Technische Universität Braunschweig
Gauß-IT-Zentrum

Technische Universität Braunschweig | Gauß-IT-Zentrum
Hans-Sommer-Straße 65 | 38106 Braunschweig | Deutschland

VESPER Plus

Abschlussbericht

Verbundprojekt: **Verbesserung der Sicherheit von Personen in der Fährschifffahrt (VESPER Plus)**

Teilvorhaben: **Simulationsbasierte Visualisierung für sicherheitsrelevante Prozesse**

FKZ: **13N11922**

Zuwendungsempfänger:	Technische Universität Carolo-Wilhelmina zu Braunschweig – Gauß-IT-Zentrum
Projektleiter:	Norbert Schenk
Laufzeit des Vorhabens:	01.09.2011 – 31.08.2014
Anlagen:	A1: Dokumentation Web-Tool Sprengstoffdetektion A2: Dokumentation Mobile 3D-Visualisierung

10.04.2015

Norbert Schenk

Inhalt

1 Kurze Darstellung

1.1 Aufgabenstellung

VESPER Plus ist das Nachfolgeprojekt von VESPER. In beiden Projekten stehen Probleme der maritimen Sicherheit von Personen in der Fährschifffahrt im Vordergrund. Wie bereits bei VESPER war die Aufgabe des Gauß-IT-Zentrums, die teilnehmenden Kooperationspartner in verschiedenen Arbeitspaketen durch innovative Software-Lösungen zu unterstützen. Konkret waren Beiträge zu den Arbeitspaketen 1, 2 und 4 geplant.

1.2 Voraussetzungen

Die personellen Voraussetzungen konnten leider nicht wie üblich vorgesehen für das Nachfolgeprojekt erhalten werden. Der durch VESPER geförderte Mitarbeiter Christian Woizischke verließ das Gauß-IT-Zentrum, weil eine Anschlussfinanzierung zunächst nicht gesichert schien. Die Stelle konnte erst 11 Monate nach Projektbeginn zum 01.08.2012 mit Alexey Beresnev durch einen geeigneten Nachfolger besetzt werden.

Eine weitere Änderung ergab sich in der Projektleitung. Der bisherige Projektleiter Carsten Hellmich verließ das Gauß-IT-Zentrum zum 30.06.2012. Als sein Nachfolger wurde Norbert Schenk benannt.

1.3 Planung und Ablauf

Die ursprüngliche Planung wurde in Absprache mit den Projektpartnern den geänderten Voraussetzungen angepasst. Dabei musste auch berücksichtigt werden, dass leider keine der in der Förderung enthaltenen studentischen Hilfskraft-Stellen geeignet besetzt werden konnten. Details finden sich in Abschnitt 2.1.

1.4 Wissenschaftlicher und technischer Stand

Die entstandenen Software-Systeme basieren auf aktuellen Technologien. Details finden sich in den jeweiligen Anhängen A1 und A2.

1.5 Zusammenarbeit mit anderen Stellen

Zum Arbeitspaket 2.1.3 „Mensch-Maschine-Schnittstelle bei Detektionsgeräten“ fand eine enge Abstimmung mit den Projektpartnern Fraunhofer-Institut für Kommunikation, Informationsverarbeitung und Ergonomie sowie mit der Hochschule Bonn-Rhein-Sieg statt. Zum Meinungsaustausch mit den übrigen Projektpartnern dienten die regelmäßigen Meilenstein-Treffen.

2 Eingehende Darstellung

2.1 Geplante und erzielte Ergebnisse

Vor allem aus den in Abschnitt 1.2 dargelegten Gründen wurden in Absprache mit den Projektpartnern der Gesamtumfang reduziert und neue Schwerpunkte festgelegt. Im Einzelnen ergaben sich folgende Abweichungen und Ergebnisse:

AP 1.1.3 Visualisierung von Prozessabläufen

Dieses Arbeitspaket mit wurde ersatzlos gestrichen.

AP 2.1.3 Mensch-Maschine-Schnittstelle bei Detektionsgeräten

Die Anforderungen der Projektpartner hatten sich im Vergleich zum Antrag geändert. Benötigt wurde eine Web-Anwendung zur Erfassung der Messdaten von Gefahrstoffdetektionsgeräten.

Durch die enge Zusammenarbeit mit der Hochschule Bonn-Rhein-Sieg und dem Fraunhofer FKI kristallisierten sich folgende Änderungen heraus: Statt des im Antrag vorgesehenen Demonstrators und virtuellen Testbetts wurde eine vollständige Web-Anwendung zur Erfassung der Messdaten von Gefahrstoffdetektionsgeräten im Hafenbetrieb und Kommunikation mit entfernten Experten zur Spektralanalyse implementiert.

Die neuen (erweiterten) Ziele wurden erreicht. Die resultierende Software wird in Anhang A1 beschrieben.

AP 4.3.1 Physikalische Simulation

Aufgrund von technischen Fortschritten ist es (selbst auf mobilen Plattformen) nicht mehr nötig, die physikalische Simulation von der Client-seitigen Darstellung zu trennen und auf dem Server zu berechnen. Die vorgesehene Simulation wurde daher Client-seitig implementiert.

AP 4.3.2 3D-Trainingsumgebung

Die notwendigen 3D-Modelle wurden für erste Tests und Machbarkeitsstudien erstellt. Professionelle Fahrzeugmodelle konnten aus der im Vorgängerprojekt erworbenen Modell-Sammlung von Dosch Design importiert und verwendet werden. Vorberechnete Sequenzen sind nach dem Stand der Technik nicht mehr notwendig.

AP 4.3.3 Mobile Endgeräte

Nach der Evaluation stellte sich heraus, dass ein plattformunabhängiger Ansatz nach dem aktuellen Stand der Technik eher geeignet ist. Die Simulation wurde entsprechend implementiert und getestet.

Die resultierende Software zu AP 4.3.x wird in Anhang A2 beschrieben

2.2 Zu den Positionen des zahlenmäßigen Nachweises

Insgesamt wurden aus den in Abschnitt 1.2 genannten Gründen nur 132.479,77 EUR statt der geplanten 212.670,- EUR verausgabt. Diese Reduktion um mehr als ein Drittel spiegelt sich auch in den geänderten Schwerpunkten (Abschnitt 2.1) wider. Details zu den Ausgaben sind dem Verwendungsnachweis zu entnehmen. Drei Finanzierungspositionen wurden genutzt:

0812 Personalkosten Wissenschaftlicher Mitarbeiter

Dieser mit Abstand größte Posten entfällt auf die Finanzierung des Wissenschaftlichen Mitarbeiters. Durch die reduzierte Laufzeit von 25 statt 36 Monaten ist er nicht ausgeschöpft.

0846 Reisekosten

Die Ausgaben fielen für Meilenstein-Treffen und Treffen mit den Projektpartnern an. Auch hier bleiben die Ausgaben unter dem Plan.

0850 Geräte

Hier wurden drei verschiedene Tablet-PCs zu Testzwecken und zum Nachweis der Plattformunabhängigkeit zu Arbeitspaket 4.3.3 beschafft. Es wurden alle verbreiteten Betriebssysteme berücksichtigt, die genauen Produkte sind dem Verwendungsnachweis zu entnehmen. Nach Erfüllung ihrer Projektaufgabe gingen die Geräte in die Multimedia-Ausleihe des Gauß-IT-Zentrums über, um dort für wissenschaftliche Projekte von Angehörigen der TU Braunschweig zusammen mit dem dort bereits vorhandenen Angebot ausgeliehen werden zu können.

2.3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Die wesentliche Arbeit bestand in diesem Teilprojekt in der Anforderungsanalyse, Implementierung und Dokumentation der beiden Software-Produkte, die in den Zielen als notwendig vereinbart wurden. Die geleisteten 25 Personenmonate scheinen angesichts der Komplexität (siehe Anhänge) als angemessen.

2.4 Verwertbarkeit des Ergebnisses

Zur Verwertbarkeit des Ergebnisses unterscheiden wir die beiden Hauptresultate:

A1: Web-Tool Sprengstoffdetektion

Das Werkzeug ist in enger Kooperation mit den potenziellen Endanwendern entstanden und könnte in dieser Form mit wenigen Anpassungen bereits zur Analyse von Gefahrstoffen an Kontrollstellen zum Einsatz kommen.

A2: Mobile 3D-Visualisierung

Die mobile 3D-Visualisierung hat eher den Charakter einer Machbarkeits-Studie. Verwertbar ist die Erkenntnis, welche Technologien geeignet und zukunftssicher sind.

2.5 Fortschritt bei anderen Stellen

Die 3D-Visualisierung auf mobilen Plattformen hat im Laufe des Projekts große Fortschritte gemacht. Dieser Entwicklung wurde Rechnung getragen, wie aus Anhang A2 ersichtlich.

2.6 Erfolgte oder geplante Veröffentlichungen

Die Ergebnisse sind als Dienstleistung an die übrigen Projektpartner zu verstehen und sollen deren Forschung unterstützen. Von Seiten des Gauß-IT-Zentrums sind daher keine Veröffentlichungen erfolgt oder geplant.

Technische Universität Braunschweig | Gauß-IT-Zentrum
Hans-Sommer-Straße 65 | 38106 Braunschweig | Deutschland

VESPER Plus

Anlage A1

Dokument:	Dokumentation der Software zur Erfassung der Messdaten von Sprengstoffdetektionsgeräten
Zuwendungsempfänger:	Technische Universität Carolo-Wilhelmina zu Braunschweig - Gauß-IT-Zentrum
Authoren:	Alexey Beresnev, Norbert Schenk
Projektleiter:	Norbert Schenk
FKZ:	13N11922

Inhaltsverzeichnis

1 Einführung

Im Rahmen der Beteiligung am VESPER^{Plus}-Projekt ist am Gauß-IT-Zentrum der TU Braunschweig (GITZ) im Zeitraum zwischen 01.09.2012 und 28.02.2013 eine mobile Web-Anwendung entstanden, welche die Erfassung, Protokollierung und Auswertung der mit Hilfe von Sprengstoffdetektionsgeräten gemessenen Daten erleichtert.

