FH JOANNEUM Gesellschaft mbH

WebGL™ – 3D Canvas Programmierung

Geschichte, Entwicklung und Zukunft

Bachelorarbeit 1

zur Erlangung des akademischen Grades

Bachelor of Science in Engineering (BSc)

eingereicht am 30. April 2012

Fachhochschul-Studiengang Internettechnik

Betreuer: DI (FH) MSc Mathias Knoll, MSc

eingereicht von: Richard Fussenegger

Personenkennzahl: 0910418059

April 2012

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Wolfurt, 30. April 2012; Fussenegger Richard

Inhaltsverzeichnis

Abbildungsverzeichnis

Quelltextverzeichnis

Zusammenfassung

In der vorliegenden Bachelorarbeit wird auf die Entwicklung, Geschichte und eine mögliche Zukunft von WebGL (World Wide Web Graphics Library) eingegangen. WebGL ist eine lizenzfreie 3D-Grafik-Programmierschnittstelle (3D-API) für Webbrowser, die im HTML5-Canvas-Kontext JavaScript erweitert und es ermöglicht 3D-Grafiken ohne zusätzliche Plug-Ins in Webbrowsern darzustellen.

Zu Beginn wird auf die Geschichte von 3D im World Wide Web eingegangen. Daraufhin wird die Entstehung des WebGL-Standards erläutert und die darunterliegenden Technologien kurz vorgestellt. Im Anschluss werden verschiedene Bibliotheken bzw. Frameworks – sogenannte WebGL-Wrapper – vorgestellt. In den darauf folgenden Kapiteln werden zwei mögliche zukünftige deklarative Standards vorgestellt und erläutert worum es sich hierbei handelt. Vor dem abschließenden Resümee werden die wichtigsten WebGL-Konkurrenten kurz vorgestellt.

Am Schluss folgt ein Resümee, in dem eine mögliche Zukunft für WebGL beschrieben wird, basierend auf den persönlichen Eindrücken und den Erfahrungen des Autors im Umgang mit WebGL.

Abstract

This present bachelor thesis deals with the development, history and possible future of WebGL (World Wide Web Graphics Library). WebGL is a royalty free 3D graphics application programming interface (3D-API) for web browser. It extends JavaScript in HTML5 canvas context and enables web browser to display 3D graphics without additional plug-ins.

At the beginning the history of 3D in the World Wide Web is described. Subsequently, the emergence of the WebGL standard will be elucidated, and the underlying technology briefly envisaged. Following various libraries and frameworks—so called WebGL wrapper—will be annotated. In subsequent chapters, two possible future declarative standards are presented and further explained. Before the final summary the main competitors of WebGL are briefly introduced.

At the end is the resume, in which a possible future of WebGL will be described, based on the personal impressions and experiences of the author in dealing with WebGL.

1. Einleitung

Das World Wide Web (WWW, oder auch Web) rückt in den letzten Jahren immer zentraler in den Mittelpunkt vieler Lebensbereiche. So ist es kaum verwunderlich, dass verschiedene Unternehmen es anstreben auch die 3D-Computergrafik im Web salonfähig zu machen, bietet doch die Auslieferung von kompletten Videospielen oder anderen Anwendungen über das Web signifikante Vorteile gegenüber *normalen* Programmen. Selbst Microsoft wird im kommenden Windows 8 Metro-Anwendungen mit Web-Technologien umsetzbar machen (vgl. Schwichtenberg, 2011).

Das steigende Interesse führt natürlich dazu, dass es viele Initiativen gibt, die eben diese Lücken zu füllen versuchen. Adobe ist hier das wohl bekannteste Unternehmen, hat sich doch die Flash-Technologie bereits vor langer Zeit als Quasi-Standard für sogenannte Browser-Games durchgesetzt. Doch das Web soll frei von proprietären Standards bleiben, da sind sich alle Beteiligten einig. Entsprechend wurde von Mozilla die Idee geboren einen freien Standard zur Realisierung von 3D-Grafiken im Web zu initiieren.

Das Resultat der Mozilla-Initiative liegt seit 3. März 2011 in der ersten Version vor. Auf den folgenden Seiten wird der WebGL-Standard auf Geschichte, Entwicklung und eine mögliche Zukunft hin durchleuchtet.

1.1 Ziel dieser Arbeit

Der Arbeit liegt zu Grunde, dem Leser / der Leserin einen Überblick über die Geschichte und Entwicklung von 3D-Grafiken im Web und im speziellen WebGL zu verschaffen. Ferner sollen die Grundlagen der WebGL-Technologie erläutert werden. Des Weiteren werden verschiedene Frameworks, deklarative Ansätze und die Konkurrenz vorgestellt. Der Autor wird, basierend auf eigenen Erfahrungen im Umgang mit WebGL, eine mögliche Zukunft des Standards im Web beschreiben. Die Be-

trachter / die Betrachterinnen sollten nach Lesen dieser Arbeit einen Überblick über 3D im Web besitzen, im Speziellen über WebGL.

1.2 Aufgabenstellung

Die Aufgabenstellung umfasst die Recherche, Zusammenfassung und Evaluierung von relevanten Artikeln, Tutorials, Standards und Technologien.

