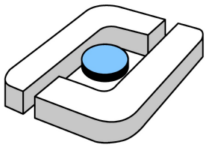

Hochschule Osnabrück

University of Applied Sciences

Fakultät

Ingenieurwissenschaften und Informatik

Bachelorarbeit

über das Thema:

Entwicklung eines WordPress Plug-ins zur Einbindung von CAD-Modellen in Webseiten

Autor:	Fide Schürmeyer
	fide.schuermeyer@hs-osnabrueck.de
1. Prüfer:	Herr Prof. Dr. Philipp Lensing
2. Prüfer:	Sven Ludwig
Abgabedatum:	24.05.2016

WICHTIG!!

An diese Stelle in der gebundenen Ausgabe der Bachelorarbeit das Themenblatt einfügen.

I Kurzfassung

Ziel dieser Bachelorarbeit ist die prototypische Implementierung eines WordPress Plug-ins zur Darstellung von dreidimensionalen CAD Modellen in Webseiten. Die Modelle werden mit WebGL und der JavaScript-Bibliothek three.js dargestellt. Dieses Dokument beschreibt zunächst die technischen Grundlagen für die Erstellung eines solchen Plug-ins. Im weiteren Verlauf wird zunächst die Modellierung und Konzeptionierung der Software, gefolgt von ihrer Implementierung beschrieben. Weiterhin wird die Brauchbarkeit des Plug-ins durch Leistungstests bewertet, die zunächst entwickelt und dann ausgeführt werden. Die Analyse der Testergebnisse zeigt eine hohe Belastung für den Browser. Diese Belastung ist vor allem auf mobilen Geräten problematisch, sodass von einer Benutzung des Plug-ins für mobile Webseiten abzuraten ist. Schlussendlich zeigt der Prototyp, dass das Plug-in mit der genutzten Technik realisierbar ist. Die Brauchbarkeit wird dadurch eingeschränkt, dass WebGL nicht darauf ausgelegt ist, viele Darstellungen in einer einzelnen Webseite zu erstellen.

Abstract

The intention of this thesis is the prototypical implementation of a WordPress plug-in that displays three-dimensional CAD models in websites. The models are displayed using WebGL, aswell as three.js, a JavaScript library for computer graphics in web browsers. This document begins by describing the basic principles needed for the implementation of the plug-in. Furthermore it describes the modeling process and the conceptual design followed by the actual implementation. The viablility of the prototype will then be evaluated by testing its performance. Therefore tests are created and then executed. Breaking down the gathered data shows a high workload for the web browser. This workload is especially problematic for mobile hardware, which leads to the conclusion that the plug-in is not viable for the use in mobile websites. In conclusion the prototype shows the implementability of the plug-in using the given software. Even though the viability is limited by WebGL not being designed to support multiple instances in one page.

II Inhaltsverzeichnis

III Abbildungsverzeichnis

IV Tabellenverzeichnis

1 Einleitung

Webseiten werden zunehmend mit aufwändigen Grafiken und Animationen versehen, um ansprechender zu werden. Nicht selten werden im Hintergrund der Webseite Filme abgespielt, die über das Unternehmen oder Produkte informieren. Um diesem Trend gerecht zu werden, soll es dem Unternehmen ermöglicht werden, eine 3D-Darstellung ihrer Produkte in eine Webseite einzubinden. Möbel, diverse Bauteile, aber auch Gebäude werden häufig am Computer mit Hilfe von CAD-Software konstruiert. Dem Unternehmen soll ermöglicht werden, bereits bei der Erstellung des Produktes genutzte 3D-Modelle in der Webseite zu präsentieren und somit zu Werbezwecken wiederzuverwerten. Die Webseiten von Unternehmen basieren häufig auf Content-Management-Systemen. Das Content-Management-System mit der höchsten Verbreitung ist WordPress.[@wps] In dieser Arbeit wird ein Plug-in für WordPress erstellt, dass die Darstellung von CAD-Dateien ermöglicht.

1.1 Vorstellung des Unternehmens

Die comlot GmbH entwickelt als Internet- und Intranetspezialist seit über 10 Jahren webbasierte Standard- und Individualsoftware für Großunternehmen, den Mittelstand im produzierenden Gewerbe und Handel sowie für Beratungsunternehmen und Agenturen. Das Dienstleistungsspektrum umfasst dabei alle Prozesse von der Frontendentwicklung, über Schnittstellen zu Backendsystemen (z.B. ERP-Systemen), sowie die Anbindung von API‘s zu Drittanbietern (z.B. Zahlungssysteme, Bewertungssysteme, u.v.m.).

1.2 Ziele und Aufgabenstellung

Ziel der Bachelorarbeit soll die prototypische Implementierung eines WordPress-Plug-ins zur Darstellung von dreidimensionalen CAD Modellen in Webseiten sein. Die Darstellung der Modelle soll mit Web GL und der Bibliothek three.js erfolgen. Der Prototyp soll sowohl verschiedene Anwendungsgebiete aufzeigen, als auch eine einfache Bedienung über das

Backend ermöglichen. Außerdem soll getestet werden, inwiefern sich die Benutzung von WebGL-Elementen die Leistung der Seite beeinträchtigt.
Die Leistung einer Webseite lässt sich anhand der Ladezeit bewerten. Die Berechnung der Grafik findet auf dem Gerät des Klienten statt. Es muss geprüft werden, ob die Seite nach Ende des Ladens noch flüssig reagiert und ob die Animationen flüssig ablaufen. Die Ergebnisse der Tests sollen zur Bewertung der Brauchbarkeit eines solchen Plug-ins herangezogen werden.

1.3 Aufbau der Arbeit

Diese Arbeit beginnt damit, dass die Grundlagen zum Verständnis der restlichen Arbeit erklärt werden, indem die verwendeten Techniken und ihre wesentlichen Konzepte vorgestellt werden. Das Kapitel Modellierung beschreibt den Entwurf der Software, mit den dahinter liegenden Konzepten. Im Kapitel Implementierung wird die Erstellung der Software beschrieben. Zudem werden einzelne Teile der Software genauer erläutert. Das fünfte Kapitel beschreibt das Testen der erstellten Software. Dazu werden zunächst die Konzepte und die Durchführung der Tests beschrieben und anschließend die Ergebnisse der Tests analysiert und bewertet. Am Ende der Arbeit werden die Ergebnisse dargestellt und bewertet. Weiterhin werden Erweiterungsmöglichkeiten der Software dargestellt. Den Schluss bildet ein persönliches Fazit.

2 Grundlagen

Dieses Kapitel legt die Grundlagen zum Verständnis der erstellten Arbeit. Dabei werden Grundkenntnisse der Web-Technologie sowie Kenntnisse von HTML, JavaScript, jQuery und PHP vorausgesetzt. Die ersten Kapitel stellen mit Web-Servern, Content-Management-Systemen und WordPress den Rahmen dar, in den die grafische Programmierung eingebettet wird. Die Abschnitte OpenGL, WebGL und three.js beschreiben die Programmierung der Grafik und ihre Konzepte.

2.1 Web-Server

Ein Webserver ist eine Software, die HTTP Anfragen bearbeitet. Die Hauptaufgaben sind das Speichern und das Ausliefern von Dateien [@serv]. Moderne Webserver lassen sich mit Skriptsprachen, wie beispielsweise ASP.NET oder PHP, programmieren. Dies wird genutzt, damit der Webserver nicht nur statische Dokumente ausliefern kann, sondern dynamische Inhalte bereitstellen kann. Diese Inhalte werden oftmals in Datenbanken gespeichert und dann vom Server ausgelesen und in das angefragte Dokument eingesetzt.
Der meist genutzte Webserver ist der Apache Webserver des gleichnamigen Herstellers. Andere viel genutzte Webserver sind der nginx von NGINX Inc. und IIS von Microsoft [@servs]. Die Webserver unterscheiden sich teilweise, für alle in dieser Arbeit beschriebenen Anwendungen genügt allerdings, dass sie PHP 5.2.4 oder höher unterstützen, damit auf ihnen WordPress ausgeführt werden kann.
Weitere Aufgaben von Webservern können Zugriffsbeschränkung, Sicherheitsgewährleistung, sowie Protokollierung und das Anzeigen von Fehlern sein.
Zugriffsbeschränkung bedeutet, dass auf ein Verzeichnis aus dem Internet nicht oder nur beschränkt zugegriffen werden kann. Dies wird auf Apache Servern beispielsweise durch eine .htaccess-Datei realisiert, in der verzeichnisbezogene Regeln definiert werden können [@htac].
Die Sicherheit der Aufrufe kann der Server gewährleisten, indem er eine Verschlüsselung der Kommunikation über HTTPS anbietet. Die zu übertragenen Daten werden beispielsweise durch SSL/TLS verschlüsselt. Der Server benötigt dafür Verschlüsselungszertifikate.[@TLS]

Die Fehlerbehandlung geschieht über HTTP-Statuscodes, die dem Browser durch den HTTP-Header mitgeteilt werden und zur Anzeige einer zu dem Fehlercode definierten HTML-Seite führen. Den Fehler *404 Not Found* wird ein Server immer dann zurück geben, wenn eine Seite oder Ressource angefragt worden ist, die der Server nicht finden kann [@http]. Häufig definieren Webseiten eine eigene Fehlerseite, auf der auf mögliche Ursachen hingewiesen wird. Eine Protokollierung der Anfragen kann interessant sein, um das Verhalten der Nutzer der Webseite zu analysieren. Des Weiteren können auf der Grundlage von Protokollen mögliche Fehlerquellen aber auch Angreifer erkannt werden.

2.2 Content-Management-Systeme

Content-Management-Systeme sind Programme, die auf Webservern ausgeführt werden und das Erstellen, Verwalten und Veröffentlichen von Inhalten erleichtern. In der Regel handelt es sich bei den Inhalten um Webseiten. Im Folgenden wird der Begriff Content-Management-System (CMS) für eben solche Content-Management-Systeme zum Verwalten von Webseiten benutzt. Die Abbildung 1 stellt den Aufbau eines CMS schematisch dar.

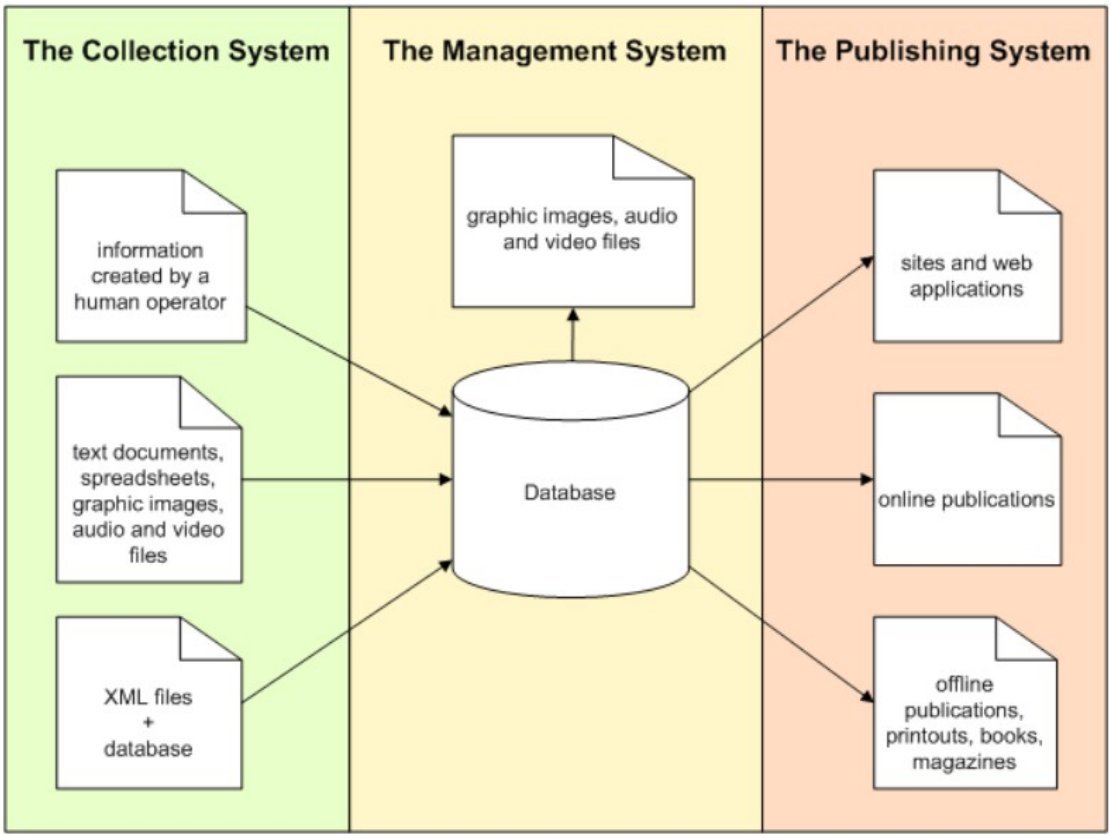

Abbildung 1: Aufbau Content-Management-System[@cms]

Das CMS besteht aus drei Teilen, dem Collection System, dem Management System und dem Publishing System. Das Collection System ermöglicht dem Benutzer das Erstellen und Einpflegen von Inhalten. Im Folgenden wird das Collection System als Backend bezeichnet. Das Publishing System ist für die Darstellung der gesammelten Inhalte zuständig. Im Fall von Web-Content-Management Systemen handelt es sich dabei um die eigentliche Webseite. Das Publishing System wird im Folgenden als Frontend bezeichnet.

Bei dem Management System handelt es sich um den Kern des CMS. Hier werden die Daten gespeichert und für die Darstellung im Frontend bereitgestellt. Das Management System verbindet somit die im Backend eingebundenen Daten mit ihrer Darstellung im Frontend. Dazu werden Layouts definiert. Layouts sind Formatvorlagen, die beschreiben, wie Inhalte aus dem Backend im Frontend angezeigt werden. Eine der Aufgaben des Management Systems kann das Verwalten von Benutzern des Backends sein. Benutzern können bestimmte Rechte und Aufgaben im Backend zugeteilt werden.[@cms]

Diese Struktur hat den Vorteil, dass ein Autor von Texten lediglich wissen muss, wie er Texte in das Backend einfügt, allerdings kein Wissen im Bereich der Erstellung von Webseiten haben muss.

Moderne CMS bieten die Möglichkeit, die Funktionalität durch das Einbinden von Plug-ins zu erweitern. Plug-ins werden in einer vom CMS abhängigen Programmiersprache erstellt.

Es existiert eine Vielzahl verschiedener Content-Management-Systeme. Die Abbildung 2 zeigt die Verbreitung der 20 meistgenutzten Open-Source Content-Management-Systeme.

