen in der baubranche. Technical report, McGRAW Hill Construction, 2014.

[JTS] Java Topology Suite (JTS). https://github.com/locationtech/jts. Zuletzt aufgerufen am 15.04.2020.

[Kiv99] Arto Kiviniemi. Iai and ifc-state-of-the-art. In *Proceedings of the 1999 CIB w78 Conference*, pages 2157–2169, 1999.

[KPIS08] Junhwan Kim, Michael J. Pratt, Raj G. Iyer, and Ram D. Sriram. Standardized data exchange of cad models with design intent. *Computer-Aided Design*, 40(7):760 – 777, 2008. Current State and Future of Product Data Technologies (PDT).

[Lad14] Daniel Ladenhauf. From building information models to simplified geometries for energy performance analysis. Master's thesis, Graz University of Technology, 2014.

[LBB+16] Daniel Ladenhauf, Kurt Battisti, René Berndt, Eva Eggeling, Dieter W. Fellner, Markus Gratzl-Michlmair, and Torsten Ullrich. Computational geometry in the context of building information modeling. *Energy and Buildings*, 115:78 – 84, 2016. "A selection of International AcademicConference "Places and Technologies 2014" Belgrade, Serbia.

[LBK+16] D. Ladenhauf, R. Berndt, U. Krispel, E. Eggeling, T. Ullrich, K. Battisti, and M. Gratzl-Michlmair. Geometry simplification according to semantic constraints: Enabling energy analysis based on building information models. *Computer Science - Research and Development*, 31(3):119–125, 2016. cited By 2.

[Lie10] Thomas Liebich. Unveiling ifc2x4 - the next generation of openbim. 2010.

[LK12] M Laakso and AO Kiviniemi. The ifc standard: A review of history, development, and standardization, information technology. *ITcon*, 17(9):134–161, May 2012.

[Loc] LocationTech. https://projects.eclipse.org/projects/locationtech. Zuletzt aufgerufen am 15.04.2020.

[Mav] Apache Maven. http://maven.apache.org/. Zuletzt aufgerufen am 21.04.2020.

[MSM19] P Mesároš, J Smetanková, and Tomáš Mandičák. The fifth dimension of bim – implementation survey. *IOP Conference Series: Earth and Environmental Science*, 222:012003, 01 2019.

[MSP+19] Maryam Montazer, Georg Suter, Filip Petrushevski, Miloš Šipetić, Max Blöchle, Stefan Gaida, Wolfgang Kastner, and Christian Schiefer. A workflow model for setup and maintenance of an integrated building model for energy management. 09 2019.

[MTX14] Gang Mei, John Tipper, and Nengxiong Xu. Numerical robustness in geometric computation: An expository summary. *Applied Mathematics & Information Sciences*, 8:2717–2727, 06 2014.

[Off] Office Building. `http://smartlab1.elis.ugent.be:8889/IFC-repo/OfficeBuilding.ifc`. Zuletzt aufgerufen am 11.08.2020.

[Ope] Open CASCADE. `https://www.opencascade.com`. Zuletzt aufgerufen am 21.04.2020.

[PBC13] Christopher Preece, Reza Barati, and Aref Charehzehi. Enhancing planning and scheduling program by using benefits of bim based applications. *Civil and Environmental Research*, 3:2–13, 01 2013.

[PLOP08] Luis Pérez-Lombard, José Ortiz, and Christine Pout. A review on buildings energy consumption information. *Energy and Buildings*, 40(3):394 – 398, 2008.

[PMB+18] A. Prouzeau, D. M.B., M. Balasubramaniam, J. Henry, N. Hoang, and T. Dwyer. Visual analytics for energy monitoring in the context of building management. In *2018 International Symposium on Big Data Visual and Immersive Analytics (BDVA)*, pages 1–9, Oct 2018.

[RB13] Cody Rose and Vladimir Bazjanac. An algorithm to generate space boundaries for building energy simulation. *Engineering with Computers*, 31, 04 2013.

[RNK] RNK Architectural. `http://smartlab1.elis.ugent.be:8889/IFC-repo/RNK/Architectural.ifc`. Zuletzt aufgerufen am 11.08.2020.

[RPL+17] Mads Holten Rasmussen, Pieter Pauwels, Maxime Lefrançois, Georg Schneider, Christian Hviid, and Jan Karlshøj. Recent changes in the building topology ontology. 11 2017.

[SELT18] Rafael Sacks, Charles Eastman, Ghang Lee, and Paul Teicholz. *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*. 08 2018.

[SFR09] Zita Sampaio, Miguel Ferreira, and Daniel Rosário. 3d virtual environment used to support lighting system management in a building. volume 5738, pages 177–184, 01 2009.

[SLL11] Yu Chih Su, Yi Chien Lee, and Yu Cheng Lin. Enhancing maintenance management using building information modeling in facilities management. In *Proceedings of the 28th international symposium on automation and robotics in construction*, volume 2, pages 752–757, 2011.

[Spaa] IfcRelSpaceBoundary. https://standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/schema/ifcproductextension/lexical/ifcrelspaceboundary.htm. Zuletzt aufgerufen am 03.08.2020.

[Spab] Space Modeler. http://spacepatterns.com/wp/. Zuletzt aufgerufen am 11.08.2020.

[Spra] Spring Boot. https://spring.io/projects/spring-boot. Zuletzt aufgerufen am 21.04.2020.

[Sprb] Spring Framework. https://spring.io. Zuletzt aufgerufen am 21.04.2020.

[TF10] Stefano Tornincasa and Di F. The future and the evolution of cad. pages II–1, 01 2010.

[TSE+19] Shu Tang, Dennis R. Shelden, Charles M. Eastman, Pardis Pishdad-Bozorgi, and Xinghua Gao. A review of building information modeling (bim) and the internet of things (iot) devices integration: Present status and future trends. *Automation in Construction*, 101:127 – 139, 2019.

[Wat10] Alastair Watson. Bim – a driver for change. 2010.

[WKO] WKO - BIM Broschüre. https://www.wko.at/branchen/gewerbe-handwerk/bau/BIM-Broschuere.pdf. Zuletzt aufgerufen am 24.08.2020.

[WS97] A. C. W. Wong and A. T. P. So. Building automation in the 21st century. In *1997 Fourth International Conference on Advances in Power System Control, Operation and Management, APSCOM-97. (Conf. Publ. No. 450)*, volume 2, pages 819–824 vol.2, Nov 1997.

[YE14] Xue Yang and Semiha Ergan. Evaluation of visualization techniques for use by facility operators during monitoring tasks. *Automation in Construction*, 44:103 – 118, 2014.