1.3 Abgrenzung

Eine klare Abgrenzung ist darin zu finden, dass es nicht gilt den Betrachtern / Betrachterinnen die WebGL- oder JavaScript-Programmierung selbst beizubringen.

Diese Arbeit erhebt keinerlei Anspruch darauf, alle relevanten Artikel, Tutorials, Standards oder Technologien, noch den kompletten Funktionsumfang von WebGL oder JavaScript zu recherchieren, zusammenzufassen oder zu evaluieren.

Des Weiteren werden nur die zum Zeitpunkt aktuellen Fassungen und Versionen von Artikeln, Tutorials, Standards und Technologien recherchiert, zusammengefasst und evaluiert. Die daraus resultierenden Ergebnisse lassen sich nicht zwangsläufig auf frühere oder zukünftige Artikel, Tutorials, Standards und Technologien umlegen.

Wichtig für die Abgrenzung ist, dass keinerlei Programmierkenntnisse als Teil dieser Arbeit den Betrachtern / Betrachterinnen vermittelt werden sollen. Im Gegenteil, im Verlauf dieser Arbeit wird davon ausgegangen, dass die Betrachter / Betrachterinnen ein grundlegendes Verständnis besitzen.

1.4 Aufbau und Struktur der Arbeit

Wie aus dem Titel *„WebGL™ – 3D Canvas Programmierung – Geschichte, Entwicklung und Zukunft“* hervorgeht, liegt der Fokus auf den geschichtlichen Hintergründen der WebGL-Technologie, respektive 3D-Technologien, im Web.

2. Die Dritte-Dimension im Web

Die Idee 3D-Computergrafiken, oder die Darstellung von 3D-Szenen, mit dem Web zu verbinden ist bereits fast so alt wie das WWW selbst. Zu einer Zeit in der das Web – so wie wir es heute kennen – seinen Siegeszug in unsere Haushalte noch nicht einmal richtig begonnen hatte, verfasste Dave Raggett den Beitrag „*Extending WWW to support Platform Independent Virtual Reality*“ (vgl. Raggett, 1994).

2.1 Virtual Reality Modeling Language (VRML)

In diesem Beitrag nannte Raggett den Begriff VRML (Virtual Reality Modeling Language) – er nannte es in seinem Dokument noch Virtual Reality *Markup* Language – zum ersten Mal und rief Interessierte dazu auf sich mit ihm in Kontakt zu setzen. Diesen Beitrag reichte er zur „*The First International WWW Conference*“[1] ein. Daraufhin wurde der „*Virtual Reality Workshop*“ abgehalten, der reges Interesse beim Publikum weckte, dies führte wiederum dazu, dass die *www-vrml mailing list* ins Leben gerufen wurde (vgl. Pesce & Behlendorf, 1995) (vgl. Thomas, 1994).

Zu dieser Zeit stand Raggett bereits mit Mark D. Pesce in Kontakt. Pesce war zusammen mit Anthony S. Parisi und Gavin Bell die treibende Kraft hinter VRML, sie gründeten auch die VRML Architecture Group (VAG), unter der Leitung von Pesce, und moderierten die *www-vrml mailing list*.

Das erste öffentliche Forum wurde noch im selben Jahr abgehalten, auf der SIGGRAPH 94 (Special Interest Group on Graphics and Interactive Techniques). Insgesamt 17 Personen diskutierten wie VRML auszusehen hat und einigten sich darauf noch bis Ende des Jahres einen Standard samt gemeinfreiem Webbrowser und Demo zu entwickeln (vgl. Pesce, 1994).

Auf dem WWW94 VRML BOF (BOF steht hier für „Birds-Of-Feather-Session“, ein informelles Treffen Gleichgesinnter bei einer Konferenz) in Chicago präsentierten

[1] http://www94.web.cern.ch/WWW94/ – Offizielle Website zur WWW94

Pesce und Parisi zusammen mit Peter Kennard das „Labyrinth“-System, ein Vorschlag für 3D-Navigation, ein Repräsentationsschema und generell 3D-Computergrafiken inklusive virtueller Realitäten (vgl. Pesce, et al., 1994) (vgl. Pesce, 1994).

Basierend auf dem Open Inventor Format von Silicon Graphics International (SGI) wurde auf der „*The Second International WWW Conference*“ bereits der erste Entwurf für einen VRML-Standard vorgestellt. Bis Januar 1996 wurde dieser weiter verfeinert und finalisiert (vgl. Bell, et al., 1996). Nun konnte Pesce weitere Größen der Industrie – namentlich Microsoft, Silicon Graphics, Netscape, Sun Microsystems und Sony – von dem Projekt überzeugen und 1997 kam es zur ISO/IEC Zertifizierung, heute bekannt als VRML97 (vgl. Web3D Konsortium, 1997).