1.1 Motivation

Der Bedarf an der vom GITZ entwickelten Software besteht aufgrund der Unzulänglichkeiten der zur Zeit verfügbaren Sprengstoffdetektionsgeräte. Diese speichern die Messspektren und Detektionsergebnisse in der Regel in einzelnen Dateien auf Speicherkarten und sind nicht für eine umfangreichere Datenverwaltung über längere Zeiträume oder hohe Verfügbarkeit von ermittelten Daten (z.B. über das Internet) ausgelegt. Darüber hinaus unterscheiden sich die Dateiformate und -namen in Messgeräten unterschiedlicher Hersteller, was eine einheitliche Auswertung erschwert. Das Einsatzszenario der Detektionsgeräte in der Fährschifffahrt stellt aber folgende Anforderungen:

- Zusätzlich zu chemischen Daten sollen auch die Daten des untersuchten Fahrzeugs erfasst und gespeichert werden.
- Viele Fahrzeuge werden untersucht, und es fallen viele Messdaten an, die auch wesentlich später noch verfügbar sein sollen (z.B. um nach Fehlerursachen zu suchen).
- Die Messungen werden nicht von erfahrenen Chemikern durchgeführt. Diese müssen aber im Zweifel sehr schnell konsultiert werden können und den Zugriff auf die Daten erhalten.
- Die Software soll alle Messungen zentral erfassen. Auch mehrere mobile Clients sollen gleichzeitig Daten hinzufügen und bearbeiten können.
- Die Benutzeroberfläche der Erfassungssoftware wird sowohl auf normalen Rechnern als auch auf mobilen Endgeräten (mit kleineren Bildschirmen) dargestellt.
- Es werden mehrere Detektionsgeräte unterschiedlicher Hersteller parallel eingesetzt, um eine möglichst große Anzahl der Gefahrstoffe detektieren zu können.
- Die Messungen sollen den Fährbetrieb nicht beeinträchtigen und daher schnell durchgeführt und ausgewertet werden.

1.2 Ziele

Aus den unter Fehler: Referenz nicht gefunden beschriebenen Anforderungen ergeben sich folgende Ziele für die Erfassungssoftware:

- **Web-Applikation:** Die klassische Architektur einer Web-Applikation mit einem zentralen Server und beliebig vielen (browserbasierten) Clients erfüllt die Anforderung des effizienten Parallelbetriebs ideal. Außerdem erlaubt die zentrale Datenspeicherung die erfassten Daten sofort einem entfernten Experten zur Verfügung zu stellen, der sich überall auf der Welt, wo es einen Internetzugang gibt, befinden kann.
- **Mobile Oberfläche:** Um die Datenerfassung vor Ort neben dem normalen Hafenbetrieb zu ermöglichen, bieten sich als Endgeräte die (heute ohnehin sehr verbreiteten) Tablet-Rechner oder Smartphones an. Diese Geräte sind heute in der Lage, beliebig komplexe Web-Oberflächen anzuzeigen, haben allerdings prinzipbedingt vergleichsweise kleine Bildschirme, was die Web-Oberfläche berücksichtigen sollte, um trotzdem gut lesbar zu bleiben. Außerdem werden die mobilen Endgeräte direkt per Fingergesten bedient, was im Vergleich mit „klassischen“ Webseiten teilweise etwas andere Bedienelemente erfordert.

- **Protokollierung:** Die früher erfassten Daten sollen in der Programmoberfläche durchsuchbar und visualisierbar sein. Neben den eigentlichen Messdaten werden zu jeder Messung die Fahrzeugdaten, die Zeit, der eingeloggte Benutzer sowie optional ein Foto gespeichert.
- **Automatische Zuordnung:** Aus dem Umstand, dass die vorhandenen Messgeräte die Ergebnisdateien lediglich auf eine Speicherkarte schreiben ergibt sich die Anforderung, dass diese Dateien erst nachträglich den bereits angelegten Messungen zugeordnet werden können. Ansonsten wäre das Herausnehmen der Speicherkarte aus dem Gerät nach jeder Messung zu umständlich.

2 Funktionsbeschreibung

2.1 Benutzerverwaltung

Die Programmoberfläche ist nur nach Eingabe eines Benutzernamens und Passworts erreichbar. Je nach Berechtigung kann der Benutzer entweder nur auf das Front-End (Fehler: Referenz nicht gefunden) oder zusätzlich auch auf das Backend (Fehler: Referenz nicht gefunden) zugreifen, wo auch neue Benutzer angelegt und existierende verwaltet werden können. Der aktuell angemeldete Benutzer wird in allen von ihm im Frontend angelegten oder veränderten Messungen zwecks Nachvollziehbarkeit als Operator eingetragen. In allen Ansichten wird oben rechts der Benutzername angezeigt. Mit einem Klick auf „Logout“ kann der Benutzer sich abmelden.

2.2 Front-End

2.2.1 „Responsive Webdesign“

Alle Ansichten des Frontends sind mit den Techniken des sogenannten „Responsive Webdesign“ gestaltet. Das bedeutet, dass die Programmoberfläche sich automatisch an die Bildschirmauflösung des verwendeten Anzeigegeräts anpasst. Mobile Endgeräte verfügen meistens über kleinere Bildschirme mit kleineren Auflösungen und werden häufig senkrecht (im Vergleich zu einem typischen Monitor) verwendet. Zudem werden sie per Fingergesten bedient, die nicht so präzise wie eine Computermaus sind. Damit herkömmliche Webseiten noch bedienbar bleiben, werden sie meistens sehr klein dargestellt und können dann in Ausschnitten vergrößert werden. Beim „Responsive Webdesign“ passt die Webseite stattdessen automatisch die Anordnung und Größe der Bedienelemente, sowie Schriftgröße und Textfluss so an, dass sie nicht verkleinert und vergrößert werden muss.

Abb. 1: Größenvergleich herkömmliche und mobile Selectbox

Die Front-End-Ansichten des Erfassungsprogramms sind meistens in Hauptbereich und Seitenleiste aufgeteilt. Steht auf dem Anzeigegerät genug Breite zur Verfügung, wird die Seitenleiste links vom Hauptbereich angezeigt. Ist der Bildschirm aber zu schmal, wandert die Seitenleiste nach oben und wird über der Hauptansicht angezeigt. Zusätzlich werden in der Oberfläche Steuerelemente verwendet, die zur Fingerbedienung besser geeignet ist. Die Abb. 1 zeigt einen Größenvergleich zwischen einer normalen Selectbox und einer, die in der Programmoberfläche verwendet wird.

2.2.2 Formular- und Listenansicht

Die Hauptansicht der Software (Abb. 2) besteht aus dem Formular für die Erfassung eines neuen Datensatzes, der Liste der zuletzt hinzugefügten Messungen und den ausklappbaren Filtereinstellungen für die Liste (Abb. 3). Für die Erfassung einer neuen Messung ist die Eingabe eines Kennzeichens und die Auswahl eines Messgeräts notwendig. Falls ein Fahrzeug mit dem eingegebenen Kennzeichen noch nicht existiert, sollte auch das entsprechende Model und die Farbe des Fahrzeugs ausgewählt werden. Wenn bereits vorhanden, können die Messdaten und ein Foto direkt hochgeladen werden, dies kann aber auch zu einem späteren Zeitpunkt geschehen. Bei der Auswahl eines Messgeräts wird das

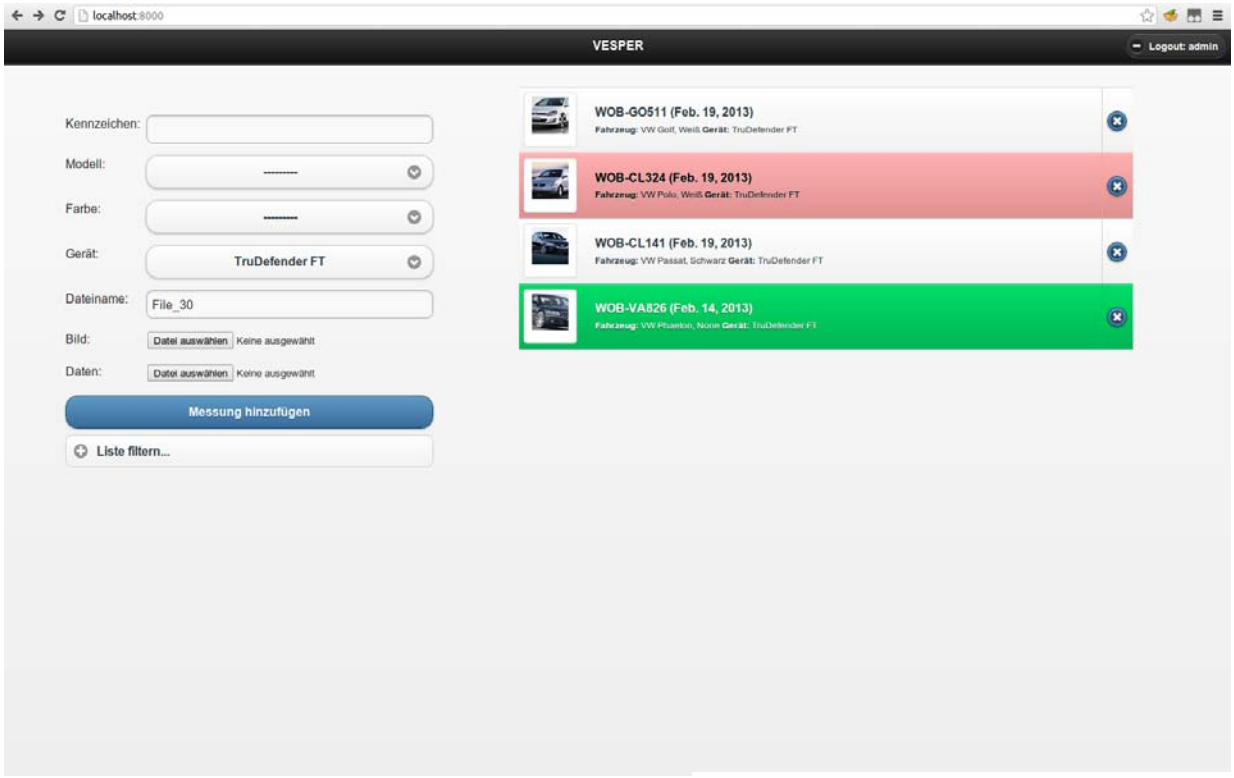

Abb. 2: Die Hauptansicht

Feld Dateiname automatisch ausgefüllt und zwar nach dem für das Messgerät hinterlegten Muster (siehe 2.3.3). Der automatisch generierte Dateiname kann aber überschrieben werden.