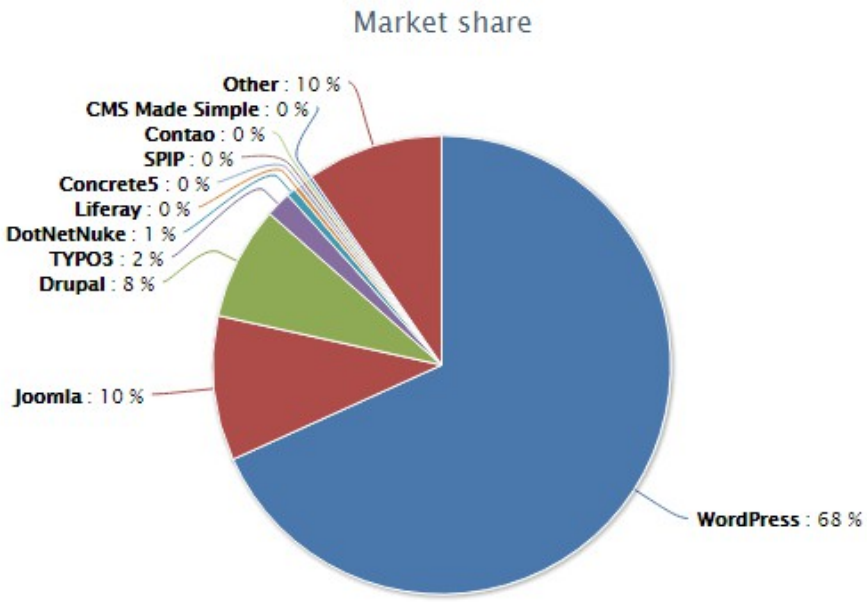

Abbildung 2: Marktanteile CMS [@wps]

Klarer Marktführer ist WordPress, gefolgt von Joomla und Drupal.[@wps] Auffällig ist, dass in Deutschland das CMS TYPO3 im Vergleich zum weltweiten Durchschnitt sehr beliebt ist. In Deutschland liegt TYPO3 mit einem Anteil von 17% als Zweitplatzierter hinter WordPress mit 35%. Auch das CMS Contao ist in Deutschland mit einem Marktanteil von 4% verhältnismäßig beliebt.[@cmss]
Im weiteren wird das Content-Management-System WordPress vorgestellt.

2.3 WordPress

WordPress ist eine freie und quelloffene Software. Aktuell ist WordPress das meist genutzte Content-Management-System [@wps]. Es basiert auf PHP und benutzt MySQL als Datenbank. Ein WordPress System unterteilt sich in die eigentliche Webseite, auch Frontend genannt, und das Backend, das zur Administration des Frontends genutzt wird. Das Backend stellt Funktionen zur Verwaltung von Posts, Medien, Webseiten und Kommentaren zur Verfügung. WordPress kann mit Hilfe von Themes und Plug-ins an die individuellen Anforderungen des Benutzers angepasst werden. Über 42000 Plug-ins sind derzeit im integrierten Plug-in-Browser verfügbar. Die Plug-ins werden in PHP geschrieben. WordPress stellt so genannte Hooks zur Verfügung, mit denen Plug-ins auf Funktionen von WordPress zugreifen können. Es handelt sich hierbei um Event Handler, die auf bestimmte Ereignisse von WordPress reagieren. Die Hooks werden in Action Hooks und Filter Hooks unterteilt. Action Hooks reagieren auf bestimmte Ereignisse, wie beispielsweise eine Benutzerinteraktion. Zum Beispiel wird der Action Hook *comment_post* aktiviert, wenn ein Kommentar abgeschickt wird, sodass auf das Verfassen eines Kommentars reagiert werden kann. Soll der Inhalt des Kommentars geprüft werden, kann ein Filter Hook verwendet werden. Der Filter Hook *comment_text* liefert beispielsweise den Inhalt eines Kommentars, sodass dieser überprüft oder sogar manipuliert werden kann.
Themes können die Gestaltung, sowie die Funktion von WordPress verändern. Sie können ausgetauscht werden, ohne den Inhalt der Seite zu verändern. WordPress wird mit einigen verschiedenen Themes ausgeliefert und es können weitere freie Themes von wordpress.org heruntergeladen werden. Zudem gibt es kostenpflichtige Themes, die lizenziert werden

können. Die Themes können nach Belieben angepasst werden und bestehen aus PHP-, HTML- und CSS-Code.[@wp]
Shortcodes sind in WordPress Schlüsselwörter oder Platzhalter, die der Benutzer in einem Post oder einer Seite zum Einbinden von Inhalten benutzen kann. Der Shortcode *[gallery]* bindet beispielsweise eine Galerie für Bilder ein. WordPress ermöglicht es, in Plug-ins eigene Shortcodes zu definieren, die dann vom Benutzer verwendet werden können. Im Folgenden wird gezeigt, wie dieses Projekt Shortcodes zur Darstellung der Modelle im Frontend benutzt. [@sc]

2.4 OpenGL

Die Open Graphics Library ist die Spezifikation einer Programmierschnittstelle zum Rendern von Grafiken. Sie definiert über 500 verschiedene Befehle, die benutzt werden können, um interaktiver dreidimensionale Computergrafiken zu erstellen. OpenGL ist dabei unabhängig von Hardware und Betriebssystem. Nicht enthalten in OpenGL sind Funktionalitäten zur Erstellung von Fenstern oder zur Verarbeitung von Benutzereingaben. Diese müssen von einer Anwendung in Abhängigkeit von dem Betriebssystem bzw. dem Fenstermanager des Betriebssystems implementiert werden. Auch nicht enthalten sind Funktionalitäten zur Erstellung kompletter dreidimensionaler Objekte. Diese müssen mit Hilfe von grafischen Primitiven wie Punkten, Linien und Dreiecken zusammengesetzt werden.[DGJ13]
Die Abbildung 3 zeigt die verschiedenen grafischen Primitive von OpenGL und deutet an, wie mit Hilfe so genannter Strips komplexere Formen entstehen.

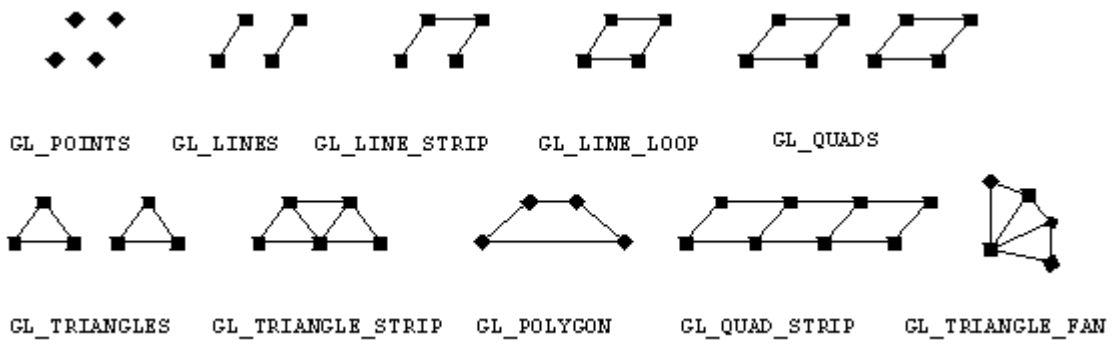

Abbildung 3: OpenGL grafische Primitive [@prim]

Der Vorgang der Erstellung eines fertigen Bildes mithilfe dieser primitiven Formen wird Rendering genannt. In OpenGL ist dazu die so genannte Rendering Pipeline spezifiziert. Sie beschreibt die zu durchlaufenden Schritte um von den Rohdaten zu einem Bild zu gelangen, wie auf der Abbildung 16 im Anhang zu sehen ist. Die blau hinterlegten Felder sind durch Shader programmierbar, Felder mit gestrichelten Linien sind optional. In dem Schritt Vertex Specification wird eine Liste von Vertices, die ein grafisches Primitiv beschreiben, an die Rendering Pipeline übergeben. Jeder der übergebenen Vertices wird durch einen Vertex-Shader bearbeitet.[@pipe]

Shader sind Programme, die zu bestimmten Zeiten im Grafikprozessor ausgeführt werden. Verschiedene Shader haben dabei unterschiedliche Aufgaben bei der Erstellung der Grafik. Dabei werden diese in OpenGL in der OpenGL Shading Language (GLSL) geschrieben.

Der OpenGL Vertex Shader bearbeitet die einzelnen Vertices. Die Aufgabe des Vertex Shaders ist dabei üblicherweise das Transformieren (Bewegung, Skallierung) oder Färben der Vertices anhand bestimmter Regeln. Dabei gilt die Beschränkung, dass es zu jedem eingehenden Vertex nur genau einen ausgehenden Vertex geben darf. Daraus folgt, dass das Ergebnis gleich bleibt, wenn der Vertex Shader mehrmals mit den gleichen Parametern aufgerufen wird. Dies ermöglicht Optimierung des Shading Vorgangs durch Caching.[@vert]

Der nächste Schritt nennt sich Tessellation und kann benutzt werden, um die grafischen Primitive weiter zu unterteilen.

Im Geometry Shader kann ebenfalls auf die grafischen Primitive zugegriffen werden. Es gibt die Möglichkeit, diese zu Löschen oder zu unterteilen. Des Weiteren kann der Geometry Shader ähnlich wie der Vertex Shader auf die einzelnen Vertices zugreifen. Dies ermöglicht ihm auch, die grafischen Primitive an sich zu ändern, indem er beispielsweise aus einzelnen Punkten Dreiecke formt.

Der Schritt des Vertex post-processsing beinhaltet einige festgelegte Funktionen zur Bearbeitung der Vectices, die hier nicht näher erläutert werden sollen.

Primitive assembly bedeutet, dass die Vertex Daten aus den vorherigen Schritten zu einer Serie aus grafischen Primitiven zusammengefasst werden. Dieser Schritt wird etwas früher ausgeführt, da Tessellation und Geometry Shader bereits zusammengesetzte Vertices benötigen.

Die Rasterization beschreibt den Prozess des Umwandelns von grafischen Primitiven zu Fragmenten. Fragmente enthalten die Infromationen zur Erstellung der Pixel.

Diese Fragmente werden im nächsten Schritt an den Fragment Shader weitergegeben. Dieser kann die Farb- und Tiefenwerte des Fragments verändern.
Der letzte Schritt enthält eine Reihe Operationen, die durchlaufen werden und an deren Ende die resultierenden Daten in die entsprechenden Bildspeicher geschrieben werden. Der Darstellungsprozess ist damit abgeschlossen.[@pipe]
Zur Erstellung von Objekten mit komplizierter Farbdarstellung, könnten die einzelnen grafischen Primitive in der Programmierung angepasst werden. Eine einfache Lösung dafür ist das Texture Mapping in OpenGL. Es bietet die Möglichkeit, ein Bild auf die Oberfläche eines Objektes zu legen. Texturen können in OpenGL ein- bis dreidimensional sein. Die Texturen werden an die Objekte gebunden und mithilfe eines Shaders dargestellt.[DGJ13b]

2.5 WebGL

Die Web Graphics Library ist eine Programmierschnittstelle zur Darstellung von hardwarebeschleunigten 3D-Grafiken in Webbrowsern. WebGL ist ein lizenzfreier Standard, der von der Khronos Group entwickelt wird. Basierend auf OpenGL ES 2.0 ermöglicht WebGL das Einbinden von 3D-Elementen in HTML. OpenGL ES ist eine Adaption des 3D-Rendering Standards OpenGL für eingebettete Systeme. Das bedeutet, dass es für die Verwendung auf kleineren Geräten, wie zum Beispiel Smartphones oder Tablets, optimiert ist. OpenGL ES wird bereits auf iPhone, iPad und Android Geräten zum Darstellen von 3D-Grafiken eingesetzt. [par12] Die Darstellung in der Webseite geschieht durch die Benutzung des HTML5 Canvas Elements. [@webGl] Der Programmcode zur Erstellung der Grafik kann in JavaScript geschrieben werden. Dadurch, dass WebGL normale HTML-Elemente benutzt, lässt es sich problemlos in HTML-Seiten einbinden. Es kann vor oder hinter beliebigen anderen HTML-Elementen benutzt werden, sodass es sich auch in bestehende dynamische Web-Anwendungen ohne größere Anpassungen integrieren lässt.[par12] Shader für WebGL werden in GLSL erstellt, welches seit OpenGL 2.0 als Standard für die Erstellung von Shadern in OpenGL genutzt wird.[@webGl][@oGL shader]

2.6 three js

Three.js ist eine JavaScript 3D Engine auf der Basis von WebGL zur Darstellung und Animation von 3D-Grafiken in Webbrowsern. Die Engine ist quelloffen und wird auf GitHub zur Verfügung gestellt.[par12b][threeGit] Three.js ist objektorientiert aufgebaut und stellt eine Vielzahl an Funktionen bereit, die das Erstellen von Grafiken vereinfachen.[par12b] Zusätzlich wird three.js mit einer Vielzahl von Beispielen ausgeliefert, die den Einstieg in die Programmierung mit three.js vereinfachen.[threeExam]
Durch den Einsatz von bewährten Verfahren zur Darstellung von 3D-Grafiken sichert three.js eine hohe Leistungsfähigkeit.[wgl] Zusätzlich abstrahiert Three.js den Rendering-Code von WebGL, sodass der Programmierer nicht zwingend Kenntnisse in WebGL benötigt.[par12b] Außerdem enthält three.js einige mathematische Funktionen, darunter sind Matrizen, Projektionen, Vektoren und die damit verknüpften üblichen Rechenverfahren.
Three.js stellt so genannte Loader zur Verfügung, die das Importieren von Dateien aus vielen 3D-Grafikprogrammen erlauben.

Dateiformat	Beschreibung
Babylon	Babylon.js ist ein JavaScript Framework zur Erstellung von 3D Spielen auf Basis von HTML5 und WebGL. Es benutzt ein JSON basiertes Dateiformat zum Speichern von Scenes. [@babylon]
Collada	COLLADA (COLLAborative Design Activity) ist ein XML-basiertes offenes Austauschformat für Daten zwischen verschiedenen 3D-Programmen.[@collada]
FBX	FBX ist ein proprietäres Dateiformat von Autodesk. Es unterstützt Objekte, Lichter, Kameras, Meshes, Materialien, Texturen und Animationen.[@fbx]
MD2	MD2 ist ein Dateiformat zum Speichern von animierten Modellen. Es wird von der id Tech 2 Engine benutzt, in der Spiele wie Quake II erstellt worden sind.[@md2]

OBJ/MTL	Das Wavefront OBJ Dateiformat ist ein Standard zum Speichern von geometrischen Daten. Optische Eigenschaften wie das Material werden in einer separaten MTL Datei gespeichert Die Materialien können in der OBJ Datei referenziert werden.[@obj]

Tabelle 1: Liste der unterstützten Dateiformate

Weitere unterstützte Dateiformate können in der Dokumentation von three.js nachgelesen werden.