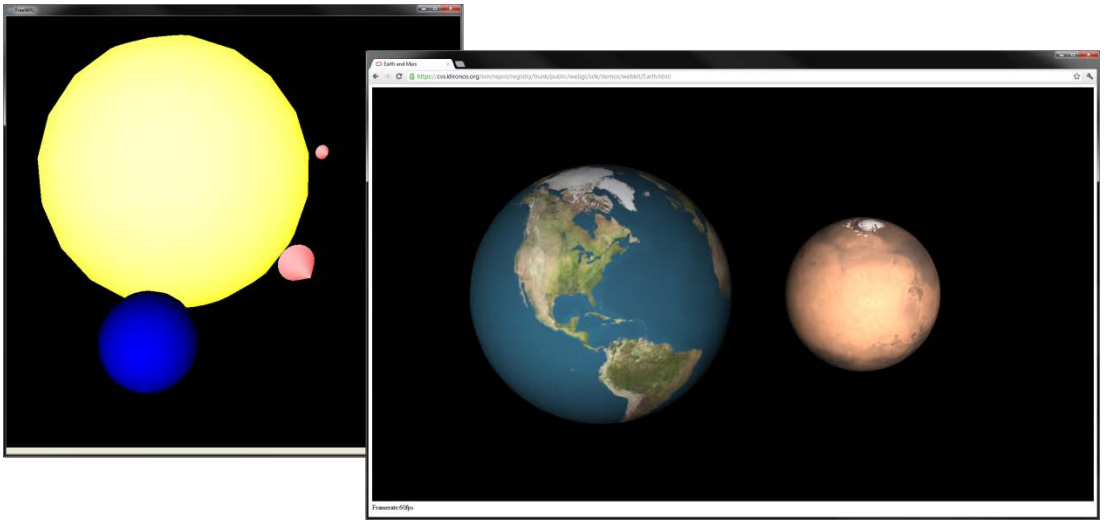

Abbildung 1 *– 3D im Web entwickelt sich, links das Sonnensystem in VRML97 und rechts eine WebGL-Sonnensystem-Demo der Khronos Group.*
(eigene Darstellung[2])

In einer **.wrl* Datei waren immer die Beschreibungen einer VRML-Szene untergebracht. Modelle und Texturen wurden über Uniform Resource Locators (URLs) referenziert und von einem VRML-Viewer automatisch nachgeladen. Die VRML-

[2] Das VRML-Sonnensystem ist gemeinfrei und stammt von der TECFA (Universität in Genf), online verfügbar unter http://tecfa.unige.ch/guides/vrml/examples/inter/. Zur Darstellung der VRML-Demo kam FreeWRL zum Einsatz, online verfügbar unter http://freewrl.sourceforge.net/.
Das WebGL-Sonnensystem ist öffentlich zugänglich über das WebGL-Demo-Repository der Khronos Group, online verfügbar unter http://www.khronos.org/webgl/wiki/Demo_Repository.

Viewer gab und gibt es als eigenständige Applikationen oder als Plug-In für Webbrowser. Für Entwickler / Entwicklerinnen war es also nicht einfach eine Kommunikation zwischen der Webseite und der 3D-Anwendung herzustellen. Hinzu kam der Umstand, dass die Rechner in der Regel damals keinen dezidierten Grafikprozessor besaßen (vgl. Völkl, 2012).

Mangelnder Hardware-Support, externe Applikationen bzw. Plug-Ins, erschwerte Integration in bestehende Websites und langsame Internetverbindungen bei den Endbenutzern führte in weiterer Folge dazu, dass VRML sich kommerziell nicht durchsetzen konnte. Es fehlte auch eine richtige „*Killer-Applikation*“, also eine Anwendung die jeder haben wollte. Durchschnittsanwender hatten keine Gründe sich mit dieser Thematik weiter auseinander zu setzen (vgl.ebd.).

2.2 Chromeffects

Microsoft war von Beginn an Mitglied im VRML-Konsortium, jedoch hatte der schlechte kommerzielle Erfolg dazu geführt, dass das Unternehmen sein Vertrauen in den Standard verlor (vbgl. Chalmers, 1998). Anfang 1998 kündigte Microsoft an, eine eigene Technologie bis zum Jahresanfang 1999 gemeinsam mit Windows 98 und Windows NT 5.0 (Windows New Technology) auszuliefern. Diese neue Technologie sollte Windows-Nutzern /-Nuterzinnen den Zugang zu 3D-Inhalten sowohl im Web als auch als eigenständige Applikation näher bringen. Die Technologie baue direkt auf der eigenen DirectX-API auf, Entwickler / Entwicklerinnen können also unter Verwendung von einfacher Hypertext Markup Language (HTML) unmittelbar imposante 3D-Szenen entwerfen, ohne direkt mit den DirectX-APIs kommunizieren zu müssen. Der Arbeitstitel lautete „Chrome“ (vgl. Lash & Kanellos, 1998) und wurde bei der finalen Ankündigung in „Chromeffects“ umgetauft (vgl. Weston, 1998).

Chromeffects war laut Microsoft nicht als VRML-Killer geplant, doch eine Verschmelzung der beiden Technologien zu einem gemeinsamen, neuen Standard könne man sich vorstellen. Microsoft kritisierte unter anderem die schlechte Integration von VRML in bestehende Web-Standards, insbesondere HTML, und die dadurch hohe

Hürde für Webdesigner VRML in ihre bestehenden Projekte zu integrieren. Chris Phillips von Microsoft hob gleichzeitig hervor, dass Chromeffects 56 Extensible-Markup-Language-Tags (XML-Tags) bereitstelle und somit sehr einfach zu integrieren sei (vgl. Chalmers, 1998).