Die Liste im Hauptbereich zeigt die zuletzt hinzugefügten Messungen. Pro Eintrag wird (falls vorhanden) eine Miniaturansicht des Photos, das Fahrzeugkennzeichen und -modell sowie das verwendete Messgerät angezeigt. Es besteht auch die Möglichkeit, einen Eintrag (durch Klick auf das X-Symbol rechts) zu löschen. Jeder Eintrag ist entsprechend seinem Status eingefärbt:

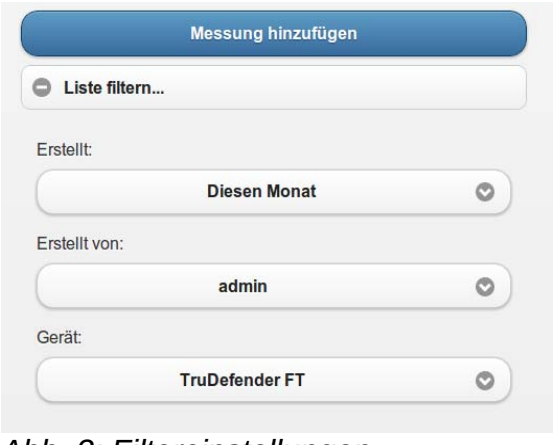

Abb. 3: Filtereinstellungen

- Grün: Status „vollständig“. Die Daten wurden hochgeladen und die Untersuchung hat keinen Verdacht erregt.

- Weiß: Status „unvollständig“. Die Messung wurde zwar angelegt, aber es wurden noch keine Messdaten hochgeladen oder automatisch zugeordnet.
- Gelb: Status „verdächtig“. Die Daten sind vollständig und die Messergebnisse sind nicht eindeutig negativ. Es wird auf die Untersuchung durch einen Sprengstoffexperten gewartet.
- Rot: Status „Alarm“. Entweder die Messung mit dem Detektionsgerät oder die Analyse des Experten hat das Vorhandensein von Gefahrstoffen ergeben.

Der in der Liste angezeigte Zeitraum und weitere Filtereinstellungen können in einem Ausklappmenü unter „Liste filtern...“ gewählt werden (Abb. 3). Standardmäßig werden in der Liste nur die Messungen des aktuellen Tages angezeigt. Zusätzlich gibt es die Möglichkeit, nur die Messungen von einem oder mehreren bestimmten Geräten anzuzeigen und/oder von bestimmten Mitarbeitern durchgeführte Messungen.

2.2.3 Messungsansicht

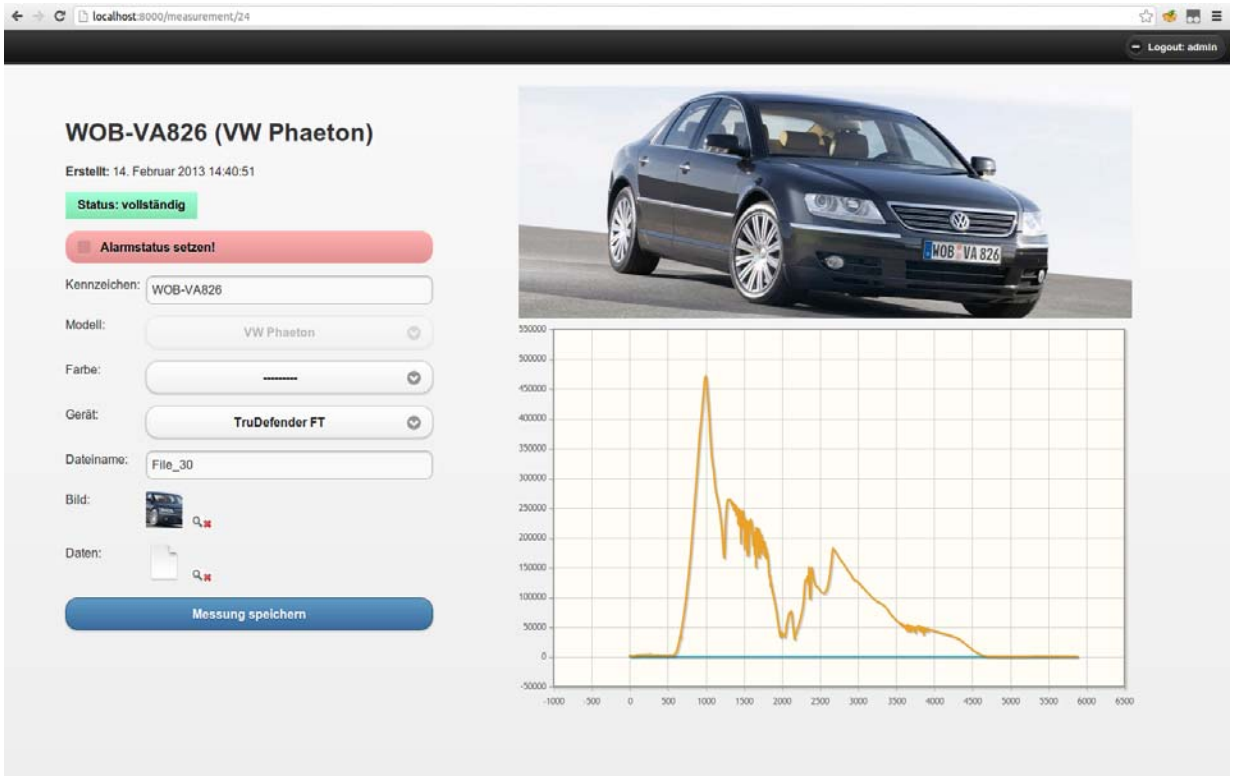

Abb. 4: Messungsansicht (Desktop/Tablet)

Mit einem Klick auf einen Listeneintrag gelangt der Benutzer in die detaillierte Ansicht einer Messung (Abb. 4 und Abb. 5). Hier werden links die eingegebenen Daten, das Erstellungs- und (falls zutreffend) Änderungsdatum sowie der aktuelle Status der Messung („vollständig“, „unvollständig“, „verdächtig“ oder „Alarm“) angezeigt. Einige der Daten (z.B. der Dateiname) lassen sich hier nachträglich ändern. Wichtig ist vor allem die rot hervorgehobene Schaltfläche „Alarmstatus setzen!“, mit der die Messung als alarmierend markiert werden kann, nachdem die Daten vollständig sind.

Wurden die Messdaten oder eine Bilddatei bereits hochgeladen, erscheint in der Hauptansicht rechts das aufgenommene Bild in höherer Auflösung (je nach der zur Verfügung stehenden Breite des Anzeigegeräts). Darunter erscheint die graphische Darstellung der Messdaten. (Hier muss die endgültige Darstellung noch mit Projektpartnern abgesprochen werden, insbesondere ob auch bekannte Gefahrstoffspektren aus einer proprietären Datenbank dargestellt werden können und sollen.)

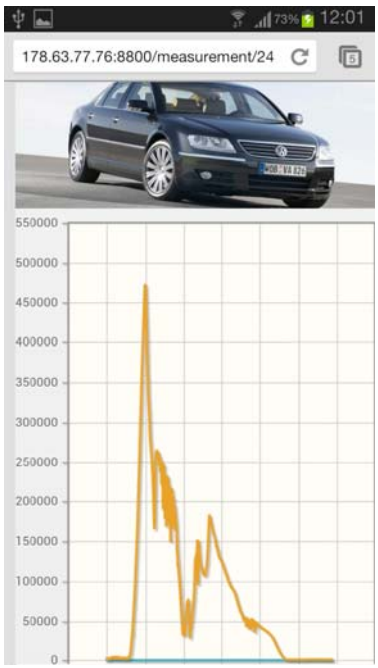

Abb. 5: Messungsansicht (Smartphone)

2.3 Back-End

Das Back-End ist eine weitere Benutzeroberfläche des Programms (Abb. 6), die den vollen Zugriff auf die Daten gestattet und für administrative Aufgaben benutzt wird. Im Back-End werden z.B. die Benutzer und ihre Berechtigungen verwaltet, aber auch die hochgeladenen Dateien und die im Front-End zur Auswahl stehenden Fahrzeugmodelle und Farben.

