

Hochschule RheinMain
Fachbereich Design Informatik Medien
Studiengang Angewandte Informatik

Bachelor-Arbeit
zur Erlangung des akademischen Grades
Bachelor of Science – B.Sc.

Konzeption und Realisierung eines Plugins für die Aufbereitung von CAD-Daten zur Nutzung in einer Game Engine

vorgelegt von *Raffael Holz*
am 20.02.2018

Referent: *Prof. Dr. Ralf Dörner*
Korreferent: *Prof. Dr. Christoph Schulz*

Erklärung gem. ABPO, Ziff. 6.4.3

Ich versichere, dass ich die Bachelor-Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Ort, Datum

Unterschrift Studierende/Studierender

Hiermit erkläre ich mein Einverständnis mit den im Folgenden aufgeführten Verbreitungsformen dieser Bachelor-Arbeit:

Verbreitungsform	ja	nein
Einstellung der Arbeit in die Hochschulbibliothek mit Datenträger	X	
Einstellung der Arbeit in die Hochschulbibliothek ohne Datenträger	X	
Veröffentlichung des Titels der Arbeit im Internet	X	
Veröffentlichung der Arbeit im Internet	X	

Ort, Datum

Unterschrift Studierende/Studierender

Inhaltsverzeichnis

1 Einleitung	1
1.1 Problemstellung.....	1
1.2 Motivation.....	2
1.3 Zielsetzung.....	3
1.4 Aufbau der Arbeit.....	4
2 Grundlagen	5
2.1 Computer-aided design (CAD).....	5
2.1.1 Definition.....	5
2.1.2 Obstruction Volumes.....	6
2.1.3 Smartplant 3D.....	6
2.2 Konvertierung mit dem VUEParser.....	7
2.3 Spiel-Engine.....	11
2.3.1 Definition.....	11
2.3.2 Unreal Engine 4.....	12
3 Ausgangslage	13
3.1 Unreal Engine 4.....	13
3.1.1 Entwicklung von Plugins.....	13
3.1.2 Darstellung von Polymeshes.....	15
3.1.3 Anlegen von Materialien.....	16
3.2 Arbeiten mit dem VUEParser.....	17
4 Anforderungsbeschreibung	19
4.1 Funktionale Anforderungen.....	19
4.1.1 Selektieren einer Datei.....	19
4.1.2 Übernahme der Metadaten.....	19
4.1.3 Platzieren der Geometrie.....	19
4.1.4 Unterstützung von Obstruction Volumes.....	20
4.1.5 Übernahme der Materialien.....	20
4.1.6 Selektieren der importierten Geometrie für Informationen.....	20
4.1.7 Nicht konvertierte Geometrie kenntlich machen.....	20
4.1.8 Bewegung des Anwenders innerhalb der Anlage ermöglichen.....	21
4.2 Nicht-Funktionale Anforderungen.....	21
4.2.1 Benutzeroberfläche.....	21
4.2.2 Stabilität.....	21
4.2.3 Performance.....	21
4.2.4 Genauigkeit.....	22
4.2.5 Virtual-Reality.....	22
4.2.6 Kosten.....	22
5 Konzeption	23
5.1 Konzeption der funktionalen Anforderungen.....	23
5.1.1 Selektieren einer Datei.....	23
5.1.2 Übernahme der Metadaten.....	24
5.1.3 Platzieren der Geometrie.....	24
5.1.4 Unterstützung von Obstruction Volumes.....	27
5.1.5 Übernahme der Materialien.....	27
5.1.6 Selektieren der importierten Geometrie für Informationen.....	27
5.1.7 Nicht konvertierte Geometrie kenntlich machen.....	28

5.1.8	Bewegung des Anwenders innerhalb der Anlage ermöglichen.....	28
5.2	Konzeption nicht-funktionaler Anforderungen.....	28
5.2.1	Benutzeroberfläche.....	28
5.2.2	Linux-Unterstützung.....	29
6	Implementierung	31
6.1	Erschaffung der Grundlage.....	31
6.2	Implementierung der funktionalen Anforderungen.....	33
6.2.1	Selektieren einer Datei.....	33
6.2.2	Platzieren der Geometrie.....	35
6.2.3	Übernahme der Metadaten.....	45
6.2.4	Unterstützung von Obstruction Volumes.....	46
6.2.5	Übernahme der Materialien.....	47
6.2.6	Selektieren der importierten Geometrie für Informationen.....	49
6.2.7	Nicht konvertierte Geometrie kenntlich machen.....	49
6.2.8	Bewegung des Anwenders innerhalb der Anlage ermöglichen.....	50
6.3	Implementierung nicht-funktionaler Anforderungen.....	51
6.3.1	Benutzeroberfläche.....	51
7	Evaluation	53
7.1	Qualität in Anbetracht der gesetzten Ziele.....	53
7.2	Stabilität.....	54
7.3	Vergleich zum Intergraph Freeview.....	55
7.4	Geschwindigkeit.....	58
7.5	Evaluation durch Fachkundige.....	61
8	Zusammenfassung und Ausblick	63
9	Literaturverzeichnis	65

1 Einleitung

1.1 Problemstellung

Die wichtigsten Bedürfnisse des Menschen sind körperliche Bedürfnisse. Dazu zählen unter anderem Nahrung, Wasser, Gesundheit und Wärme.[1] Damit die Güter, die diese Bedürfnisse befriedigen, ausreichend zur Verfügung stehen, werden meist Anlagen zur Produktion verwendet. Der Anlagenbau beschäftigt sich mit der Planung, Produktion und Wartung dieser Anlagen.

Beim Jahreswechsel 2016/2017 umfasste die Weltbevölkerung rund 7,4 Milliarden Menschen.[2] Rund 83 Millionen Menschen kommen gegenwärtig jährlich dazu.[3] 2050 erwarten die Vereinten Nationen eine globale Bevölkerungsdichte zwischen 8,7 und 10,8 Milliarden Menschen.[3] Durch den raschen Zuwachs der Bevölkerung, lässt sich vermuten, dass auch die Anzahl an Anlagen steigen muss. Eine solche Vermutung entsteht daraus, dass mehr Anlagen für die Produktion von notwendigen Gütern gebaut werden müssen. In Bezug auf den deutschen Anlagenbau bedeutete das konkret, dass 2002 ein Auftragseingang von 17,6 Milliarden Euro verzeichnet wurde[4], es 2008 aber schon 32,8 Milliarden waren. Die Steigerung von 115%[4] lässt vermuten, dass eine weitere Steigerung zu erwarten ist. Eine Steigerungen, die durch den Anlagenbau gesättigt werden muss. Bevor jedoch eine Anlage in Betrieb genommen werden kann, muss sie geplant und produziert werden. Die Planung geschieht heutzutage vor allem mit *computer-aided design* kurz CAD. Darunter versteht man eine softwaregestützte Hilfestellung zur dreidimensionale Planung einer Konstruktion. Umfangreichere Anlagen werden oft gespalten, deren Bereiche an verschiedene Parteien verteilt[5] und später wieder zusammengesetzt. Eine Partei kann dabei eine Person, ein Team oder sogar ein Drittunternehmen darstellen. Wir gehen einfacheitshalber nun davon aus, dass die Person oder das Team, welche für das gleiche Unternehmen arbeiten, auch die gleiche Software zur Planung verwenden. Anders sieht es beim Drittunternehmen aus. Hier ist es durchaus denkbar, dass eine alternative Anwendung für die Planung verwendet wird, die wiederum ihr eigenes Dateiformat verwendet. Durch die verschiedenen Dateiformate ist eine Kompatibilität untereinander nicht zwangsläufig gegeben und das spätere Zusammenfügen wird erschwert.

1 Einleitung 1.1 Problemstellung

Die später zusammengefügte Zeichnung wird auch für Präsentationszwecke verwendet. CAD-Anwendungen sind aber vor allem für das Ergebnis des Prozesses gedacht und nicht für die Präsentation.[6] Auch muss für Präsentationen die lizenzierte und voll funktionsfähige Anwendung verwendet werden, obwohl gar nicht geplant, sondern lediglich präsentiert werden möchte. Manche Hersteller geben zwar teilweise kostenfreie Viewer für ihre Dateiformate heraus, unterstützen aber meist nur ihre hauseigenen Formate.[7]

1.2 Motivation

Nicht jeder Anwender möchte für ein einziges Projekt sein etabliertes CAD-System gegen ein Anderes tauschen. Die Anschaffung der neuen Software kann mit Kosten verbunden sein. Preis der Software, Installation und Wartung, aber auch notwendige Mitarbeiterschulungen, können zusätzliche Kosten darstellen. Durch Installation und Schulungen können außerdem Verzögerungen entstehen, bis die Mitarbeiter genug geschult sind, um die Software in ihrem vollen Umfang nutzen zu können. Zudem ist eine Simulation von Anlagen auch für Rettungskräfte hilfreich. Es können Brände und Explosionen simuliert werden und eine Lösung der Situation im Vorfeld geplant werden.[8] Virtual Reality ist zudem ein Thema, das auch für den Anlagenbau interessant ist.[8]

1.3 Zielsetzung

Ziel der Arbeit ist die Prüfung, inwieweit ein Plugin für eine bestehende Spiel-Engine als Softwarelösung in Betracht gezogen werden kann. Es gilt unter anderem die folgenden Fragen zu beantworten:

- Wie hoch ist die Performance in Relation zu bestehenden Lösungen?
- Können alle notwendigen Daten aus dem Anlagenbau übernommen werden?
Dazu zählen auch Daten, die keine geometrische Darstellung haben.
- Wie sieht es mit der Bedienbarkeit aus? Auch für einen Anwender ohne CAD-Erfahrung?
- Wie kann eine Interaktivität, Animation und Simulation umgesetzt werden?
Zum Beispiel eine Simulation mit Animation durch einfachen Knopfdruck zu starten, die eine Explosion betätigen simulieren würde.
- Ist die Lösung auch portabel?
- Wie sieht es mit Virtual Reality aus?

Um dies zu bewerten wird ein Plugin für die Unreal Engine 4 entwickelt, das CAD-Daten importiert und sie präsentiert. Damit das Ergebnis möglichst dem Original ähnelt, werden auch Metadaten und Materialien übernommen. Die CAD-Daten kommen aus der Anwendung SmartPlant 3D[9], einer Software von Intergraph[10] und werden in das proprietäre Format .VUE gespeichert. Das Format enthält vor allem die Geometrie der Anlage, beschrieben als Primitive. Der von dem Unternehmen Unitec Informationssysteme GmbH zur Verfügung gestellte Parser liest diese ein und konvertiert sie zu einfachen Strukturen. Der Parser wird fester Bestandteil des neuen Plugins, da die Geometrien benötigt werden.

1.4 Aufbau der Arbeit

Die Arbeit wird in sieben Kapitel unterteilt und beginnt mit der Einleitung. Die Einleitung gibt einen kurzen Überblick über das Themengebiet, die Motivation und die gesetzten Ziele der Arbeit.

Das zweite Kapitel geht näher auf die Grundlagen und die dazu verwendete Software ein, für die sich in dieser Arbeit entschieden wird. Auch wird kurz auf das Produkt VUEParser eingegangen, das CAD-Daten aus dem VUE-Format importieren kann. In den folgenden Kapiteln wird die Unreal Engine 4 fortan als Spiel-Engine bezeichnet.

Die Ausgangslage wird in Teil drei näher analysiert. Es wird mit der Anforderungsanalyse begonnen und näher beschrieben, welche Bestandteile für das Plugin gegeben und notwendig sind. So wird speziell näher auf die zu verwendenden Schnittstellen eingegangen, damit die Daten importiert werden können.

Der vierte Teil beschreibt die Anforderungen an das Plugin.

Der fünfte Teil der Arbeit beschäftigt sich mit der Konzeption des Plugins und der generellen Tesselierung der vom VUEParser stammenden 3D-Daten.

Nachdem die notwendige Planung des Plugins definiert wurde, geht es im sechsten Teil um die Implementierung.

Der letzte Teil geht näher auf das Ergebnis der Arbeit ein. So wird geprüft, ob das am Anfang gesetzte Ziel erreicht wurde und es auch in Hinsicht auf die Problemstellung, eine tatsächlich mögliche Lösung für Unternehmen darstellt.

2 Grundlagen

2.1 Computer-aided design (CAD)

Da die Daten, die das Plugin für die Präsentation benötigt, aus einer CAD-Anwendung stammen, werden die Grundlagen hier genauer beschrieben.

2.1.1 Definition

Als Teil der CAx-Technologien, steht Computer-aided design, kurz CAD oder CADD (letztes D für drafting), für die computerunterstützte Konstruktion und Planung von Produkten (z.B. Anlagen, Autos, Bauwerke). Das x in den CAx-Technologien steht als Platzhalter für andere Bereiche, wie etwa der rechnergestützten Entwicklung (computer-aided engineering – CAE) oder der rechnergestützten Erfindung (computer-aided innovation – CAI). Bei der CAD-Technologie versteht man streng genommen die Erstellung und Manipulation geometrischer Modelle. In der Praxis gehören hier aber noch weitere Teile des Konstruktions- und Planungsprozesses dazu. So werden auch die Gewinnung und Bereitstellung von Informationen, die Simulation und das Rendering als Teil des Prozesses angesehen. Zusammengefasst werden Produkte von der Planung bis zur Übergabe an die Hersteller begleitet.[11]

Die Marktverteilung (Tabelle T.1) zeigt, dass die Wahl der Software auch abhängig von der Größe des Projekts ist.[12]

Klein - Mittel	Mittel - Groß	Groß
Autodesk Plant 3D	AVEVA PDMS or Everything3D	Intergraph SmartPlant 3D
Intergraph CADWorks/PDS	Autodesk Plant 3D	AVEVA PDMS and Everything3D
	Bentley OpenPlant and AutoPLANT	Cadmatic
	Intergraph CADWorx/PDS	TriCAD MS

Tabelle T.1 zeigt die Verbreitung von CAD-Anwendungen auf dem Markt [12]

2.1.2 Obstruction Volumes

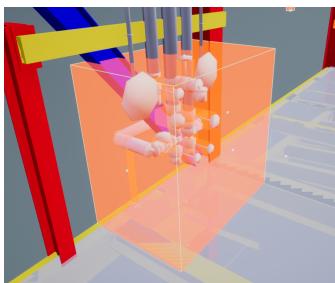


Abbildung 1: Zeigt eine Obstruction Volume

Obstruction Volumes stehen für Geometrien, die vorwiegend im Planungsprozess Verwendung finden. So stellen sie zum Beispiel Bereiche dar, die für eine spätere Wartung freigehalten werden müssen. Oder den Bereich, den eine Maschine zum Arbeiten benötigt. Sie können auch mit transparenten Materialien verbunden werden, damit sie sich augenscheinlich von anderer Geometrie unterscheiden.

Drei häufig vorkommende Bereiche sind:

Maintenance

Dieser Bereich ist für die mögliche Wartungen freizuhalten. Dabei sollte genug Platz eingeplant werden, um jedes Bauteil auswechseln zu können.

Operation

Dieser Bereich stellt den notwendigen Platz dar, den eine Maschine für die Arbeit benötigt. Zur Sicherheit kann beitragen werden, indem durch Markierung des Bereichs auf die mögliche Gefahr hingewiesen wird.

Die Nutzung dieses Bereichs endet nicht bei Maschinen. Der Bereich wird zudem für Türen eingesetzt, damit sie ungehindert geöffnet werden können.

Insulation

Auch im Anlagenbau ist eine Isolierung wünschenswert. Durch passende Isolation kann zum Beispiel der Verlust von Wärme reduziert werden. Auch der Schutz vor Hitze spielt eine Rolle.

2.1.3 Smartplant 3D

Eine sehr häufig eingesetzte Software, aus dem oberen Preissegment, ist die von Intergraph[10] entwickelte CAD-Anwendung Smartplant 3D[9]. Sie existiert bereits seit über 20 Jahren und wird rund um den Globus verwendet. Der Entwickler, Intergraph[10], bestätigt 2013 den Anwenderkreis von über 16000 Anwender.[13]

Ein weiterer Vorteil ist die Wiederverwendung von Daten, offiziell Model Data Reuse genannt. Diese Funktion ermöglicht den Import aus verschiedenen anderen

etablierten CAD-Anwendungen und bietet so die Möglichkeit, mit den Daten der Partnerunternehmen zu arbeiten. Auf diesem Weg können die Daten später zusammengefügt werden und als ein ganzes betrachtet und bearbeitet werden. Einer der wichtigsten Exportformate ist das binäre Format .VUE. Es bietet die Möglichkeit die Geometrien einer Zeichnung, durch Filter, zu exportieren.

2.2 Konvertierung mit dem VUEParser

Smartplant 3D[9] speichert die geometrischen Daten in ein Dateiformat, dass die Endung .vue besitzt. Es handelt sich hierbei um ein binäres Dateiformat, das nicht öffentlich dokumentiert ist. Betrachtet man die gängigen Produkte aus T.1 und deren unterstützen Dateiformate, so stellt man fest, dass nur die Produkte des Unternehmens Intergraph das Dateiformat unterstützen.

Autodesk Plant 3D[14]	3D Studio (*.3ds) ACIS (*.sat) Autodesk Inventor (*.ipt), (*.iam) CATIA V4 (*.model; *.session; *.exp; *.dlv3) CATIA V5 (*.CATPart; *.CATProduct) DGN (*.dgn) auch mit der Unterstützung von Seed-Dateien FBX (*.fbx) IGES (*.iges; *.igs) JT (*.ij) Parasolid (*.x_b) Parasolid (*.x_t) PDF (*.pdf) Pro/ENGINEER (*.prt*; *.asm*) Pro/ENGINEER Granite (*.g) Pro/ENGINEER Neutral (*.neu) Rhino (*.3dm) SolidWorks (*.prt; *.sldprt; *.asm; *.sldasm) Metafile (*.wmf) STEP (*.ste; *.stp; *.step) Drawing (*.dwg)
-----------------------	--

	IDF an PCF durch ISO2PLANT[15]
TriCAD MS[16]	*.dwg, *.dng, *.dx, *.iges, *.step
AVEVA PDMS[17]	Es konnten keinen Informationen gefunden werden.
AVEVA Everything3D[17]	Es konnten keinen Informationen gefunden werden.
Bentley OpenPlant[18]	<p>Durch den Bentley OpenPlant Modeler V8<i>i</i>:</p> <p>Export: *.dng, *.dwg, *.dx, *.iges, *.cgm, *.stl, *.svg, *.obj, *.u3d</p> <p>Import: *.iges, Parasolids, ACIS *.sat, *.cgm, *.step AP203/AP214, *.stl, Terrain Model Land XML</p> <p>Durch AutoPIPE[19] außerdem:</p> <p>Import:</p> <ul style="list-style-type: none"> AutoPIPE Model (*.dat) Model From STAAD Using PipeLink (*.pipelink) Coordinates (*.xls, *.xlsx, *.txt) AutoPLANT (PXF) OpenPlant Modeler (PXF) Caesar II Neutral (CII) Aveva PDMS (CII) Intergraph PDS (N) ADLPipe (ADI) Smart Plant (PCF) CADWorx (PCF) SolidWorks (PCF) AutoDesk Inventor (PCF) CATIA (PCF) Autodesk Plant 3D (PCF) AutoCAD Koordinaten durch MS Excel Koordinaten von Excel <p>Export:</p> <ul style="list-style-type: none"> AutoPLANT (PXF) - Geometry/Results AutoCAD (DXF) Caesar II neutral (*.cii)

	Isogen (*.pcf) JSpace Model (*.jsm) DGN Model (*.dgn) LiCAD (*.apxt) Export to LiCAD (*.txt) Input Database (*.mdb) Results Database (*.mdb) Nozzle Loads to AutoPIPE Nozzle Nozzle Loads to AutoPIPE Vessel Model to STAAD Using PipeLink (*.pipelink)
Bentley AutoPLANT[20]	<p>Es konnten keinen Informationen zu den direkt unterstützten Formaten gefunden werden.</p> <p>Es gilt die Unterstützung durch AutoPIPE[19], wie bei Bentley OpenPlant[18]</p>
Cadmatic[21]	*.3dd, *.3dc, *.3dp, *.asc, *.pts, *.xyz Point Clouds, *.cii, *.dgn, *.dwg/*.dx (benötigt den Exhanger von AutoCAD), *.ebm, *.ifc, *.jt, *.mdl, *.pcf, *.pst, *.pdf, *.pst, *.rvm, *.txt, *.xls(x), *.xml. Durch den CADMATIC Polytrans eXhanger werden weitere Formate unterstützt.[22]
Intergraph SmartPlant 3D[9]	<p>Es wird der SmartPlant Interop Publisher[23] verwendet, um in verschiedene Formate zu exportieren oder aus ihnen zu importieren. Darunter fällt auch das .vue-Dateiformat.</p>
Intergraph CADWORX[24]	<p>Es wird der SmartPlant Interop Publisher[23] verwendet, um in verschiedene Formate zu exportieren oder aus ihnen zu importieren. Darunter fällt auch das .vue-Dateiformat.</p>
Intergraph PDS[25]	<p>Es wird der SmartPlant Interop Publisher[23] verwendet, um in verschiedene Formate zu exportieren oder aus ihnen zu importieren. Darunter fällt auch das .vue-Dateiformat.</p>

Tabelle T.2 zeigt die Auflistung der gängigen Produkte auf dem Markt und deren unterstützten Dateiformate.