Trotz des vielversprechenden Vorhabens seitens Microsoft sollte Chromeffects niemals zu Stande kommen. Nur vier Monate nach Ankündigung von Chromeffects wurde von Microsft auch bereits wieder die Einstellung auf unbestimmte Zeit bekannt gegeben (vgl. Festa, 1998). Während Microsofts Chris Phillips noch im September Interview gegenüber CBR Online News angab wie gut die Integration von Chromeffects mit bestehenden Web-Standards sei (vgl. Chalmers, 1998), war der Tenor der Web-Entwickler / -Entwicklerinnen die das Chromeffects-SDK testen durften völlig anders. Zu langsam sei die Performance, zu schlecht die Integration mit bestehenden und zukünftigen Web-Standards des World Wide Web Consortiums (W3C). Auch die anstehende Wettbewerbsklage wegen Monopolismus soll eine große Rolle bei dieser Entscheidung gespielt haben (vgl. Festa, 1998).

Im April 2007 kündigte Microsoft „Silverlight“ an, ein Anwendungs-Framework das sehr viele der Chromeffects-Vorhaben umsetzt, jedoch neun Jahre später andere Konkurrenz zu bekämpfen hat (vgl. dazu Kapitel 2.4.2 „Microsoft Silverlight“ und Kapitel 4.2 „Microsoft XNA 3D“).

2.3 Extensible 3D (X3D)

Aus dem VRML-Konsortium wurde das Web3D-Konsortium, als erste Amtshandlung wurde am 16. Februar 1999 angekündigt, dass sich eine auf XML basierende VRML-Erweiterung in Arbeit befindet. Des Weiteren war es Ziel der Erweiterung, aus dem monolithischen VRML eine leichte Komponentensammlung zu stricken. Die je nach Einsatzgebiet genau die Elemente enthalten sollte, die benötigt werden. Dabei sollten so viel etablierte Standards wie möglich unterstützt werden (vgl. Auel, 1999).

Um jedoch das von Beginn an gesteckte Ziel zu erreichen – 3D für Jedermann im Web – musste eine Plattformunabhängigkeit erreicht werden. Deshalb wurden die Java3D-Klassenbibliotheken[3] aufgenommen. Java und Java3D waren somit die wichtigsten Programmiersprachen für VRML (vgl. Jahrmann, 1999), später kam auch noch JavaScript dazu. Zur Kommunikation zwischen 3D-Szene und Webseite wurde speziell das Scene Access Interface (SAI) eingeführt, Programmierer / Programmiererinnen können über diese API mit Java, JavaScript oder anderen kompatiblen Programmiersprachen während der Laufzeit auf die X3D-Szene zugreifen, Knoten erstellen und zerstören, Events triggern, Routen erstellen, uvm. (vgl. Web3D Consortium, 2004).

Über die Jahre kamen somit immer mehr Erweiterungen zu VRML respektive X3D hinzu, selbst Adobes Flash wurde unterstützt. Die Anwendungsgebiete blieben jedoch immer auf Nischen beschränkt (vgl. Hager, 2000) und die Webbrowserumgebung wäre für keines der Programme wirklich notwendig gewesen (vgl. Völkl, 2012).

Eine der wohl berühmtesten X3D-Anwendungen soll an dieser Stelle nicht unerwähnt bleiben. Im November 2003 spielte X3D Fritz in New York gegen den mehrfachen Schachweltmeister Garry Kasparov. X3D Fritz war dabei eine angepasste Version von ChessBases[4] Fritz Schachcomputer, den es so bereits seit 1991 gab und als der wohl beste Schachcomputer bis dato gilt. Das Match zwischen Mensch und Maschine endete mit einem vier zu vier Remis.

2.4 Adobe & Microsoft

Zwischen der Ankündigung von X3D 1999 und der Finalisierung des Standards samt ISO/IEC-Zertifizierung im Jahr 2004, ist viel geschehen. Sehr viele große Unternehmen kündigten diverse Projekte an mit denen sie 3D ins Web bringen wollten, u. a. IBM und Pelican Crossing mit inDuality (vgl. Terdiman, 2007) oder 3Di mit Moveable Life (vgl. Wagner, 2007). Aber auch X3D ist noch nicht gestorben. Doch

[3] http://java3d.java.net/ – Offizielle Website von Java 3D.
[4] http://www.chessbase.de/ – Offizielle ChessBase Website.

der gesamte Markt um virtuelle Realitäten im Web stagnierte Zusehens aus Sicht der Endanwender. Zu dieser Zeit entwickelten sich jedoch zwei Technologien in Richtung 3D-Darstellung, die eigentlich vorerst gar nicht dafür geplant waren. Sowohl Adobes Flash als auch Microsofts Silverlight wurden sukzessive mit 3D-Technologien erweitert.