Die Hauptansicht des Back-Ends zeigt die zu verwaltenden Objekte in drei Gruppen unterteilt:

- Auth: Die Benutzer- und Gruppenverwaltung
- Filer: Dateiverwaltung
- Main: Hauptprogrammdaten (Messungen, Fahrzeugdaten, Messgeräte)

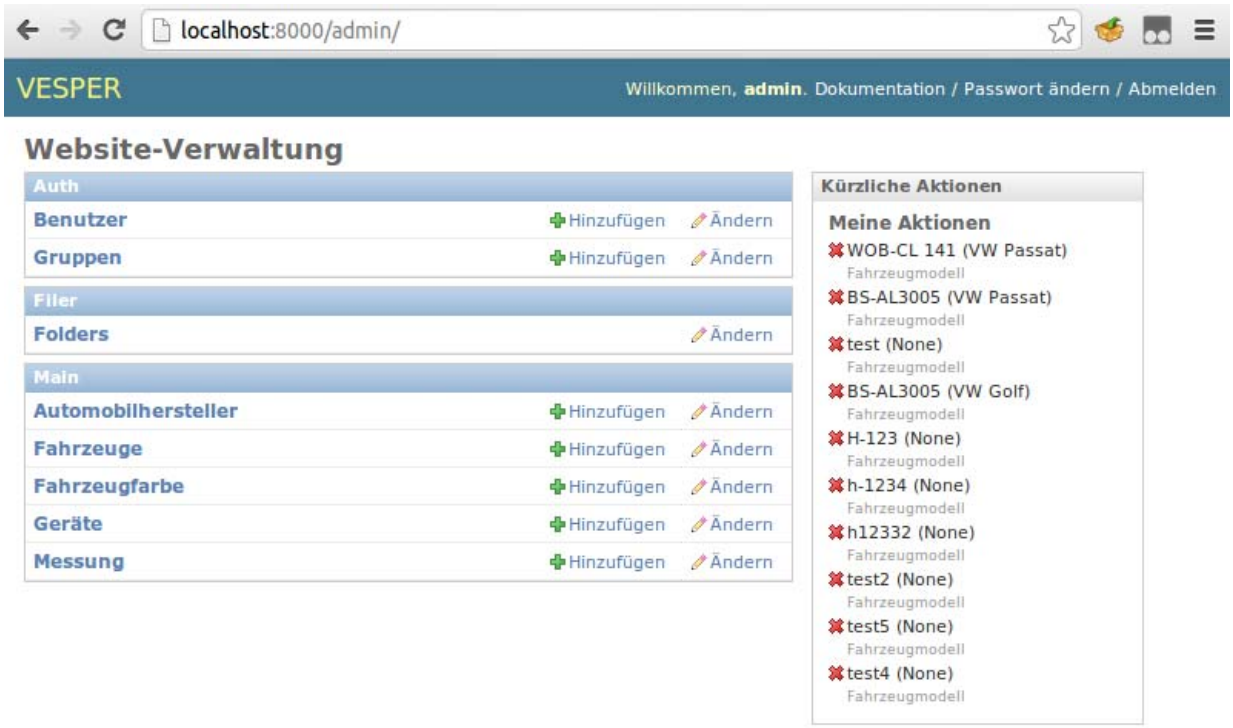

Abb. 6: Back-End

2.3.1 Die Benutzer- und Gruppenverwaltung

Die Benutzerverwaltung erlaubt eine sehr feine Zuweisung von Berechtigungen zu den einzelnen Benutzern des Programms oder auch zu Benutzergruppen. Eine Berechtigung erlaubt (wenn sie zugewiesen wurde) eine bestimmte Operation an einem Objekt des Programms. Wird eine Berechtigung nicht zugewiesen und der Benutzer ist kein Administrator, bedeutet es, dass die Operation dem Benutzer nicht erlaubt ist. So gibt es z.B. eine Berechtigung, die das Hinzufügen von neuen Messungen erlaubt und eine Berechtigung für das nachträgliche Bearbeiten von Messungen. Abb. 7 zeigt einen Ausschnitt aus der Liste der verfügbaren Berechtigungen. Außerdem kann der Benutzer Mitglied in beliebig vielen Gruppen sein (z.B. Hafenmitarbeiter, Chemiker usw.). Weil die direkte Zuweisung von Berechtigungen für jeden Benutzer aufwendig und fehleranfällig wäre, bietet es sich an, die wichtigsten Berechtigungen an die Gruppen zu knüpfen und die Benutzer nach Bedarf in diese einzusortieren.

Abb. 7: Berechtigungen

Darüber hinaus existieren im „Benutzer ändern“-Formular unter dem Menüpunkt „Benutzer“ noch die Einstellungen „Administrator-Status“ und „Redakteur-Status“. Der Administrator-Status verleiht dem Benutzer alle möglichen Berechtigung, ohne dass sie explizit zugewiesen werden müssten. Der Redakteur-Status ermöglicht den Zugang zum Back-End, allerdings reglementiert durch die Berechtigungen des Benutzers oder seiner Gruppen. Durch das Entfernen des Aktiv-Status eines Benutzers kann er vorübergehend gesperrt werden, ohne seine Daten zu löschen.

2.3.2 Die Dateiverwaltung

Im Back-End-Abschnitt „Folders“ können alle im Front- oder Back-End hochgeladenen Dateien durchsucht werden (Abb. 8). Die Dateien sind wie gewöhnlich in Verzeichnissen organisiert, die wiederum Unterverzeichnisse enthalten können. Von Bildern werden verkleinerte Vorschau-Versionen direkt in der Dateiliste angezeigt. Dabei ist diese Verzeichnisstruktur im Back-End rein virtuell und hat nichts mit der tatsächlichen Ablage der Dateien auf dem Web-Server gemeinsam. So erlaubt es die Dateiverwaltung, Zugriffsrechte auf Verzeichnisse und Dateien an die Programmbenutzer zu binden, ohne dass entsprechende

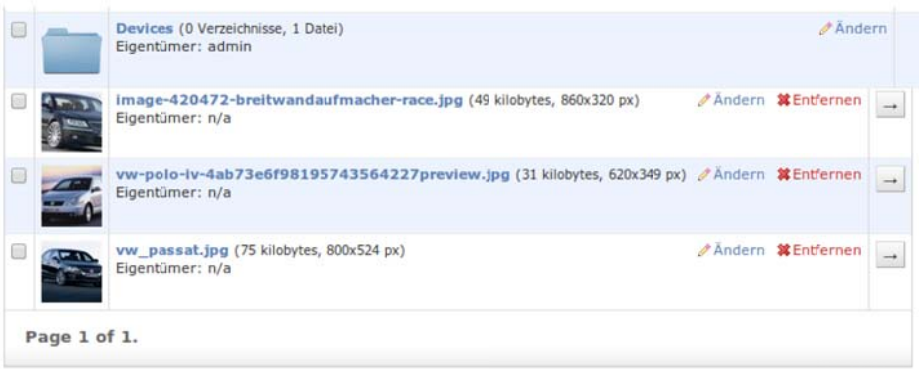

Abb. 8: Die Dateiverwaltung

Benutzer auch im Betriebssystem des Servers angelegt werden müssten. Prinzipiell können beliebig viele zusätzliche Verzeichnisse angelegt und Benutzern zugewiesen werden (z.B. für zusätzliche Dokumente), standardmäßig sind aber die Verzeichnisse „Fotos“ und „Messdaten“ vorhanden, in denen die entsprechenden Dateien gespeichert werden.

2.3.3 Programmdaten

Im Hauptabschnitt „Main“ der Back-End-Oberfläche werden folgende Objekte verwaltet:

- **Fahrzeuge:** In diesem Bereich werden alle Fahrzeuge, die in den Messungen erfasst wurden mit ihrem Kennzeichen, Hersteller, Model und Farbe aufgelistet. Die Eigenschaften können hier auch nachträglich geändert werden und die neuen Werte sind sofort im Front-End sichtbar.
- **Automobilhersteller:** Die Objekte dieses Typs verfügen über einen Namen und eine beliebig lange Liste von Modellbezeichnungen. Zwecks Datenkonsistenz ist im Front-End derzeit für die Hersteller- und Modellbezeichnung keine freie Eingabe, sondern nur die Auswahl aus den hier angelegten Werten möglich, daher sollten hier nach Möglichkeit alle existierenden Hersteller und Modelle eingegeben (oder importiert) werden.
- **Fahrzeugfarben:** Auch für die Farbbezeichnung ist im Front-End keine freie Eingabe, sondern nur die Auswahl aus den hier definierten Werten möglich. Zusätzlich zu einem Namen kann pro Farbe auch ein Farbcode eingegeben werden, damit später auch ein Beispiel für die Farbauswahl dargestellt werden kann. Der Farbcode wird in der vom HTML-Standard bekannten Notation eingegeben: entweder als englischer Name ('white', 'black', 'red' usw.) oder als sechs-stelliger hexadezimaler RGB-Code, angefangen mit dem '#'-Zeichen. (Also: weiß='#FFFFFF', schwarz='#000000', rot='#FF0000' usw.)
- **Geräte:** Jedes Objekt in der „Geräte“-Liste hat außer eines Namens und eines optionalen Bildes ein Muster für die Vorhersage der vom entsprechenden Gerät generierten Dateinamen.