Die folgende Tabelle gibt einen Überblick über weitere Funktionen von three.js.[@threef]

Funktion	Beschreibung
Szenen	Szenen verwalten die Darstellung, die von Three.js gerendert wird. Dazu gehören Objekte, Licht und Kameras. Des Weiteren wird Nebel unterstützt.
Kameras	Three.js stellt perspektivische Kameras und orthografische Kameras zur Verfügung. Zusätzlich gibt es “Controls”, die die Steuerung der Kamera übernehmen können.
Lichter	Three.js stellt Ambient-Lights, Directional-Lights, Hemisphere-Lights, Point-Lights und Spotlights zur Verfügung.
Materialien	Es werden Materialien mit Lambert und Phong Reflexionsmodellen zur Verfügung gestellt. Diese unterstützen Texturen und erweiterte Einstellungen zur Beleuchtung.
Shader	Es wird die OpenGL Shading Language GLSL unterstützt
Objekte	Bereit gestellt werden unter anderem: Meshes, Partikel, Sprites, Linien und Bones.
Geometrien	Bereitgestellt werden unter anderem: Ebenen, Quader, Sphären, Tori und Texte.
Animation	Es werden Morph- und Keyframeanimationen unterstützt

Tabelle 2: Funktionen von three.js

Im Folgenden wird ein typischer Programmaufbau mit three.js beschrieben.
Zunächst wird ein Renderer benötigt, welcher für die Darstellung aller mit three.js erstellten Objekte zuständig ist. Die Größe des Renderers wird über seine Attribute festgelegt und entspricht der Größe des Fensters auf der Webseite, in dem die Darstellung stattfindet. Ist der Renderer fertig konfiguriert, kann er in die Webseite eingebunden werden.
Damit der Renderer eine Darstellung erstellen kann, benötigt er eine Szene und eine Kamera, durch die die Szene betrachtet wird.
Die Szene ist ein Objekt, welches alle darzustellenden Objekte enthält und sie verwaltet. Zu den darzustellenden Objekten gehören geometrische Objekte, sowie Meshes und Lichter. In three.js gibt es sowohl orthografische Kameras als auch perspektivische Kameras. Die Kameras können frei positioniert und ausgerichtet werden.
Sind sowohl Szene als auch Kamera vorhanden, kann die Render-Funktion des Renderers aufgerufen werden, um eine Darstellung zu erstellen. Damit diese erneuert wird, sollte die Funktion in einem regelmäßigen Intervall aufgerufen werden. Dazu wird die *requestAnimationFrame* Funktion des Browsers benutzt.[@threeex]

3 Modellierung

In diesem Kapitel wird die Modellierung des Plug-ins vorgestellt. Dazu werden die grundlegenden Konzepte erläutert. Die Implementierung dieser Konzepte folgt im nächsten Kapitel. Zunächst wird das Konzept der Funktionalität und der Benutzung des Plug-ins beschrieben. Im zweiten Abschnitt wird die Softwarearchitektur der einzelnen Module beschrieben.

3.1 Konzeption des Plug-ins

Das Plug-in soll es ermöglichen, CAD-Dateien mit dreidimensionalen Modellen in eine WordPress Seite einzubinden.
Als zu unterstützendes Dateiformat wurde FBX ausgewählt. FBX eignet sich aufgrund seiner hohen Verbreitung, sowie der Speicherung von Materialien, Texturen und Modell in einer einzelnen Datei. Die hohe Verbreitung entsteht dadurch, dass der Marktführer im Bereich CAD, Autodesk, dieses Dateiformat nutzt. Auch andere 3D-Programme, wie Cinema4D, 3D Studio MAX und Blender unterstützen dieses Dateiformat. Weiterhin bietet Autodesk einen FBX Converter an. Dieser erlaubt das Umwandeln von unter anderem Wavefront OBJ-Dateien in FBX-Dateien. Auch von three.js wird das Einlesen von FBX-Dateien unterstützt.
Das Plug-in besteht aus einem Frontend- und einem Backendmodul. Dabei ist das Frontend-Modul für die Darstellung der Modelle auf der Webseite verantwortlich. Durch den Einsatz von Shortcodes wird es in Webseiten eingebunden, wobei der Shortcode die nötigen Informationen zur erfolgreichen Darstellung enthält. Die Implementierung durch Shortcodes bietet den Vorteil, dass diese frei in WordPress platziert werden können.
Das Backend-Modul, im Folgenden auch Model Viewer genannt, stellt eine grafische Oberfläche zur Erstellung des Shortcodes bereit. Eine Vorschau zeigt dabei die Auswirkungen der Änderungen an den Einstellungen. Sind die gewünschten Einstellungen getätigt, kann über einen Button der Shortcode für die aktuelle Vorschau erstellt werden.
In diesem Abschnitt wird zunächst die Konzeption der Benutzeroberfläche des Backend-Moduls und anschließend der Workflow bei der Benutzung des Plug-ins vorgestellt.

3.1.1 Konzeption der Oberfläche

Die Benutzeroberfläche des Backend-Moduls erlaubt dem Benutzer die Szene, in der das Modell dargestellt wird, zu verändern. Strukturell ist die Benutzeroberfläche in zwei Bereiche unterteilt wobei die linke Seite die verschiedenen Regler anzeigt und die rechte Seite zur Darstellung der Vorschau verwendet wird.

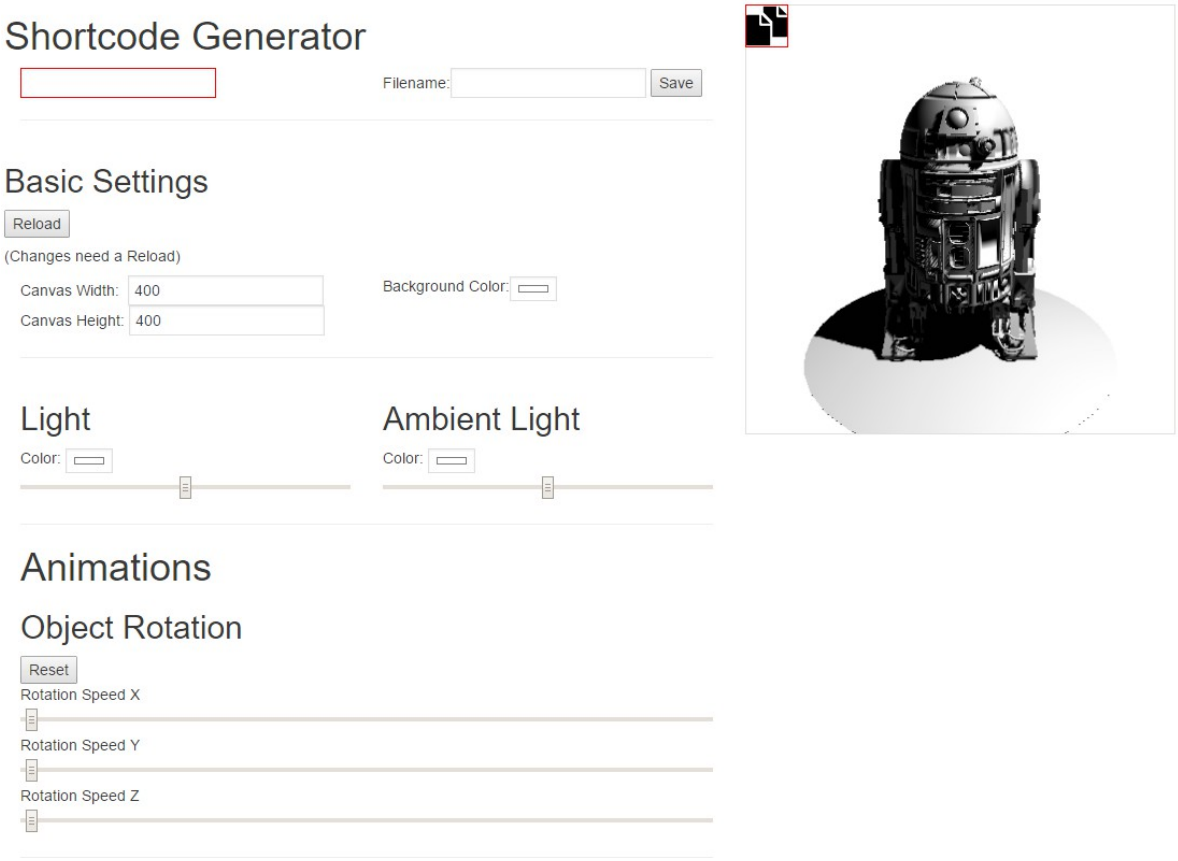

Abbildung 4: Benutzeroberfläche

Im Umfang von three.js ist ein Editor für Szenen enthalten. Dieser wurde nicht als Basis für die Benutzeroberfläche genutzt, um diese so übersichtlich wie möglich zu gestalten. Der Benutzer soll nicht mit der Menge der Einstellungsmöglichkeiten überfordert werden, sondern mit wenigen Einstellungen eine fertige Szene erstellen können.
Folgende Einstellungsmöglichkeiten werden dem Benutzer angeboten:

- Breite und Höhe der Darstellung

- Hintergrundfarbe der Darstellung
- Farbe und Intensität des Lichts
- Farbe und Intensität der Umgebungsbeleuchtung
- Rotationsgeschwindigkeit des Objekts
- Rotationsgeschwindigkeit der Kamera

In den erweiterten Einstellungen kann der Benutzer die Farbe der Plattform ändern und diese nach Wunsch deaktivieren. Zudem kann er das Beleuchtungsmodell des Modells ändern, sowie die Farbe des Modells anpassen.

3.1.2 Workflow

Dieser Abschnitt beschreibt die benötigten Arbeitsabläufe, um mit Hilfe des Plug-ins Modelle in eine WordPress Seite einzubinden. Die Abbildung 5 zeigt die Arbeitsschritte aus Sicht des Benutzers, welche sich in zwei Kategorien unterteilen lassen: Dem Erstellen des Shortcodes und dem Einbinden des Shortcodes. Im Folgenden wird zunächst das Erstellen der Shortcodes und dann das Einbinden der Shortcodes erläutert.

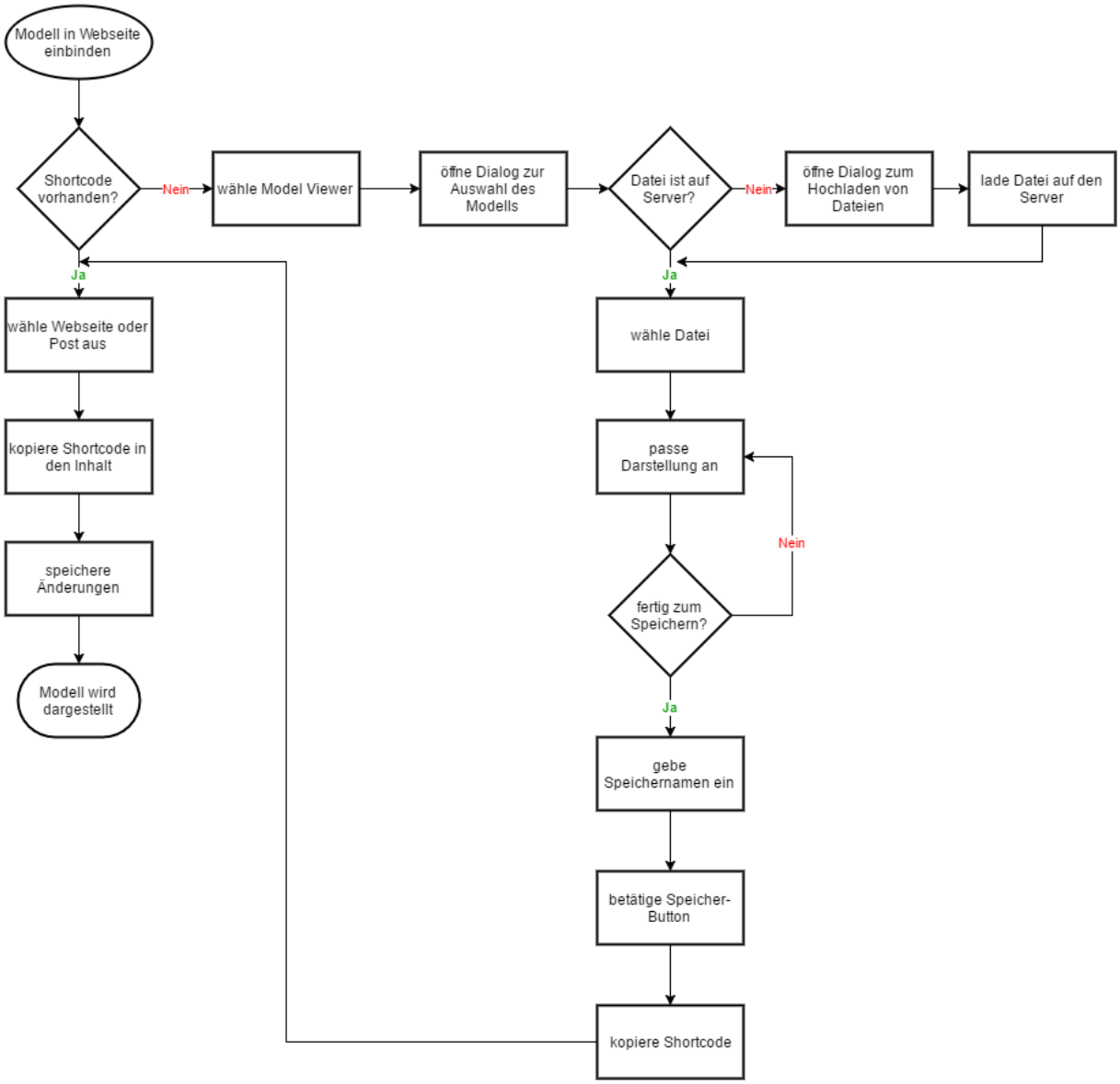

Abbildung 5: Workflow

Um einen Shortcode zu erstellen wählt man im Backend von WordPress den Model Viewer aus. Der nächste Schritt ist das öffnen des Dialoges zur Dateiauswahl. Ist die gewünschte Datei noch nicht auf dem Server vorhanden, kann sie über den Reiter Hochladen auf den Server geladen werden. Mit dem Bestätigen der Auswahl wird der Dialog geschlossen und die ausgewählte Datei wird dargestellt. Im nächsten Schritt wird die Darstellung über die Benutzeroberfläche den Wünschen angepasst. Nach Tätigung der Einstellungen muss die Szene gespeichert werden. Dazu wird zunächst ein Name in das vorgesehene Feld eingegeben und danach der Button zum Speichern betätigt. Nach erfolgreichem Speichern wird der generierte Shortcode angezeigt. Dieser wird jetzt vom Benutzer aus dem Feld kopiert und

kann jetzt benutzt werden, um die gerade erstellte Darstellung in eine Seite oder einen Post einzubinden.