2.4.1 Adobe Flash

Adobe Flash wurde als erstes um 3D-Darstellung erweitert und zwar bereits im Dezember 2005. Es zeichnete sich jedoch nicht Adobe selbst dafür verantwortlich. Der Entwickler Carlos Ulloa Matesanz, der mit dem privaten Projekt Papervision3D eine Erweiterung für Flash programmierte, die 3D-Darstellungen ermöglichte, war der Erste der dafür sorgte, dass es ein 3D-Framework für Adobes Flash gab. Laut eigenen Angaben wollte er ursprünglich lediglich die Illusion von 3D schaffen, es wurde jedoch schnell mehr daraus. In besagtem Dezember 2005 veröffentlichte er dann die erste öffentliche Beta-Version 0.9 von Papervision3D für Flash 7 (vgl. Matesanz, 2006). Mit dem von Matesanz in ActionScript entwickelten Framework ist es möglich 3D-Modelle in Flash-Animationen zu integrieren und ganze 3D-Szenen aufzubauen. Bis einschließlich der Adobe Flash Creative Suite 3 (CS3) war dies nicht mit den von Adobe veröffentlichten Flash-Versionen möglich. Dabei muss das Framework jedoch auf jegliche Hardware-Unterstützung von dedizierter Peripherie verzichten und kann lediglich auf den Hauptprozessor (CPU) zugreifen. Dies bedeutet natürlich, dass die 3D-Fähigkeiten sehr eingeschränkt waren und bis heute sind. Die Entwicklung von Papervision3D wurde Ende 2006 eingestellt. Diverse andere Projekte entwickelten sich aus Papervision3D heraus, so z. B. Away3D das von Alexander Zadorozhny und Rob Bateman 2007 ins Leben gerufen wurde und Code-Bestandteile von Papervision3D übernahm. Die Ausrichtung von Away3D liegt jedoch ausschließlich auf der Entwicklung von 3D-Spielen, Papervision3D hingegen war von Beginn an als Erweiterung für Websites angedacht gewesen (vgl. Zadorozhny, 2009).

Adobe erkannte schnell selbst, dass 3D im Web eine interessante Zukunftstechnologie darstellt und erweiterte bereits 2008 mit der Veröffentlichung von Flash 10 den eigenen Player mit grundlegenden 3D-Darstellungsfähigkeiten. Dabei waren die von Adobe implementierten 3D-Fähigkeiten zum Zeichnen von Objekten jedoch derart beschränkt, dass Adobe diese nicht einmal bewarb (vgl. Nicollet, 2008).

Das steigende Interesse an sogenannten Browser-Games (Videospiele die direkt im Webbrowser gespielt werden) durch Plattformen wie Facebook (als größter Marktplatz für erfolgreiche Browser-Games) und dedizierten Browser-Game-Websites führte dazu, dass Flash immer mehr zur Spieleplattform avancierte. Adobe initiierte daraufhin die Flash Gaming Summit (FGS)[5]. Dieses jährlich stattfindende Event diente als Plattform für Entwickler / Entwicklerinnen um sich über die neuesten Entwicklungen im Bereich der sogenannten Flash-Games zu informieren.

Auf der Flash Gaming Summit 2011 (FGS2011) die am 27. Februar 2011 in San Francisco stattfand, veröffentlichte Adobe die erste vollwertige Version ihrer 3D-API mit dem Codenamen „Molehill“ für Adobe Flash und ermöglichte interessierten Entwicklern und Entwicklerinnen diese vorab zu testen. Zur Präsentation gab es bereits erste Demo-Spiele, die die Möglichkeit der neuen API untermauern sollten. So portierte das Unternehmen Firma Studios ihr Sony PSP Spiel „Zombie Tycoon“ und zeigte eine lauffähige Demo auf der FGS2011 (vgl. Goel, 2011). Im Kapitel 4.1 „Adobe Stage3D“ wird weiter auf Adobes gegenwärtige 3D-API eingegangen.

2.4.2 Microsoft Silverlight

Auf der hauseigenen Entwickler- / Entwicklerinnenkonferenz MIX09[6] die im März 2009 in Las Vegas stattfand, kündigte Microsoft die 3. Version von Silverlight an. Unter vielen anderen Neuerungen wurde auch eine 3D-API mit dem Codenamen „Perspective 3D“ angekündigt, dadurch war Silverlight auf Augenhöhe mit dem zuvor von Adobe veröffentlichen Flash Player 10 (vgl. Braun, 2009). Im Gegensatz zu Adobe bewarb Microsoft die neue 3D-API jedoch immens und bereits vier Monate

[5] http://www.flashgamingsummit.com/ – Offizielle Website
[6] http://channel9.msdn.com/Events/MIX/MIX09 – Offizielle Website

nach der Ankündigung auf der MIX09 war Silverlight 3 mit Perspective 3D erhältlich (vgl. Braun, 2009).

Mit der Veröffentlichung von Silverlight 4 im Jahr 2010 und Silverlight 5 im Jahr 2011 erweiterte Microsoft die 3D-Fähigkeiten des Webbrowser-Plug-Ins. Jedoch konnten bis heute nicht sehr viele Entwickler / Entwicklerinnen davon überzeugt werden, Silverlight statt Adobes Flash für ihre Browser-Games in Erwägung zu ziehen. Im Kapitel 4.2 „Microsoft XNA 3D“ wird weiter auf die 3D-Fähigkeiten von Microsofts Silverlight eingegangen.

3. WebGL

Nicht nur die freie 3D-Entwicklugn im Web bewegte sich auf der Stelle, die Weiterentwicklung der freien Web-Standards durch das W3C kam auch nicht voran. Das Konsortium rundum Tim Berners-Lee (der Erfinder des WWW und dem darunter liegenden Hypertext Transfer Protocol „HTTP“) konzentrierte sich voll und ganz darauf HTML mit dem neuen Standard Extensible Hypertext Markup Language 2.0 (XHTML 2.0) völlig abzulösen. Die Entwicklung von diesem zog sich jedoch in die Länge und es zeigte sich bereits früh, dass es sehr kompliziert werden würde mit XHTML 2.0 Dokumente zu erstellen. Zudem war der geplante neue Standard nicht vollständig abwärtskompatibel zu älteren Standards wie HTML 4.