2 Grundlagen 2.2 Konvertierung mit dem VUEParser

Da die Arbeit mit den aus SmartPlant 3D[9] stammenden Daten dennoch erforderlich ist, hat das Unternehmen Unitec Informationssysteme GmbH[26] eine Softwarelösung entwickelt, die das Dateiformat einliest und es zukünftigen Entwicklern über eine C#-API zur Verfügung stellt. In diesem Prozess wird keine Intergraph Software mehr benötigt. Es entfallen also Lizenzkosten und, falls eine Migration zu einer etablierten Anwendung durchgeführt wird, die etablierte Anwendung auch weiterhin verwendet werden kann.

So beinhaltet eine .vue-Datei zum Beispiel ein Primitiv vom Typ *Line3D*. An der passenden Stelle, mit einem Hexeditor betrachtet, zeigt sich folgendes Bild:

4:5010h:	00 00 00 00 00 00 01 00 F6 45 F4 0E C2 57 A3 40
4:5020h:	E9 26 31 08 CC B9 A6 40 99 99 99 99 99 83 83 40
4:5030h:	00 00 00 00 00 00 F0 BF 00 00 00 00 00 00 00 00 00 00 00
4:5040h:	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
4:5050h:	00 E0 4A 71 F8 13 0B 40 30 03 00 00 18 03 00 00
4:5060h:	28 00 00 00 70 00 00 00 7B 31 46 C7 C0 57 A3 40

Abbildung 2: zeigt einen Line3D-Primitiv, blau eingefärbt, mit einem Hexeditor (Editor 010[25]).

Es lässt sich vermuten, dass ein Arbeiten über einen Hexeditor zwar möglich, aber nicht sehr effizient ist, da die Daten jedes Mal neu interpretiert werden müssen.

Der Parser präsentiert die Daten dagegen, menschlich lesbar, als folgende C#-Klasse:

```
public class Line3D : GeometryPrimitive
{
    public Math.Vector3D Position;           // {(2475,87902034143, 2908,8985, 624,45)}
    public Math.Vector3D Direction;          // {(-1, 0, 0)}
    public double StartLength, EndLength; // 0 und 3.3847512103056943
}
```

Da es sich beim VUEParser um ein proprietäres Produkt handelt und das Unternehmen darum bat nicht zu sehr ins Detail zu gehen, wird das Thema VUE nicht vertieft.

2.3 Spiel-Engine

Für die Umsetzung des Plugins wird eine Spiel-Engine verwendet, um die Geometrien anzeigen zu können.

2.3.1 Definition

Unter Spiel-Engine wird ein wiederverwendbares Grundgerüst verstanden, das die wichtigen technischen Grundbausteine enthält, die für die Umsetzung einer Videospielidee benötigt werden. Die Implementierung einer Spiel-Engine wird dabei möglichst abstrakt gehalten, damit auch mehrere Videospielideen damit entwickelt werden können, ohne, dass Änderungen an der Spiel-Engine selbst durchgeführt werden müssen. Vermehrt werden Spiel-Engines auch für die Renderings von Filmen verwendet.[27][28][29] In dem Fall wird sich meist auf die Grafik-Engine beschränkt.

Die Grafik-Engine beschäftigt sich mit der Präsentation auf dem Bildschirm. Damit die Performance möglichst gut ist und die neusten Effekte richtig funktionieren, arbeiten die Entwickler häufig nah mit den Grafikkartenherstellern zusammen.[30] Eine Grafik-Engine arbeitet parallel zur Spiellogik, da sie streng genommen nichts mit dem Spiel zu tun hat.

Ein weiterer wichtiger Teil ist das Soundsystem. Immer wenn Bilder dem Menschen präsentiert werden, spielt auch die Akustik eine mögliche hilfreiche Rolle. So beschrieb z.B. Hansjörg Pauli[31], die Filmmusik als eine Rolle zur möglichen Steigerung des Erfolgs. Damit das Erlebnis des Nutzers möglichst realistisch gehalten wird, verfügen die meisten Soundsysteme über eine 5.1 oder 7.1-Raumklang Unterstützung.

Für Interaktionen mit dem Nutzer steht auch das Steuersystem zur Verfügung. Es nimmt Eingaben des Nutzers entgegen und leitet diese an das oberere Systeme, z.B. das Spiel, weiter. Dabei steht, je nach Projekt, dem Nutzer nicht nur klassisch die Maus und Tastatur zur Verfügung, sondern auch alternative Peripherie wie Joysticks, Gamepads, Leapmotion, und weitere.

Für die Mehrspielererfahrung, in verteilten Systemen, sorgt das Netzwerksystem. Es stellt nicht nur die Verbindung, meist UDP oder TCP, her, sondern sorgt mit verschiedenen Techniken auch für eine Kompensierung der unterschiedlichen

Verbindungen. Auf Grund verschiedener geometrischer Abstände der Verbindungen zum Server, kann die Synchronisation leiden. Es entstehen merkliche Verzögerungen zwischen den einzelnen Spielabläufen. Speziell entwickelte Techniken, genannt Lag Compensation, gehen gezielt gegen dieses Problem vor.

Das Physiksystem ist ein, vor allem in heutigen Spielen, komplexes System zur Berechnung von realistischen physikalischen Bewegungen. Heute reicht es nicht mehr aus, nur einfache Kollisionsabfragen durchzuführen. Gerade durch die sogenannten Physikbeschleuniger, die in Grafikkarten gegossen sind, werden auch komplexe Physik wie z.B. Soft Bodies unterstützt.[32]

2.3.2 Unreal Engine 4

Die Unreal Engine 4, kurz UE4, ist eine der meist eingesetzten Spiel-Engines[33], die auf dem Markt verfügbar sind. Der Sourcecode ist öffentlich verfügbar. Die Nutzung ist solange kostenfrei, bis die Einnahmen 3000€ pro Quartal übersteigen. Ab dieser Summe verlangt der Entwickler, Epic Games, einen Gewinnanteil von 5%.

Ihre Grafik-Engine wird auch in der Filmindustrie eingesetzt. So wurde der Druide K-2S0 aus dem bekannten Film Rogue One: A Star Wars Story mit dem Hilfe der Unreal Engine 4 in Echtzeit gezeichnet. Tim Sweeney, der CEO von Epic Games, beschreibt mit den folgenden Worten den Schritt, weshalb es für Disney möglich war: „*Unreal Engine is the only way to achieve these final, photorealistic pixels in real time. It's the only renderer that does that.*“[34]

In den folgenden Kapiteln wird die Unreal Engine 4 fortan als *Spiel-Engine* bezeichnet.

3 Ausgangslage

Die Ausgangslage beschreibt die fertig entwickelte Software, die für die Entwicklung des Plugins benötigt wird und so als Grundlage dient. Als Frage formuliert: „*Mit welchen vorliegenden Bauteilen wird das Plugin entwickelt?*“

3.1 Unreal Engine 4

Die Spiel-Engine Unreal Engine 4 unterstützt die Nutzung von selbst entwickelten Projekten und Plugins. Da Plugins in beliebigen Projekten verwendet werden können, befasst sich dieser Bereich mit der Entwicklung von Plugins.[35]

3.1.1 Entwicklung von Plugins

Die Erstellung von Plugins kann wahlweise als Blueprint-Plugin oder als CPP-Plugin erfolgen. Dafür öffnet man die Unreal Engine 4 und erstellt ein neues Projekt:

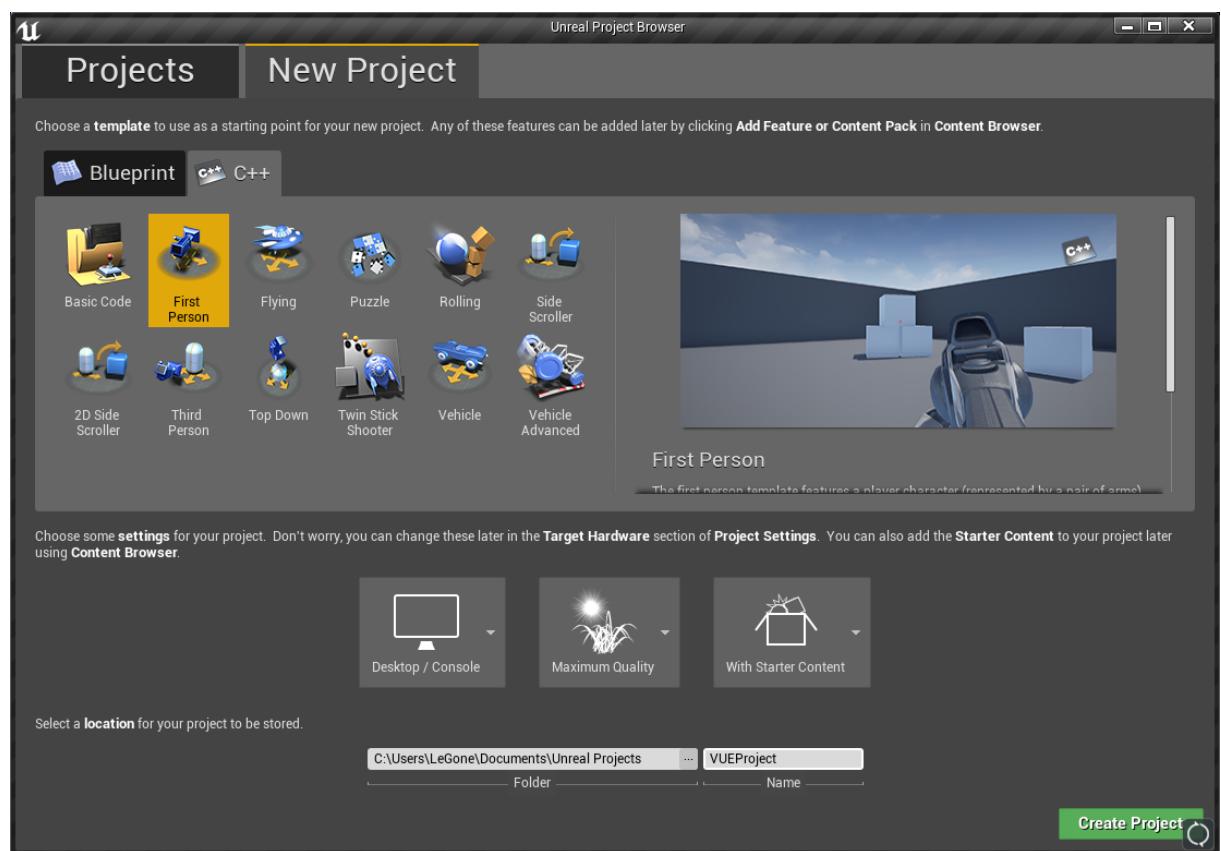


Abbildung 3: Erstellung eines neuen Projekts in der Unreal Engine 4

3 Ausgangslage 3.1 Unreal Engine 4

Nach der Erstellung wird im Editor unter Plugins ein neues Plugin erstellt. Es gibt verschiedene Startmöglichkeiten für neue Plugins.[36]

- Blank
Ein leeres Plugin, mit minimalem Code.
- Content Only
Die Wahl fällt hierauf nur dann, falls das Plugin nur aus Daten bestehen soll.
- Blueprint Library
Wird zur Verwendung von statischen Blueprint Nodes verwendet. Es handelt sich um eine Library für Blueprint-Funktionen.
- Editor Toolbar Button
Erstellt ein Plugin, das einen Schalter in der Toolbar vom Editor erstellt.
- Editor Standalone Window
Erstellt ein Plugin, das einen Schalter in der Toolbar vom Editor erstellt. Mit dem Schalter wird ein leerer Tab geöffnet.
- Editor Mode
Das Plugin ist im Editor-Modus. Zudem inkludiert es ein Toolkit-Beispiel, das eine UI erstellt, die unter dem Modes-Tab erscheint. Die grundlegende UI zeigt beispielhaft, wie mit dem Editor interagiert wird (auch Undo/Redo).
- Third Party Library
Hierbei handelt es sich um ein Plugin, das eine Drittbibliothek inkludiert. Es zeigt genauer wie man eine Library inkludiert, lädt und nutzt.

Damit das Plugin auch kompilieren kann, wird das Windows SDK 8.1 benötigt.[37] [38][39] Der Entwickler der Spiel-Engine, Epic Games[40], setzt bei der Kommentierung auf an JavaDoc[41] angelehnten Kommentaren.[42]

3.1.2 Darstellung von Polymeshes

Mit dem *ProceduralMeshComponent*[43] bietet die Unreal Engine 4 ein Plugin zur Erstellung von Meshes an. Bevor das *ProceduralMeshComponent*-Plugin[43] jedoch verwendet werden kann, muss es dem Projekt hinzugefügt werden.

MyProject.Build.cs:

```
PublicDependencyModuleNames.AddRange
(
    new string[]
    {
        "Core",
        "DesktopPlatform",
        ...
        "ProceduralMeshComponent"
    }
);
```

MyProject.uproject:

```
"AdditionalDependencies":
[
    "Engine",
    "ProceduralMeshComponent"
]
```

Nach dem Einfügen der passenden Header-Datei (`#include "RuntimeMeshComponent.h"`), kann auch schon eine Instanz erstellt werden:

```
Mesh = CreateDefaultSubobject<URuntimeMeshComponent>(TEXT("GeneratedMesh"));
```

Ein Dreieck kann wie folgt erstellt werden:[43]

```
TArray<FVector> Vertices;
Vertices.Add(FVector(0, 0, 0));
Vertices.Add(FVector(0, 100, 0));
Vertices.Add(FVector(0, 0, 100));

TArray<int32> Triangles;
Triangles.Add(0);
Triangles.Add(1);
Triangles.Add(2);

TArray<FVector> Normals;
Normals.Add(FVector(1, 0, 0));
Normals.Add(FVector(1, 0, 0));
Normals.Add(FVector(1, 0, 0));

TArray<FVector2D> UV0;
UV0.Add(FVector2D(0, 0));
UV0.Add(FVector2D(10, 0));
UV0.Add(FVector2D(0, 10));

TArray<FProcMeshTangent> Tangents;
Tangents.Add(FProcMeshTangent(0, 1, 0));
```

3 Ausgangslage 3.1 Unreal Engine 4

```
Tangents.Add(FProcMeshTangent(0, 1, 0));
Tangents.Add(FProcMeshTangent(0, 1, 0));

TArray<FLinearColor> VertexColors;
VertexColors.Add(FLinearColor(0.75, 0.75, 0.75, 1.0));
VertexColors.Add(FLinearColor(0.75, 0.75, 0.75, 1.0));
VertexColors.Add(FLinearColor(0.75, 0.75, 0.75, 1.0));

// 0 for the section, true for creating the collision
Mesh->CreateMeshSection_LinearColor(0, Vertices, Triangles, Normals, UV0,
                                     VertexColors, Tangents, true);

// Enable collision data
Mesh->ContainsPhysicsTriMeshData(true);
```

3.1.3 Anlegen von Materialien

In der Unreal Engine 4 werden Materialien mit dem Typ **UMaterial** behandelt.

UMaterial wird einem Mesh zugewiesen, um dessen Aussehen zu beeinflussen.[44]

Die Dokumentation gibt hier aber eine Warnung: *Ein neues Material zu erstellen, beeinflusst die Zeit negativ, die ein Shader zur Kompilierung benötigt. Stattdessen wird empfohlen, auf eine dynamische Instanz auszuweichen.*[44]

Mit der Warnung wird auf **UMaterialInstanceDynamic** angespielt. Damit lassen sich Instanzen von vorhandenen Shadern erstellen, ohne, dass der Shader neu kompiliert werden muss. Das Problem: Die Kompilierung eines Shaders muss immer vor der Laufzeit durchgeführt werden.[45] Durch die dynamische Instanz können aber Veränderungen und Berechnungen auch *während* der Laufzeit vorgenommen werden.[45] Hingegen Instanzen vom Typ **UMaterialInstanceConstant** nur innerhalb des Editors angelegt und verändert werden können, da sie vor der Laufzeit berechnet werden.[45]

Die Instanziierung eines dynamischen Materials könnte, um ein Beispiel zu nennen, über ein vorhandenes **UMaterial** aus dem Contentbrowser geschehen. Dafür würde man die Blueprint-Referenz eines vorhandenen Materials verwenden und eine dynamische Instanz daraus generieren:

```
BaseMaterial = Cast<UMaterial>(StaticLoadObject(UMaterial::StaticClass(), NULL,
                                                TEXT("Material'/Game/VUEImporter/VUEMaterial.VUEMaterial'")));

DynamicMaterial = UMaterialInstanceDynamic::Create(BaseMaterial, NULL);
```

Die Referenz **"Material'/Game/VUEImporter/VUEMaterial.VUEMaterial'"** steht für den relativen Blueprintpfad, zum bereits vorhandenen Material. Die zurückgegebene Instanz vom Typ **UMaterialInstanceDynamic** kann nun beliebig manipuliert werden.

3.2 Arbeiten mit dem VUEParser

Der VUEParser liefert, wie bereits angesprochen, die Daten aus dem VUE-Dateiformat. Er liegt als managed C# Library vor und besitzt zudem einen Wrapper, der als Brücke zwischen unmanaged C++ und managed C# fungiert. Wie wir festgestellt haben, werden Plugins der Unreal Engine 4 in C++ entwickelt. Daher wird an dieser Stelle auf den Wrapper eingegangen und nicht auf den Parser selbst.

Nach der Einbindung der Bibliothek wird eine Instanz des Parsers vom Typ `vUEParser` instanziert:

```
VUEParserW::VUEParser* AVUEParser = new VUEParserW::VUEParser();
```

Es ist möglich bei der Instanziierung einen Wert namens `LinearMemorySize` zu übergeben. Dieser Wert wird nur dann benötigt, wenn die Verwendung des linearen Allokierers gewünscht ist. Der lineare Allokierer ist Teil des Wrappers und ersetzt den Standard-Allokierer. Dafür muss der Wrapper mit `LINEAR_ALLOCATION` kompiliert werden.

Den Prozess zum Einlesen und Parsen startet man mit:

```
HRESULT Result = AVUEParser->Parse(L"D:\\test1.vue");
```

Die Methode Parse benötigt einen String vom Typ `LPCWSTR`. Der Rückgabewert muss geprüft werden, da der Parser auch die Lizenzprüfung übernimmt und den Status über den Wrapper weiterreicht.

Die möglichen Rückgabewerte sind:

`S_OK`, wenn alles in Ordnung war.