Zum Einbinden eines vorhandenen Shortcodes, wählt der Benutzer zunächst eine Seite oder einen Post über die entsprechenden WordPress Module aus und öffnet dann den Dialog zum Bearbeiten des Inhalts. Der Shortcode wird an die gewünschte Stelle im Inhalt eingefügt und nach dem Speichern des Inhalts wird die eingebundene Szene im Frontend angezeigt.

3.2 Software-Architektur

Wie bereits erwähnt ist das Plug-in in ein Frontend- und ein Backendmodul unterteilt. Diese Module funktionieren unabhängig voneinander.

3.2.1 Backend-Modul

Das Backend-Modul ist eine zusätzliche Seite im WordPress Backend, die es dem Benutzer ermöglicht, Shortcodes für das Frontend zu erstellen. Das Backend-Modul hat eine Model-View-Controller Architektur. Die Zusammenarbeit der Module wird in der Abbildung 6 dargestellt.

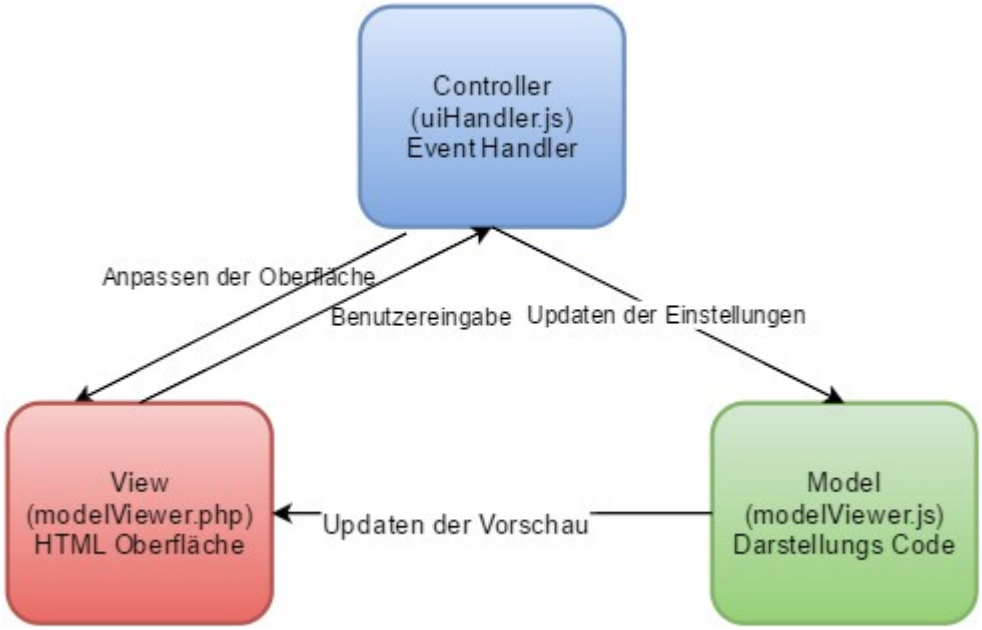

Abbildung 6: MVC Übersicht

Das Modell ist dabei die Klasse Model Viewer. Sie enthält die Daten, den Darstellungs-Code für die Vorschau, sowie die Logik um die Darstellung anzupassen.
Die View oder auch Präsentation ist die HTML-Seite im Backend von WordPress, die eigentliche Benutzeroberfläche. Diese enthält die Steuerungselemente für die Anpassungsmöglichkeiten, sowie das Fenster in dem die Vorschau dargestellt wird.
Der Controller ist eine Sammlung von JavaScript Event-Handlern, die die Steuerungselemente überwachen und Änderungen an die Model Viewer Klasse weitergeben. Des weiteren enthält der Controller die Logik zur Erstellung der Shortcodes.

3.2.2 Frontend-Modul

Das Frontend-Modul besteht aus zwei Funktionen. Die erste Funktion ist die Definition des Shortcodes in der Haupt-Datei des Plug-ins. In dieser Funktion ist der Code festgelegt, der Anstelle des Shortcodes in die Websesite eingesetzt wird. In der Regel handelt es sich dabei um HTML-Code.
Die zweite Funktion ist der Scene Reader. Diese JavaScript-Funktion durchsucht die Webseite nachdem sie geladen wurde auf HTML-Code, der durch Shortcodes in die Seite eingesetzt worden ist. Der Scene Reader liest diesen Code aus und erstellt an der entsprechenden Stelle eine Darstellung anhand des Shortcodes.

4 Implementierung

In diesem Kapitel wird gezeigt, wie das in Kapitel drei vorgestellte Plug-in implementiert wird. Dies ist nicht der Ort für eine detaillierte Erklärung des gesamten Quellcodes, diese findet sich im Quellcode selbst. Im Folgenden werden die wichtigsten Funktionalitäten des Quellcodes erklärt, sodass das Zusammenspiel der verschiedenen Teile des Programms verstanden werden kann. Dazu werden immer wieder Beispiele aus dem Quellcode gezeigt, welche zum besseren Verständnis vereinfacht sein können. Weiterhin werden Kenntnisse in JavaScript, jQuery und PHP benötigt, um die Beispiele zu verstehen. Zuletzt werden in diesem Kapitel auch einige interessante Funktionen genauer erklärt. Das Kapitel beginnt mit der Erstellung des eigentlichen WordPress Plug-ins. Anschließend wird das Backend-Modul, gefolgt von dem Frontend-Modul vorgestellt.

4.1 Erstellung des WordPress Plug-ins

Ein Plug-in in WordPress besteht aus mindestens einer PHP Datei, die im Plug-in Ordner der WordPress-Installation liegt. Der Name des Plug-ins bzw. der Datei muss eindeutig sein. Soll das Plug-in aus mehreren Dateien bestehen, können diese in Verzeichnissen angeordnet sein.
Im Folgenden wird zunächst die Struktur des Plug-ins beschrieben, gefolgt von der Registrierung der Module und dem Einbinden der Bibliotheken.

4.1.1 Dateistruktur des Plug-ins

Die Dateistruktur des Plug-ins zeigt die Abbildung 7. Die Datei *cad-model-viewer.php* ist die Hauptdatei des Plug-ins. Alle Funktionalitäten werden hier definiert. Weitere PHP-Dateien befinden sich im Unterordner */php*. Das Verzeichnis */js* beinhaltet die JavaScript-Dateien des Plug-ins, sowie die Bibliotheken three.js und stats.js.
Im Verzeichnis */files* befinden sich Beispieldateien und sonstige benötigte Dateien. Das Unterverzeichnis */scenes* ist der Speicherort für Dateien des Model Viewers.

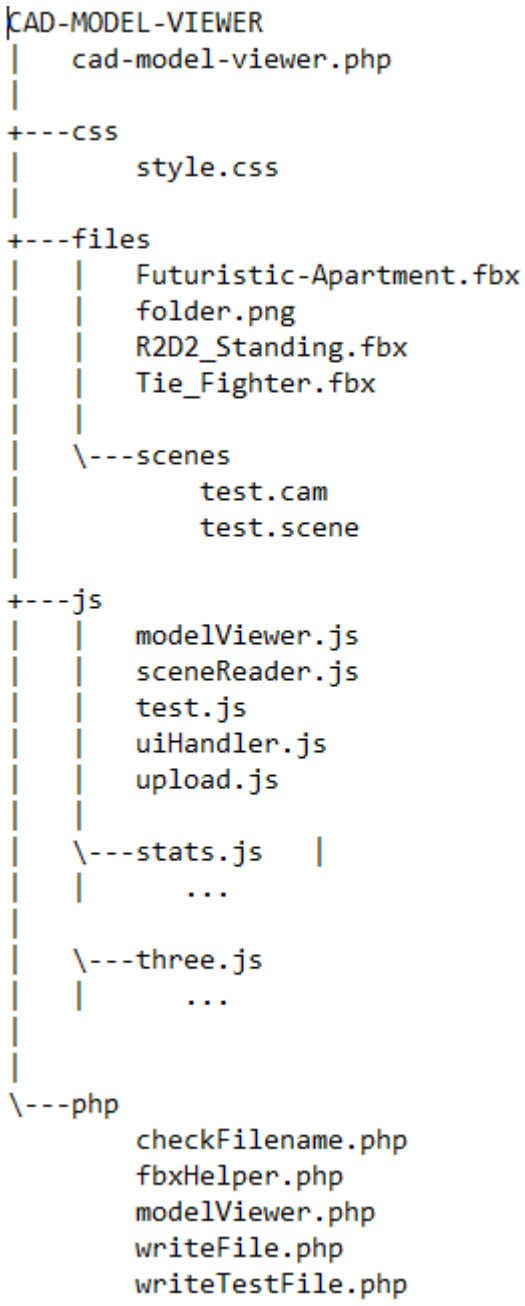

Abbildung 7: Ordnerstruktur

4.1.2 Registrierung der Module

Das Backend-Modul befindet sich auf einer Seite, die dem WordPress Backend hinzugefügt wird. Dazu wird ein Action Hook auf des Laden des Admin-Menüs registriert, welcher die WordPress-Funktion *add_object_page()* aufruft. Dadurch wird eine neue Seite im WordPress

Backend erstellt. Der Funktion wird der Name einer Funktion übergeben, die den Inhalt der Seite bestimmt.
Das Frontend Modul wird über einen Shortcode registriert. Dieser Vorgang wird im Abschnitt Implementierung des Frontend-Moduls genauer beschrieben.

4.1.3 Implementierung der Bibliotheken

Damit three.js eingesetzt werden kann, müssen die benötigten Bibliotheken in den HTML-Code eingebunden werden. WordPress bietet dazu die Funktion *wp_enqueue_script()* an, die im PHP-Code des Plug-ins ausgeführt werden kann. Die benötigten Bibliotheken werden in dem Ordner der Plug-ins abgelegt und können dann eingebunden werden. Der folgende Code ist ein Beispiel für die Benutzung der Funktion und bindet den Kern von three.js in den HTML-Code ein

```
wp_enqueue_script('threejs',plugin_dir_url(__FILE__)  ."/js/Thr
eeJs/build/three.min.js")
```

Als ersten Parameter erhält die Funktion einen String, der dem einzubindenden Script einen eindeutigen Namen gibt. Der zweite Parameter ist ebenfalls ein String, der den Pfad zu der Datei enthält. *plugin_dir_url(__FILE__)* ist dabei ein weiterer Aufruf einer Wordpress-Funktion, die den Pfad zu dem Plug-in liefert. WordPress bringt außerdem einige Scripts mit, die ohne Angabe eines Dateipfades eingebunden werden können. Dazu zählt unter anderem jQuery.[@wpe]

4.2 Implementierung des Backend-Moduls

Dieser Abschnitt beschreibt die Implementierung des Backend-Moduls. Dazu wird zunächst die Erstellung der Benutzeroberfläche und des Dialoges zum Hochladen von Dateien erklärt. Anschließend wird die Implementation des Model Viewers gezeigt. Der Model Viewer ist für

die Darstellung der Vorschau für das Frontend zuständig. Zuletzt wird das Speichern dieser Vorschau, sowie die Generierung des Shortcodes gezeigt.

4.2.1 Erstellung der Benutzeroberfläche

Die Benutzeroberfläche(View) ist eine Ansammlung von HTML-Formular-Elementen. Der Benutzer kann durch diese Elemente die Szene beeinflussen und abspeichern. Das Formular wird durch jQuery Event-Handler(Controller) überwacht. Im Folgenden wird beispielhaft das Zusammenspiel von View und Contoller gezeigt.

```
<input type="number" required name="width" class="parameter"
 id="c_width"  value="400"/>
```

Dies ist die Definition eines Eingabefeldes zur Steuerung der Breite das Model-Viewers.
type=”number” beschränkt die Eingabe auf ganze Zahlen.
value=”400” setzt einen Standard-Wert.
class=”parameter” wird vom Controller genutzt um auf das Feld zuzugreifen.
name=”width” gibt dem Feld einen Namen. Der Name wird vom Controller als Variablenname für den Wert das Feldes benutzt.
Der folgende Code ist ein Ausschnitt aus der Funktion zur Überwachung aller Eingabefelder, die die Klasse *parameter* haben.

```
jQuery('.parameter').change(function(){
     ...
     modelViewer.init[jQuery(this).attr("name")]=
     jQuery(this).val();
     ...
});
```

Es wird der Event-Handler Change benutzt, um auf Änderungen am Wert des Feldes zu reagieren. Bei *modelViewer.init[]* handelt es sich um ein JavaScript Objekt, das wie ein assoziatives Array beschrieben werden kann. Der Name des Eingabefeldes wird dabei als

Schlüssel für das Datenfeld benutzt und der Wert des Feldes wird dadurch dem Namen zugeordnet.
Da nicht alle Eingabefelder gleich verwaltet werden können, gibt es für einige Felder individuelle Event-Handler. Diese Event-Handler werden anhand der ID des Eingabefeldes registriert.