3.1 HTML5 verändert alles

Das W3C geriet durch diese zuvor genannten Umstände und die langsame Weiterentwicklung vom Cascading Style Sheets Standard (CSS) immer mehr in Kritik. Mozilla und Opera Software ASA (Opera) – beide bekannt für ihre Webbrowser – erstellten deshalb am 4. Juni 2004 die Web Hypertext Application Technology Working Group (WHATWG) Mailinglist. Das WHATWG wurde hierbei nicht als Konkurrenz zum W3C eingerichtet, vielmehr sollten in dieser Arbeitsgruppe Ideen zu Papier gebracht werden, die dann vom W3C in zukünftige Standards einfließen könnten (vgl. WHATWG, 2004). Kurze Zeit später im selben Jahr reichte die WHATWG den ersten Vorschlag zu HTML5 beim W3C ein, damals noch unter dem Namen *Web Applications 1.0*.

Das W3C übernahm die Spezifikationen als Grundlage zur Entwicklung von HTML5 (vgl. W3C, 2007), seit Anfang 2011 lautet die Bezeichnung nur noch HTML um zu zeigen, dass es sich nicht um etwas Fixes handelt, sondern um einen sogenannten *lebendigen* Standard – der der ständigen Korrektur und Weiterentwicklung unterliegt (vgl. Hickson, 2011) (vgl. Diverse, 2012).

Mit dem neuen Standard HTML5 – respektive HTML – wurde der Entwicklung Rechnung getragen, dass sich das Web immer mehr in Richtung einer interaktiven Anwendung entwickelt, weg von den statischen Inhalten die keine Interaktion ermöglichten. Benutzer erwarten sich vom Web das Musik, Videos, Bilder und andere Medienformen direkt integriert sind.

Teil des zukünftig neuen Webstandards war auch das sogenannte *canvas*–Element (zu Deutsch „Leinwand“). Ursprünglich von Apple als Erweiterung der hauseigenen Webkit-Browserengine entworfen, wurde es später vom WHATWG und W3C in den HTML5-Standard aufgenommen. Das *canvas*-Element ermöglicht es, mit Hilfe von JavaScript, zur Laufzeit innerhalb von HTML-Dokumenten zweidimensionale Zeichnungen zu erstellen. Für dreidimensionale Inhalte fehlte zum Zeitpunkt der Adaptierung durch das WHATWG und W3C jedoch ein freier Standard.

Auf der XTech 2006 hielt Vladimir Vukićević eine Präsentation unter dem Titel „*SVG and Canvas: Graphics for Web Apps*“ ab (vgl. Diverse, 2006), in dieser Präsentation schlug Vukićević vor, aufbauend auf dem offenen OpenGL™ (Open Graphics Library) Standard einen 3D-Kontext für das *canvas*-Element zu implementieren. Es sollte jedoch noch über ein weiteres Jahr dauern bis er eine erste Demo vorzeigen konnte. Am 25. November 2007 veröffentlichte Vukićević ein Mozilla Firefox Add-On das gleich zwei leicht unterschiedliche 3D-Kontexte bereitstellte. *moz-gless11* folgte dabei dem OpenGL ES 1.1™ Standard und *moz-glweb20* dem OpenGL ES 2.0™ Standard, um das Add-On testen zu können musste ein *nightly build* von Mozilla Firefox installiert werden. Zum Testen der Add-Ons stellte Vukićević gleich mehrere kleinere Demos bereit (vgl. Vukićević, 2007).

Zu diesem Zeitpunkt wurde bereits in einem eigens dafür eingerichteten Forum innerhalb der Mozilla Website (das komplette Forum wurde mittlerweile gelöscht) diskutiert und an Canvas 3D gearbeitet. Anfang 2009 zog die Gruppe vom Mozilla Forum zu Google Code um (auch diese Seite wurde komplett gelöscht) und der Support für OpenGL ES 1.1 wurde zugunsten von ES 2.0 wegrationalisiert. Die Entwick-

ler / Entwicklerinnen entschieden, dass ES 2.0 zukunftsträchtiger sei als ES 1.1 (vgl. Vukićević, 2009).

Am 24. März 2009 gab die Khronos Group bekannt, dass auf Anfrage von Mozilla eine neue Arbeitsgruppe mit dem Titel „*Accelerated 3D on Web*“ initiiert wurde. Die Aufgabe der Arbeitsgruppe sei es, innerhalb von weniger als zwölf Monaten einen funktionierenden ersten Entwurf für einen Standard zur Realisierung von 3D-Grafiken im Web mit Hilfe des *canvas*-Elements und JavaScript vorzustellen. Gleichzeitig riefen sie interessierte Unternehmen und Entwickler / Entwicklerinnen dazu auf der Arbeitsgruppe beizutreten. Das Mozilla an die Khronos Group herantrat war nicht verwunderlich, basierte die Arbeit von Vladimir Vukićević doch bereits auf dem offenen Grafikstandard OpenGL ES 2.0, welcher von der Khronos Group erarbeitet und verwaltet wird (vgl. The Khronos Group, 2009) (vgl. Ihlenfeld, 2009).