`E_ABORT`, falls der Parser selbst nicht vom Wrapper instanziert werden konnte.

`E_ACCESSDENIED`, beschreiben den Fehler, wenn die Lizenzprüfung fehlerhaft verlief.

Nach erfolgreichem Parsing kann nun auf die Daten aus der .vue-Datei zugegriffen werden.

Danach steht die folgende Hierarchie zur Verfügung:

- Header
- Materials
- Container
 - Gruppe
 - Primitive

4 Anforderungsbeschreibung

Dieses Kapitel dient dazu, die Anforderungen an das Plugin näher zu definieren.

Als Frage ausgedrückt: „*Welche Punkte sollen später in der Software umgesetzt sein?*“

4.1 Funktionale Anforderungen

Die funktionalen Anforderungen werden in diesem Unterkapitel als kurze signifikante Aufgaben zusammengefasst.

4.1.1 Selektieren einer Datei

Der Anwender muss die Datei selektieren können, die er importieren möchte. Das Selektieren soll über eine Oberfläche erfolgen und den Import anstoßen.

4.1.2 Übernahme der Metadaten

Die Daten aus dem VUE-Header müssen einsehbar sein. Sie beinhalten wichtige Daten über die Datei.

4.1.3 Platzieren der Geometrie

Die geometrischen Primitive, die der Parser zur Verfügung stellt, müssen übernommen werden. Dabei sollte die vorgegebene Hierarchie *Container*→*Gruppe*→*Primitive* erhalten bleiben. Damit die geometrischen Daten auch passend skaliert werden können, muss eine editierbare Variable zur Verfügung stehen.

4.1.4 Unterstützung von Obstruction Volumes

Die in Intergraph SmartPlant 3D[9] vorkommenden Obstruction Volumes heißen Display Aspects. Da Obstruction Volumes ein wichtiger Bestandteil des Planungsprozesses sind, gilt es diese zu übernehmen.

- SimplePhysical
- DetailedPhysical
- Insulation
- Operation
- Maintenance
- ReferenceGeometry
- Centerline

4.1.5 Übernahme der Materialien

Materialien können sowohl in den Container, als auch in den Primitiven vorkommen. Beide Varianten müssen unterstützt werden. Die Materialien in den Primitiven, falls vorhanden, überschreiben die des Containers.

4.1.6 Selektieren der importierten Geometrie für Informationen

Die aus der VUE-Datei stammenden Daten eines Containers, müssen später durch Anklicken einsehbar sein. Darunter fallen auch Werte wie der *Container Type* und der *Definition Type*.

4.1.7 Nicht konvertierte Geometrie kenntlich machen

Jede nicht erfolgreich konvertierte oder unterstützte Geometrie, sollte über eine Boundingbox dargestellt werden können. Der Anwender soll die Möglichkeit haben diese Funktion abzuschalten.

4.1.8 Bewegung des Anwenders innerhalb der Anlage ermöglichen

Damit sich der Anwender die Anlage betrachten kann, ist es notwendig, dass er sich innerhalb der Anlage frei bewegen kann. Frei bewegen bedeutet, dass er sowohl fliegen, als auch laufen können muss.

4.2 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen stellen die Kriterien da, die allgemein gehalten für die Qualität sorgen.

4.2.1 Benutzeroberfläche

Die Benutzeroberfläche (UI) sollte ansprechend und verständlich gestaltet werden. So verständlich, dass sich sowohl erfahrene CAD-Anwender schnell einfinden, als auch Unerfahrene.

4.2.2 Stabilität

Fehler, die durch den Parser oder dem Import entstehen, sollten dem Benutzer klar ersichtlich angezeigt werden. Dadurch wird verhindert, dass fehlerhafte Bauteile an die Hersteller geleitet werden, da sie nicht vorher im Planungsprozess entdeckt wurden.

4.2.3 Performance

Das zu entwickelnde Plugin soll auch größere Zeichnungen, z.B. über 1000 Container, mit Bildwiederholungsrate von mindestens 60 anzeigen können. Da diese Bedingung abhängig von der verwendeten Hardware ist, wird die empfohlene Zusammensetzung für die Unreal Engine 4 verwendet:[46]

- Desktop PC oder Mac
- Windows 7 64-bit oder Mac OS X 10.9.2 oder schneller
- Quad-core Intel oder AMD processor, 2.5 GHz oder schneller
- NVIDIA GeForce 470 GTX oder AMD Radeon 6870 HD Grafikkarte oder höher
- 8 GB RAM

4.2.4 Genauigkeit

Da es sich bei CAD-Daten auch um Daten handelt, die nicht selten an externe Hersteller übergeben werden, ist es notwendig, dass die Genauigkeit durch den Konvertierungsprozess nicht negativ belastet wird. Es ist daher auf den Maßstab der Quell- und Zielsoftware zu achten.

4.2.5 Virtual-Reality

Das zu entwickelnde Plugin sollte über eine Unterstützung von Virtual Reality verfügen.

4.2.6 Kosten

Ein sehr wichtiger Aspekt, der aus der Problemstellung hervorgeht, sind die Kosten der vorgegebenen SmartPlant 3D[9] Software. Daraus entwickelt sich die Anforderung, dass die summierten Kosten unterhalb der SmartPlant 3D[9] Lizenzkosten bleiben müssen. Zur Summe der Kosten zählen die Kosten des VUEParsers, die Kosten der Spiel-Engine, und die des Plugins selbst.

5 Konzeption

In diesem Kapitel wird das Konzept beschrieben, wie die Anwendung zu implementieren ist. Dafür werden die vorherigen Kapitel als Grundlage genutzt. Damit das Konzept auch für andere Spiel-Engines verwendet werden kann, wird es möglichst abstrakt gehalten.

Als Frage formuliert: “Wie ist die neue Anwendung zu implementieren?”

5.1 Konzeption der funktionalen Anforderungen

In den folgenden Unterkapiteln wird auf das Konzept eingegangen, dass sich aus den notwendigen funktionalen Anforderungen entwickelt, die aus dem vorherigen Kapitel Anforderungsbeschreibung stammen.

Alle angelegten Objekte sollen passend benannt werden.

5.1.1 Selektieren einer Datei

Die Selektion einer Datei soll über ein Durchsuchen-Fenster erfolgen, so wie in Abbildung 4 gezeigt:

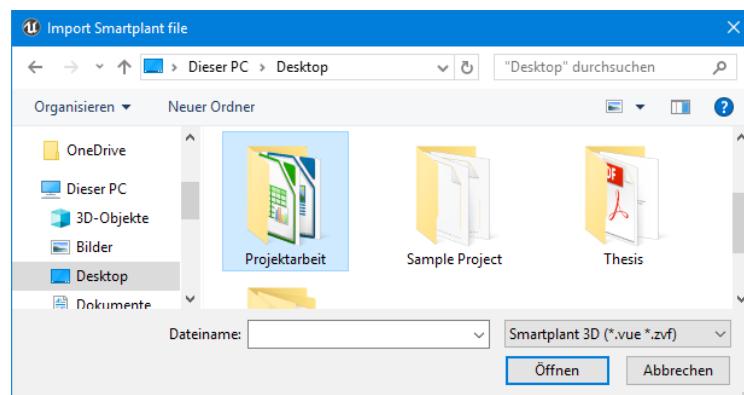


Abbildung 4: Zeigt einen möglichen OpenFileDialog

Die ausgewählte Datei soll danach an den VUEParser übergeben und konvertiert werden. Einen solchen OpenFileDialog bietet die Spiel-Engine selbst. Gefunden wird er unter IDesktopPlatform::OpenFileDialog.

5.1.2 Übernahme der Metadaten

Der Wrapper bietet die Metadaten ein Mal für den Header und dann noch für jeden Container. Für die Header-Daten soll ein neues Objekt erstellt werden, dass in der Welt platziert wird und alle Daten enthält. Für jeden Container soll dies ebenfalls passieren. Allerdings sollte der Header den Namen der Datei haben und die Container die passenden UIDs.

5.1.3 Platzieren der Geometrie

Jeder Container besitzt Gruppen, jede Gruppe besitzt Primitive. Damit nicht jedes Primitive einzeln in die Welt gesetzt werden muss, bietet das ProceduralMeshComponent die Möglichkeit, Sektionen zu erstellen. Damit die Positionsdaten und Längen auch später noch passen, müssen sie passend skaliert werden. Bei der Spiel-Engine ist eine Einheit 1 cm und bei den vom VUEParser stammenden Daten 1 mm. Ein Slider in der Oberfläche soll hier für eine variable Lösung sorgen. Vor allem müssen die Container passend benannt werden. Dafür ist die gegebene UID zu verwenden. Damit die Qualität der Geometrie angepasst werden kann, soll die Anzahl der Eckpunkte variable gehalten werden.

Torus

Das Zeichnen des Torus beginnt mit der Position, schwarz gezeichnet in Abbildung 5, in Verbindung mit einem schwarz gezeichneten Richtungsvektor. Rotiert man um diesen Richtungsvektor, im Abstand des Radius, erhält man die grün gezeichneten Punkte auf dem Kreis. Rotiert man zudem die zweite Orientierung mit, erhält man die grün gezeichneten Richtungsvektoren. Mit der neuen Position und der neuen Orientierung, kann man nun die Kreis-Eckpunkte zeichnen, die später miteinander verbunden werden. Dafür wird als Radius die Dicke verwendet. Die Eckpunkte sind rot eingefärbt.

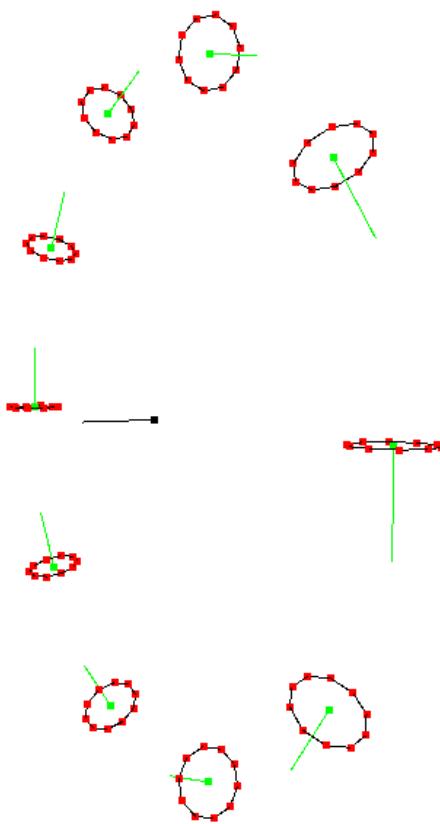


Abbildung 5: Skizze des Torus

Revolution

Bei diesem Konzept wird sich auf die elliptische Grundfläche begrenzt. Die Triangulierung ist ähnlich dem Torus, mit dem Unterschied, dass beim Revolution nicht um 360° gedreht wird, sondern mit einem gegebenen Winkel.

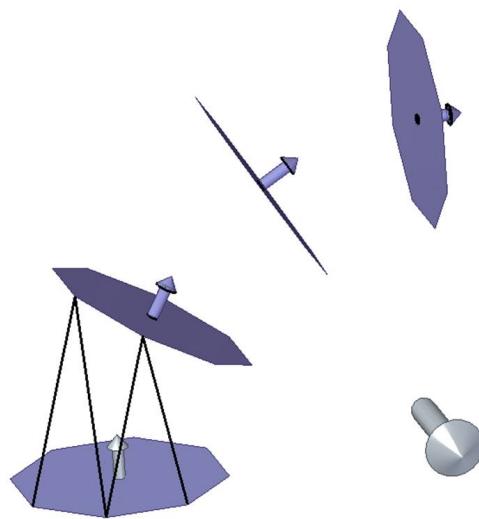


Abbildung 6: Zeigt Revolution per Skizze



Abbildung 7:
Projection als
Skizze

Die Kugel wird mit einem Mittelpunkt und einem Radius gezeichnet. Dafür sollen zweidimensionale Kreise gezeichnet werden, die per Transformation passend gedreht und skaliert werden. In Abbildung 8 ist der Mittelpunkt grün gezeichnet.

Projection

Das Konzept bezieht sich auf die spätere Implementierung einer projizierten elliptischen Fläche. Für die elliptische Grundfläche wird eine Position verwendet und der grün eingezeichnete Richtungsvektor, um den der Kreis gezeichnet wird. Die Extrusion wird mit einem Richtungsvektor und einer Länge beschrieben. Zuletzt werden wird die Grundfläche als Deckel erneut gezeichnet. Hier ist darauf zu achten, dass die Indizierung des Deckels in gegenteiliger Reihenfolge abläuft, als die des Bodens. Der Boden muss so gezeichnet werden, dass er von Unten sichtbar ist. Für die Indizierung sind die Eckpunkte der Grundflächen zu verwenden.

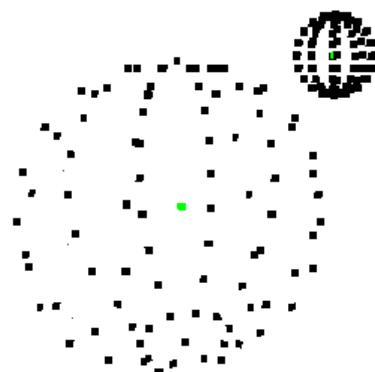


Abbildung 8: Zeigt
Eckpunkte der Sphere

Point

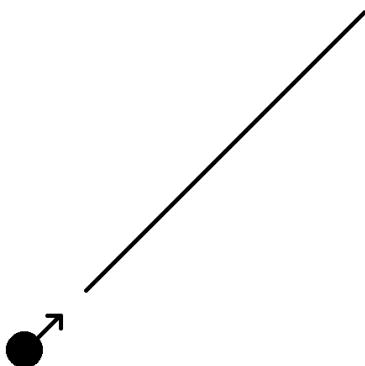
Der Punkt wird mit Hilfe der Position gezeichnet.



Er besitzt keine weiteren Eigenschaften.

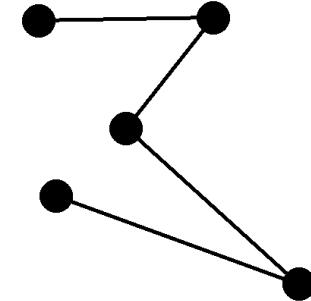
Line

Die Linie wird mit von einer Position in eine Richtung gezogen. Die Startlänge beschreibt, wann die Linie anfängt und die Endlänge, sobald sie endet.



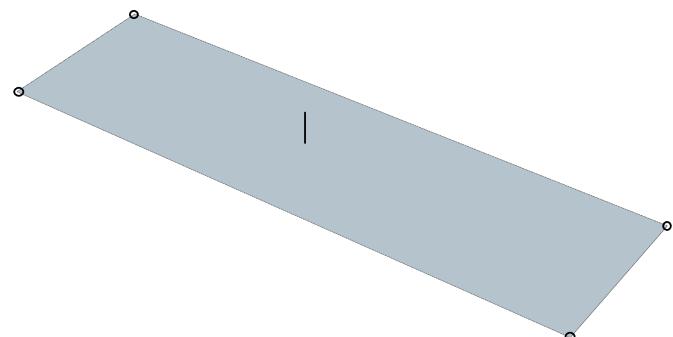
LineString

Als Kette von *Lines*, wird der LineString als aneinander nicht notwendigerweise geschlossene, Linien gezeichnet. Jede Linie verfügt demnach über die selben Eigenschaften, wie die *Line* des vorherigen Typs.



Plane

Als Polygonfläche nutzt der Plane nicht nur Ecken vom Typ Point, sondern auch andere Typen. In diesem Konzept werden allerdings nur die Point-Ecken unterstützt, da die Mehrheit der Primitive in den Testdateien Point-Ecken verwenden.



Das zu erzeugende Polygon ist generell geschlossen und muss mit einem passenden Algorithmus trianguliert werden. Durch die gegebene Normale können auch Algorithmen der zweidimensionalen Triangulierung durchgeführt werden, wenn die Geometrie passend auf eine Ebene reduziert wird.

5.1.4 Unterstützung von Obstruction Volumes

Damit die Obstruction Volumes später auch einsehbar sind, sollen sie an das Container-Objekt angeheftet werden. Die Obstruction Volumes können für jede Gruppe anders sein. Daher sind sie im VUEParser ebenfalls Teil des Gruppen-Objekts und sitzen hier in dem Feld `DisplayAspect`.

Die folgenden Obstruction Volumes sollen keine Kollisionsabfrage besitzen:

- Maintenance
- Operation
- Insulation
- CenterLine

5.1.5 Übernahme der Materialien

Jeder Container, aber auch jedes Primitiv, kann ein eigenes Material besitzen. So soll es auch in unserem Plugin umgesetzt werden. Dafür werden die in der Ausgangslage beschriebene *Anlegen von Materialien* Programmsnipsel verwendet, um neue Materialien anzulegen. Da der VUEParser aber auch ein globales Array mit allen Materialien besitzt und jedes Objekt nur eine Referenz zu diesem Array besitzt, soll das in unserem Plugin ebenfalls umgesetzt werden. Dafür werden vor der Erstellung der Geometrie alle Materialien erstellt und in einem Array gespeichert. Später bekommen die Geometrie dann eine passende Referenz zum verwendeten Material.

5.1.6 Selektieren der importierten Geometrie für Informationen

Damit die Metadaten auch angezeigt werden können, wird der Details-Tab des Editors verwendet. Dort sollen die Daten gelistet werden, wie auch viele allgemeine Eigenschaften dort bereits gelistet werden. Eine neue Rubrik *VUE* gilt als passende Gruppierung. Würde man auf das passende geometrische Bauteil klicken, wären die Daten somit einsehbar und die Anforderung umgesetzt.

5.1.7 Nicht konvertierte Geometrie kenntlich machen

Sobald ein Fehler auftreten sollte oder das Primitiv nicht unterstützt werden, soll die Boundingbox mit einem speziellen Material gezeigt werden. Geschieht ein Fehler auf Gruppenebene, wird die Boundingbox der Gruppe verwendet. Bei einem Fehler auf Containerebene, passend die Boundingbox des Containers. Das Material der später platzierten Boundingbox soll orange leuchten und transparent sein, damit es sich kenntlich von anderer Geometrie abhebt. Dazu soll ein Reiter in der Oberfläche dafür sorgen, dass dieses Feature eingeschalten und ausgeschaltet werden kann.

5.1.8 Bewegung des Anwenders innerhalb der Anlage ermöglichen

Die vom Editor verfügbaren Steuerelemente werden für diese Anforderung ausreichen. Dadurch kann der Anwender bereits durch die Anlage fliegen.

5.2 Konzeption nicht-funktionaler Anforderungen

Dieses Unterkapitel beschäftigt sich mit der Konzeption der nicht-funktionalen Anforderungen.

5.2.1 Benutzeroberfläche

Die Benutzeroberfläche sollte einen eigenen Tabulator im Editor erhalten. Dort sollen die einzelnen Steuerelemente gelistet sein. Es befindet sich dort der Import-Schalter und die in dem vorherigen Unterkapitel der *Konzeption der funktionalen Anforderungen* geplanten Slider und Checkboxen.

5.2.2 Linux-Unterstützung

Die Unreal Engine 4 unterstützt die Linux-Plattform.[47][48] Der folgende Test zeigt, dass der VUEParser ebenfalls unter dieser Plattform laufen könnte.

Da es sich bei dem VUEParser um eine .Net-4.0-Anwendung handelt, kommt hierbei Wine in der Verbindung mit Wine-Mono zum Einsatz. Wine ist hier möglicherweise optional, denn es wird nur für native Aufrufe verwendet. Allerdings benötigen manche .NET-Aufrufe auch native Funktionen.

Wine[49] ist für die Ausführung von Windows-Applikationen unter POSIX-konformen Betriebssystemen gedacht. Es arbeitet als eine Kompatibilitätsschicht. Anders als virtuelle Maschinen oder Emulatoren, wird hier die Windows-Logik nicht simuliert, sondern übersetzt. Wine kann die Windows-API-Aufrufe in Echtzeit zu den entsprechenden POSIX-Aufrufen übersetzen. Dadurch entstehen keine unnötigen Performance- und Speichereinbußen.