4.2.2 Erstellung des Dialogs zum Verwalten von Dateien

Der Dialog zum Verwalten der Dateien ist mit Hilfe der WordPress Medien Bibliothek erstellt. Er ermöglicht das Hochladen von Dateien, das Entfernen von Dateien, sowie das Auswählen von Dateien für die Benutzung im Model Viewer. Diese Funktionalitäten werden von WordPress bereitgestellt. Im Folgenden ist erläutert, wie auf diese Funktionalitäten zugegriffen werden kann.
Damit die JavaScript-Funktionen benutzt werden können, muss die Funktion *wp_enqueue_media()* aufgerufen werden. Der Aufruf dieser Funktion bindet die benötigten JavaScript-Dateien in den HTML-Code ein.
Im JavaScript kann durch Aufruf der Funktion *wp.media()* auf den Dialog zur Medienverwaltung von WordPress zugegriffen werden. Der Funktion können Parameter übergeben werden, um den Dialog anzupassen.[@wpm] Der gesamte Funktionsaufruf sieht wie folgt aus.

```
var media_uploader = wp.media({
     title: "Select or upload a file",
   button: {
   text: 'Load this file'
   },
   multiple: false
});
```

Es werden der Titel und der Text des Buttons angepasst. Zudem wird durch *multiple: false* verhindert, dass mehrere Dateien ausgewählt werden können.

```
media_uploader.open()
```

Dieser Funktionsaufruf öffnet den vorher angelegten Dialog, sodass er dem Benutzer angezeigt wird.
Um auf das Auswählen einer Datei reagieren zu können wird der Event Handler *on.(select)* registriert. Dieser wird aktiviert sobald der Button zum Laden einer Datei betätigt wird.
Der folgende Funktionsaufruf ermöglicht den Zugriff auf die Eigenschaften der ausgewählten Datei. Die Eigenschaften enthalten Informationen wie den Namen, die Größe und die URL der Datei.

```
media_uploader.state() .get('selection').first().toJSON();
```

Die URL wird im weiteren Verlauf der Programms benötigt, damit die ausgewählte Datei vom Browser geladen werden kann. Damit sichergestellt ist, dass es sich um eine FBX Datei handelt, wird über eine Regular-Expression geprüft, ob die URL die Endung *.fbx* hat.
Ist die URL gültig, werden die Eigenschaften der Datei in ein unsichtbares Input-Feld im HTML geschrieben und das Change Event des Feldes wird ausgelöst. Dieses Verfahren ermöglicht dem Controller, auf die Auswahl einer neuen Datei zu reagieren, indem es eine Änderung des Feldes registriert. JavaScript bietet keine Reaktion auf das Ändern einer Variable an, sodass der Umweg über den Wert eines unsichtbaren Feldes benötigt wird.
Zu den genannten Funktionen unterstützt die Media Library die Suche anhand von Dateinamen, sowie das Sortieren nach Datum des Hochladens. Das Sortieren der Dateien in Ordnern wird nicht unterstützt.
Beim Hochladen wird der Datei-Typ nicht überprüft. Dies hätte auch wenig Auswirkung, da die Media Library auch direkt im WordPress Backend, sowie in anderen Plug-ins benutzt werden kann und dort keine Beschränkungen beim Upload hat. Dies führt dazu, dass in der Liste der Dateien auch andere Dateien, wie zum Beispiel Bilder, auftauchen. Des Weiteren wird die Gültigkeit der FBX-Dateien nicht überprüft.

4.2.3 Erstellung des Model Viewers

Der Model Viewer ist eine JavaScript-Klasse zur Darstellung von FBX-Modellen in Web-Browsern. Sie verwaltet die dargestellte Szene sowie das Einlesen der Modell-Dateien.
Um mit three.js eine Darstellung zu erhalten wird ein Renderer benötigt. Der folgende Code erstellt einen Renderer und tätigt Einstellungen zur Größe, Hintergrundfarbe und Schatten.

```
renderer = new THREE.WebGLRenderer();
renderer.setSize(this.CANVAS_WIDTH, this.CANVAS_HEIGHT);
renderer.setClearColor(this.init["bg_color"], 1);
this.renderer.shadowMap.enabled = true;
this.renderer.shadowMap.type = THREE.PCFSoftShadowMap;
```

Zusätzlich muss der Renderer in den HTML-Code eingebettet werden. Dies geschieht mit folgendem Funktionsaufruf:

```
container.append(renderer.domElement);
```

Es wird eine Szene erstellt, damit im Renderer Objekte dargestellt werden können. Diese Szene wird im Folgenden alle Objekte und Lichter verwalten. Der folgende Code zeigt beispielhaft, wie ein Ambient-Light erstellt und der Szene hinzugefügt wird. Dem Licht wird ein Name gegeben um es in der Szene referenzierbar zu machen:

```
var ambiLight = new THREE.AmbientLight(color,intensity);
ambiLight.name = "AmbientLight";
scene.add(ambiLight);
```

Zur Darstellung der Szene benötigt der Renderer zusätzlich eine Kamera, durch die die Szene betrachtet wird:

```
camera = new THREE.PerspectiveCamera(50, this.CANVAS_WIDTH / this.CANVAS_HEIGHT, 1, 10000);
```

Dieser Funktionsaufruf erstellt eine perspektivische Kamera. Der erste Parameter ist das vertikale Field-of-View in Grad. Als nächsten Parameter erhält die Kamera das Seitenverhältnis des Canvas-Elements des Renderers. Die letzten beiden Parameter sind die minimale und die maximale Entfernung der dargestellten Objekte.
Sind Renderer, Szene und Kamera erstellt kann mit folgendem Funktionsaufruf die Darstellung erfolgen:

```
renderer.render(scene, camera);
```

Um Veränderungen in der Szene sichtbar zu machen muss diese Funktion für jeden darzustellenden Frame aufgerufen werden. Die Funktion *requestAnimationFrame()* informiert den Browser, dass eine Animation ausgeführt werden soll. Sie übergibt dem Browser als Parameter einen Funktionsnamen. Diese Funktion wird vom Browser ausgeführt bevor er das nächste mal seine Darstellung aktualisiert. Im Folgenden wird eine Funktion gezeigt, die die Szene regelmäßig neu rendert:

```
animate() {
renderer.render(scene, camera);
     requestAnimationFrame(animate);
}
```

Die Funktion übergibt ihren eigenen Namen an die *requestAnimationFrame()* Funktion und sorgt somit dafür, dass sie selbst wieder aufgerufen wird.
Der Renderer stellt jetzt eine leere Szene dar. Im Folgenden wird gezeigt wie ein Objekt aus einer FBX-Datei eingelesen und der Szene hinzugefügt wird. In three.js wird das Einlesen von diversen Dateitypen unterstützt, dazu zählen auch FBX-Dateien.

```
var loader = new THREE.FBXLoader();
loader.load(this.filepath, function(object){
     var ob = new THREE.Object3D();
     object.traverse(function(child){
     if(child instanceof THREE.Mesh){
        if(this.init["material"] == "lambert"){
          var m = new THREE.Mesh(child.geometry, new
                        THREE.MeshLambertMaterial({
            color: this.init["material_color"]}));
         }
               else if(this.init["material"] == "phong"){
          var m = new THREE.Mesh(child.geometry, new
                        THREE.MeshPhongMaterial({
            color: this.init["material_color"]}));
         }
         m.castShadow = true;
         m.receiveShadow = true;
         ob.add(m);
       }
     }.bind(this));
     this.scene.add(ob);
}
```

Zunächst wird ein *THREE.FBXLoader* erstellt. Dieser stellt eine *load()* Funktion zur Verfügung. Als Parameter erhält diese Funktion den Pfad zu der zu lesenden Datei und eine Callback-Funktion, die nach erfolgreichem Einlesen mit dem eingelesenen Objekt als Parameter ausgeführt wird. Der Ablauf der *load()* Funktion ist dabei asynchron. Da die Modelle in der Regel aus mehreren geometrischen Objekten zusammengesetzt sind, ist das eingelesene Objekt eine Gruppe eben dieser Objekte. Die einzelnen Objekte sind Polygonnetze vom Typ *THREE.Mesh* und als Kinder der Gruppe zugänglich. Der FBX-Loader unterstützt derzeit noch keine Texturen und Materialien, wodurch eingelesene FBX-Modelle einfarbig blau dargestellt werden und keine Interaktion mit dem Licht stattfindet. Um die Modelle ansehnlicher zu machen wird eine Kopie erstellt, die nach Wahl ein Material mit

einer lambertschen oder phong Reflexion erhält. Zudem wird die Farbe des Modells bestimmt und das Werfen und Empfangen von Schatten wird aktiviert. Nach dem Erstellen der Kopie wird diese in die Szene eingefügt.
Um das Modell in der Szene besser zu präsentieren wird die Kamera eingestellt, ein Licht erstellt und eine Plattform erstellt, auf der das Modell steht, wofür die ungefähren Maße des Modells benötigt werden. Die Bounding Box ist ein Quader, der das Objekt einschließt. In three.js gibt es die Möglichkeit über einen *THREE.BoundingBoxHelper* eine solche Box zu erstellen, die an den Achsen des Welt-Koordinatensystems ausgerichtet ist. Diese Bounding Box wird im Weiteren für die oben genannten Aufgaben benutzt.

Kamera

Für das einstellen der Kamera wird die Höhe und die Breite in X-Richtung des Modells benötigt. Diese können mit Hilfe der Bounding Box wie folgt berechnet werden.

```
var height = Math.abs(bbox.box.min.y - bbox.box.max.y);
var width = Math.abs(bbox.box.min.x – bbox.box.max.x);
```

Die Kamera wird auf der Hälfte der Höhe platziert. Des Weiteren wird die Distanz der Kamera von dem Modell berechnet. Die folgende Skizze soll den Zusammenhang der Distanz der Kamera, dem Field of View und der Höhe des Modells verdeutlichen.

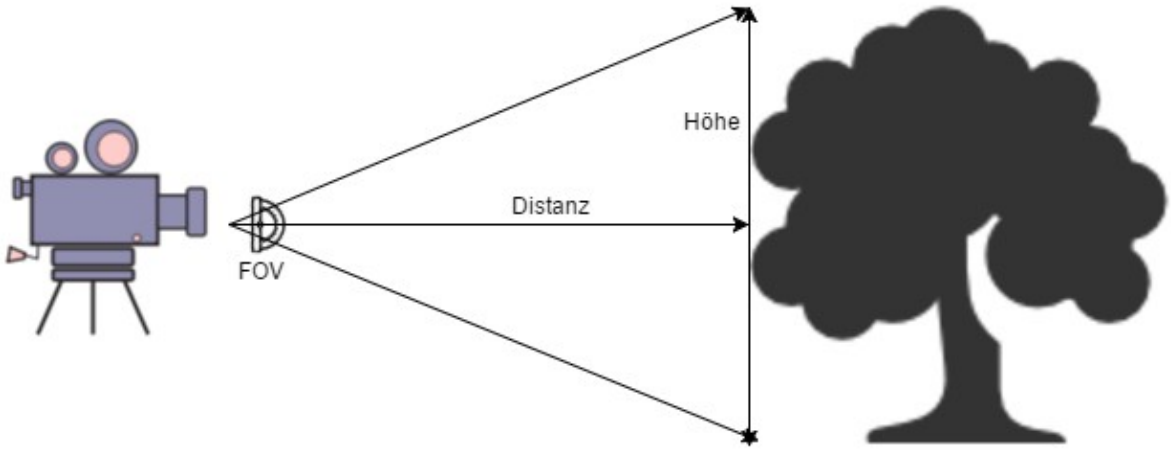

Abbildung 8: Berechnung der Kameradistanz

Die Höhe des Modells und das Field of View der Kamera sind bekannt. Die benötigte Distanz zum Modell ergibt sich durch folgende Formel.

$$Distanz = \frac{Höhe/2}{\tan(fov/2)}$$

Da sich die Kamera auf der halben Höhe des Modells befindet, werden sowohl der Winkel des Field of View als auch die Höhe halbiert. Durch den Tangens des Field of View- Winkels lässt sich die Distanz bestimmen. In der Programmierung wird die Größe des Modells etwas erhöht, damit das Modell nicht den kompletten Bereich ausfüllt. Des Weiteren wird die Distanz nach Höhe oder Breite berechnet, sodass Modelle die breiter als hoch sind möglichst viel Platz in der Breite ausfüllen.

Beleuchtung

Damit Modelle verschiedener Größen ähnlich beleuchtet werden, wird das Licht an Hand der Bounding Box positioniert. Der Szene werden zwei Lichter hinzugefügt: Ein Point-Light und ein Directional-Light. Die folgende Abbilung zeigt die Bounding Box und ihren Schatten aus der Draufsicht, sowie das Point-Light als Sonne und das Directional-Light als Taschenlampe. Das Point-Light ist die Haupt-Lichtquelle und für das Werfen der Schatten verantwortlich. Es wird in Richtung der X und Z-Achse um die jeweilige Breite der Bounding Box verschoben. Zudem wird es auf der 1,5 fachen Höhe der Bounding Box positioniert. Point-Lights sind Lichtquellen die aus einem Punkt hervorgehen und in alle Richtungen gleichermaßen Strahlen, dadurch wirft es perspektivische Schatten. Das Directional-Light ist für das ausleuchten der Schatten zuständig. Es wird auf der gegenüber liegenden Seite positioniert und ist deutlich schwächer als das Point-Light. Das Directionl-Light wirft keine Schatten. Directional-Lights sind Lichter die aus einer bestimmten Richtung kommen und nicht aus einem Punkt. Die Lichtstrahlen dieses Lichtes sind alle parallel. Diese Eigenschaft ist nützlich für gleichmäßiges Ausleuchten.

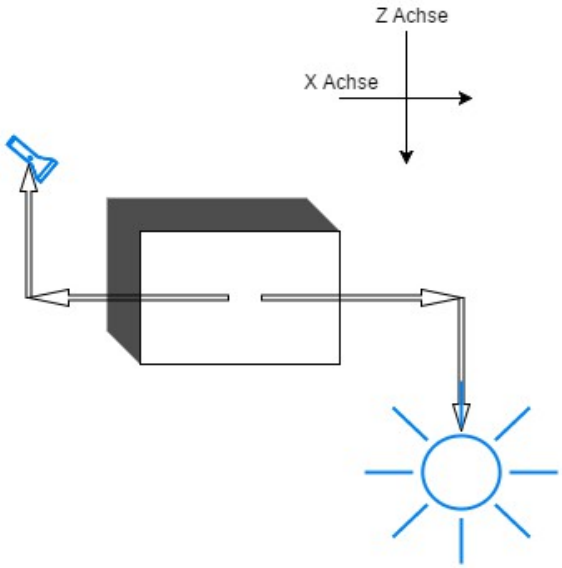

Abbildung 9: Beleuchtung der Szene

Die Plattform ist eine Scheibe auf der die Modelle stehend präsentiert werden können. Des Weiteren wirkt die Szene räumlicher durch den Schattenwurf des Modells auf der Plattform und einer deutlicheren Abgrenzung vom Hintergrund. Abbildungen 10 und 11 sollen diesen Unterschied verdeutlichen, indem die gleiche Szene mit und ohne Plattform gezeigt wird.
Die Plattform erhält den doppelten Radius der Bounding Box des Modells und wird unterhalb der Boundig Box platziert.