Abb. 9: Gerätekonfiguration

Das Feld ist eine Zeichenfolge, die im normalen Text auch Variablen der Form „$name“ enthalten kann. Diese Variablen werden durch entsprechende Werte ersetzt, um den nächsten Dateinamen des Gerätes vorherzusagen. Als Beispiel kann der Standardwert des Muster-Feldes dienen. Er lautet „File_$counter“. Für die Variable „$counter“ wird der aktuelle Wert der Eigenschaft „Zähler“ des Geräts eingesetzt, welche ebenfalls in den Einstellungen des Geräteobjekts eingestellt werden kann. Die Eigenschaft „Zähler“ wird nach jedem Einsetzen automatisch um

eins erhöht, so dass als Dateinamen folgende Werte vorhergesagt werden: „File_1“, „File_2“, „File_3“ usw. Außerdem können folgende Variablen im Dateinamenmuster verwendet werden:

- $day – der zum Zeitpunkt der Vorhersage aktuelle Tag des Monats (z.B. 21)
- $month – die Monatszahl zum Zeitpunkt der Vorhersage (z.B. 02)
- $year – die Jahreszahl zum Zeitpunkt der Vorhersage (z.B. 2013)
- $hour – die Stunde zum Zeitpunkt der Vorhersage (z.B. 13)
- $min – die Minute zum Zeitpunkt der Vorhersage (z.B. 01)
- $sec – die Sekunde zum Zeitpunkt der Vorhersage (z.B. 59)

- **Messungen:** Die Messungen können auch im Back-End bearbeitet werden. Der administrative Benutzer hat hier allerdings keine Einschränkungen und kann alle Werte des Objekts auch nachträglich ändern.
-

3 Ausblick Februar 2013

Zum aktuellen Zeitpunkt (Februar 2013) hat die beschriebene Software den ersten lauffähigen Stand erreicht, in dem bereits viele der Hauptfunktionen umgesetzt sind. In Absprache mit den Projektpartnern werden sich sicherlich noch weitere Anforderungen ergeben. Folgende Punkte müssen aber auf jeden Fall noch bearbeitet werden:

- Der Bereich für die automatische Zuordnung der Messdaten zu Messungen fehlt noch vollständig. Hier ist das praktische Vorgehen der Operateure bei der Erfassung, sowie die Besonderheiten der verwendeten Geräte (Dateinamen, Verzeichnisstruktur) noch mit den Projektpartnern von der Hochschule Bonn-Rhein-Sieg abzustimmen.
- Spätestens für die grafische Darstellung der Messdaten, wahrscheinlich aber auch für die Analysesoftware der Projektpartner aus Fraunhofer FKIE, sind die Messdaten von verschiedenen geräte-spezifischen Formaten in ein einheitliches Format umzuwandeln. Hier ist ebenfalls mit der Hochschule Bonn-Rhein-Sieg abzustimmen, welche Formate die verwendeten Messgeräte schreiben.
- Die genaue Berechtigungsstruktur und die benötigten Benutzergruppen müssen noch abgestimmt werden.
- Die Auswahl des Fahrzeugherstellers und -modells sollte auch die Möglichkeit bieten, einen noch nicht in der Datenbank vorhandenen Hersteller bzw. Modell anzulegen.
- Die juristische Problematik der Datenerhebung und -speicherung im für das Projekt (und speziell für die beschriebene Software) benötigten Rahmen ist noch zu klären.

4 Stand zu Projektende

Von den offenen Punkten im Ausblick (Abschnitt 3) wurden noch erreicht:

- Die automatische Zuordnung der Messdaten zu Messungen wurde auf dem Abschlusstreffen vorgestellt. Erahrungen aus der Praxis fehlen noch.
- Ein Rollen-/Rechte-System wurde implementiert, konnte aber noch nicht abgestimmt werden- Die genaue Berechtigungsstruktur und die benötigten Benutzergruppen müssen noch abgestimmt werden.
- Die Auswahl des Fahrzeugherstellers und -modells bietet nun auch die Möglichkeit, einen noch nicht in der Datenbank vorhandenen Hersteller bzw. Modell anzulegen.

Offen bleibt weiterhin:

- Spätestens für die grafische Darstellung der Messdaten, wahrscheinlich aber auch für die Analysesoftware der Projektpartner aus Fraunhofer FKIE, sind die Messdaten von verschiedenen geräte-spezifischen Formaten in ein einheitliches Format umzuwandeln. Hier konnte nicht mehr mit den Projektpartnern abgestimmt werden, welche Formate die verwendeten Messgeräte schreiben.
- Die juristische Problematik der Datenerhebung und -speicherung im für das Projekt (und speziell für die beschriebene Software) benötigten Rahmen ist noch nicht geklärt..

Technische Universität Braunschweig | Gauß-IT-Zentrum
Hans-Sommer-Straße 65 | 38106 Braunschweig | Deutschland

VESPER Plus

Anlage A2

Dokument:	Plattformunabhängige 3D-Visualisierung mit WebGL
Zuwendungsempfänger:	Technische Universität Carolo-Wilhelmina zu Braunschweig - Gauß-IT-Zentrum
Authoren:	Alexey Beresnev
Projektleiter:	Norbert Schenk
FKZ:	13N11922

Inhaltsverzeichnis

1. Motivation

Dieses Dokument erläutert die Ergebnisse der Untersuchung der Einsatzmöglichkeiten mobiler Plattformen für eine 3D-Simulation im Rahmen des Projekts VESPER[Plus]. Ursprünglich sollten zwei mobile Plattformen untersucht werden: Microsoft XNA (für Windows Phone 7) und Google Android. Seit der Antragstellung hat sich die Situation aber stark verändert: Micorosoft hat die Entwicklung von XNA eingestellt[1] und mit WebGL hat sich eine neue plattformunabhängige 3D-Programmierumgebung etabliert, die neben Android auch auch auf den meisten anderen mobilen und klassischen Betriebssystemen zur Verfügung steht. Im folgenden wird die Technologie WebGL näher beschrieben, sowie die komplette Entwicklung eines Prototypen der 3D-Trainingsumgebung für die Arbeitspakete 4.3.1, 4.3.2 und 4.3.3 dokumentiert.

1.1. WebGL

WebGL (Web-based Graphics Library) ist eine Programmierschnittstelle der modernen Browser, die es Webseiten erlaubt, ohne zusätzliche Plug-ins (wie Adobe Flash oder Microsoft Silverlight) interaktive und hardwarebeschleunigte 3D-Grafik auszugeben. Die Schnittstelle basiert auf der OpenGL ES 2.0 Spezifikation[2] (Open GL for Embedded Systems) und wird ebenfalls von der non-profit Organisation Khronos Group[3] entwickelt, die über 100 namhafte Mitglieder (wie AMD, Intel, Nvidia, Google und Oracle) zählt.

1.2. Unterstützung

Die Abb. 1 zeigt eine mittlerweile fast flächendeckende Unterstützung für WebGL in den aktuellen Versionen der wichtigsten Browser, auch auf mobilen Plattformen. Mit Microsoft und Apple ziehen auch die beiden großen Hersteller der proprietären Browser im Bezug auf WebGL nach: die Version 11 des Microsoft Internet Explorers unterstützt WebGL (zumindest auf dem Desktop) vollständig und die nächste iOS Version 8 schaltet die Schnittstelle auch auf mobilen Geräten von Apple frei (inoffiziell war die Implementierung auch schon in früheren Versionen enthalten).

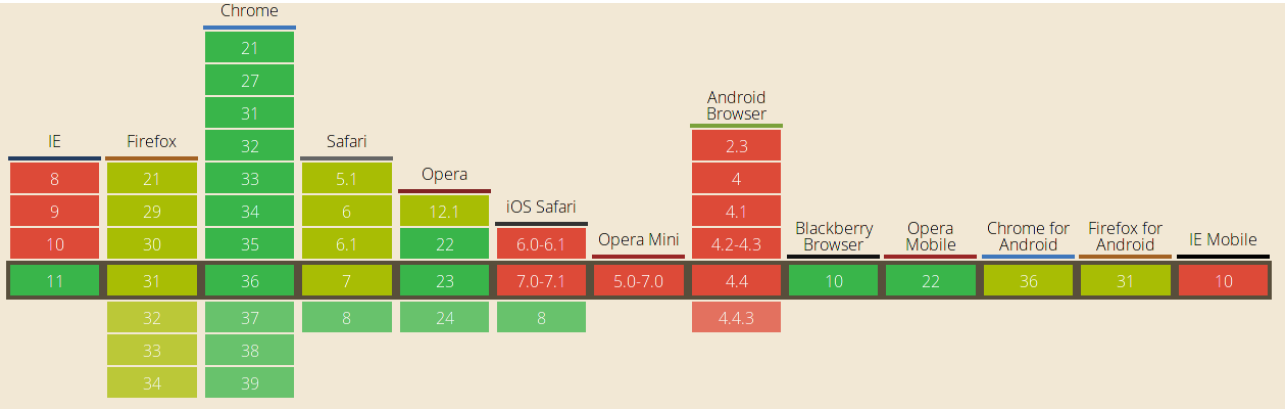

Abb. 1: WebGL Unterstützung in Browsern (Quelle: caniuse.com/webgl)

1 http://www.computerandvideogames.com/389018/microsoft-email-confirms-plan-to-cease-xna-support/
2 http://www.khronos.org/opengles/
3 http://www.khronos.org

Abb. 1 berücksichtigt allerdings nicht die tatsächliche Leistungsfähigkeit der Implementierung. In der Praxis ist sie zumindest auf mobilen Plattformen häufig noch nicht ausreichend, um komplexe 3D-Szenen mit akzeptabler Bildwiederholrate anzuzeigen. Bei dem aktuellen Tempo der Hardwareentwicklung (gerade im mobilen Segment) und der optimistischen Einstellung aller Hersteller zu WebGL, ist abzusehen, dass WebGL bald ähnlich performant wird, wie native native Grafik.

1.3. Alternativen

3D-Schnittstellen waren vor WebGL nativen (nicht Web-) Anwendungen vorbehalten. Hier gibt es zwei bedeutende Alternativen:

- OpenGL – eine offene, mit WebGL eng verwandte Spezifikation, die bis jetzt allerdings eher im professionellen Bereich relevant war, von Consumer-Grafikkarten aber schlechter unterstützt und von Computerspielen selten benutzt wurde.
- DirectX – eine 3D-Schnittstelle von Microsoft, die auch nur unter Windows und Xbox nutzbar ist, jedoch gut von Grafikkarten unterstützt und von den meisten Computerspielen benutzt wird.