Wine-Mono[50] wird für die Simulierung des .NET-Frameworks verwendet. Hier ist darauf zu achten, dass es sich um das Wine-Mono-Paket handelt. Es gibt auch eine entkoppelte Variante, die hier keine Verwendung findet.

Damit möglichst schnell zwischen den einzelnen Installationen (Konfigurationen) gewechselt werden kann, wird PlayOnLinux verwendet.

PlayOnLinux[51] ist eine Software, womit ganz einfach Spiele und Programme, die normalerweise nur unter Windows laufen, installiert und verwendet werden können.

Nach der erfolgreichen Installation von .Net-4.0, startet die Software auf Anhieb. Das Parsen läuft erfolgreich durch, verabschiedet sich allerdings mit einem Absturz.

```
Unhandled Exception: System.ComponentModel.Win32Exception: No associated application
  at System.Diagnostics.Process.StartWithShellExecuteEx(ProcessStartInfo startInfo)
  at System.Diagnostics.Process.Start()
  at System.Diagnostics.Process.Start(ProcessStartInfo startInfo)
  at System.Diagnostics.Process.Start(String fileName)
  at ...Viewer.MainWindow.Parse..._Click(...)
```

Dieser Fehler entsteht, da Wine versucht – über den Editor – die Logdatei anzuzeigen. Da jener Editor nicht geöffnet werden kann, folgt der bedingte Absturz. Die Logdatei selbst wurde erfolgreich geschrieben. Das Ergebnis ist daher dennoch als positiv anzusehen, denn die tatsächliche Funktionalität der Software ist gegeben.

6 Implementierung

Dieses Kapitel beschäftigt sich mit der konkreten Implementierung der im vorherigen Kapitel beschriebenen Konzepte. Dazu wird zunächst die Grundlage des Plugins geschaffen und danach die einzelnen Teile des Konzepts übernommen.

Als Frage formuliert: „Wie wird die Software tatsächlich umgesetzt?“

6.1 Erschaffung der Grundlage

Das neu zu erstellende *Standalone Window Plugin* dient als erste Anlaufstelle, um eine Grundlage für das zu entwickelnde Plugin zu schaffen. Da die Entwicklung unter *Windows 10* und mit der IDE *Visual Studio 2017[52]* geschieht, ist das installierte *Windows SDK[53]* in der Version 10 installiert. Erst nach der Installation des benötigten Windows SDKs 8.1 ist eine Kompilierung der Spiel-Engine möglich.

Der VUEParser ist schnell eingebunden. Bei der ersten Verwendung des Parsers aber, stürzt die Spiel-Engine ab. Die Suche nach dem Fehler erweist sich als schwierig, da keine aussagekräftige Fehlermeldung erscheint. Der Debugger zeigt aber hingegen, dass bei der Instanziierung des VUEParsers stehen geblieben wird. So kommt die Vermutung auf, dass die notwendige .dll nicht gefunden wird. Bei einem Versuch, das Problem zu lösen, werden die beiden Dateien (*VUEParserCPPWrapper.dll* und *VUEParser.dll*) in das Verzeichnis kopiert, worin sich auch die Runtime der Spiel-Engine befindet. Da dadurch das Problem behoben wird, zeigt sich, dass tatsächlich nicht die passenden Dateien gefunden werden. Es ist allerdings wünschenswert, dass sich alle Dateien des Plugins innerhalb des Plugin-Verzeichnis befinden. Die Spiel-Engine bietet hier die Möglichkeit die Bibliotheken manuell zu laden:

```
LibraryPath = FPaths::Combine
(
    *BaseDir,
    TEXT("/Source/ThirdParty/VUEParser/Binaries/VUEParserCPPWrapper.dll")
);
LoadLibrary(*LibraryPath);

LibraryPath = Fpaths::Combine
(
    *BaseDir,
    TEXT("/Source/ThirdParty/EarClipper/Binaries/EarClipperWrapper.dll")
);
LoadLibrary(*LibraryPath);
```

Dadurch werden die Wrapper nun auch im Plugin-Verzeichnis gefunden.

Leider gilt diese Änderung nicht für die vom Wrapper geladenen Assemblies. Diese werden nach wie vor nicht im Plugin-Verzeichnis gefunden. Das Problem: Windows sucht nach der Assembly nicht in dem Verzeichnis, von der referenziert wird, sondern von dem die Anwendung ausgeführt wird.[54][55][56] Es ist möglich ein passendes Event abzufangen und die Suche manuell zu verändern.[56]

Das AssemblyResolve-Event wird immer dann ausgelöst, wenn die Common Language Runtime versucht eine Assembly einzubinden, es aber nicht gelingt. Also müssen wir unseren eigenen Event-Handler entwickeln:

```
System::Reflection::Assembly^ currentDomain_AssemblyResolve(Object^ sender,  
ResolveEventArgs^ args)  
{  
    // Load the assembly from the specified path  
    String^ finalPath = nullptr;  
    try  
    {  
        finalPath = gcnew String("Source/ThirdParty/EarClipper/Binaries/")  
            + args->Name->Substring(0, args->Name->IndexOf(',')) + ".dll";  
  
        System::Reflection::Assembly^ retval =  
            System::Reflection::Assembly::LoadFrom(finalPath);  
  
        return retval;  
    }  
    catch (...)  
    {  
    }  
  
    return nullptr;  
}
```

Damit die Common Language Runtime auch weiß, wo sich unser Event-Handler befindet:

```
AppDomain::CurrentDomain->AssemblyResolve += gcnew ResolveEventHandler(AssemblyResolve);
```

Dieser „Workaround“ kommt für den VUEParser allerdings nicht in Frage, da es sich um kein öffentlich zugängliches Projekt handelt und nur die API zur Verfügung steht. Nachdem dieser Punkt abgearbeitet ist, kann mit der weiteren Entwicklung begonnen werden. Es stellt sich heraus, dass der Charakter zu breit ist. Zur Kollisionsabfrage wird hier eine Kapsel verwendet. Die passende Stelle ist schnell gefunden und der Wert für die Breite 55.f wird auf **45.f** reduziert. 96.f steht für die Höhe:

```
GetCapsuleComponent()->InitCapsuleSize(55.f, 96.f); // 55.f to 45.f
```

6.2 Implementierung der funktionalen Anforderungen

Dieses Kapitel beschäftigt sich mit den implementierten funktionalen Anforderungen.

Da sowohl der Header-Actor, als auch die Container-Actors, laut Konzept passend benannt werden sollen, wird die Methode Rename vom Actor verwendet:

```
VUEHeader->Rename(*HeaderName);
```

Wie sich herausstellt, führt dies zu keiner Änderung. Nach einer Suche stellt sich heraus, dass das passende Label unbenannt werden muss:

```
VUEHeader->SetActorLabel(*HeaderName);
```

Zur Erinnerung: Der Header-Actor ist das Objekt in der Spiel-Engine, dass die Metadaten des Headers beinhaltet. Als Container-Actor wird das erstellte Objekt genannt, dass die Daten des Containers beinhaltet.

6.2.1 Selektieren einer Datei

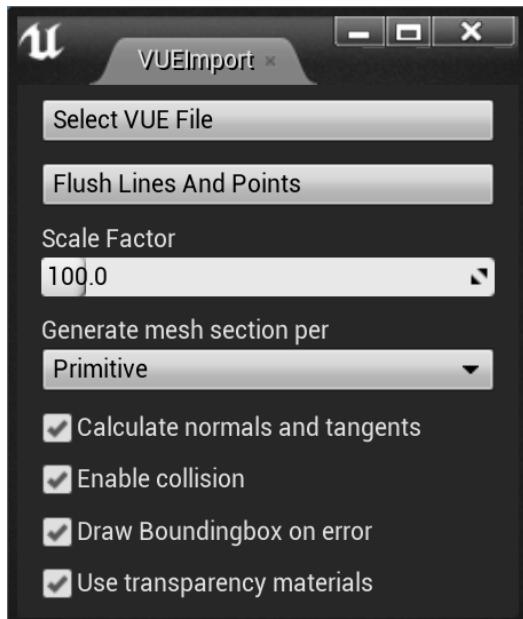


Abbildung 9: UI mit Button

Die Selektion einer Datei und der darauf folgende Import, ist einer der wesentlichen Funktionen, die ein Anwender vermutlich häufig verwenden wird. Daher stehen sie in der Oberfläche an der ersten Stelle. Die Oberfläche für den Import-Schalter ist schnell geschaffen:

```
SNew(SButton)
    .Text(FText::FromString("Select VUE File"))
    .OnClicked_Raw(this,
        &FVUEImporterModule::OnClickedImportButton)
```

OnClickedImportButton wird aufgerufen, sobald ein Anwender den Schalter drückt. Damit der Anwender dann ein Dateiauswahlfenster erhält, bietet die Spiel-Engine ein OpenFileDialog an. Der folgende Programmcode zeigt die umgesetzte Implementierung im Plugin:

6 Implementierung 6.2 Implementierung der funktionalen Anforderungen

```
bool VUEHelper::OpenVUEFileDialog(TArray<FString>& OutFiles)
{
    void* ParentWindowPtr = FslateApplication::Get().
        GetActiveTopLevelWindow()>GetNativeWindow()>GetOSWindowHandle();
    return (FDesktopPlatformModule::Get()>OpenFileDialog
    (
        ParentWindowPtr,
        TEXT("Import Smartplant file"),
        TEXT("C:/VUEFiles"),
        TEXT("Test.vue"),
        TEXT("Smartplant 3D (*.vue *.zvf)|*.vue;*.zvf"),
        EFileDialogFlags::None,
        OutFiles
    )));
}
```

Nachdem die Datei selektiert ist und `OutFiles` den Dateinamen enthält, muss der Dateiname noch an den VUEParser übergeben werden und der Parsevorgang gestartet werden. Dafür wird die Klasse `VUEImporter` implementiert. Sie enthält den Programmcode, der mit dem Konvertierungsprozess zu tun hat. Die Methode `ImportVUEFile` nimmt den Dateinamen entgegen und spricht dann den VUEParser an.

```
void VUEImporter::ImportVUEFile(FString Filename)
{
    VUEParserW::VUEParser VUEParser;
    VUEParser.Parse(*Filename);
    ...
}
```

Danach kann mit den Daten des VUEParsers gearbeitet werden.

6.2.2 Platzieren der Geometrie

Die Triangulierung der Geometrie funktioniert für jeden Primitiv unterschiedlich. Alle Eckpunkte müssen wegen der Skalierung passend multipliziert werden. Generell gilt bei allen Primitiven, dass zuerst die Eckpunkte berechnet werden müssen und danach, falls notwendig, die Indizes. Da die Geometrie erst sichtbar wird wenn auch die korrekten Indizes gesetzt sind, werden Debug-Linien und Debug-Punkte als Hilfestellung gezeichnet:

```
ULineBatchComponent* LineBatcher = GetDebugLineBatcher(World, true, -1.f, false);
if (LineBatcher)
{
    // Draw debug line
    LineBatcher->DrawLine(Pos1, Pos2, FLinearColor::Green, 0, 0, -1.f);

    // Draw debug point
    LineBatcher->DrawPoint(Pos1, FLinearColor::Green, 5.0f, 0, -1.f);
}
```

Die Linien und Punkte sind nämlich auch dann sichtbar, wenn noch keine Indizes gesetzt sind. Wobei die Primitive Line, LineString und Point sowieso ohne Geometrien auskommen. Hier reichen die oben aufgeführten Zeichnungen von Linien und Punkten vollkommen aus. Es gilt aber zu erwähnen, dass DrawPoint keine Punkte zeichnet, sondern Quadrate.

Wie sich herausstellt gibt es eine verbesserte Bibliothek zur Erstellung von prozeduralen Meshes. Die besagte Bibliothek, entwickelt von Chris Conway[57], nennt sich RuntimeMeshComponent[58], kurz RMC. Für uns ist vor allem die automatische Generierung von Normalen und Tangenten interessant, denn die werden für die Lichtberechnung benötigt. Zwar beschreibt der Entwickler der Bibliothek die Geschwindigkeit der automatischen Generierung als nicht optimal[58], aber das stört uns nicht, da nicht die Schnelligkeit des Imports entscheidend ist, sondern die Bildwiederholungsrate zur Laufzeit.

6 Implementierung 6.2 Implementierung der funktionalen Anforderungen

Im folgenden Abschnitt wird näher auf die Triangulierung der einzelnen umzusetzenden Primitive eingegangen:

Plane

Der Plane wird als Polygon mit endlichen Ecken beschrieben. Gegeben ist die Liste der Eckpunkte und weitere nicht dokumentierte Vektoren. Damit für jede Anzahl von Eckpunkten eine passende Triangulierung durchgeführt werden kann, wird die API der Spiel-Engine nach fertigen Implementierung durchsucht. Die API scheint hier auch eine mögliche Triangulierung von Polygonen zu bieten. Die Implementierung sieht wie folgt aus:

```
void VUETriangulation::TriangulatePolygon(UWorld* World, TArray<FVector>& InVertices,
                                         TArray<int32>& InTriangles)
{
    TArray<FVector> OutVertices;

    FPoly NewPolygon;

    for (int Index = 0; Index < InVertices.Num(); Index++)
    {
        NewPolygon.InsertVertex(Index, InVertices[Index]);
    }

    TArray<FPoly> OutPolys;

    // I do not know what GetDefaultBrush means ---
    NewPolygon.Triangulate(World->GetDefaultBrush(), OutPolys);

    for (int Index = 0; Index < OutPolys.Num(); Index++)
    {
        OutVertices.Add((FVector)OutPolys[Index].Vertices[0]);
        OutVertices.Add((FVector)OutPolys[Index].Vertices[1]);
        OutVertices.Add((FVector)OutPolys[Index].Vertices[2]);

        InTriangles.Add(InTriangles.Num() - 1);
        InTriangles.Add(InTriangles.Num() - 1);
        InTriangles.Add(InTriangles.Num() - 1);
    }

    // InVertices contains the result as a list of points
    InVertices = OutVertices;
}
```

Es stellt sich allerdings heraus, dass die Implementierung nicht das ist, was benötigt wird. So werden in Tests fast keine Polygone erfolgreich trianguliert. Durch die Notwendigkeit der Triangulierung, muss ein Algorithmus und eine dazu passende Bibliothek gefunden werden. Die Wahl fällt auf den Ear Clipping Algorithmus[59], implementiert in der earclipper-Library[60] von Martin Ennemoser[61]. Da die Bibliothek in managed C# geschrieben ist, sorgt ein für das Plugin entwickelter

Wrapper, für die notwendige Brücke.

Aber warum Earclipping und die gewählte Bibliothek?

Natürlich gibt es noch alternative Algorithmen wie den von Delaunay[62]. Allerdings wird nicht primär nach einem Algorithmus gesucht, sondern nach einer einfach zu bedienenden Bibliothek. Zudem unterstützen viele der gefundenen Lösungen nicht direkt die dreidimensionale Triangulierung. Die gewählte Bibliothek benötigt dafür nur eine Liste von Eckpunkten und eine Normale, die die Richtung beschreibt, von der trianguliert werden soll. Wie sich herausstellt, ist einer der noch unbekannten Werte des Plane-Primitives die besagte notwendige Normale. Der resultierende Programmcode sieht wie folgt aus:

```
void VUETriangulation::TriangulatePolygon(TArray<FVector>& InVertices, TArray<int32>& OutIndices, FVector Normal)
{
    EarClipperW::EarClipper EarClipper;

    // Feed the EarClipper library with our vertices.
    for (int Index = 0; Index < InVertices.Num(); Index++)
    {
        FVector& Vertex = InVertices[Index];
        EarClipper.AddVertex(Vertex.X, Vertex.Y, Vertex.Z);
    }

    // EarClipper needs to know what perspective it has to triangulate.
    EarClipper.SetNormal(Normal.X, Normal.Y, Normal.Z);

    EarClipper.Triangulate();

    // If the result is valid and we have at least one triangle,
    // return the indices.
    if (EarClipper.ResultIndices && EarClipper.ResultIndicesCount >= 3)
    { // At least one triangle
        for (short Index = 0; Index < EarClipper.ResultIndicesCount; Index++)
        {
            OutIndices.Add(IndexOfVertices+EarClipper.ResultIndices[Index]);
        }

        IndexOfVertices += InVertices.Num();
    }
}
```

Die Resultate sehen generell sehr vielversprechend aus. Bei manchen Planes kommt es allerdings zu Problemen. Gerade wenn es sich um Polygone mit Löchern handelt. Die Bibliothek unterstützt zwar auch Löcher, aber der VUEParser bietet bislang keine Liste mit Löschern. Es kommt daher selten zu einer schlechten Triangulierungen. Abbildung 10 zeigt, in rot markiert, eine solche schlechte Triangulierung. Hier liegt es allerdings nicht an den fehlenden Löchern, sondern einfach daran, dass die Bibliothek hier scheinbar die konvexe Hülle verwendet:

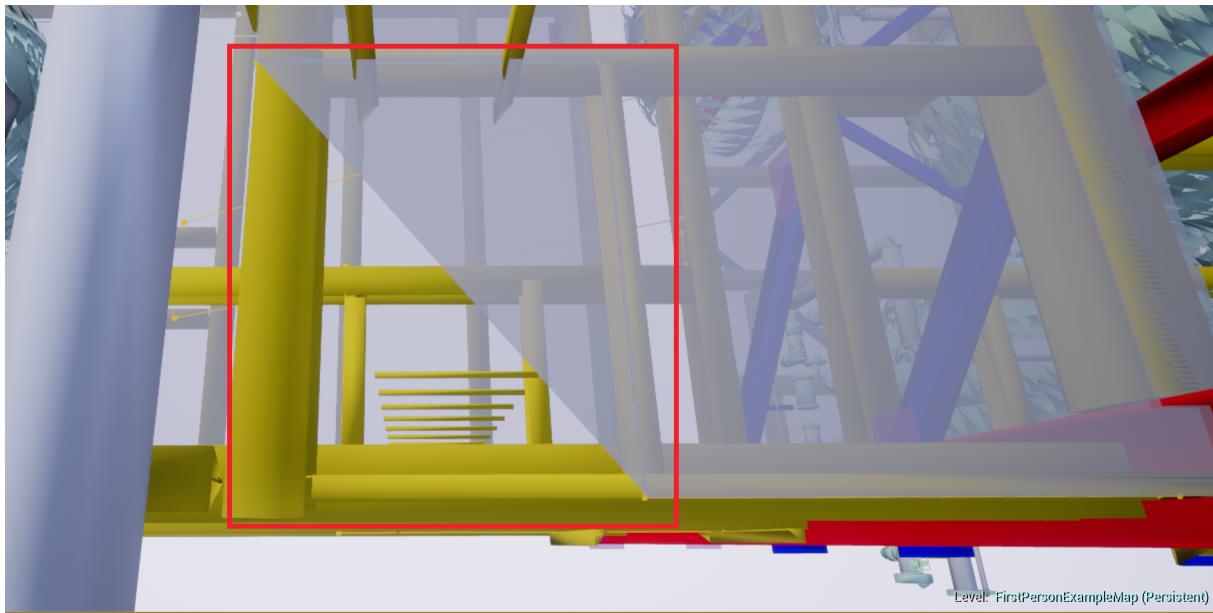


Abbildung 10: Zeigt nicht optimale Triangulierung

Wie sich herausstellt, stürzt das Plugin ab, wenn nicht wenigstens 3 Eckpunkte übergeben werden. Das liegt daran, dass es nicht möglich ist, ein Dreieck mit weniger als 3 Eckpunkten zu erstellen. Die Lösung:

```
if (Plane3DIndices.Num() >= 3) // We need at least 3 Points.  
{  
    VUETriangulation::TriangulatePolygon(Plane3DVertices, Plane3DIndices, Normal);  
}
```