Abbildung 10: Szene mit Platform

Abbildung 11: Szene ohne Platform

Animationen

Das Plug-in unterstützt zwei Animationen der Modelle: Einerseits die Rotation des Modells um eine oder mehrere Achsen und andererseits die Rotation der Kamera um das Modell.
Die Rotation des Modells geschieht über den Zugriff auf die Rotation des Modells. Das *rotation* Objekt ist ein Eulerscher Winkel, der die Lage das Modells in der Szene beschreibt. Auf die X,Y und Z Anteile kann einzeln zugegriffen werden, was eine Drehung um einzelne Achsen des Modells ermöglicht.
Die Rotation der Kamera um das Modell benutzt die *THREE.OrbitControls*. Damit eine automatische Rotation erfolgt muss das Attribut *autoRotate=true* gesetzt werden und die Geschwindigkeit der Rotation muss über die Variable *autoRotateSpeed* gesetzt werden. Sind diese Einstellungen getätigt, rotiert die Kamera um die Y-Achse des Punktes, auf den die Kamera gerichtet ist. Die Geschwindigkeit wird wie bei der Rotation des Modells über einen Regler im HTML-Code bestimmt.

4.2.4 Speichern der Szene

Damit eine im ModelViewer erstelle Szene in die Web-Seite eingebunden werden kann, muss der Benutzer sie abspeichern. Dazu gibt er der Szene einen Namen, unter dem sie gespeichert werden soll.
Three.js ermöglicht es Objekte wie Szenen, Kameras und 3D-Objekte in JSON umzuwandeln. Dies soll genutzt werden, um die Szene aus dem Model-Viewer in einer Datei auf dem Server abzuspeichern. Das Frontend-Modul liest diese Datei wiederum ein und benutzt so die vorher abgespeicherte Szene.
Im Folgenden wird die Funktion zum Speichern der Dateien vorgestellt:

```
var temp = mv.scene.clone();
temp.remove(temp.getObjectByName("obj"));
```

Zunächst wird eine Kopie der eigentlichen Szene erstellt und aus dieser Kopie wird das aus der FBX-Datei eingelesene Objekt entfernt. Dies ist nötig, da die eingelesenen Objekte sehr groß sein können und der Server somit sehr lange Anfragen bekommen würde. Für ein WordPress Plug-in wäre das eine schlechte Eigenschaft, da viele Besitzer einer WordPress-Seite keine Möglichkeiten haben, den Server für den Umgang mit großen Anfragen zu konfigurieren. Des Weiteren wird Speicherplatz auf dem Server gespart, da Modelle bereits in Form von FBX-Dateien auf dem Server vorhanden sind. Die Szene enthält nun lediglich die Lichter, den Boden und die Bounding-Box des Modells. Durch den Aufruf der Funktion *.toJSON()* werden Szene und Kamera in das benötigte Format umgewandelt und anschließend mit Ajax an eine PHP-Datei auf dem Server geschickt.

```
jQuery.ajax({
     type: 'POST',
   url: ajaxUrl.url+"php/writeFile.php",
   data: {
     "filename": jQuery("#filename").val(),
      "scene": ,JSON.stringify(temp.toJSON()),
      "cam": JSON.stringify(mv.camera.toJSON()),
     }
});
```

Auf dem Server werden zwei Dateien erstellt, eine *filename.scene*, in der die Szene gespeichert ist und eine *filename.cam*, in der die Kamera gespeichert ist.
Im nächsten Schritt wird der so genannte Shortcode erstellt, mit dem die benötigten Werte des Model Viewer in eine Vorlage eingesetzt werden können.

4.3 Implementierung des Frontend-Moduls

Das Frontend Modul basiert auf den bereits häufig erwähnten Shortcodes. Dieser Abschnitt beginnt mit einer ausführlichen Erklärung der Implementation und Funktion des Shortcodes. Anschließend wird erklärt wie aus den Shortcodes wiederum eine dreidimensionale Darstellung wird.

4.3.1 Shortcodes

Zunächst muss der Shortcode für WordPress registriert werden. Dies geschieht in der *cad-model-viewer.php* Datei durch folgenden Funktionsaufruf.

```
add_shortcode( 'cad_modelviewer','modeviewer_shorcode_func' )
```

Der erste Parameter der Funktion ist der Name des Shortcodes. *[cad_modelviewer]* ist hiermit ein gültiger Shortcode.

Der zweite Parameter ist der Name der Funktion, die die Funktionalität des Shortcodes beinhaltet. Die Funktion wird immer dann ausgeführt wenn eine Seite dargestellt werden soll, in der der zuvor definiertet Shortcode benutzt wird. Der Funktion kann ein Array übergeben werden, welches Parameter enthält. Ein Shortcode mit Parameter kann wie folgt aussehen:

```
[cad_modelviewer width=”400” height=”400”]
```

Der Programmierer kann diese Parameter auslesen und dadurch den Inhalt anpassen.

In der Shortcode-Funktion werden zunächst die benötigten Bibliotheken aus three.js, sowie die JavaScript-Datei *sceneReader.js* eingebunden. Die Datei *sceneReader.js* enthält den Code, der zu den jeweiligen Shortcodes die entsprechende Szene darstellt.

Des Weiteren erstellt die Funktion das HTML-Gerüst, in dem die Darstellung erfolgt. Das Gerüst besteht aus einem einfachen Div-Container und einem Script-Tag, das zur Übergabe der Parameter des Shortcodes genutzt wird.

Die Folgende Zeile Code wandelt das Array mit den Parametern in ein JSON-Objekt um, dass in JavaScript benutzt werden kann.

```
$output  .=json_encode($atts);
```

Der Rückgabewert der Funktion enthält den HTML-Code der an die stelle des Shortcodes eingesetzt wird.

Im Folgenden wird beispielhaft ein Shortcode und das Ergebnis im HTML-Code gezeigt:

```
[cad_modelviewer  file="test"
fbx_file="/path/to/R2D2_Standing.fbx"
```

```
width="400" height="400" bg_color="#ffffff" material="phong"
cam_rotation_speed="0" rot_speed_x="0" rot_speed_y="0"
rot_speed_z="0"]

<div class="model-viewer-canvas">
     <script type="json">
          {"file":"test",

     "fbx_file":"\/path\/to\/R2D2_Standing.fbx",
          "width":"400",      "height":"400",
"bg_color":"#ffffff",              "material":"phong",
     "cam_rotation_speed":"0",
     "rot_speed_x":"0", "rot_speed_y":"0", "rot_speed_z":"0"}
     </script>
</div>
```

Das Beispiel zeigt einen Shortcode mit einigen Paramentern. Die folgende Tabelle zeigt eine Übersicht über alle Parameter und deren Bedeutungen.

Parameter Name	Bedeutung
file	Name der Datei, in der Szene und Kamera gespeichert sind.
fbx_file	Pfad zu der auszulesenden FBX-Datei
width	Breite der Darstellung
height	Höhe der Darstellung
bg_color	Hex-Wert der Hintergrundfarbe der Darstellung
cam_rotation_speed	Rotationsgeschwindigkeit der Kamera
rot_speed_x	Rotationsgeschwindigkeit des Modells um die X Achse
rot_speed_y	Rotationsgeschwindigkeit des Modells um die Y Achse
rot_speed_z	Rotationsgeschwindigkeit des Modells um die Z Achse
material	Gibt an, ob ein Lambertsches Material oder ein Phong Material erstellt werden soll
material_color	Hex-Wert der Farbe des Modells

Tabelle 3: Bedeutung der Shortcode Parameter

4.3.2 SceneReader

Der SceneReader durchsucht das HTML-Dokument nach den durch den Shortcode generierten Containern. Über den Aufruf der folgenden Funktion werden die zuvor in JSON umgewandelten Parameter des Shortcodes in ein JavaScript-Objekt umgewandelt:

```
var sc = jQuery.parseJSON(c.children("script").text())
```

Mit Hilfe der Parameter werden die Einstellungen des ModelViewers wieder hergestellt. Anschließend werden die gespeicherten Szene- und Kamera-Dateien eingelesen. Zum Laden und Interpretieren der Dateien wird der *THREE.ObjectLoader* benutzt. Die Pfade der Dateien stammen aus den Parametern des Shortcodes. Die Darstellung und Animation der Modelle erfolgt analog zum Model Viewer.

5 Testen und Bewerten der Leistung des Plug-ins

Die Leistung des Plug-ins soll an Hand von verschiedenen Merkmalen bewertet werden. Ein wichtiges Merkmal für die Leistung des Plug-ins ist, dass die Webseite reaktionsfähig bleibt und somit weiterhin flüssig bedienbar ist. Ein weiteres Merkmal ist die Ladezeit der Seite. Im Folgenden werden diese Merkmale und ihre Tests beschrieben und bewertet. Getestet wird lediglich die Implementierung für das Frontend, da diese kritisch für die Benutzbarkeit der Webseite ist. Das Backend-Modul kann aus Sicht der Leistung ähnlich wie eine Webseite, in die ein Shortcode eingebunden ist, betrachtet werden.

5.1 Testen der Ladezeit und Größe der Webseite

Die Ladezeit der Webseite ist davon abhängig, wie viele Ressourcen vom Server heruntergeladen werden müssen und wie schnell die Anbindung an den Server ist. Werden die Ressourcen nicht schnell genug vom Server geladen, dauert es länger, bis sich die Webseite nach und nach aufbaut. Dieses Verhalten soll vermieden werden. Da die Anbindung zum Server nicht beeinflusst werden kann, geht es im Folgenden hauptsächlich um die Analyse der zu ladenden Ressourcen.

5.1.1 Durchführung des Tests

Die Tests werden mit der Netzwerküberwachung von Google Chrome durchgeführt. Diese gibt detaillierte Informationen über die geladenen Ressourcen einer Webseite. Um die Tests einfacher zu gestalten, werden die gleichen Modelle mehrmals eingebunden. Damit das Caching des Browsers das Ergebnis nicht verfälscht, wird es deaktiviert. Gleiche Modelle werden dadurch mehrmals heruntergeladen, was dem Laden von mehreren ähnlichen Modellen im Bezug auf die Ladezeit und auf die zu ladende Menge entspricht.
Tests auf mobilen Geräten sind nicht nötig, da der Unterschied zu einem PC lediglich in der Anbindung zum Server liegt. Die zu ladenden Ressourcen im Bezug auf das erstellte Plug-in

sind identisch. Die Ressourcen von WordPress für mobile Geräte können sich geringfügig unterscheiden. Google Chrome bietet die Möglichkeit den Netzwerkverkehr zu drosseln um mobile Übertragungsgeschwindigkeiten zu simulieren.

5.1.2 Bewertung der Ergebnisse

Bereits kleine FBX-Dateien, wie die Datei *Futuristic-Apartment.fbx* haben bereits eine Größe von 3 MB. Da diese Dateien vom Browser heruntergeladen werden müssen ist die Größe der Datei maßgeblich für die gesamte Größe der Webseite.
Auf der Abbildung 17 im Anhang wird eine aufwändigere Datei heruntergeladen, wodurch die gesamte Größe der Webseite 7,1MB beträgt. Dies entspricht einem Aufruf von Youtube und dem Anschauen eines dreieinhalb minütigen Musikvideos bei einer Auflösung von 360p.
Selbst beim heutigem Standard bei der Internet-Geschwindigkeit kommt es somit zu kurzen Verzögerungen beim Laden der Webseite.
Auf mobilen Geräten hingegen sind derartige Größen sehr problematisch. Befindet sich das Gerät im Mobilfunknetz, ist die Anbindung an das Internet deutlich langsamer und zudem wird häufig nach Datenvolumen bezahlt. Bei einer 2G-Verbindung führt dies beispielsweise zu einer Ladezeit zwischen drei und vier Minuten. Damit für den Besucher der Webseite keine hohen Kosten aufkommen, sollten die Modelle erst auf Wunsch des Besuchers geladen werden.

5.2 Testen der Reaktionsfähigkeit der Webseite

Die Reaktionsfähigkeit der Webseite lässt sich an den Frames per Second (FPS) messen. Diese beschreiben, wie oft die Darstellung der Webseite pro Sekunde erneuert wird und somit auch wie häufig die Webseite auf Benutzereingaben reagieren kann. Der gewollte Zustand ist eine Bildwiederholungsrate von 60 FPS. Während der Entwicklung des Plug-ins hat sich gezeigt, dass dieser Zustand stabil ist sobald er einmal erreicht wird. Diese Aussage impliziert, dass die Webseite nicht ständig stabil läuft. Eine Tatsache, die hauptsächlich dem

Laden und Parsen der FBX-Datein geschuldet ist. Der Folgende Test will diesen Zeitpunkt in Abhängigkeit der Anzahl von Modellen feststellen.

5.2.1 Erstellung des FPS Tests

Um die Leistungs-Tests durchzuführen wird der Shortcode *[cad_modelviewer_test]* in WordPress registriert. Dieser Shortcode hat die gleiche Funktionlität wie der Shortcode zur Benutzung im Frontend. Zusätzlich wird die JavaScript Datei test.js zum Durchführen und Kontrollieren der Tests eingebunden.
Google Chrome bietet Entwicklern eine Anzeige der Framerate der Webseite, welche sich sehr gut eignet, um einen Überblick über die Leistung der Seite zu erhalten. Im Folgenden wird zur Kontrolle der FPS die Bibliothek stats.js benutzt. Der Vorteil von stats.js ist, dass die Überwachung durch ein Programm gesteuert werden kann. Dies verbessert die Möglichkeiten zur Definition von Tests und verhindert Fehler beim Ablesen.
Stats.js ist eine JavaScript Bibliothek von dem Hersteller von three.js zur Überwachung der Leistung des Codes. Nach Implementation in den JavaScript-Code wird auf der Webseite eine Box dargestellt, die Informationen zur Leistung zeigt. Standardmäßig werden das Anzeigen von Frames per Second, Renderzeit pro Frame und Speicherverbrauch unterstützt. Zusätzlich kann die Box um eigene Anzeigen erweitert werden. Weiterhin wurde der Code von stats.js erweitert, sodass die aktuellen FPS abgefragt werden können.
Um stats.js zu benutzen, wird die JavaScript-Datei in die Seite eingebunden.
Mit folgende Funktionsaufrufen wird eine neues Objekt erstellt und die Box wird an den HTML-Code angefügt.

```
var stats = new Stats();
document.body.appendChild( stats.dom );
```

Der Code, der überwacht werden soll wird nun mit den Funktionen *stats.begin()* und *stats.end()* eingeschlossen.[@statsjs]

Der Test erstellt eine festgelegte Anzahl von Darstellungen auf einer Webseite und misst die

vergangene Zeit vom Erstellen bis zum erstmaligen Erreichen von 60 FPS. Werden 60 FPS nach 60 Sekunden nicht erreicht, wird der Test abgebrochen und die aktuellen FPS gespeichert. Zur Erstellung der Darstellungen werden festgelegte Shortcodes in die Webseite eingesetzt und anschließend mit dem Frontend-Modul (Scene Reader) dargestellt. Die Ergebnisse des Test werden auf dem Server gespeichert, was den Vorteil hat, dass die Tests auch auf Geräten ausgewertet werden können, die keine Entwickler-Konsole haben. Dies betrifft vor allem mobile Geräte. Die Daten werden im JSON-Format auf dem Server gespeichert und können mit der Hilfe der Webseite https://json-csv.com/ in das CSV-Format umgewandelt werden, wodurch sie in Excel und ähnliche Programme importiert werden können.