OpenGL ist zwar an sich eine plattformunabhängige Schnittstelle, sie beschreibt aber nur die für 3D relevanten Funktionsaufrufe, um das Öffnen eines Fensters und die Kommunikation mit dem Betriebssytem, muss sich ein OpenGL-Programm aber selbst kümmern. Der dazu nötige Code ist aber so unterschiedlich, wie die Betriebssysteme selbst: iOS schreibt z.B. als Progammiersprache Objective-C vor, während ein Android-Programm zwingend in Java geschrieben sein muss und es handelt sich dabei um eine andere Java-Umgebung als die, die auf den meisten Desktop-Systemen installierbar ist. Letztlich führt kein Weg an der Pflege einer Softwareversion für jede Zielplattform vorbei und echte plattformunabhängige 3D-Schnittstellen wie WebGL existierten bislang nicht. Dabei ist die Anzahl unterschiedlicher Plattformen in den letzten fünf Jahren mit dem Siegeszug der mobilen Geräte stark gewachsen. Nachfolgend seien die wichtigsten mit ihrem aktuellen Marktanteil aufgelistet[4]:

Desktop

- Windows (91,68%)
- MacOS (6,64%)
- Linux (1,68%)

Mobil

- Android (41,58%)
- iOS (48,34%)
- Windows Phone (2,1%)

4 http://www.netmarketshare.com/

- Blackberry (1,14%)

1.4. Pro und Contra

Neben dem wichtigsten Vorteil von WebGL, nämlich der Möglichkeit, mit einer einzigen Codebasis, alle WebGL unterstützende Plattformen zu erreichen, gibt es noch viele andere:

Vorteile

- Plattformunabhängigkeit – bereits jetzt können alle im vorigen Absatz aufgelisteten Plattformen mit Ausnahme von Windows Phone mit einer einzigen Code- und Datenbasis unterstützt werden
- Zukunfssicherheit – auch Browser der noch nicht existierenden zukünftigen Betriebssysteme werden Web-Standards wie WebGL unterstützen. Andere 3D-Schnittstellen (wie z.B. 3dfx Glide) sind dagegen deutlich kurzlebiger.
- Neue Geräteklassen (wie Fernseher) sind teilweise auch schon WebGL tauglich.
- Einfache Verteilung – wie jede andere Web-Seite, kann eine WebGL-Anwendung sehr einfach erreicht und genutzt werden. Keine Datenträger, keine Installation, keine höheren Benutzerrechte sind notwendig. Auch keine Browser-Erweiterungen (wie Adobe Flash) müssen installiert sein.
- Client-Server-Kommunikation – weil WebGL-Anwendungen auch Web-Seiten sind, können sie sehr einfach Inhalte nachladen, Code aktualisieren, Daten zum Server schicken oder mit anderen Instanzen austauschen.
- GUI – die Gestaltung der zusätzlichen nicht- 3D-Benutzeroberfläche (Menüs, Formulare usw.) ist mit HTML-Mitteln am einfachsten und kann auch in 3D-Szenen integriert werden. Auch Video-Inhalte können nachgeladen und abgespielt werden.

Nachteile

Der größte Nachteil von WebGL ist sein jünges Alter. Es gibt noch wenige andere, für die es aber Lösungen entweder bereits jetzt gibt oder in nächster Zukunft zu erwarten sind.

- Kinderkrankheiten – WebGL ist eine komplexe Spezifikation und manche Implementierung sind entweder noch nicht vollständig oder zeigen noch unzureichende Performance
- Performance – Probleme mit der Performance sind momentan noch vor allem durch noch nicht optimierte Implementierungen in Browsern, Betriebssystemen und Grafiktreibern begründet. Weil WebGL-Anwendungen aber mit einer interpretierten Skriptsprache JavaScript programmiert werden, sind sie auch prinzipiell etwas langsamer als native Anwendungen. Durch immer bessere JIT-Compiler und stetigen Zuwachs an Rechenleistung, spielt das allerdings keine große Rolle.
- Komplexität – als OpenGL-Abkömling ist auch WebGL eine Schnittstelle, die auf einem niedrigen Level mit der Grafikhardware kommuniziert und ziemlich umständlich zu

programmieren ist. So erfordern selbst primitive Operationen wie Drehung die Berechnung einer Transformationsmatrix. Andererseits lässt gerade dieser wenig abstrahierter Zugriff auf die Grafikhardware viel Raum für eigene Optimierungen und für alltägliche WebGL-Aufgaben gibt es inzwischen mächtige Libraries wie Threejs, auf die in Abschnitt 2.1. noch näher eingegangen wird.

2. Logik

Die 3D-Umgebung im Projekt VESPER[Plus] wurde nicht direkt mit WebGL entwickelt, sondern mit der Open-Source JavaScript-Library Three.js[5]. Diese bietet im Vergleich zum reinen WebGL eine viel höhere Abstraktionsebene. Außerdem müssen die Objekte in der 3D-Umgebung physikalische Gesetze einhalten. Darauf achtet die offene Physik-Simulation für JavaScript – Physi.js[6]. Im Folgenden werden diese zwei Libraries näher beschrieben.

2.1. Graphische Ausgabe – Three.js

WebGL arbeitet sehr nah an der Grafikhardware und ermöglicht der Web-Anwendung aus dem Browser heraus durch sogenannte Shader sogar Code direkt auf der GPU der Grafikkarte auszuführen. Allerdings müssen selbst einfachste Objekte wie Spheren durch relativ komplexe Vektor- und Matrix-Berechnungen erzeugt werden, weil die Grafikhardware nur auf dieser Ebene arbeitet. Diese Hardwarenähe hat auch Vorteile aber eine komplexe 3D-Szene mit so primitiven Mitteln zu erzeugen und zu steuern wäre unnötig umständlich. Deshalb kommt in der 3D-Umgebung zusätzlich die JavaScript-Library Three.js[5] zum Einsatz. Sie stellt einen objektorientierten Szenegraph bereit, dem alle Komponenten der 3D-Welt hierarchisch gegliedert hinzugefügt werden. Three.js ermöglicht einfache Erzeugung graphischer Primitive aber auch das Laden kompletter 3D-Modelle aus einiger Formate. Sie kümmert sich um Materiale und Texturen, Kameras, Lichtquellen und vieles mehr.

Für Three.js-Programme üblich ist folgende Grundstruktur:

```
var camera, scene, renderer;

init();
render();
```

Listing 1: Grundstruktur

Globale Objekte (camera, scene, renderer), auf die von vielen Stellen aus zugegriffen wird, sind im obersten Context definiert. Die Funktion init() wird ein mal aufgerufen und initialisiert die Szene. render() wird anschließend aufgerufen und registriert sich selbst bei jedem Aufruf für das Anpassen der Szene vor dem nächsten Animationsschritt. Ihr Inhalt sieht schematisch aus wie in Listing 2. requestAnimationFrame ist dabei eine neuere Methode der modernen Browser, welche die angegebene JavaScript-Funktion (in diesem Fall animate) genau in dem Moment aufruft, wenn der Seiteninhalt neu gezeichnet werden kann. Sie vermeidet ein zu häufiges Neuzeichnen, indem sie unter anderem die Bildwiederholrate des Monitors in Betracht zieht und unterbindet das Neuzeichnen ganz wenn das Browser-Fenster oder Tab gerade nicht sichtbar ist. Normalerweise aber wird render() etwa 60 mal in der Sekunde aufgerufen, um flüssige Animationen zu ermöglichen. Das eigentliche Neuzeichnen findet dann in der Methode des Renderer-Objekts render() statt. Als Parameter übergeben wird ihr eine Referenz auf den Szenegraph (also die

5 http://threejs.org/
6 http://chandlerprall.github.io/Physijs/

komplette 3D-Szene mit allen Objekten) und eine Kamera, die darauf gerichtet ist (eine Web-Anwendung kann die Szene auch aus mehreren Perspektiven gleichzeitig anzeigen). Vor dem render()-Aufruf sollten aber alle Szeneanpassungen durchgeführt werden, die zum nächsten Bildwechsel angefallen sind: z.B. können Objekte modifiziert oder ihre Position verändert werden, um eine Bewegung im Raum darzustellen. Damit Three.js automatisch die Animationen anwendet, die zu jedem Szene-Objekt hinterlegt (und aus dem Modell geladen) werden können, wird in der render()-Funktion auch die Methode update() des globalen AnimationHandler aufgerufen. Als Parameter erhält sie die Anzahl der Animationsschritte, die sie anwenden soll.

```
function render() {

    requestAnimationFrame( animate );

// hier können Objekte verändert
// und Koordinaten angepasst werden

        THREE.AnimationHandler.update(1);
    renderer.render( scene, camera );

}
```

Listing 2: Die Funktion render()

Ein komplettes Beispiel der init()-Funktion ist in Listing 3 dargestellt. Hier wird als erstes ein neuer WebGL-Renderer erzeugt (wie bereits angesprochen kennt Three.js noch andere Renderer z.B. Canvas-Renderer). Die Größe der Zeichenfläche wird ihm bekanntgegeben. Sie ergibt sich hier aus der Fenstergröße. Anschließend wird das HTML-Element des Renderers in die Seite ingefügt.