Passend zu den noch folgenden Primitiven

Damit bei den noch folgenden Primitiven die Vektoren um eine beliebige Achse gedreht werden können, benötigen wir den folgenden Code, adaptiert von OpenTK[63]:

```
FMatrix CreateFromAxisAngle(FVector Axis, double Angle)
{
    double Cos = FMath::Cos(-Angle);
    double Sin = FMath::Sin(-Angle);
    double T = 1.0 - Cos;

    Axis.Normalize();

    return Fmatrix(
        FPlane(t * Axis.X * Axis.X + Cos, t * Axis.X * Axis.Y -
               Sin * Axis.Z, t * Axis.X * Axis.Z + Sin * Axis.Y,
               0.0),
        FPlane(t * Axis.X * Axis.Y + Sin * Axis.Z, t * Axis.Y *
               Axis.Y + Cos, t * Axis.Y * Axis.Z - Sin * Axis.X,
               0.0),
        FPlane(t * Axis.X * Axis.Z - Sin * Axis.Y, t * Axis.Y *
               Axis.Z + Sin * Axis.X, t * Axis.Z * Axis.Z + Cos,
               0.0),
        FPlane(0, 0, 0, 1)
    );
}

{
    FMatrix Rotation = CreateFromAxisAngle(Orientation, Angle);
    FVector4 Rotated = Rotation.TransformVector(Radius);
}
```

Wie sich herausstellt, muss bei der Adaption etwas schief gelaufen sein, denn die Werte scheinen nicht immer korrekt zu sein. Zudem bietet die Spiel-Engine eine eigene Implementierung. Dadurch reduziert sich der Code auf eine einzige Zeile:

```
FVector RotatedRadius = Radius.RotateAngleAxis(Angle, Orientation);
```

Bei dieser Umstellung ist auch darauf zu achten, dass hier Gradmaß erwartet wird und nicht mehr Bogenmaß. Da die Daten aus dem VUEParser allerdings in Bogenmaß kommen, hilft eine Funktion der API bei der Umrechnung:

```
RevolutionAngle = FMath::RadiansToDegrees(RevolutionAngle);
```

Torus

Wie im Konzept beschrieben, werden zuerst die Zentren im Umlauf des Kreises berechnet:

```
FinalRadiusPosition =
    Orientation2.RotateAngleAxis(RadiusCurrentAngle,
                                   Orientation1);
FinalRadiusPosition = FinalRadiusPosition *
    Radius + Position;
```

Danach wird noch der Vektor vom Mittelpunkt zur gerade berechneten Position berechnet und damit das Kreuzprodukt mit der ersten Orientierung gebildet. Das Resultat ist die Orientierung, um die der Kreis gebildet werden kann:

```
DirectionFromCenterToRadius = FinalRadiusPosition - Position;
DirectionFromCenterToRadius.Normalize();
FinalRadiusDirection =
    Fvector::CrossProduct(Orientation1, DirectionFromCenterToRadius);
```

Dadurch besitzen wir nun den Mittelpunkt des Kreises und die notwendige Orientierung. Es folgt die Berechnung eines Eckpunktes des zu erstellenden Kreises:

```
FinalCirclePosition =
DirectionFromCenterToRadius.RotateAngleAxis(CircleCurrentAngle, FinalRadiusDirection);
FinalCirclePosition = FinalCirclePosition * Thickness + FinalRadiusPosition;
```

In Abbildung 11 ist zu sehen, wie die Debug-Linien als Hilfestellung verwendet werden. Zu diesem Zeitpunkt wird gerade entwickelt, wie die Eckpunkte des ersten Außenkreises mit den des letzten Außenkreis verbunden werden. Das erspart zusätzliche Eckpunkte, die ineinander sitzen würden und damit doppelt wären.

Damit die Qualität anpassbar ist, werden 2 Variablen für eine mögliche Optimierung bereitgestellt:

<code>int VUEGlobals::TorusNumberOfEdges</code>	<code>= 5;</code>
<code>int VUEGlobals::TorusNumberOfCircleEdges</code>	<code>= 5;</code>

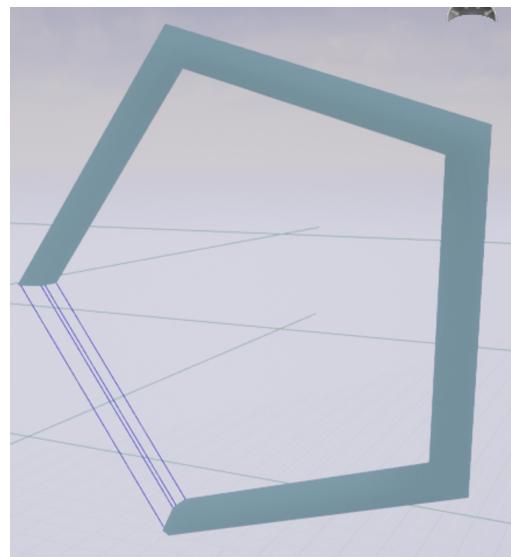


Abbildung 11: Die Debug-Linien beim Torus

Projection

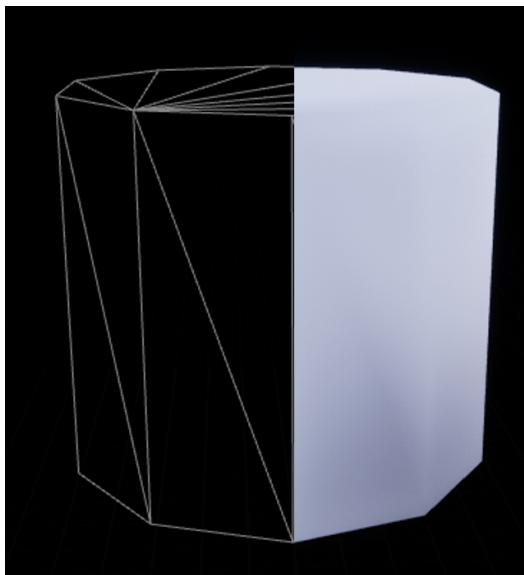


Abbildung 12: Zeigt *Projection* in wireframe und beleuchtet

Die Projektion wird für die elliptische Grundfläche implementiert, wie im Konzept vorgesehen. Der elliptische Boden, der Zylinder und der elliptische Deckel. Auch hier wird darauf geachtet, dass der Zylinder die Eckpunkte der Ellipsen wiederverwendet, um nicht unnötige Eckpunkte zu berechnen. Zur Berechnung des Kreises wird kein PI verwendet, wie man hätte annehmen können. Stattdessen wird der gegeben Radius passend rotiert. Der folgende Programmcode zeigt

zudem, dass beide Ellipsen in einer Schleife berechnet werden können:

```
for (int Index = 0; Index < VUEGlobals::ProjectionNumberOfEdges; Index++)
{
    Angle += Step;
    FVector RotatedRadius = Radius.RotateAngleAxis(Angle, Orientation);

    FVector Position1;
    Position1.X = Position.X + RotatedRadius.X;
    Position1.Y = Position.Y + RotatedRadius.Y;
    Position1.Z = Position.Z + RotatedRadius.Z;

    // Calculate the position of the second ellipse. ExtrusionDepth is a vector,
    // that is being used as the projection. It cannot be normalized, because the
    // length of ExtrusionDepth is needed to project correctly.
    FVector Position2 = Position1 + ExtrusionDepth;

    OutVertices.Add(Position1);
    OutVertices.Add(Position2);
}
```

Revolution

Der Revolution beginnt mit der Rotation um den Mittelpunkt mit einem bestimmten Winkel. Dabei werden immer wieder, wie beim Torus, neue Kreise gezeichnet. Die in Abbildung 14 gezeigten Abstände sind aber ungleichmäßig, da sie nach oben hin vom Abstand abnehmen. Hier wurde noch der adaptierte Programmcode von OpenTK verwendet. Nach der Umstellung aber, so zu sehen in Abbildung 13, sind die Kreise deutlich gleichmäßiger verteilt. Damit die Qualität beeinflussbar bleibt, werden die folgenden globalen Variablen angelegt:

```
VUEGlobals::RevolutionNumberOfEdges  
VUEGlobals::RevolutionNumberOfCircleEdge
```

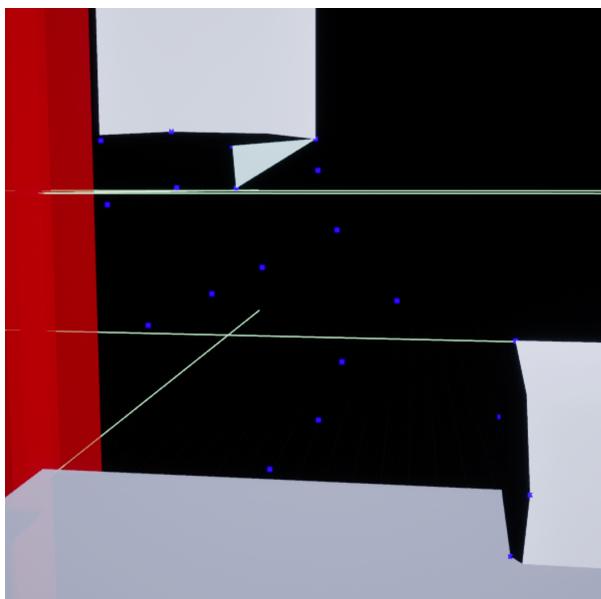


Abbildung 14: Schlechte Berechnung der Eckpunkte

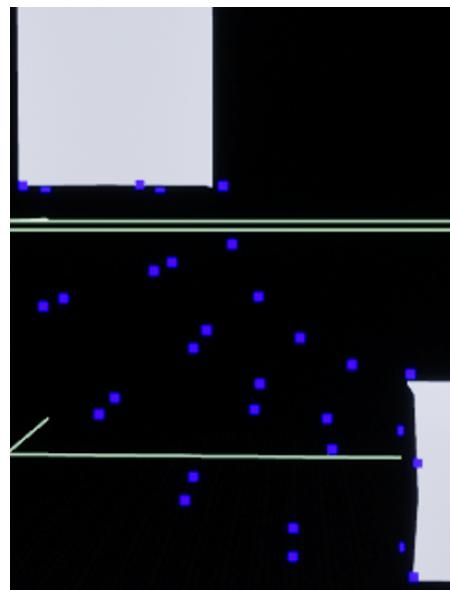
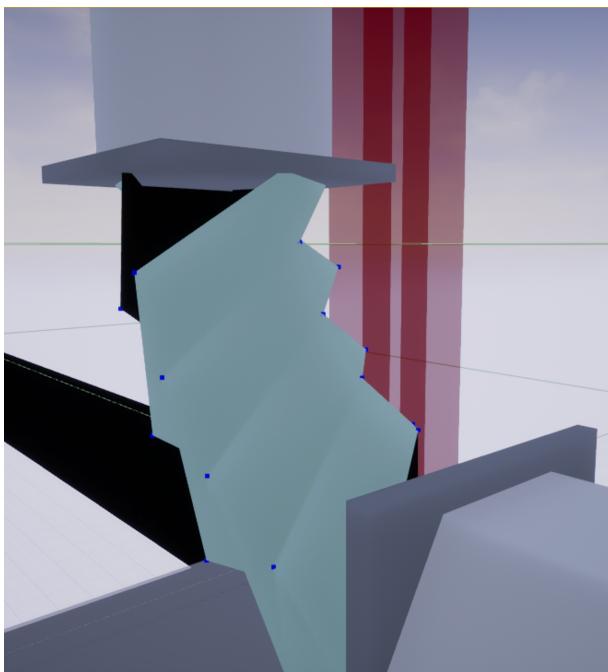


Abbildung 13: Gleichmäßige Berechnung der Eckpunkte

Fehler bei der Triangulierung



Durch einen Fehler in der Indizierung kommt es zu einer Darstellung, die einer Schraube ähnlich sieht. Abbildung 15 zeigt dieses Problem. Das liegt daran, dass der Index um eins verschoben ist. Damit wird nicht der gewünschte Eckpunkt verwendet, sondern der benachbarte Eckpunkt.

*Abbildung 15: Zeigt falsche Indizierung
beim Revolution*

Sphere

Die Implementierung der Kugel basiert auf dem Quellcode der Spiel-Engine. Dort wird sie in Verbindung mit Texturen verwendet. Dieser Teil ist für uns nicht relevant, da unsere Materialien keine Texturen verwenden. Der implementierte Programmabschnitt sieht wie folgt aus:

```

for (int32 Index = 0; Index < VUEGlobals::SphereNumberOfEdges + 1; Index++)
{
    FDynamicMeshVertex* ArcVert = &ArcVerts[Index];

    float Angle = ((float)Index / VUEGlobals::SphereNumberOfEdges) * PI;

    Wir berechnen erst einmal nur zweidimensionale Punkte
    ArcVert->Position.X = 0.0f;
    ArcVert->Position.Y = FMath::Sin(Angle);
    ArcVert->Position.Z = FMath::Cos(Angle);
}

Danach werden die Punkte gedreht.
for (int32 S = 0; S < VUEGlobals::SphereNumberOfEdges + 1; S++)
{
    Es wird SphereNumberOfEdges Mal durchgelaufen und jedes Mal
    S /SphereNumberOfEdges mehr rotiert
    FRotator ArcRotator(0, 360.f * (float)S / VUEGlobals::SphereNumberOfEdges, 0);
    FRotationMatrix ArcRot(ArcRotator);

    Nun wird jede Position noch transformiert
    for (int32 V = 0; V<VUEGlobals::SphereNumberOfEdges + 1; V++)
    {
        int32 VIx = (VUEGlobals::SphereNumberOfEdges + 1)*S + V;

        Die final transformierte Position wird übertragen
        Verts[VIx].Position = ArcRot.TransformPosition(ArcVerts[V].Position);
    }
}

Wichtig: Der Radius muss dazu multipliziert werden, da die Kugeln sonst
immer gleich groß wären. Auch der Skalierungsfaktor ist notwendig.
for (int32 VertIdx = 0; VertIdx < NumVerts; VertIdx++)
{
    OutVertices.Add(Verts[VertIdx].Position * (Radius * VUEGlobals::ScaleFactor) +
                    Position * VUEGlobals::ScaleFactor);
}

```

6.2.3 Übernahme der Metadaten

Die Datenmigration der Metadaten werden an zwei verschiedene Stellen geschrieben: Dem *Header-Actor* und *Container-Actor*. Für die Anzeige des Headers wird ein Aktor vom Typ `AActor` instanziiert, der im Treeview angeklickt werden kann. Damit ein Objekt im Spiel platziert werden kann, darf es nicht von der Klasse `UObject` ableiten, sondern muss von `AActor` erben.[64] Durch das Anklicken des Headers, in Abbildung 16 `.Header-SPRF` genannt, bekommt der Anwender nun wichtige Daten über die .vue-Datei gelistet. Abbildung 17 zeigt genauer, welche Daten übermittelt werden. Wie sich feststellen lässt, werden neben den Header-Daten auch die

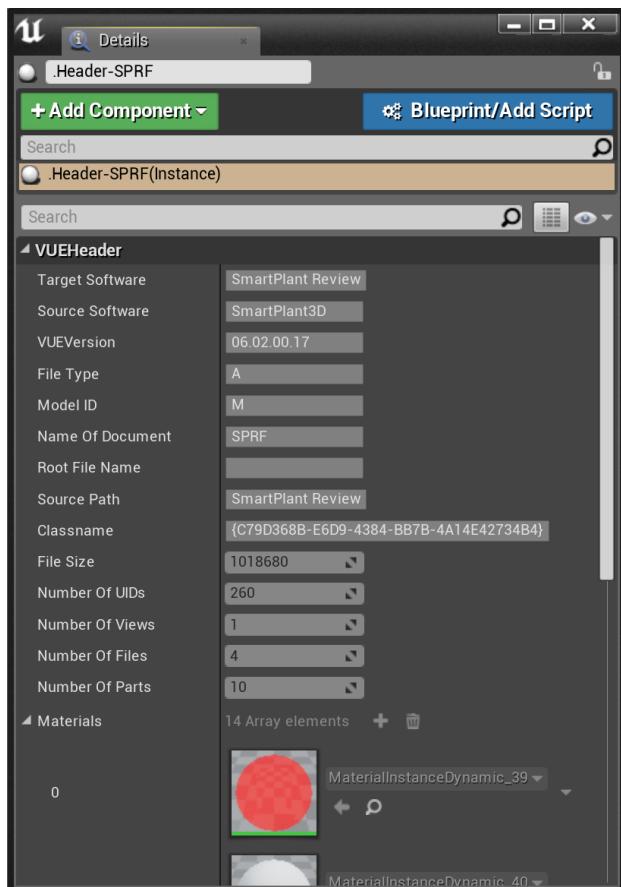


Abbildung 17: Header-Actor in Details-Tab

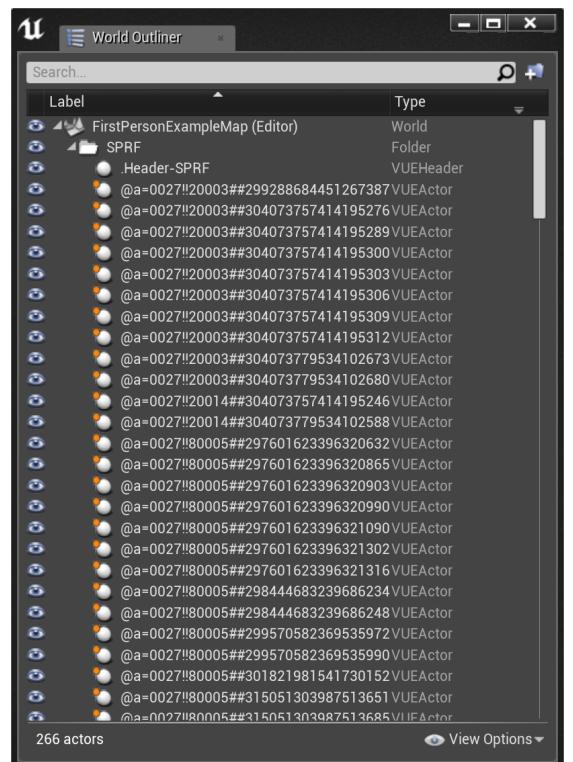


Abbildung 16: Treeview zeigt eine Importierte Datei

Materialien gelistet.

Damit diese Auflistung möglich ist, werden dem Aktor Eigenschaften angelegt. In der Unreal Engine 4 nennt sich eine solche Eigenschaft `UPROPERTY` und wird als Makro über die Variable geschrieben. Dadurch weiß die Spiel-Engine, dass es sich hier um eine Eigenschaft handelt, die zum Beispiel im Editor angezeigt werden soll.

```
UPROPERTY(VisibleAnywhere)
FString TargetSoftware;
```

`VisibleAnywhere` bedeutet, dass es in Eigenschaften-Fenstern zwar angezeigt wird, aber nicht editierbar ist.[65]

Der Implementierung der

Datenmigration bezüglich des *Container-Actors*, entspricht nahezu dem des *Header-Actors*.

6.2.4 Unterstützung von Obstruction Volumes

Die Obstruction Volumes, auch Display Aspects genannt, werden wie die Metadaten im *Container-Actor* gespeichert. Abbildung 18 zeigt, grün umrandet, die Display Aspects. Da es sich bei der Natur der Obstruction Volumes oftmals um Geometrien handelt, die passierbar sind, wird dies in der Implementierung berücksichtigt. So gibt es keine Kollisionsabfragen, wenn es sich um Obstruction Volumes der Typen Insulation, Operation, Maintenance, ReferenceGeometry und CenterLine handelt. Kollisionsabfragen können aber auch für das gesamte Modell ausgeschaltet werden.