Bereits am 10. Dezember 2009 wurde der erste öffentliche Entwurf zu einem WebGL-Standard veröffentlicht (vgl. Ranganathan, 2009). Zu diesem Zeitpunkt waren bereits alle großen Webbrowser Hersteller Mitglieder der „*Accelerated 3D on Web*“-Arbeitsgruppe, mit der Ausnahme von Microsoft (siehe auch Kapitel 3.5 „WebGL-Sicherheit“). Weitere Unternehmen umfassen nVIDIA und AMD, die zwei großen Hersteller von Grafiklösungen im Desktop- und Mobilmarkt. Auch boten die neuesten Webbrowser Versionen von Mozilla Firefox und Webkit (Apple Safari und Google Chrome) bereits Unterstützung von WebGL an. Diese musste jedoch noch explizit in den erweiterten Einstellungen des jeweiligen Programmes aktiviert werden.

Am 3. März 2011 wurde auf der Game Developers Conference (GDC) von der Khronos Group der finale WebGL 1.0-Standard veröffentlicht (vgl. Khronos Group, 2011) (vgl. Neumann, 2011). Nun veröffentlichten auch Opera und Apple Versionen ihrer Browser mit WebGL-Unterstützung, bis zum heutigen Zeitpunkt jedoch nur als Vorabversionen und mit eingeschränktem Umfang, verglichen mit Mozillas Firefox und Googles Chrome (vgl. Ihlenfeld, 2011).

3.2 Funktionsweise

WebGL basiert auf OpenGL ES 2.0, dieser Standard stammt auch von der Khronos Group und wurde speziell für eingebettete Systeme wie Mobiltelefone, Personal Digital Assistants (PDAs), Konsolen oder Avionik entwickelt. Die OpenGL ES Standards zielen also bewusst auf Plattformen ab die in der Rechenleistung limitiert sind. Während im Desktopbereich OpenGL und Microsofts DirectX zum Einsatz kommen, war dies für Geräte der genannten Klassen keine Option. Zum einen ist DirectX an Microsoft Windows gebunden, was nun mal nur für den Desktopbereich erhältlich ist, zum anderen ist OpenGL viel zu umfangreich, als dass es in einem eingebetteten Systemen effektiv genutzt werden könnte. Um dieses Ziel – eine spezielle API für eingebettete Systeme – zu erreichen wurden von der Khronos Group die folgenden Kriterien in der OpenGL ES Spezifikation festgelegt (vgl. Munshi, et al., 2008):

- Die OpenGL API ist groß und komplex, um eingebetteten Systemen mit all ihren Einschränkungen gerecht zu werden sollen alle Redundanzen aus der OpenGL ES API entfernt werden. Immer wenn es zum Erreichen eines Zieles mehrere Wege gibt, soll die effizienteste beibehalten werden und der Rest entfernt werden. Ein gutes Beispiel ist hierbei die Spezifizierung von Geometrie, in einer OpenGL-Applikation kann der Entwickler / die Entwicklerin auf den *Immediate Mode*, die *Display Lists* oder Vertex Arrays zurückgreifen. In OpenGL ES existieren hingegen nur noch Vertex Arrays und der *Immediate Mode* und die *Display Lists* wurden komplett aus der API entfernt (vgl. ebd.).

- Die Kompatibilität mit OpenGL zu erhalten war jedoch auch klarer Bestandteil der Spezifikation. Dies sollte sicherstellen, dass Applikationen die für eine API geschrieben wurden, sich einfach auf die Andere portieren lassen. Mit OpenGL ES 2.0 wurde dieses Kriterium jedoch sehr oft gebrochen (vgl. ebd.).

- Neue Eigenschaften sollten eingeführt werden, die speziell auf die Einschränkungen von eingebetteten Systemen abzielen. Z. B. wurden Präzisions-

vermerke für Shader eingeführt, um die Leistungsaufnahme von Programmen senken zu können (vgl. ebd.).

- Die Designer sollten ein Minimum an Bildqualität über die API sicherstellen. Der Grund dafür ist denkbar einfach, eingebettete Systeme haben oft nur kleine Bildschirmgrößen und deshalb ist es wichtig, dass jeder gezeichnete Pixel die maximale Qualität aufweist (vgl. ebd.).

- Damit all diese Kriterien auch eingehalten werden, sollten Konformitätstest entwickelt werden die sicherstellen, dass jede OpenGL-ES-Implementierung die Kriterien für Bildqualität, Korrektheit und Robustheit erfüllt (vgl. ebd.).

Es gibt insgesamt drei OpenGL ES Spezifikationen: OpenGL ES 1.0, OpenGL ES 1.1 und OpenGL ES 2.0. Im Folgenden wird nur auf OpenGL ES 2.0 eingegangen, da WebGL direkt von dieser Spezifikation abgeleitet wurde.

OpenGL ES 2.0 ist wiederum direkt von OpenGL 2.0 abgeleitet. Ableitung in diesem Zusammenhang bedeutet, dass ein Dokument erstellt wird, das die genauen Unterschiede und Ergänzungen anführt. Basierend darauf wird dann die abgeleitete Spezifikation erstellt.