Als nächstes wird eine perspektivische Kamera erzeugt und positioniert. Neben dem Kameratyp (die perspektivische Kamera verkleinert Objekte mit zunehmender Entfernung) wählt man den Öffnungswinkel (70°), Seitenverhältnis und die Entfernung zur vorderen (1) und hinteren (1000) Clipping-Ebene. Nur Objekte, die sich aus der Sicht der Kamera zwischen den Clipping-Ebenen befinden werden dargestellt, Objekte die zu nah oder zu weit sind fallen heraus.

Danach wird die Szene selbst (also die Wurzel der Objekt-Hierarchie) und ein einfaches geometrisches Objekt (eine Box) angelegt. Für die Box wird aus einer Bild-Datei die Textur geladen. Die Textur beschreibt zwar die Farbe für jeden Punkt der Objektoberfläche, zusätzlich ist aber noch das Material der Oberfläche wichtig – es kann nämlich mehr oder weniger Licht reflektieren, brechen oder transparent sein. Deshalb wird die Textur nicht direkt der Box zugewiesen sondern einem Material, welches dann zusammen mit der Geometrie der Box ein fertiges Objekt (*mesh*) ergibt. Dieses wird dann, genau wie die Kamera mit der methode add() der Szene hinzugefügt.

Als Ergebnis erscheint im Browser ein mit der Textur bezogener Kubus, der sich auch um die eigene Achse drehen wird, wenn man in der render()-Funktion die Zeile mesh.rotation.y += 0.01; einfügt.

```
function init() {
    renderer = new THREE.WebGLRenderer();
    renderer.setSize( window.innerWidth, window.innerHeight );
    document.body.appendChild( renderer.domElement );


    camera = new THREE.PerspectiveCamera( 70, window.innerWidth / window.innerHeight, 1, 1000 );
    camera.position.z = 400;

    scene = new THREE.Scene();

    var geometry = new THREE.BoxGeometry( 200, 200, 200 );

    var texture = THREE.ImageUtils.loadTexture( 'textures/crate.gif' );

    var material = new THREE.MeshBasicMaterial( { map: texture } );

    mesh = new THREE.Mesh( geometry, material );
    scene.add( mesh );
}
```

Listing 3: Die Funktion init()

2.2. Physikalische Simulation – Physi.js

Wie im letzten Abschnitt beschrieben, erlaubt Three.js mit relativ wenig Code 3D-Objekte zu erzeugen und zu bewegen. Dabei setzt Three.js der Bewegungsfreiheit der Objekte keine Grenzen: beliebig viele Objekte können sich auf einem einzigen Punkt befinden, es gibt keine Erde und keine Erdanziehung, da es sich bei Three.js und WebGL um reine Grafikschnittstellen handelt. Für viele Anwendungsfälle sind sie auch vollkommen ausreichend, bei Spielen oder 3D-Simulationen, ist aber die Einhaltung einer ganzen Reihe allgemeiner physikalischer Gesetze notwendig, wofür es eigene (zum Teil sehr komplexe) Software- und sogar Hardwarelösungen gibt – die sogenannten Physics Engines. Wie der Rest der WebGL-Anwendung muss auch seine Physik-Simulation im Browser und in JavaScript laufen. Trotz des jungen Alters von Three.js und WebGL gibt es dafür bereits eine passende und ebenfalls offene JavaScript-Library Physi.js. Physi.js erweitert das Objektmodell von Three.js um physikalische Eigenschaften wie Masse, Impuls oder Materialreibung. Die komplexen Berechnungen der Physik-Simulation werden dabei in einem separaten Thread parallel zur grafischen Ausgabe durchgeführt.

Mit Physi.js im Einsatz wird die Szene wie in Listing 4 initialisiert: statt der normalen THREE.Scene wird Physijs.Scene instanziiert und die Richtung der Gravitation wird mit einem Vektor festgelegt.

```
scene = new Physijs.Scene;
scene.setGravity(new THREE.Vector3( 0, -30, 0 ));
```

Listing 4: Die Initialisierung der Szene mit Physi.js

Auch die Materiale werden für Physi.js etwas anders initialisiert (Listing 5). Zusätzlich zum normalen Three.js-Material muss noch die Reibung (0,4) und die Stoßzahl[7] (0,6) angegeben werden.

7 http://de.wikipedia.org/wiki/Sto%C3%9F_(Physik)#Realer_Sto.C3.9F

```
box_material = Physijs.createMaterial(
    new THREE.MeshLambertMaterial({ map: THREE.ImageUtils.loadTexture(
'models/plywood.jpg' ) }),
    .4, // friction
    .6 // restitution
);
```

Listing 5: Material mit Physi.js

Schließlich müssen auch die Objekte der Szene in physikalische Objekte umgewandelt werden. Eine rechteckige Box wird wie in Listing 6 initialisiert.

```
var box = new Physijs.BoxMesh(
    new THREE.CubeGeometry( 200, 200, 200 ),
    box_material
);
```

Listing 6: Box mit Physi.js

Für fast jede graphische Form, die Three.js kennt, hat auch Physi.js eine Entsprechung. Allerdings muss nicht unbedingt jedes graphische Objekt in ein physisches umgewandelt werden. Zum einen sind nicht alle Objekte für die Physik-Simulation relevant und normale Three.js-Objekte können mit Physi.js-Objekten problemlos in einer Szene koexistieren, auf sie wirken nur keine Kräfte und sie stellen für „physische“ Objekte keine Hindernisse dar. Zum anderen ist die Kollisionsberechnung der Physik-Simulation umso komplizierter, je mehr und je komplexere Formen in der Szene vorhanden sind. Es ist daher meistens empfehlenswert die physischen Objekte etwas gröber zu modellieren als ihre graphischen Repräsentationen. So lässt sich aus einem beliebig komplexen Model sehr schnell die sogenannte Bounding Box berechnen, die für eine grobe Kollisionserkennung benutzt werden kann.

Mit den beschriebenen Modifikationen wirkt in der 3D-Welt jetzt die Gravitation und die Box wird bis zur Unendlichkeit (für JavaScript-Zahlen durchaus erreichbar) in Richtung des Gravitationsvektors beschleunigen. Es muss also die Erdoberfläche der Szene hinzugefügt werden. Im einfachsten Fall ist es eine Ebene (Physijs.PlaneMesh) oder Physijs.Heightfield für unebenes Gelände, das durch eine Höhenmatrix repräsentiert ist. Um die Gestaltung des Geländes für das VESPER-Projekt auch ohne Programmierkenntnisse zu ermöglichen, wird es komplett aus einem separaten Modell geladen, wobei für alle Objekte, deren Namen im Modell mit „Box_“ oder „Plane_“ beginnen, automatisch unbewegliche Physik-Objekte des entsprechenden Typs angelegt werden (siehe auch 3.1)

2.3. Fahrzeugphysik

Eine wichtige Klasse von Objekten, insbesondere bei einer Hafensimulation, stellen Fahrzeuge dar. Physi.js bietet zwar neben Impuls-, Kollisions- und Abprallberechnungen auch Einschränkungen der Bewegungsfreiheitsgrade wie Scharniere oder Schienensysteme, für die Simulation realistischer Fahrzeugbewegung reichen diese einfacheren allgemeinen Physik-Regeln nicht aus, weil unter anderem auch Federung und Fliehkräfte simuliert werden müssen. Zum Glück verfügt Physi.js

bereits über eine brauchbare Simulation der Fahrzeug-Physik. Neben ein paar Parametern wie Radstand oder Federungshub und -steifheit benötigt das durch die Vehicle-Klasse repräsentierte Objekt natürlich auch das graphische Modell des Fahrzeugs und seiner Räder. Dieses wird für die Hafensimulation aus einer separaten Modell-Datei geladen, die ebenalls in Blender erstellt werden kann und auf der obersten Ebene der Objekt-Hierarchie ein Objekt namens Car und eins namens Wheel haben muss. Bei dem (einzigen) Rad muss es sich um ein linkes Rad handeln, welches dann für die Rechte Seite automatisch gespiegelt wird.

Die Physijs.Vehicle Klasse wird durch die Vesper.Vehicle Klasse abgeleitet, welche das projektspezifische Parsing der Modell-Datei implementiert. Diese kann neben dem Fahrzeug selbst und seinem Rad auch Animationen enthalten, die sich öffnende und schließende Türen, sowie Kofferraumklappe zeigen. Dabei muss es sich um sogenannte Key-Frame-Animationen handeln. Sie bestehen mehreren Zeitslots, die jeweils Koordinaten-, Rotations- oder Skalierungsänderungen für bestimmte Objektteile beschreiben. Wenn diese Animationen bei dem globalen AnimationHandler-Objekt von Three.js registriert sind und aktiviert werden, wendet der Animation-Handler die Änderungen der Zeitslots nacheinander mit jedem Animationsschritt an. Die Animationsschritte werden in der render()-Funktion fortgezählt. Damit die Tür-Animationen korrekt zugeordnet werden können, müssen sie im Modell folgende Namen tragen:

(0) "animation_Trunk_lid" – Kofferraumklappe

(1) "animation_Door_front_left" – Tür vorne links

(2) "animation_Door_front_right" – Tür vorne richts

(3) "animation_Door_back_left" – Tür hinten links

(4) "animation_Door_back_right" – Tür hinten rechts

Bei Fahrzeugen ohne die hinteren Türen, können entsprechende Animationen natürlich fehlen. Aktiviert werden die Tür-Animationen durch die Methode toggleDoor() der Vesper.Vehicle-Klasse. Sie erhält als Parameter die Zahl in Klammern aus der oberen Auflistung.

Die Vesper.Vehicle-Klasse übernimmt auch die (Tastatur-)Steuerung des Fahrzeugs. Diese Funktionalität könnte später leicht in eine Fernsteuerung durch andere Teilnehmer (z.B. mit WebSockets[8]) oder durch ein zuvor festgelegtes Skript ausgebaut werden. In der Methode update(), die in jeder Physi.js-Klasse enthalten ist und durch die Physik-Engine bei jedem Simulationsschritt aufgerufen wird, fragt die Vehicle-Klasse den Zustand der festgelegten Tasten ab und steuert das Fahrzeug durch die Physijs.Vehicle-Methoden applyEngineForce(), setBrake() und setSteering().