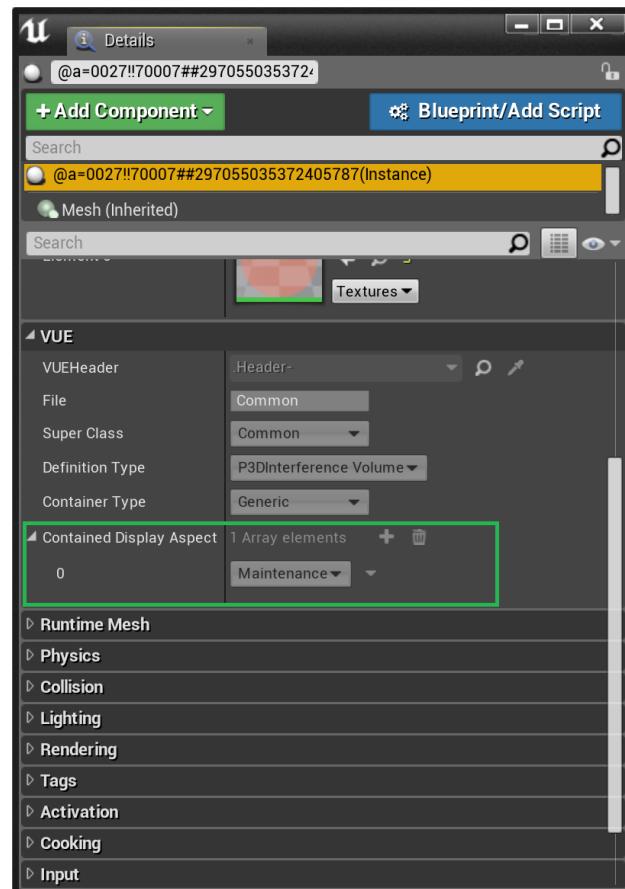


Abbildung 18: Zeigt die Display Aspects eines Objekts

6.2.5 Übernahme der Materialien

Bei den Materialien, die der VUEParser liefert, benötigen wir vor allem die `AmbientColor`, `SpecularColor` und `DiffuseColor` Werte. Damit diese übernommen werden können, müssen Materialien angelegt werden. Dem Konzept nach handelt es sich hier um dynamische Instanzen des Basismaterials. Das Basismaterial sieht wie folgt aus:

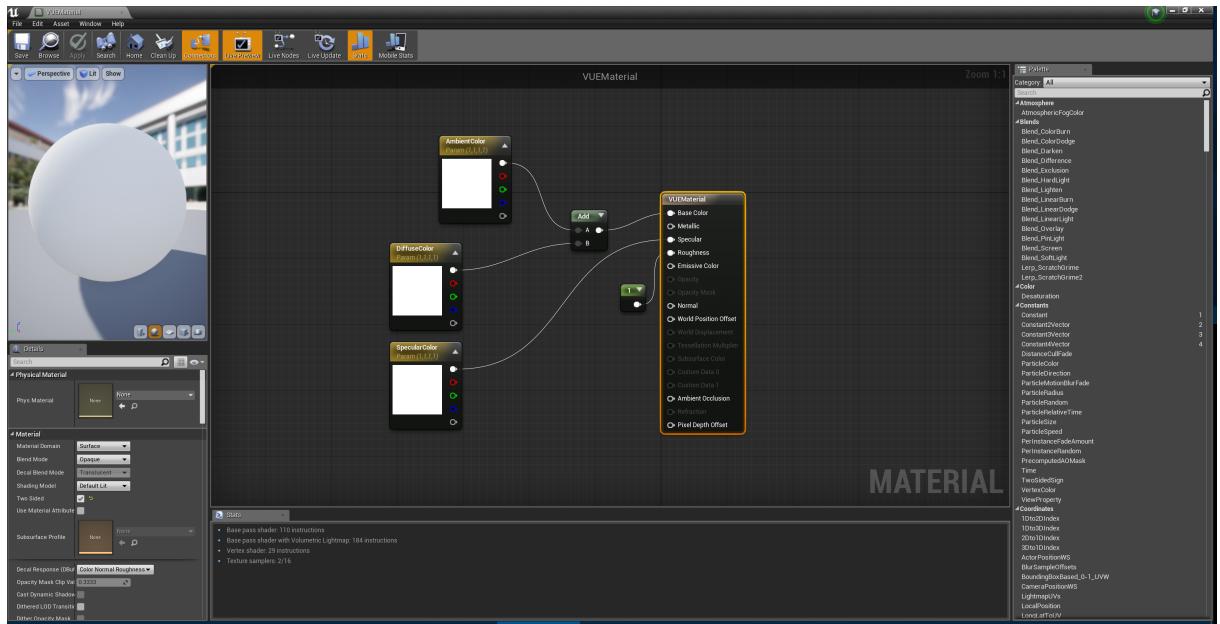


Abbildung 19: Das Basismaterial im Materialeditor

Es ist zu erkennen, dass hier die 3 Basiswerte vorkommen und passend in die dazu gehörigen Punkte verbunden werden. `AmbientColor` und `DiffuseColor` werden hier addiert, da es sich um den Aufbau nach Phong handelt. Wie sich herausstellt ist das Material damit falsch angelegt. Da die ambiente Farbe nicht der Lichtberechnung unterliegen darf, muss es mit dem Punkt `Emissive Color` verbunden werden. Abbildung 20 zeigt das korrigierte Material. Mit diesem Material ist es aber nicht möglich Transparenz darzustellen.

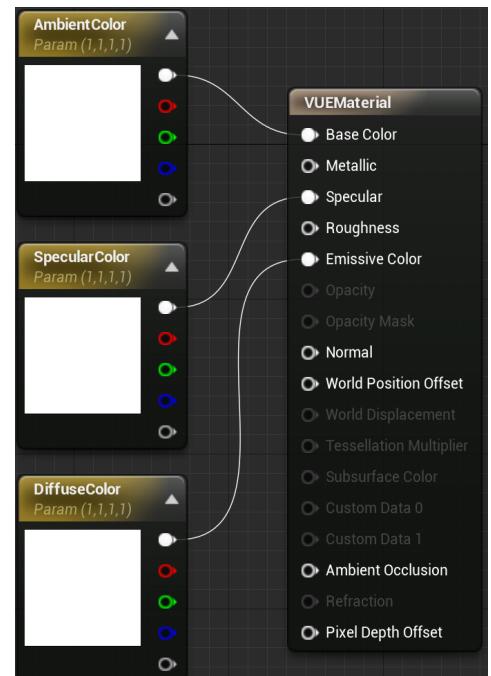


Abbildung 20: Besseres Material

Das Transparente Material

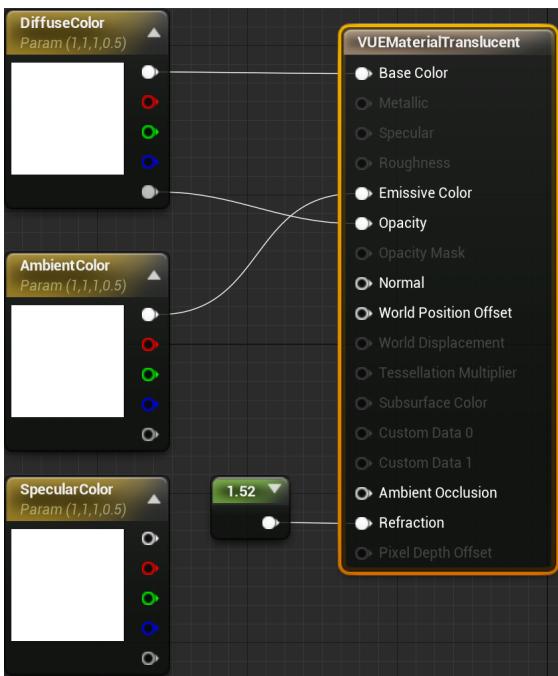


Abbildung 21: Zeigt das Basismaterial für Transparenz

Umsetzung die Nutzung eines experimentellen Features voraussetzt[70] und sich höchstwahrscheinlich negativ auf die Performance auswirken würde. Die Verwendung von experimentellen Features spricht gegen die nicht-funktionale Anforderung der **Stabilität**. Die Daten allerdings nicht zu übergeben, spricht gegen die nicht-funktionale Anforderung **Genauigkeit**. Daher ist es möglich, das Material so abzuändern, damit es dennoch den spekulativen Farbanteil unterstützt. Dafür muss der *Lighting Mode* von *Volumetric NonDirectional* auf *Surface TranslucencyVolume* gestellt werden. Abbildung 22 zeigt das daraus resultierende Material.

Damit auch transparente Materialien unterstützt werden, muss das Material ein bisschen verändert werden. Hier wird der *Blend Mode* von *Opaque* auf *Translucent* gestellt. Dadurch kann eine *Opacity*, zu deutsch *Opazität*[66], verbunden werden. Den Wert dafür erhalten wir durch den Alpha-Wert der diffusen Farbe. Da die Spiel-Engine auch die Unterstützung von Refraktion[67] anbietet, wird dieser Wert auf 1.52 gesetzt.[68] Der Wert 1.52 steht für den Refraktionsindex[69] von Flachglas. Abbildung 21 zeigt das implementierte Material. Für den spekulativen Farbanteil ist keine Verbindung vorgesehen, da diese

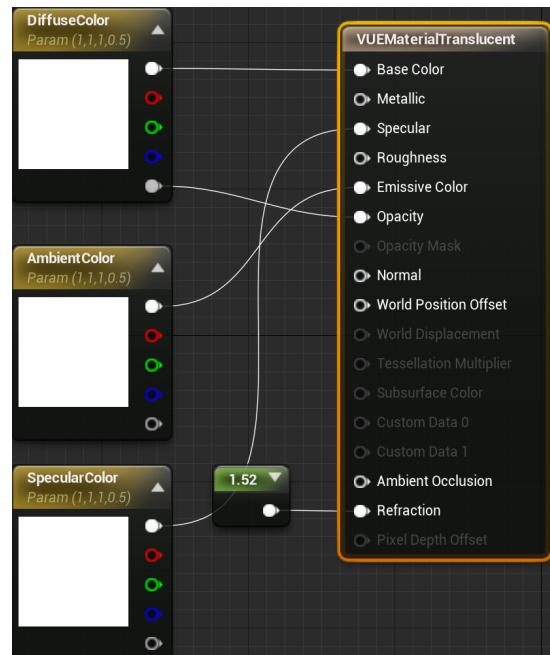


Abbildung 22: Das Material für Transparenz mit der Unterstützung des spekulativen Farbanteils

Dynamische Materialien und wie sie passend manipuliert werden

Die Implementierung instanziert ein dynamisches Material, wie in der Ausgangslage beschrieben. Danach kann auf die einzelnen Vektoren zugegriffen werden:

```
MaterialInstance->SetVectorParameterValue("AmbientColor",
    FLinearColor(AmbientColor.R, AmbientColor.G, AmbientColor.B, AmbientColor.A));
MaterialInstance->SetVectorParameterValue("DiffuseColor",
    FLinearColor(DiffuseColor.R, DiffuseColor.G, DiffuseColor.B, DiffuseColor.A));
MaterialInstance->SetVectorParameterValue("SpecularColor",
    FLinearColor(SpecularColor.R, SpecularColor.G, SpecularColor.B, SpecularColor.A));
```

Die dynamischen Instanzen haben auf Grund ihrer Natur dynamische Namen. Das Umbenennen zu passenderen Namen funktioniert über Rename. Anders als beim Actor, funktioniert es hier wie erwartet:

```
MaterialInstance->Rename(*FString::Printf(TEXT("Material %s [%i]"),
    InMaterial->File->Name, InMaterial->SimpleIndex));
```

Allerdings gibt es an dieser Technik einen Haken: Kommt es zu zwei Materialien, die den gleichen Namen besitzen sollen, stürzt die Spiel-Engine ab. Damit also auch die dynamischen Material passend benannt werden können, muss vorher nach einer Technik gesucht werden, die alle verfügbaren Materialien durchgeht und nach bereits vorhanden Instanzen sucht.

6.2.6 Selektieren der importierten Geometrie für Informationen

Das Selektieren der Geometrie um Informationen zu erhalten, ist überwiegend durch die Verwendung des Editors in Verbindung mit der Übernahme der Metadaten gedeckt. Dadurch reicht das Anklicken der Geometrie im Editor, um die relevanten Metadaten im Details-Tab zu sehen.

6.2.7 Nicht konvertierte Geometrie kenntlich machen

Die Implementierung verwendet ein Fallback-System, dass bei Fehlern auf einen Fehlerzustand fällt. Ist der Fehlerzustand erst einmal ausgelöst, wird die Boundingbox als Platzhalter angezeigt. Durch ein spezielles Material wird dem Anwender dann kenntlich gemacht, dass es sich hierbei um einen Platzhalter handelt.

6.2.8 Bewegung des Anwenders innerhalb der Anlage ermöglichen

Die Umsetzung nutzt den Editor-Viewport als Grundlage der Fortbewegung. Dies ermöglicht dem Spieler, sich fliegend durch die Anlage zu bewegen. Durch den „Play“-Schalter kann er außerdem durch die Anlage laufen. Da es sich um eine Implementierung seitens der Unreal Engine 4 handelt, können nähere Informationen auf der Webseite gefunden werden.[71]

6.3 **Implementierung nicht-funktionaler Anforderungen**

Dieses Unterkapitel beschäftigt sich mit der Implementierung der in der Konzeption beschriebenen Lösungen der nicht-funktionalen Anforderungen.

6.3.1 Benutzeroberfläche

Die Benutzeroberfläche nutzt die vom Editor gegeben Oberfläche und addiert die neu hinzugefügten Funktionen des Plugins. Diese Funktionen und deren grafische Darstellung, wurden bereits in den vorherigen dazu gehörigen Unterkapiteln hinreichend angedeutet. Da das Fenster während des Importvorgangs aber keine Reaktion zulässt, wird ein Ladebalken den Prozessfortschritt dokumentieren. So kann ein Anwender auch abschätzen, wie lange der Vorgang noch in etwa dauern wird.

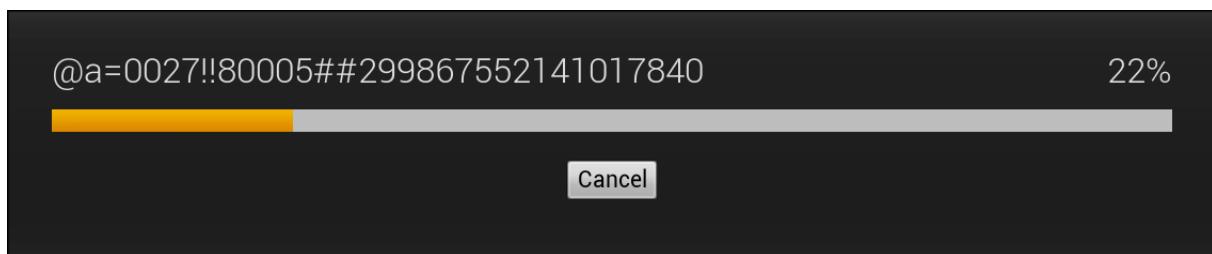


Abbildung 23: Zeigt den Ladebalken während des nicht reagierenden Imports

7 Evaluation

Dieses Kapitel beschäftigt sich mit der Beurteilung der fertig implementierten Software. Es wird auf die einzelnen Fragen eingegangen, die in der Zielsetzung gestellt wurden und auch wie stark die entwickelte Software zu bereits angebotenen Lösungen dasteht. Zudem wird die Geschwindigkeit beurteilt.

Als Frage formuliert: „Wie gut ist die Software-Lösung in Anbetracht der Anforderung?“

7.1 Qualität in Anbetracht der gesetzten Ziele

Die aus der Zielsetzung gestellten Fragen werden im Folgenden kurz beantwortet:

- Wie hoch ist die Performance in Relation zu bestehenden Lösungen?

Diese Frage wird in den 2 Unterkapiteln *Vergleich zum Intergraph Freeview* und *Geschwindigkeit* beantwortet.

- Können alle notwendigen Daten aus dem Anlagenbau übernommen werden?

Es wurden erfolgreich alle Daten übernommen, die benötigt werden und vom VUEParser unterstützt werden.

- Wie sieht es mit der Bedienbarkeit aus? Auch für einen Anwender ohne CAD-Erfahrung?

Die Bedienbarkeit leidet unter der Implementierung im Editor. Es wäre deutlich sinnvoller, den Viewer als eigenständige Software zu erstellen. Dadurch kann die Oberfläche auf das Wesentliche reduziert werden und der Anwender wird nicht durch Funktionen irritiert, die für ihn nicht relevant sind.

- Wie kann eine Interaktivität, Animation und Simulation umgesetzt werden?

Zum Beispiel eine Simulation, mit Animation, durch einfachen Knopfdruck zu starten. Das könnte eine Explosion betätigen.

Da sich Unternehmen wie Dynamic Vision[72] darauf spezialisiert haben und sich daher ein möglicher Bedarf vermuten lässt, ist auch hier eine Beurteilung notwendig. Tatsächlich lassen sich Simulationen durch das Blueprint-System der Unreal Engine durchführen. Allerdings gibt es Probleme beim Abspeichern der Datei: Die dynamisch erzeugten Material-Instanzen sind nach dem

erneuten Laden der Datei nicht mehr vorhanden. Sie existieren schließlich nur so lange, wie die Instanz eines Spiels vorhanden ist und werden nicht in die Welt-Datei gespeichert.

- Ist die Lösung auch portabel?

Die momentane Lösung ist nur bedingt portabel. Wenn die Bedingungen erfüllt sind, dass die Engine bereits installiert ist und die VUEParser.dll sich im Engine-Runtime-Verzeichnis befindet, wäre die Lösung portabel.

- Wie sieht es mit Virtual Reality aus?

Da keine VR-Hardware zur Verfügung steht, konnte dieser Punkt nicht getestet werden. Es gilt daher die nicht getestete, von der Spiel-Engine kommende, VR-Unterstützung.[73]

7.2 Stabilität

Die Stabilität ist bei den Testmodellen als positiv zu betrachten, da die Software nicht abstürzt. Leider ist die Reihe an Testmodellen sehr begrenzt, da Kundenmodelle nicht Teil dieser Ausarbeitung sein dürfen. Es lässt sich hier aber dennoch erwähnen, dass zu große Dateien zum Absturz führen können. So zeigt Abbildung 24 die mögliche Fehlermeldung, wenn die Datei zu groß für einen erfolgreichen Import ist:

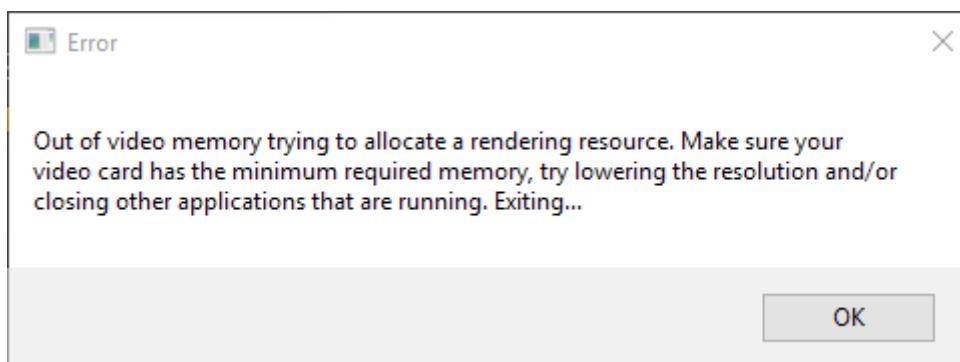


Abbildung 24: Eine Datei die zu groß ist, führt zu einem Fehler.

Laut dieser Fehlermeldung wird sich explizit auf den Videospeicher bezogen, der als nicht ausreichend angesehen wird. Es ist aber auch möglich, dass der Fehler wegen dem generell zu geringen Arbeitsspeicher des Systems erscheint. Abbildung 25 zeigt die Auslastung im Windows Taskmanager, während die Fehlermeldung erscheint.