5.2.2 Durchführung des Tests

Zur Durchführung des Test wird der Shortcode in eine WordPress Seite eingebunden. In der Datei test.js wird durch einen im Model-Viewer erstellten Shortcode festgelegt, welche Modelle dargestellt werden sollen. Des Weiteren wird festgelegt, wie oft der Shortcode in die Seite eingebunden werden soll. Dem Testlauf kann weiterhin noch ein Name gegeben werden. Nach Festlegung dieser Einstellungen wird die Webseite auf der der Test eingebunden ist aufgerufen und durch Betätigung des Test Buttons der Test gestartet. Vor der Durchführung eines neuen Tests muss die Webseite neu geladen werden.

5.2.3 Bewertung der Ergebnisse

Wie Eingangs erwähnt entstehen die Einbrüche der Leistung aus dem Parsen der FBX-Dateien. Dieser Effekt wird durch die Abbildungen 12 und 13 sehr deutlich. Die Abbildungen wurden mit Hilfe der Google Chrome Timeline bei einem Test erstellt, bei dem fünf Modelle dargestellt werden. In der Abbildung 12 zeigt der orange Balken die CPU Auslastung und der grüne Balken die FPS zur gleichen Zeit an. An der 6500ms Marke sinkt die CPU Last und die FPS steigen. Anschließend bleiben die FPS stabil.

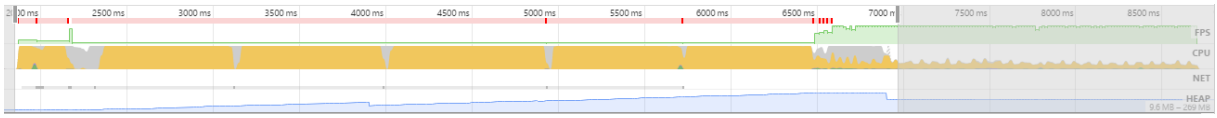

Abbildung 12: Zusammenhang CPU Last und FPS

Die Abbildung 13 schlüsselt die CPU Auslastung genauer auf. Es ist deutlich zu sehen, wie die Geometry-Parser beim Einlesen der fünf Modelle die CPU auslasten. Nach dem Einlesen des letzten Modells sinkt die Belastung der CPU sichtlich.

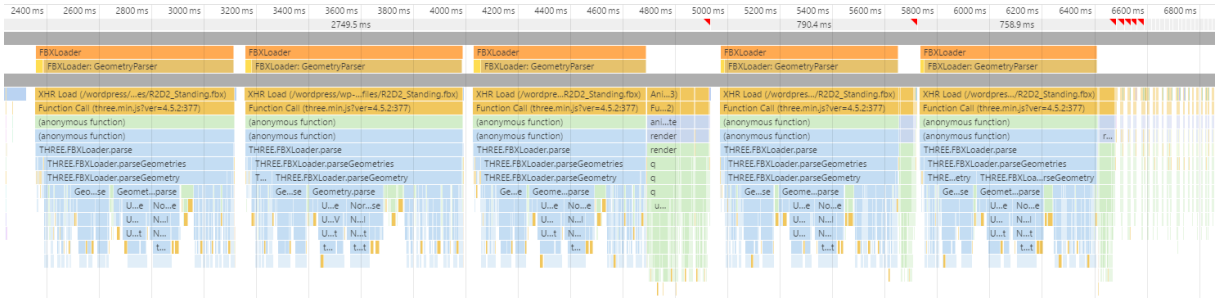

Abbildung 13: CPU Last detailliert

Zu erwarten ist ein linearer Anstieg der Dauer bis zum erreichen eines stabilen Zustandes. Das Diagramm in Abbildung 14 zeigt die durchschnittliche Zeit auf der Y-Achse in Abhängigkeit zu den darzustellenden Modellen. Die blauen Säulen sind dabei der Durchschnittswert, die roten Säulen sind der erwartete Durchschnittswert.

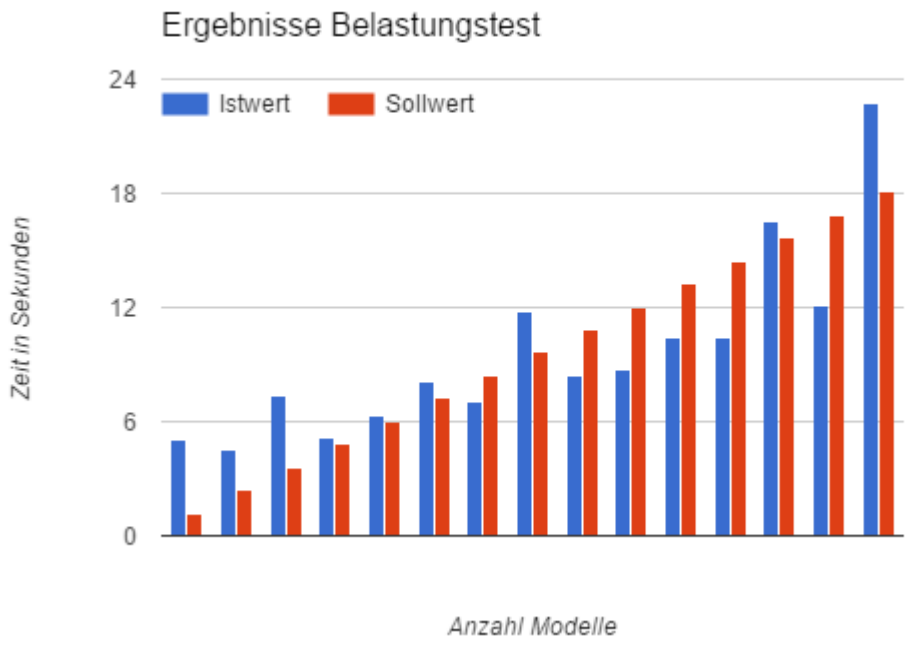

Abbildung 14: Ergebnisse Belastungstest

Die Abweichungen sind mit den unterschiedlichen Ausgangssituationen der Testläufe zu erklären. Obwohl die Testläufe der oben dargestellten Daten alle mit dem gleichen Gerät und der gleichen Internetanbindung ausgeführt worden sind, gibt es erhebliche Unterschiede beim Messwert. So haben Testläufe mit einem Modell Werte von 1-7 Sekunden und auch Testläufe mit sechs Modellen ergeben Werte von 6-11 Sekunden. Je mehr Modelle zu laden sind, desto geringer werden diese Schwankungen. Mit Hilfe der Timeline und Netzwerkanalyse in Google Chrome wurden die Testläufe überwacht, um den Grund für die Unterschiede festzustellen.

Einen deutlichen Unterschied mach das Caching des Browseres. Es sorgt dafür, dass die FBX-Dateien nicht mehr heruntergeladen werden müssen. Dies reduziert den Testwert, da die Zeit zum Herunterladen der Dateien wegfällt.

Unterschiede in der Geschwindigkeit der Internetanbindung führen dazu, dass der FBX-Loader erst verspätet einsetzt und somit die angegebene Zeit zu hoch ist. Dies ist als Fehler im Test anzusehen, wenn nur die Leistungsfähigkeit des JavaScript-Codes getestet werden soll.

Allerdings wurden auch bei Testläufen auf einem lokalen Server ähnliche Schwankungen festgestellt. Diese lassen sich auf die Speicherverwaltung von Google Chrome bzw dem

unregelmäßigen Einsetzen der Garbage Collection zurückführen. Dies führt dazu, dass ein vorangegangener Testlauf den Nachfolger belasten kann. Allerdings können diese Unterschiede auch durch anderweitige Belastung der Hardware, etwa durch Hintergrundprozesse des Betriebssystems verursacht werden.

Ein weiteres Problem, dass dieser Test hervorgebracht hat, ist, dass WebGL nur 16 eigenständige Darstellungen zulässt. Dies wird durch die folgende Fehlermeldung gekennzeichnet:

```
Error: WebGL: Exceeded 16 live WebGL contexts for this
principal, losing the least recently used one.
```

Wird diese Zahl überschritten, wird die Darstellung, die am längsten nicht mehr benutzt wurde, entfernt. Dieses Limit sollte beim Einbinden der Shortcodes in die Webseite beachtet werden.
Im Firefox tritt zudem ein Fehler auf, der dafür sorgt, dass auch nach Neustart des Browsers weiterhin diese Fehlermeldung ausgegeben wird und keine Darstellungen erstellt werden.

Im Browser Microsoft Edge sind die FPS Werte auf dem gleichen Testgerät deutlich niedriger. Firefox und Google Chrome erreichen auch mit 15 Darstellungen noch 60 FPS. Im Microsoft Edge werden bereits bei 5 Darstellungen keine 60 FPS mehr erreicht.

Noch niedriger ist die Leistung auf mobilen Geräten. Auf dem Nexus 4 werden bei einer Darstellung im Schnitt nur 27 FPS gemessen. Ab fünf Darstellungen liegen die FPS unter 10. Zieht man die Ergebnisse der Ladezeit-Messung mit in Betracht sollte auf das einbinden der Darstellungen auf mobilen Webseiten verzichtet werden.
Bei der Benutzung auf einem PC kommt es in Abhängigkeit von der Hardware und dem Browser zu unterschiedlich starken Leistungseinbrüchen beim Laden der Webseite. Diese Einbrüche verhindern kurzzeitig ein flüssiges Benutzen der Webseite. Mit Rücksicht auf Benutzer auf leistungsschwachen Geräten sollte daher auf das Einbinden vieler Darstellungen verzichtet werden.

6 Ergebnisse und Ausblick

6.1 Bewertung der Ergebnisse

Insgesamt hat das Projekt gezeigt, dass die Umsetzung eines WordPress Plug-ins zur Darstellung von CAD-Dateien mit three.js möglich ist. WordPress stellt Funktionen zur Verfügung, die problemlos zur Umsetzung benutzt werden können.
Auch three.js enthält die benötigte Funktionalität.
Während des Projekts kam es jedoch zu Problemen, die im Hinblick auf die Benutzbarkeit des Plug-ins betrachtet werden sollten.
Die Wahl, FBX-Dateien zu unterstützten, stellte sich im Verlauf des Projektes als problematisch heraus, da diese nicht im vollen Umfang von three.js unterstützt werden. Da three.js ein Beispiel zum Einlesen von FBX-Dateien beinhaltet, wurde dieser Entwicklungsschritt ohne nähere Prüfung als unkritisch eingestuft. Zu dem Zeitpunkt der Feststellung des Problems ist das Implementieren oder Erweitern des FBX-Parsers aus zeitlichen Gründen nicht mehr möglich gewesen. Die Tatsache, dass FBX ein proprietäres Dateiformat ist, ist vermutlich ein Grund dafür, dass es keine fertigen Lösungen gibt. Autodesk stellt ein FBX SDK zur Verfügung, dass das Lesen, Schreiben und Konvertieren von FBX-Dateien ermöglicht. Diese sind allerdings nur für C++ und Python verfügbar, [@fbxsdk] was ein sinnvolles Einbinden des SDK in dieses Projekt verhindert.
Das Ausweichen auf ein anderes Dateiformat ist möglich, wurde aber abgelehnt, da es wenig Mehrwert für das Projekt hätte. Eine Umstellung auf OBJ/MTL Dateien, würde das Darstellen von Modellen mit Materialien ermöglichen. Dies würde für eine ansehnlichere Darstellung sorgen, aber nichts an der Umsetzbarkeit und der Leistung des Plug-ins ändern.
Zuletzt unterstützt der FBX-Parser von three.js auch nur FBX-Dateien im ASCII Format, die deutlich größer sind als ihre Gegenstücke im Binär Format. Dies führt zu unnötig großen Dateien, die vom Browser heruntergeladen werden müssen.
Diese Problematik wurde in den Ladezeit- und Leistungstests verstärkt festgestellt. Ein einzelnes eingebundenes Modell ist im Bezug auf die Ladezeit mit einem Video zu vergleichen. Zudem kommt es zu Einbrüchen der Reaktionsfähigkeit der Webseite sobald der FBX-Parser einsetzt. Die niedrige Leistung beim Einbinden vieler Modelle und das Limit von

WebGL von 16 Contexten deuten an, dass WebGL nicht auf diese Art der Benutzung ausgelegt ist.

6.2 Erweiterungsmöglichkeiten

Verbesserung des Shortcode Workflows

Die Verwendung der Shortcodes lässt sich durch einige Erweiterungen benutzerfreundlicher gestalten.

Es könnte ermöglicht werden, dass ein bereits erstellter Shortcode vom Model Viewer wieder eingelesen werden kann, sodass der Shortcode mit dem Model Viewer verändert werden kann. Ein Liste, die die erstellten Shortcodes speichert ist hilfreich, da der Benutzer die Shortcodes nicht mehr selbst für späteren Gebrauch speichern muss. Die Shortcodes können in der Datenbank gespeichert werden.

Unterstützung des three.js Editors

Der Editor von three.js erlaubt das erstellen komplexerer Szenen in denen mehrere Objekte hierarchisch angeordnet werden können. Weiterhin werden viele Dateitypen zum Einlesen von Modellen unterstützt. Eine eingeschränkte Version der Editors wäre zu dem Model Viewer kompatibel, indem man den Model Viewer um Kompatibilität zum Objekt Format von three.js erweitert und den Editor auf den Export dieses Formats beschränkt.

Unterstützung weiterer Datei-Typen

In three.js gibt es bereits Unterstützung für weitere Dateitypen. Das Plug-in kann um diese Unterstützungen erweitert werden. In dem three.js Editor existiert Code zur Auswahl des richtigen Loaders.

Hochladen der Dateien

Der Dialog zum Hochladen für Dateien könnte verbessert werden, indem das Verwalten der Dateien in Verzeichnissen ermöglicht wird. Die WordPress Medien Bibliothek kann mit

bestehenden Plug-ins um diese Funktion erweitert werden. Weiterhin ist eine Umsetzung des Dialogs ohne die Medien Bibliothek zu prüfen, da die maximale Dateigröße beschränkt ist und Dateien die über FTP hochgeladen wurden nicht erkannt werden.