8 http://de.wikipedia.org/wiki/WebSockets

3. Modelle

Zwar bietet Three.js im Gegensatz zu WebGL bequeme Möglichkeiten, geometrische 3D-Objekte zu erzeugen, daraus eine realistische und detailreiche Darstellung zu kombinieren wäre aber ineffizient und aufwendig. Außerdem arbeiten in einem Projekt meistens auch Grafiker und Gestalter, denen Code-Änderungen nicht zuzumuten sind. Modelle werden daher in Modeling-Software geformt, texturiert und animiert. Neben kommerzieller Produkte wie Autodesk Maya und 3DS Max stellt die quelloffene und für alle Betriebssysteme verfügbare Software Blender[9] eine durchaus vergleichbare Alternative dar.

3.1. Blender

Wie der Szene-Graph von Three.js ordnet auch Blender alle Elemente einer 3D-Szene in einer Objekt-Hierarchie an. Zwar könnten im Prinzip auch alle Vektoren in einem einzigen Objekt enthalten sein, macht es für die saubere Modellierung Sinn, größere Objekte aus kleineren Details zusammenzusetzen und zusammenhängende Objekte zu Gruppen zu kombinieren.

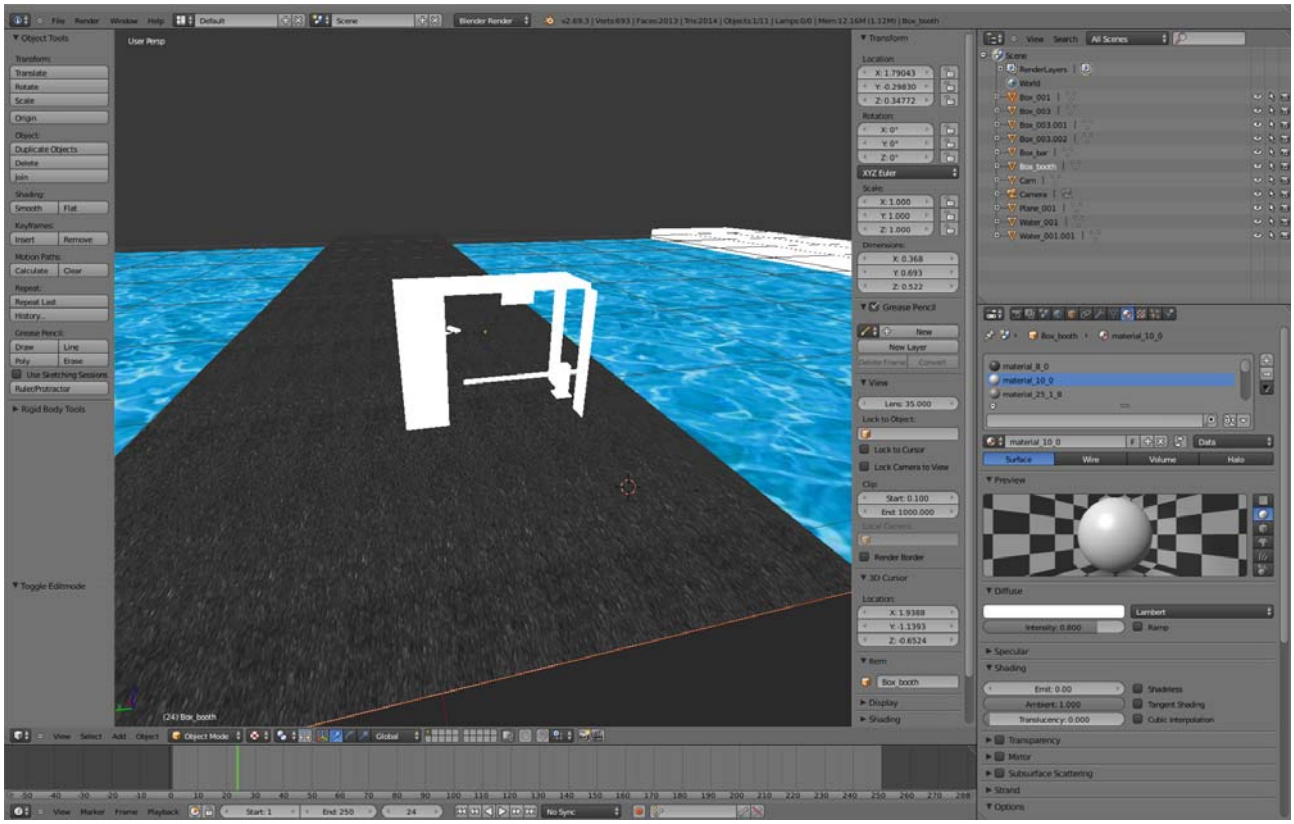

Abb. 2: Das Hafenmodell in Blender

Auf der obersten Hierarchieebene hat eine Blender-Szene standardmäßig eine Kamera und eine sonnenartige (also stark gestreute) Lichtquelle. Ohne sie stellt der Blender-eigene Renderer in der Vorschau natürlich nur ein schwarzes Bild dar. Für den Import in die Hafensimulation sind die in der Szene definierten Kameras und Lichtquellen aber vorerst nicht relevant und werden einfach

9 http://www.blender.org/

ignoriert. Der Grund dafür ist, dass bei der Simulation die Kamera keine feste Position hat und stattdessen über Maus und Tastatur gesteuert wird, während das Licht als Umgebungslicht (THREE.HemisphereLight) dargestellt wird.

Blender erlaubt nicht nur jedes Objekt oder Gruppe sinnvoll zu benennen, sondern darüber hinaus auch beliebige Attribute (als Name-Wert-Paare) anzuhängen. Diese Speichermöglichkeit ist auch notwendig, um Zusatzinformationen an die Physik-Engine zu übergeben, etwa ob es sich bei einem Objekt um Wasser, Gebäude oder einen beweglichen Gegenstand handelt. Leider werden die Zusatzattribute von dem derzeitigen Collada-Exporter ignoriert, was im Prinzip relativ einfach zu ändern wäre, zumal das XML-basierte Collada-Format ebenfalls Platz für Zusatzinformationen vorsieht. Vorerst war es jedoch ausreichend, eine Namenskonvention für Gelände-Elemente festzulegen und die Objektnamen nach dem Laden zu untersuchen.

Blender speichert das komplette Projekt, an dem ein Modellierer arbeitet in einer einzigen .blend-Datei. Dabei handelt es sich um ein eigenes Format, das zwar alle Daten und Einstellungen enthält, aber eher nicht als Modell-Austauschformat geeignet ist, weil es zum einen einige nur für Blender relevante Daten enthält und zum anderen nicht von anderen Programmen unterstützt wird.

3.2. Das COLLADA-Format

Three.js verfügt über ein eigenes Modell-Format, was auf JSON (JavaScript Object Notation)[10] basiert und durch ein Export/Import-Plugin für Blender generiert werden kann. Die Tests haben jedoch gezeigt, dass dieser Weg noch nicht sehr zuverlässig funktioniert: das Three.js-Plugin für Blender entwickelt sich langsamer als Blender selbst und ist mit neueren Versionen noch nicht kompatibel. Außerdem exportierte es für das (relativ komplexe) Fahrzeug-Modell riesige Dateien (über 50Mb groß). Wesentlich besser funktionierte der in Blender bereits enthaltene Exporter für das COLLADA-Format[11], für das es auf der Three.js-Seite auch einen Loader gibt.

Das offene und erweiterbare 3D-Datenformat COLLADA hat sich neben einiger proprietären Formate der verbreiteten 3D-Software für den Austausch, Speicherung und Verteilung der Modelle etabliert. Ursprünglich von Sony entwickelt wird es inzwischen ebenfalls von der Khronos-Group verwaltet. Das Format ist XML-basiert und wird von sehr vielen 3D-Anwendungen unterstützt. Durch die XML-Struktur lässt sich das Format beliebig erweitern, doch der Standardumfang ist in der Lage eine komplette 3D-Szene, und alle wichtigen Aspekte der 3D-Daten beschreiben:

- Geometrieen
- Materiale und Texturen (meistens in separaten Bild-Dateien)
- Animationen (Morph- und Keyframes)
- Kameras und Lichtquellen
- Physik-Daten

Schon seit der Version 1.4 des Standards aus dem Jahr 2008 ist auch eine sehr umfassende

10 http://de.wikipedia.org/wiki/JSON
11 http://collada.org/

Definition der Physik-Eigenschaften in COLLADA-Dateien möglich. Auch in den aktuellen Blender-Versionen können sie angegeben werden. Lediglich der COLLADA-Exporter von Blender ignoriert diese Informationen bislang. Auch der COLLADA-Loader von Three.js müsste entsprechend angepasst werden.

4. Fazit

Der im VESPER^{Plus}-Projekt verfolgte Ansatz, eine plattformunabhängige 3D-Simulation basierend auf dem modernen WebGL-Standard zu entwickeln, statt für wenige ausgewählte mobile Plattformen mehrere untereinander inkompatible Software-Versionen zu erstellen, hat sich bewährt. Zwar ist die Technologie noch sehr neu und die verwendeten Werkzeuge und Libraries teilweise nicht ganz ausgereift, jedoch konnte gezeigt werden, dass bereits heute damit möglich ist, interaktive 3D-Szenen plattformübergreifend in modernen Browsern darzustellen und Modelle dafür in frei verfügbarer Software zu entwickeln.