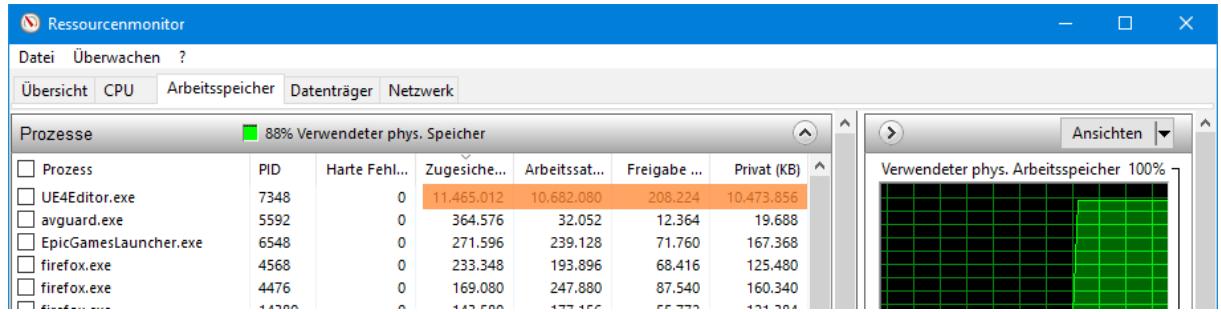


Abbildung 25: Zeigt die Auslastung durch den Windows Task Manager

Lässt man den Grund des Absturzes außer Acht, bleibt immer noch der Wunsch danach auch große Dateien betrachtet zu können. Im momentanen Zustand des Plugins wäre es daher möglicherweise für ein kommerzielles Produkt ungeeignet.

7.3 Vergleich zum Intergraph Freeview

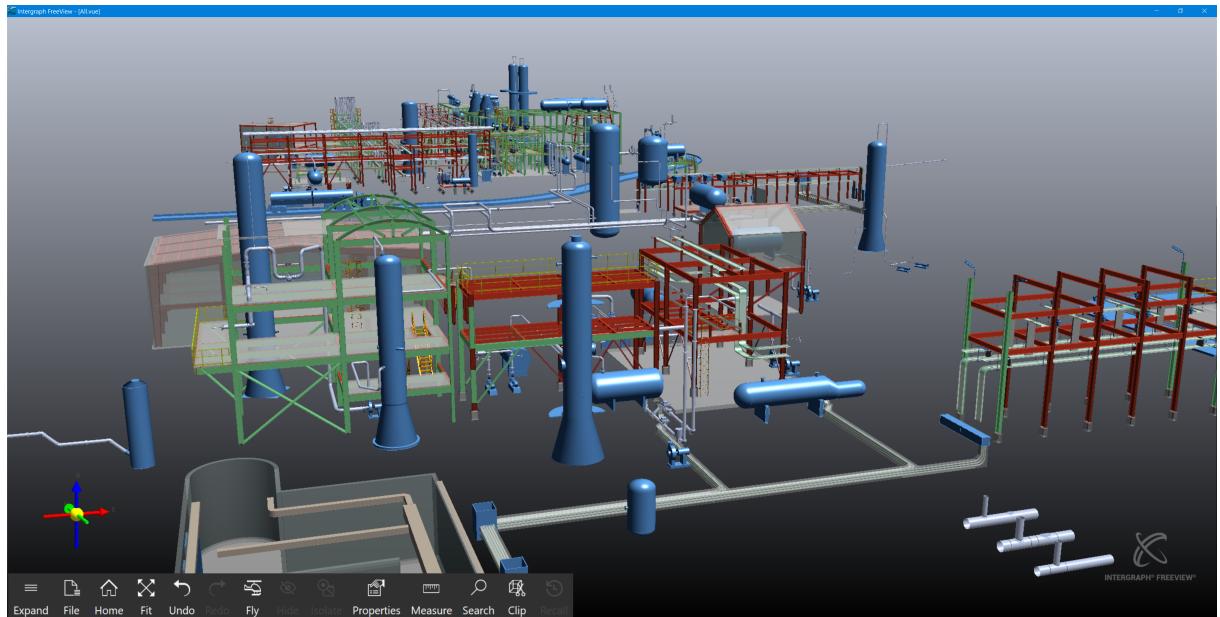


Abbildung 26: Screenshot aus Intergraph® FreeView® (12.00.00.0062)

Die kostenfrei verfügbare, aber proprietäre Intergraph® FreeView®[7] Software, gilt als Referenz für die volle Unterstützung des .vue-Formats. Laut mitgelieferter Hilfe, sollen auch andere Formate unterstützt werden: „*Intergraph FreeView can open 3D .vue/xml, .vue/.mdb2, and .svf files from Smart 3D. You can also open third-party 3D files translated to Smart Models using Intergraph SmartPlant Interop Publisher.*“ Bei SmartPlant Interop Publisher[74] handelt es sich aber wahrscheinlich um eine kostenpflichtige Anwendung, denn sie ist nicht Teil der Freeview Software und nicht von der Produktseite herunter ladbar. Abbildung 26 zeigt die vollständige Zeichnung, wie sie

7 Evaluation 7.3 Vergleich zum Intergraph Freeview

auch für die Tests dieser Ausarbeitung verwendet wurde. Die Software gibt dem Anwender die Möglichkeit, sich fliegend durch die Anlage zu bewegen und einzelne Container anzuklicken. Es folgt eine Auflistung der Daten.

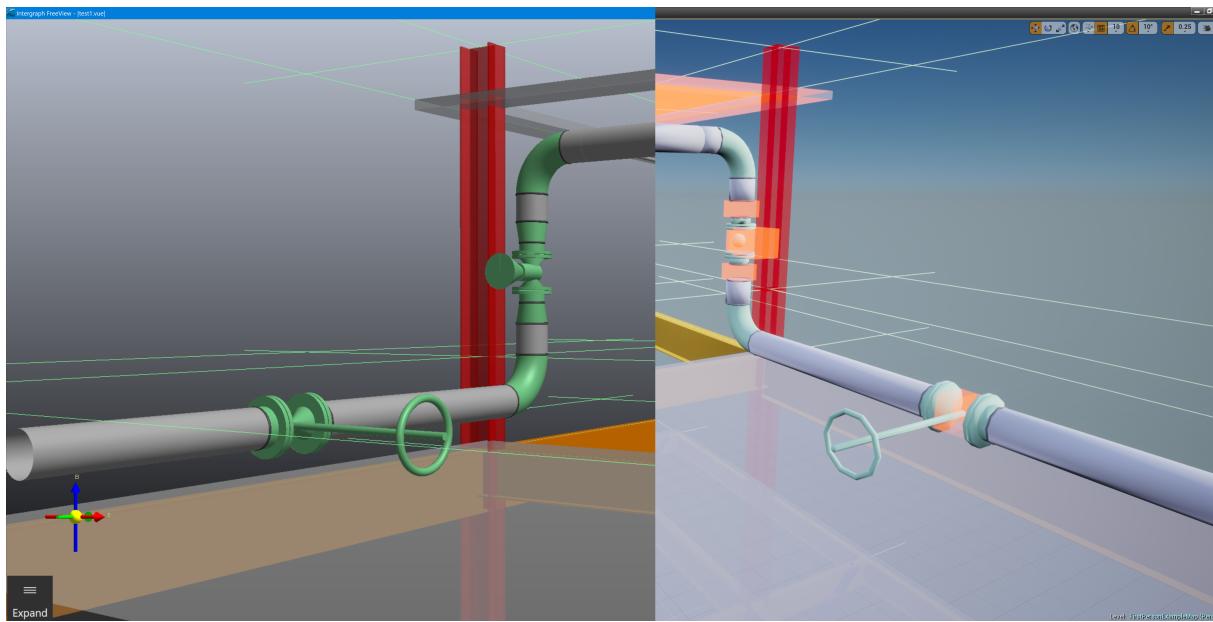


Abbildung 27: Zeigt sichtlich Differenzen zwischen beiden Programmen

Abbildung 27 zeigt auffallenden Unterschieden zwischen der Freeview Software und unserem Plugin. Der größte Unterschied: *Die Spiegelung an der XZ-Ebene*. Diesem Fehler könnte man entgegen wirken, indem die Y-Achse invertiert wird. Das liegt daran, dass in der Spiel-Engine die positive Richtung der Y-Achse aus dem Bildschirm zeigt, bei der verwendeten CAD-Anwendung es aber umgekehrt ist. Weitere Unterschiede sind die als Platzhalter dargestellten Container, leicht unterschiedliche Farben und weniger Polygone am Torus. Die unterschiedlichen Farben kommen möglicherweise durch unterschiedliche Belichtungsmodelle zustande. Hier scheint die ambiente Farbe eine wichtige Rolle zu spielen. Möchte man mehr Polygone am Torus sehen, ist es möglich, die passenden globalen Variable zu inkrementieren:

```
int VUEGlobals::SphereNumberOfEdges = 10;  
int VUEGlobals::TorusNumberOfEdges = 10;  
int VUEGlobals::TorusNumberOfCircleEdges = 10;  
int VUEGlobals::ProjectionNumberOfEdges = 10;  
int VUEGlobals::RevolutionNumberOfEdges = 10;  
int VUEGlobals::RevolutionNumberOfCircleEdges = 10;
```

Unterschied der Geschwindigkeit

Freeview	Primitive Ebene	Gruppen-Ebene	Container-Ebene
14 Bilder	6 Bilder	14 Bilder	17 Bilder

Die oben aufgeführte Tabelle zeigt die Bildwiederholungsrate der beiden Programme. Dabei beschreibt die erste Spalte die Bilder pro Sekunde in der Freeview Software, die anderen 3 Spalten gelten für unser Plugin. Getestet wurde die Datei All.vue, an der Stelle, die auch Abbildung 26 zeigt. Der Unterschied der Geschwindigkeit ist allerdings nur bedingt als aussagekräftig zu betrachten, da die nicht unterstützten Primitive als Platzhalter dargestellt werden und somit die Qualität unterschiedlich ist.

Während der Nutzung von Freeview musste es einmal neu installiert werden, da die Fehlermeldung aus Abbildung 28 die Nutzung unterbunden hatte:

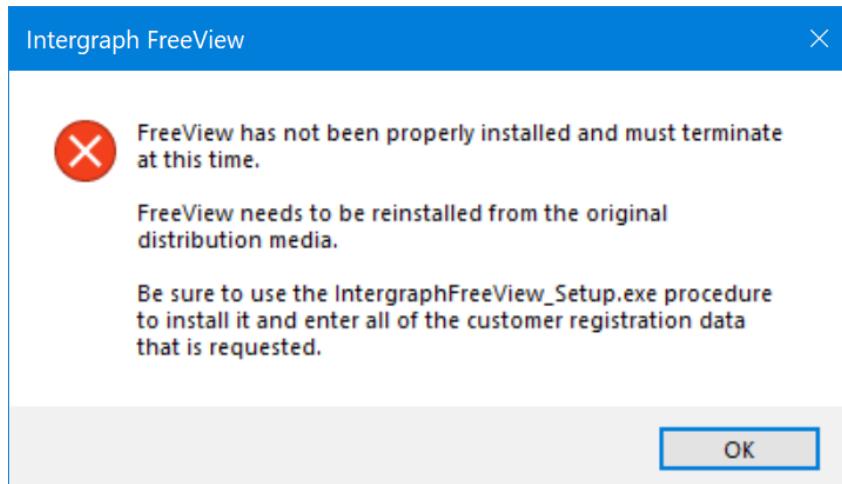


Abbildung 28: Fehlermeldung beim Starten von Freeview

7.4 Geschwindigkeit

Auf Grund der aufgestellten Erwartung, dass Spiel-Engines gut für die Anzeige von CAD-Daten sind, zeigt sich bei der Beurteilung ein anderes Bild. So ist die erbrachte Leistung unerwartet schlecht. Wie im vorherigen Unterkapitel *Vergleich zum Intergraph Freeview* angedeutet, ist die Performance nicht besser, als die der bereits verfügbaren Software. Das könnte daran liegen, dass für jeden Container ein Actor-Objekt erstellt wird. Auch spielen die prozedural erstellten Meshes eine Rolle. Jene sind für Dynamik ausgelegt und nicht für statisch hochperformante Objekte.[75] So stellt man fest, dass die Sektionen und Materialien einen großen Unterschied in der Geschwindigkeit darstellen. Erstellt man für jedes Primitiv eine Sektion im Container-Mesh, so wird auch für jede Sektion ein Material angelegt. Abbildung 29 zeigt den GPUProfiler[76] der Spiel-Engine, wenn für jedes Primitiv eine Sektion angelegt wird:

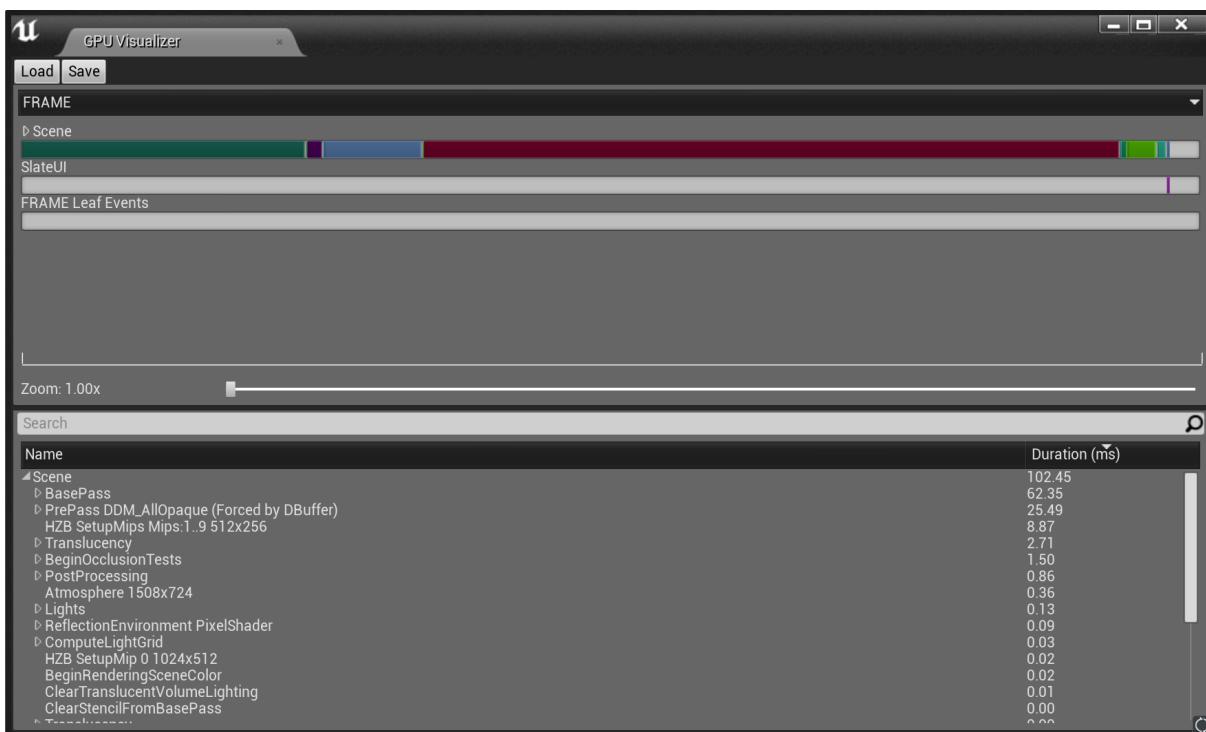


Abbildung 29: Der GPUProfiler, wenn für jedes Primitiv eine Sektion erstellt wird.

Der grüne Balken zeigt die Auslastung des PrePass und der rote Balken den des Basepass. Beide machen die größten Teile der Berechnungen aus. Importiert man die Daten allerdings auf Gruppen-Ebene, wirkt das Schauspiel bereits performanter auf Abbildung 30:

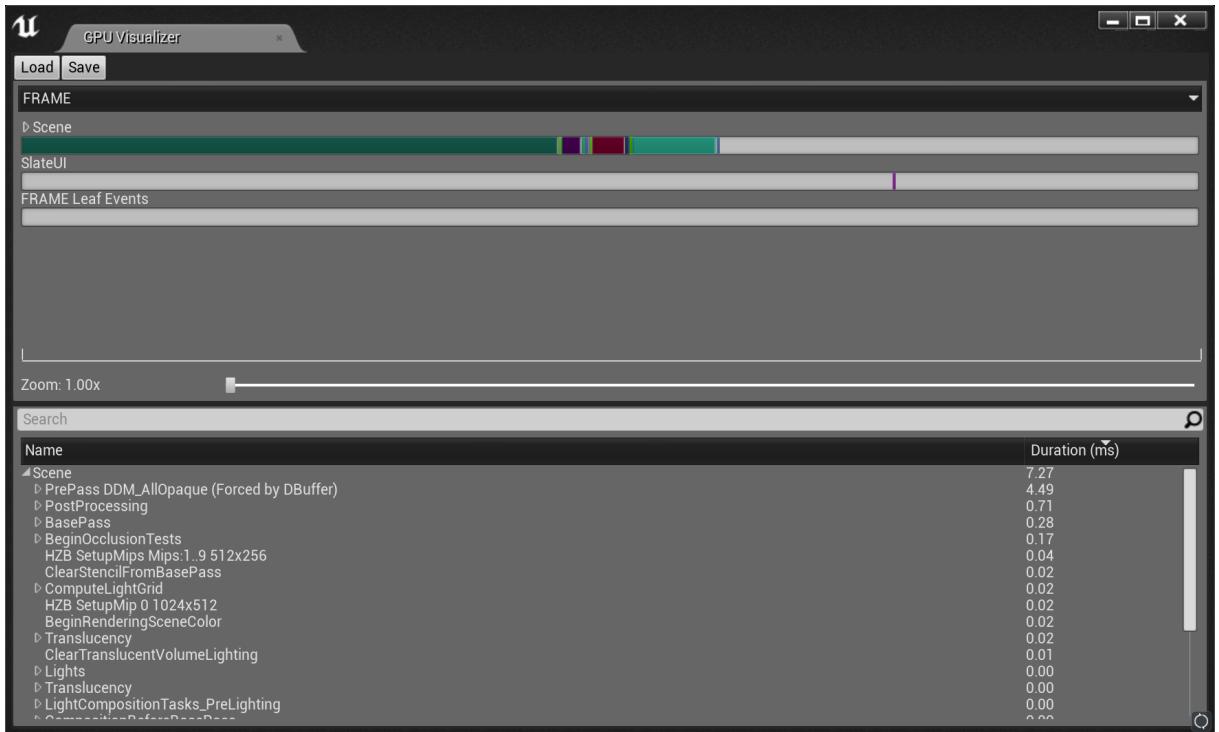


Abbildung 30: GPUProfiler, bei Import auf Gruppen-Ebene

Noch immer ist der Prepass sehr auslastend. Allerdings ist der Basepass jetzt deutlich entspannter. Das liegt möglicherweise daran, dass weniger Sektionen und vor allem weniger einzelne Materialien genutzt werden. Importiert man nun auf Container-Ebene, zeigt sich ein noch etwas performanteres Bild:

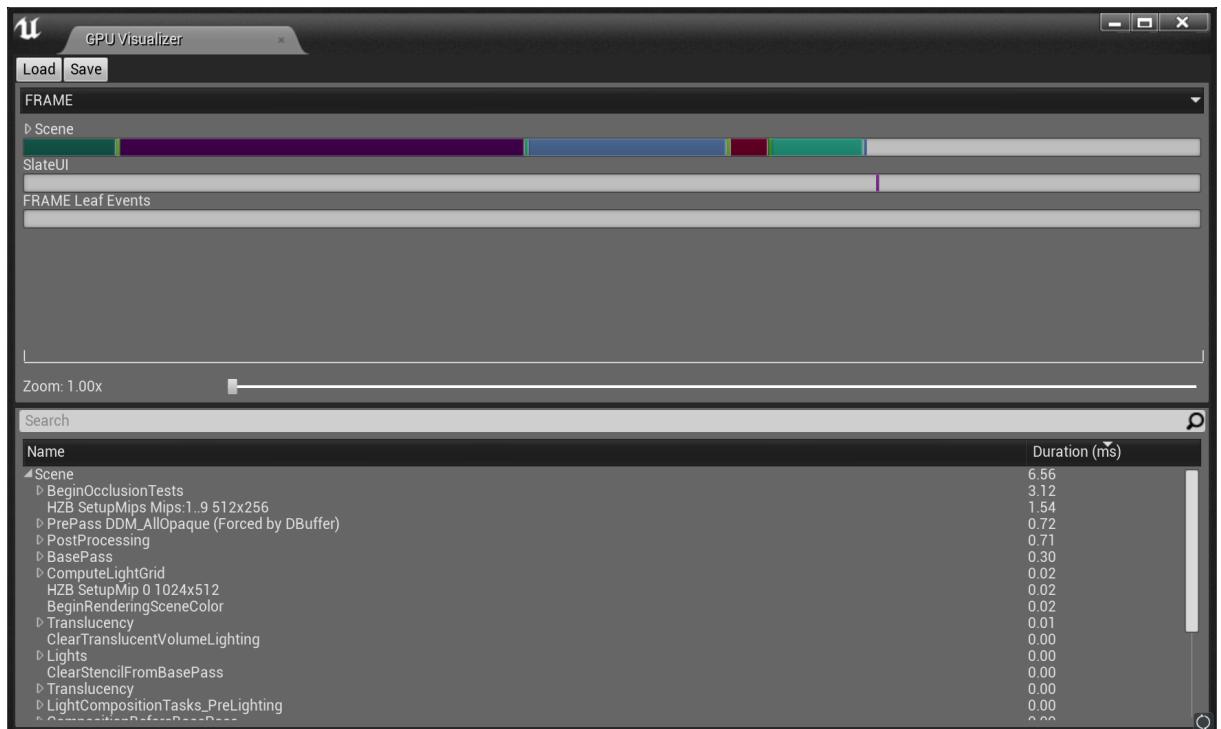


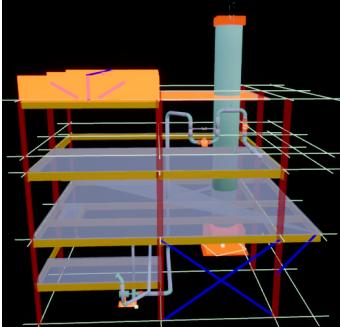
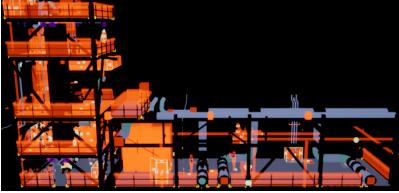
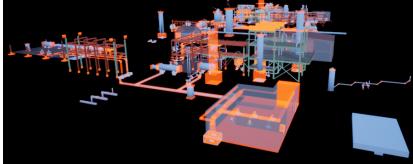
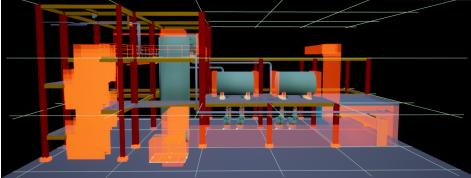
Abbildung 31: GPUProfiler, beim Import auf Container-Ebene

7 Evaluation 7.4 Geschwindigkeit

Auf Abbildung 31 ist außerdem zu sehen, dass nun nicht mehr der Basepass und der Prepass die meiste Zeit in Anspruch nehmen, sondern der BeginOcclusionTest.

Durch das Umschalten auf den Import in Gruppen- oder Container-Ebene, entfällt aber die Möglichkeit, pro Primitive ein eigenes Material zu haben. Der VUEParser bietet aber dieser Möglichkeit.

Folgende Tests wurden mit 4 verschiedenen Dateien durchgeführt:

Zeichnung	Primitiv	Gruppe	Container
	65 FPS	120 FPS(Limit)	120 FPS(Limit)
Zeichnung mit 260 Containern			
	8 FPS	40 FPS	55 FPS
Zeichnung mit 2682 Containern			
	6 FPS	16 FPS	19 FPS
Zeichnung mit 9757 Containern			
	28 FPS	100 FPS	120 FPS(Limit)
Zeichnung mit 687 Containern			

Die FPS-Limitierung liegt im Editor bei 120 FPS.

7.5 Evaluation durch Fachkundige

Die Beurteilung der Software durch Fachkundige übernimmt das Unternehmen, das auch den VUEParser entwickelt hat. Hier stand der Geschäftsführer, Herr Mayland, freundlicherweise für eine Beurteilung bereit. Die folgenden Anregungen sind aus Notizen entstanden, die von mir während des Gesprächs notiert wurden.

Bei der Selektion eines Materials, wäre es wünschenswert, wenn auch weitere Filter-Optionen zur Verfügung stehen würden. Mit Filter-Option ist gemeint, dass man z.B. alle Container selektieren kann, die das gleiche Material besitzen. Es sollte auch möglich sein nach weiteren Merkmalen zu suchen. Dazu zählen zum Beispiel Layer, Farbe, Objekttyp und weitere Merkmale.

Die Einstellung der Oberfläche „Generate mesh section per:“ ist irreführend. Es wäre sinnvoller die Qualität auswählen zu können. Es folgt ein Beispiel des Drop-Down-Menüs mit den möglichen Einträgen:

Choose Quality:

- High Quality
- Balanced
- High Performance

Des Weiteren wurde angemerkt, dass es empfehlenswert sei, die UID als Namen im Treeview anzuzeigen. Es wäre viel mehr wünschenswert, könnte der Kunde hier den Definitionstyp erkennen. Mit der UID kann nicht direkt etwas angefangen werden.

Die Waffe im Spielmodus sei zwar lustig, ist aber verschwendetes Potential. Viel sinnvoller wäre es, könnte man hier verschiedene Werkzeuge auswählen, wie zum Beispiel ein Messgerät.

Die Mehrspieler-Unterstützung könnte für interessante Projekte verwendet werden. So könnte die Anlage zusammen begutachtet werden oder besser simuliert werden. Das Ergebnis sollte nicht nur im Editor laufen, sondern auch direkt als eigenständiger Viewer. Der Editor sei nur dann attraktiv, wenn der Anwender eine Simulation erstellen möchte. Der allgemeinere Fall sei aber, dass der Anwender die Anlage nur begutachten möchte. Es kommt auch die Frage auf, ob die Anlage nicht auch mit einem Tablet begutachtet werden könnte. Ein kurzer Test dazu zeigt, dass das

7 Evaluation 7.5 Evaluation durch Fachkundige

Starten als eigenständiger Viewer zwar funktioniert, aber keine Anlage zu sehen ist. Das liegt daran, dass die Anlage nur im Editor generiert wird, nicht aber zur Laufzeit des Spiels. Das Starten im Web zeigt ein äquivalentes Bild. *Als Lösung könnte die Anlage nach dem Event „BeginPlay“ des Container-Actors erstellt werden.*

8 Zusammenfassung und Ausblick

Die Arbeit zeigt, in wie weit Spiel-Engines für die Verarbeitung von CAD-Daten geeignet sind. Dafür wird die Unreal Engine 4 als Spiel-Engine verwendet. Die Daten kommen aus, über einen Parser vermittelt, der Software Intergraph Smartplant 3D. Auch wenn bei dieser Arbeit nur ein geringer Teil der Geometrie unterstützt wird, zeigt sich, dass die gewählte Spiel-Engine dieser Herausforderung durchaus gewachsen ist. Zwar war die Performance, gemessen an den Bildern pro Sekunde, geringer als angenommen. Aber es handelt sich dabei auch um eine nicht optimierte Anwendung. Wichtige Aspekte, wie die mögliche Simulation und Virtual Reality, können mit Hilfe der ausgewählten Spiel-Engine umgesetzt werden. Die Evaluation durch die Fachkundigen zeigt, dass die Software als eigenständige Anwendung laufen sollte. Also nicht als Plugin innerhalb des Editors. Da die Unreal Engine 4 auch die Entwicklung für die HTML5-Plattform unterstützt[77][78], könnten Anlagen auch direkt im Web betrachtet werden, ohne, dass der Anwender die Anwendung vorher installieren müsste. Hier ist eine Cloudspeicherung durchaus denkbar.

Das Unternehmen *Unitec Informationssysteme GmbH* hat sein Interesse an der Arbeit bereits bestätigt. Es hält eine optimierte und eigenständige Anwendung für ein potentiell durchaus attraktives Produkt.

Der Sourcecode zur Ausarbeitung befindet sich auch Online unter Github:
<https://github.com/LeGone/Bachelor-Thesis-CAD-in-Game-Engine>

Der Parser ist lizenpflichtig und daher nicht Teil der Ausarbeitung.

9 Literaturverzeichnis

Literaturverzeichnis

- [1]: Müller-Schoppen, Erik, and Beate Kesper. *Managementwissen-kompakt*. BoD—Books on Demand, 2011, Seite 19-20
- [2]: Über 7,55 Milliarden Menschen auf der Welt [11.12.2017]
<http://www.br.de/themen/wissen/weltbevoelkerung-bevoelkerungswachstum-menschen-erde-welt-100.html>
- [3]: Bevölkerungsentwicklung[12.12.2017]
<http://www.bpb.de/wissen/I6T8RL>
- [4]: Voigt, Kai-Ingo. Risikomanagement im industriellen Anlagenbau: Konzepte und Fallstudien aus der Praxis. Erich Schmidt Verlag GmbH & Co KG, 2010. Seite 1
- [5]: Rodríguez, Juan M. Corchado, et al. "Trends in Practical Applications of Agents and Multiagent Systems." *Springer* 10 (2012): 978-3, Seite 58
- [6]: Kosmadoudi, Zoe, et al. "Game interactivity in CAD as productive systems." *Procedia Computer Science* 15 (2012): 285-288.
- [7]: Intergraph® FreeView®[12.12.2017]
<https://hexagonppm.com/products/3d-product-family/intergraph-freeview>
- [8]: Interview und eMail-Kontakt mit Unitec Informationssysteme GmbH am 20.1.2018
- [9]: Dynamic Vision[5.1.2018]
<http://www.dynamicvision.ca>
- [10]: Smartplant 3D Produktseite[4.2.2017]
http://www.intergraph.com/products/ppm/smart_3d/plant/default.aspx
- [11]: Intergraph[4.2.2017]
<http://www.intergraph.com/>
- [12]: Wikipedia: CAD[15.2.2018]
<https://de.wikipedia.org/wiki/CAD>
- [13]: Moran, Sean. *Process Plant Layout*. Butterworth-Heinemann, 2016. Seite 550 Tabelle A.1.
- [14]: Learn more about Intergraph Smart 3D - Intergraph PP&M[12.12.2017]
<https://www.youtube.com/watch?v=heA7PwMn3lI>

[15]: AutoCAD Formats[15.12.2017]

<https://knowledge.autodesk.com/support/autocad/learn-explore/caas/sfdcarticles/sfdcarticles/What-file-formats-can-AutoCAD-import.html>

[16]: UNITEC Anlagenbaulösungen[22.12.2017]

https://www.unitec.de/content/2-technologies/2-interfaces/UNITEC_Interface_Brochure.pdf

[17]: Tricad MS[15.2.2018]

<https://www.cadison.com/pdf/brochures/tricad-ms-building-services-engineering.pdf>

[18]: AVEVA[15.2.2018]

www.aveva.com

[19]: Bentley® OpenPlant Modeler[15.2.2018]

https://www.bentley.com/-/media/files/documents/product-data-sheet/pds_openplantmodeler_ltr_en_lr.pdf

[20]: AutoPIPE[15.2.2018]

https://communities.bentley.com/products/pipe_stress_analysis/w/pipe_stress_analyses__wiki/9620/01-what-file-formats-can-autopipe-importexport

[21]: Bentley AutoPLANT[15.2.2018]

<https://www.bentley.com/en/products/brands/autoplant>

[22]: Cadmatic Interoperability, eXchangers – File formats[15.2.2018]

https://www.cadmatic.com/en/products/cadmatic-3d-plant-design/interoperability_exchangers/interoperability-exchangers-file-formats.html

[23]: Polytrans[15.2.2018]

https://www.okino.com/conv/filefrmt_3dimport.htm

[24]: SmartPlant Interop Publisher 2015R[15.2.2018]

http://www.intergraph.com/assets/pdf/SPIOP_2015R1_3DModelFormatsandFiles.pdf

[25]: INTERGRAPH CADWORX[1.1.2018]

<http://www.intergraph.com/global/eu/go/cadworx.aspx>

[26]: Intergraph PDS[15.2.2018]

<http://www.intergraph.com/products/ppm/pds/>

[27]: Unitec Informationssysteme GmbH[15.2.2018]

<https://unitec.de/>

[28]: Unity is the little game engine that could revolutionize animated

movies[5.1.2018]

<https://www.theverge.com/2017/6/30/15899446/unity-cinemachine-unite-europe-2017-animation>

[29]: Unity Game Engine Just Democratized High-profile Film Making[15.2.2018]

<https://edgylabs.com/unity-game-engine-just-proved-that-film-making-could-be-democratized>

[30]: Lucasfilm using game engine to render movie scenes in real-time [15.2.2018]

https://www.gamasutra.com/view/news/200959/Lucasfilm_using_game_engine_to_render_movie_scenes_in_realtime.php

[31]: nVidia bestätigt Optimierung der Unreal Engine für deren Chips[15.2.2018]

<https://developer.nvidia.com/unrealengine>

[32]: Sperl, Stephan. *Die Semantisierung der Musik im filmischen Werk Stanley Kubricks*. Königshausen & Neumann, 2006. Seite 30

[33]: K Allm Ann, M. Arcelo, and Kost As Bekris. "Motion in games.". Seite 229

[34]: Which Game Engine Should I Choose?[15.2.2018]

<https://www.pluralsight.com/blog/film-games/unity-udk-cryengine-game-engine-choose>

[35]: Why Two Disney Films Rendered Scenes In Unreal Engine 4 [15.2.2018]

<https://www.cinemablend.com/games/1631230/why-two-disney-films-rendered-scenes-in-unreal-engine-4>

[36]: Unreal Engine 4 Plugins -

<https://docs.unrealengine.com/latest/INT/Programming/Plugins/> [15.2.2018]

[37]: An Introduction to UE4 Plugins[6.2.2018]

https://wiki.unrealengine.com/An_Introduction_to_UE4_Plugins

[38]: Error windows SDK v8.1 must be installed in order to build this target[6.2.2018]

<https://answers.unrealengine.com/questions/538760/error-windows-sdk-v81-must-be-installed-in-order-t.html>

[39]: "SDK v8.1 must be installed", even tho it already is[6.2.2018]

<https://answers.unrealengine.com/questions/589800/sdk-v81-must-be-installed-even-tho-it-already-is.html>

[40]: Why I need windows 8.1 sdk? - <https://forums.unrealengine.com/development-discussion/content-creation/88892-why-i-need-windows-8-1-sdk> [15.2.2018]

[41]: Epic Games[15.2.2018]

<https://www.epicgames.com>

[42]: JavaDoc[15.2.2018]

<https://en.wikipedia.org/wiki/Javadoc>

[43]: Kommentare der Unreal Engine 4[15.2.2018]

<https://docs.unrealengine.com/latest/INT/Programming/Development/CodingStandards/#comments>

[44]: Procedural Mesh Component[15.2.2018]

https://wiki.unrealengine.com/Procedural_Mesh_Component_in_C%2B%2B:Getting_Started

[45]: UE4: Umaterial[15.2.2018]

<https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/Materials/UMaterial/>

[46]: UE4: Material Instances[4.2.2017]

<https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/MaterialInstances/>

[47]: UE4: Empfohlene Hardware[8.2.2018]

https://wiki.unrealengine.com/Recommended_Hardware

[48]: UE4: Linux Support[15.2.2018]

https://wiki.unrealengine.com/Linux_Support

[49]: UE4: Building On Linux[15.2.2018]

https://wiki.unrealengine.com/Building_On_Linux

[50]: WineHQ[15.2.2018]

<https://www.winehq.org/>

[51]: Mono[15.2.2018]

<https://wiki.winehq.org/Mono>

[52]: PlayOnLinux[15.2.2018]

<https://www.playonlinux.com/en/>

[53]: Visual Studio Produktseite

<https://www.visualstudio.com> [15.2.2018]

[54]: Wikipedia zur Information über das Windows SDK[7.2.2018]

https://en.wikipedia.org/wiki/Microsoft_Windows_SDK

[55]: Why my .NET App does not load an assembly located in a folder added to the PATH environment variable?[6.2.2018]

<https://stackoverflow.com/questions/7342052/why-my-net-app-does-not-load-an-assembly-located-in-a-folder-added-to-the-path?rq=1>

[56]: DLL reference location - <https://stackoverflow.com/questions/12464743/dll-reference-location> [15.2.2018]

[57]: How to load an assembly at runtime that is located in a folder that is not the bin folder of the application[6.2.2018]

<https://support.microsoft.com/en-us/help/837908/how-to-load-an-assembly-at-runtime-that-is-located-in-a-folder-that-is>

[58]: Chris Conway[12.2.2018]

<https://github.com/Koderz>

[59]: UE4 Runtime Mesh Component[12.2.2018]

<https://github.com/Koderz/RuntimeMeshComponent>

[60]: Christer Ericson, Real-Time Collision Detection, Volume 1, Seite 496

[61]: earclipper library[15.2.2018]

<https://github.com/NMO13/earclipper>

[62]: Martin Ennemoser bei Github[4.2.2017]

<https://github.com/NMO13>

[63]: Cheng, Siu-Wing, Tamal K. Dey, and Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2012.

[64]: OpenTK: CreateFromAxisAngle (Method)[12.2.2018]

http://www.nudoq.org/#!/Packages/OpenTK_GLControl/OpenTK/Matrix4/M/CreateFromAxisAngle

[65]: AActor[6.2.2018]

<https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/GameFramework/AActor/>

[66]: UE4-Property: VisibleAnywhere[6.2.2018]

<https://docs.unrealengine.com/latest/INT/Programming/UnrealArchitecture/Reference/Properties/Specifiers/VisibleAnywhere/>

[67]: Opazität[8.2.2018]

<https://de.wikipedia.org/wiki/Opazit%C3%A4t>

[68]: Refraction[8.2.2018]

<https://en.wikipedia.org/wiki/Refraction>

[69]: UE4: Refraction[15.2.2018]

<https://docs.unrealengine.com/latest/INT/Engine/Rendering/Materials/HowTo/Refraction>

[70]: Liste der Refractionsindexe[8.2.2018]

https://en.wikipedia.org/wiki/List_of_refractive_indices

[71]: specular highlights on translucent material[8.2.2018]

<https://answers.unrealengine.com/questions/257409/specular-highlights-on-translucent-material.html>

[72]: UE4: Navigating the Viewport[10.2.2018]

<https://docs.unrealengine.com/latest/INT/Engine/QuickStart/2/>

[73]: Unreal Engine VR Mode[11.2.2018]

<https://docs.unrealengine.com/latest/INT/Engine/Editor/VR/>

[74]: SmartPlant Interop Publisher[10.2.2018]

<http://www.intergraph.com/products/ppm/smartplant/interop/publisher/default.aspx>

[75]: UE4: Procedural Mesh Component Rendering Performance

Concerns[11.2.2018]

<https://answers.unrealengine.com/questions/412702/procedural-mesh-component-rendering-performance-co.html>

[76]: UE4: GPU Profiler[11.2.2018]

<https://docs.unrealengine.com/latest/INT/Engine/Performance/GPU/>

[77]: UE4: Getting Started: Developing HTML5 Projects[6.2.2018]

<https://docs.unrealengine.com/latest/INT/Platforms/HTML5/GettingStarted/>

[78]: UE4: HTML5 Game Development[6.2.2018]

<https://docs.unrealengine.com/latest/INT/Platforms/HTML5/>