Fehleranzeige für das Frontend

Die Tests haben ergeben, dass es verschiedene Fälle gibt, die dazu führen, dass die Darstellungen ausfallen. Zu den Fällen gehört das Erstellen von zu vielen WebGL Contexten und eine Überlastung des Browsers. Zudem gibt es Browser, die WebGL nicht unterstützen. In diesen Fällen sollte es eine Fehleranzeige im Frontend geben, sodass der Besucher darauf aufmerksam gemacht wird, dass an der Position eigentlich ein Modell dargestellt werden sollte.

Fehleranzeige für das Backend

Da der FBX-Parser nicht alle FBX-Dateien unterstützt, kann es sein, dass der Benutzer eine ungültige FBX-Datei auswählt. Er sollte mit einer Fehlermeldung auf diesen Umstand aufmerksam gemacht werden. Des Weiteren sollte der Benutzer darüber informiert werden, dass er mit Hilfe des Autodesk FBX Converters ungültige FBX-Dateien in gültige Dateien umwandeln kann.

CSS Einstellungen

Der Shortcode könnte um CSS Einstellungen erweitert werden, sodass der Container besser platziert werden kann. Einstellung zu Außen- und Innenabstand würden die Einsatzmöglichkeiten und das Verhalten zu anderen Elementen deutlich verbessern. Auch die Möglichkeit der Definition eines Rahmens ist denkbar.

Responsive Webdesign

Damit die Darstellungen auch in Webseiten, die dem Paradigma des Responsive Webdesign folgen, eingesetzt werden können, müssten die Darstellungen an die Größe des Containers angepasst werden, in dem sie eingebunden sind. Dazu müsste für JavaScript eine Funktionalität hinzugefügt werden, die auf Änderungen der Größe des Fensters reagiert.

Verbesserung der Beleuchtung

Bei beistimmten Modellen funktioniert der genutzte Ansatz zur Beleuchtung der Szene nicht wie gewünscht. Die Abbildung 15 zeigt ein Modell, dass aufgrund seiner Struktur unzureichend beleuchtet wird. Eine Erweiterung des Codes zur automatischen Positionierung des Lichts kann dieses Problem lösen.

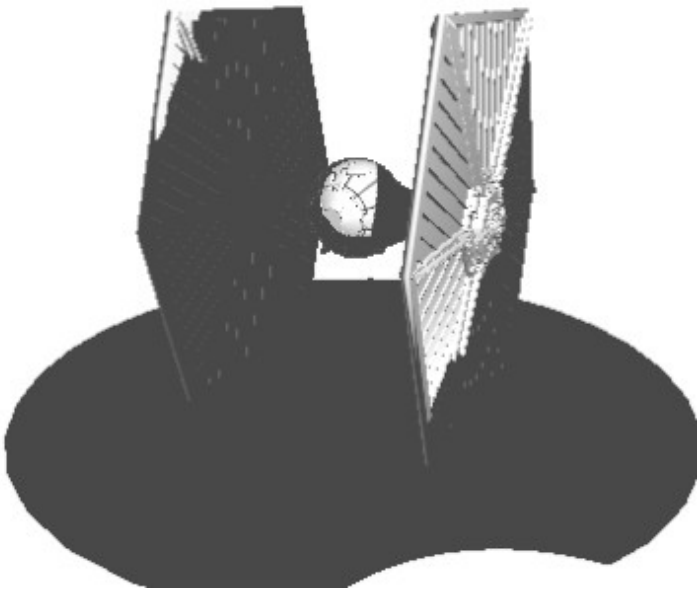

Abbildung 15: Fehlerhafte Beleuchtung

Grafik Effekte

Durch die Möglichkeit, in three.js eigene Shader zu definieren, sind den Grafik Effekten kaum Grenzen gesetzt. Dem Benutzer könnte eine Liste verschiedener Shader angeboten werden, die unterschiedliche Grafik Effekte beinhalten.

6.3 Zusammenfassung und Fazit

In diesem Projekt wird ein WordPress Plug-in zur Darstellung von CAD-Dateien in Webbrowsern entworfen und implementiert. Wordpress bewährt sich als frei erweiterbares Content-Management-System. Der Einstieg in die Entwicklung eines WordPress Plug-in ist

durch die Erstellung einer einzigen PHP-Datei getan. WordPress stellt durch Hooks passende Einstiegspunkte für den Code des Plug-ins bereit und auch das Einbinden von fremden Bibliotheken geschieht ohne Probleme.

Three.js ist eine mächtige Bibliothek, die eine Lösung für jede Problematik, die im Laufe des Projekts aufgetreten ist, bereitgestellt hat. Für die Entwicklung mit three.js werden keine Kenntnisse von OpenGL bzw WebGL benötigt. Die Funktionalitäten von three.js werden in diesem Projekt bei weitem nicht ausgeschöpft und interessante Funktionalitäten wie Shader hatten keinen Platz in diesem Projekt. Die Probleme beim Einlesen von FBX-Dateien zeigen aber auch, dass three.js noch erweiterbar ist.

Mit ECMAScript 6 wurde eine neue Syntax für Klassen in JavaScript eingeführt. Die Klasse Model Viewer wurde mit Hilfe dieser Syntax erstellt. Mit dieser Syntax lassen sich Klassen und Methoden für die Klassen definieren, es fehlt allerdings die Möglichkeit, Member Variablen zu definieren. Über das *this* Objekt kann allerdings auf Variablen der Klasse zugegriffen werden. Durch den Gebrauch von anonymen Funktionen geht aber der Kontext für dieses Objekt verloren und es muss stets darauf geachtet werden, dass dieser Kontext wiederhergestellt wird. Dies führt zu unübersichtlicher Syntax.

Die durchgeführten Tests und Analysen attestieren dem Plug-in eine zu hohe Belastung des Browsers. Gerade für mobile Geräte sind Ladezeit, Datenlast und Belastung der Hardware zu hoch. Für Desktop Computer ist eine Benutzung in gewissem Maße möglich. Abschließend bleibt festzustellen, dass die Umsetzung möglich ist, WebGL aber nicht auf diese Benutzung ausgelegt ist.

Anhang A Referenzen

Web-Seiten zuletzt am 23.05.2016 abgerufen.

[@babylon] Babylon Datei Format
https://doc.babylonjs.com/generals/File_Format_Map_(.babylon)
[@cms] Web Content Management Systems, a Collaborative Environment in the Information Society http://revistaie.ase.ro/content/50/003%20-%20Mican.pdf
[@cmss] CMS Marktanteile in Deutschland http://www.cmscrawler.com/country/DE
[@collada] Collada Dateiformat https://www.khronos.org/collada/
[@fbx] FBX Süezifikation von Blender
https://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Import-Export/Autodesk_FBX
[@fbxsdk] FBX SDK http://usa.autodesk.com/adsk/servlet/pc/item?siteID=123112&id=10775847
[@htac] htaccess mit Apache https://httpd.apache.org/docs/2.4/howto/htaccess.html
[@http] Dokumention HTTP https://tools.ietf.org/html/rfc2616
[@md2] MD2 Dateiformat http://tfc.duke.free.fr/old/models/md2.htm
[@obj] OBJ Dateidormat http://www.fileformat.info/format/wavefrontobj/egff.htm
[@ogl shader] Shader in OpenGL
https://www.opengl.org/registry/doc/GLSLangSpec.Full.1.10.59.pdf
[@pipe] OpenGL Rendering Pipeline https://www.opengl.org/wiki/Rendering_Pipeline_Overview
[@prim] Grafik Open GL Primitive http://vision-simulator.tripod.com/report/section_7.htm
[@sc] WordPress Shortcode API https://codex.wordpress.org/Shortcode_API
[@serv] Webserver erklärt http://www.webdevelopersnotes.com/basics/what_is_web_server.php
[@servs] Verbreitung Webserver http://news.netcraft.com/archives/2015/03/19/march-2015-web-server-survey.html
[@statsjs] stats.js GitHub https://github.com/mrdoob/stats.js
[@three] three.js GitHub https://github.com/mrdoob/three.js/
[@threeex] three.js Beispiele
https://github.com/mrdoob/three.js/blob/master/examples/webgl_animation_scene.html
[@threef] three.js features https://github.com/mrdoob/three.js/wiki/Features
[@TLS] TLS/SSL faq http://httpd.apache.org/docs/2.4/ssl/ssl_faq.html
[@vert] Open GL Vertex Shader https://www.opengl.org/wiki/Vertex_Shader
[@webGl] WebGL Spezifikation https://www.khronos.org/webgl/

[@wp] WordPress Codex https://codex.wordpress.org/About_WordPress

[@wpe] WordPress enqueue script Referenz

https://developer.wordpress.org/reference/functions/wp_enqueue_script/

[@wpm] WordPress Medien Bibliothek Referenz

https://codex.wordpress.org/Javascript_Reference/wp.media

[@wps] Verbreitung CMS http://www.opensourcecms.com/general/cms-marketshare.php

[par12] T. Parisi, WebGL Up and Running, 1. Auflage, O'Reilly Media, Sebastopol Californien, 2012. Seite 2 - 4

[par12b] T. Parisi, WebGL Up and Running, 1. Auflage, O'Reilly Media, Sebastopol Californien, 2012. Seite 17 - 20

[DGJ13] D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane OpenGL Programming Guide, Version 4.3 , Addison-Wesley, 2013. Kapiel 1

[DGJ13b] D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane OpenGL Programming Guide, Version 4.3 , Addison-Wesley, 2013. Kapiel 6

Anhang B Grafiken

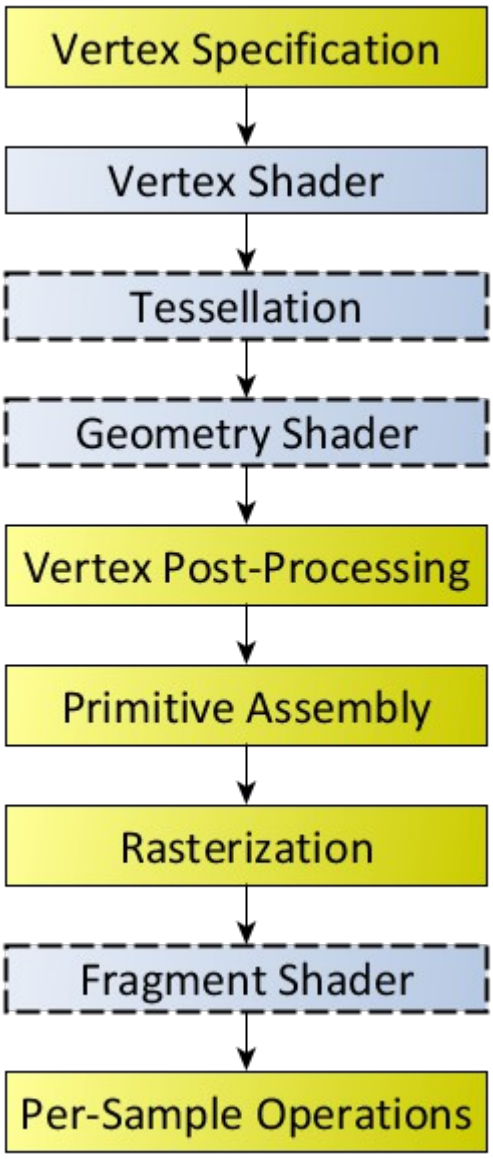

Abbildung 16: OpenGL Rendering Pipeline [@pipe]

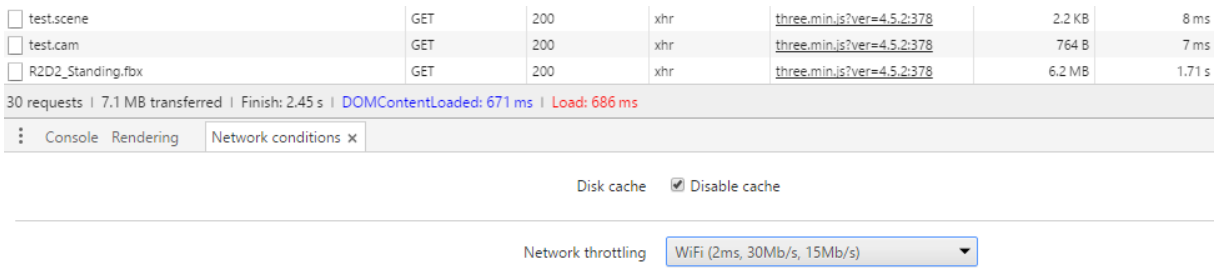

Abbildung 17: Ladezeit der Webseite 1 Modell

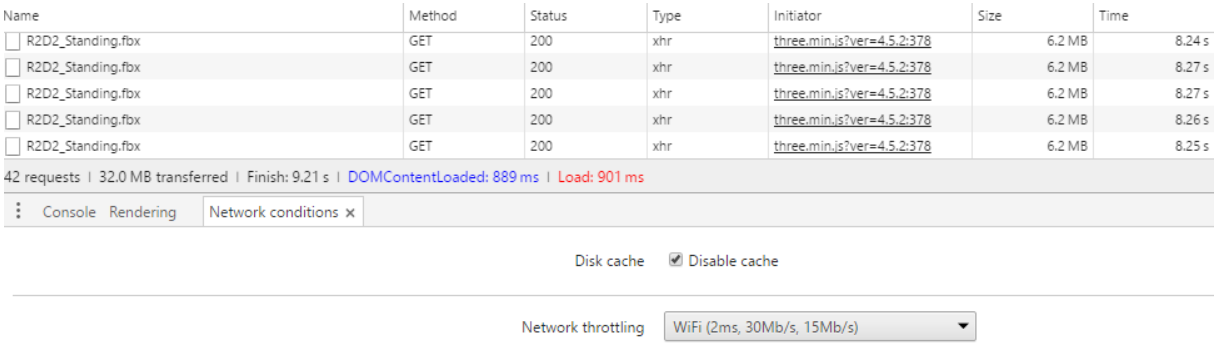

Abbildung 18: Ladezeit der Webseite 5 Modelle

Anhang C Inhalt der CD

In der beigefügten CD sind folgende Ordner und Dateien enthalten.

Ordnerverzeichnis	Dateien	Beschreibung
\Dokumentation	Bachelorarbeit_Schuermeyer.pdf	Die Bachelorarbeit im Portable Document Format (PDF)
	Bachelorarbeit_Schuermeyer.doc	Die Bachelorarbeit im Microsoft Word Format
	Bachelorarbeit_Schuermeyer.odt	Die Bachelorarbeit im Open Document Text Format
	Handbuch.pdf	Handbuch zur Installation und Benutzung des Plug-ins
\Quellen	Quellen.zip	Archiv mit den genutzten Quellen im PDF Format
\Software	cad-model-viewer.zip	Archiv mit dem erstellten Plug-in

Tabelle A.1: Inhalt der CD

Erklärung

Hiermit versichere ich, dass ich meine Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Datum:

…….....

(Unterschrift)