3.2.1 WebGL-Grafik-Pipeline

WebGL implementiert eine Grafik-Pipeline mit programmierbaren Shadern und setzt sich somit aus zwei Spezifikationen zusammen: der *WebGL Specification*[7] und der *OpenGL ES Shading Language* in Version 1.00[8] (OGLSL ES) (vgl. Marrin, 2012).

[7] http://www.khronos.org/registry/webgl/specs/latest/ – Aktuellste Version der WebGL Spezifikation.
[8] http://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf – GLSL ES Spezifikation (Format: PDF).

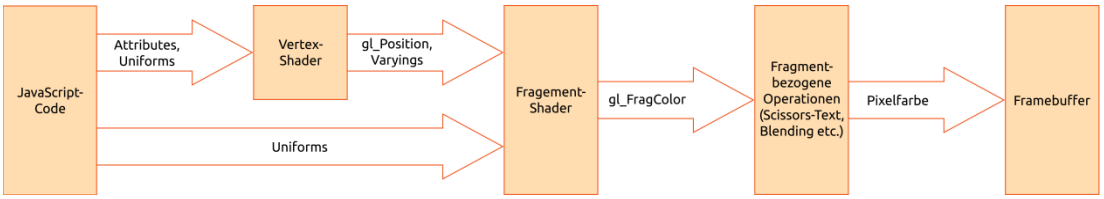

Abbildung 2 *– Die WebGL-Rendering-Pipeline, vereinfachte Darstellung (vgl. ebd.) (vgl. Strohm, 2011)*
(eigene Darstellung vgl. ebd.)

WebGL ist eine hardwarenahe und generische Schnittstelle, Entwickler / Entwicklerinnen müssen entsprechend ein solides Grundverständnis für den Ablauf der Berechnungen auf der GPU, der hier beschriebenen Grafik-Pipeline, besitzen. Des Weiteren müssen beide programmierbaren Shader mit funktionstüchtigem Code gespeist werden damit überhaupt die gewünschte Computergrafik am Bildschirm dargestellt wird. Die Shader selbst werden hierbei mit besagter OGLSL ES (oder kurz einfach GLSL) geschrieben, dieser C ähnliche Code wird im HTML-Dokument gemeinsam mit dem JavaScript-Code eingebettet oder referenziert.

Die zwei programmierbaren Shader-Einheiten sind hierbei – wie in der Grafik oben ersichtlich – der Vertex- und der Fragment-Shader. Der Vertex-Shader berechnet die Position eines jeden Vertex und definiert Attribute, diese werden interpoliert an den Fragment-Shader weitergegeben, es können auch direkt interpolierte Attribute an den Fragment-Shader übergeben werden. Der Fragment-Shader berechnet Farbe und Tiefe eines jeden Pixels. Im Folgenden werden die einzelnen Einheiten der WebGL-Grafik-Pipeline weiter erläutert. Dabei soll dies nur ein Überblick darstellen und begrenzt sich auf das wesentliche.

3.2.1.1 JavaScript-Code

WebGL wird anhand von JavaScript-Code programmiert. Als Teil dieses Dokuments wird nicht näher auf die WebGL-Programmierung mit JavaScript eingegangen, da hier ausschließlich die Geschichte und die Möglichkeiten von 3D im Web behandelt werden. Es existieren unzählige Bücher und Online-Quellen zum Erlernen von JavaScript selbst. Bei WebGL sieht es im Moment noch etwas dürftig aus im Bereich der Literatur. Online gibt es bereits diverse Quellen, die bekannteste und beliebteste

innerhalb der Community ist hierbei *Learning WebGL* von Giles Thomas.[9] Allerdings ist es nicht zwangsläufig notwendig auf spezielle WebGL-Literatur zurückzugreifen. Da WebGL sehr nah an OpenGL ES 2.0 angelehnt ist kann auch entsprechend Literatur für diesen Standard herangezogen werden. An dieser Stelle sei im Besonderen auf *OpenGL ES 2.0 Programming Guide* erschienen über Addison-Wesley (siehe Quellen für alle Details) hingewiesen. Dieses Buch dient auch als eine der Hauptquellen für die technischen Hintergründe dieser Arbeit und wird in der Community als *das* Standardwerk angesehen.

Die Verwendung von JavaScript ermöglicht eine nahtlose Integration in bestehende Webseiten. JavaScript selbst erlebt in den letzten Jahren einen enormen Anstieg an Interesse (vgl. TIOBE, 2012), nicht nur im Bereich des Webs. Mit *node.js* z. B. gibt es bereits die Möglichkeit JavaScript-Code direkt am Server auszuführen, statt ausschließlich clientseitig wie bisher (vgl. Joyent, 2012).

Dabei bringt JavaScript, im Besonderen im Zusammenhang mit Hardware hungrigen 3D-Anwendungen, einige Probleme mit sich, die selbst bei OpenGL ES 2.0 Applikationen so nicht vorhanden sind. C- bzw. C++-Code wird kompiliert und kann auf die ausführende Plattform hin optimiert werden. JavaScript hingegen wird von einem Parser interpretiert und ist plattformunabhängig ausführbar. Die Ausführungsgeschwindigkeit von JavaScript hat in den letzten Jahren enorm zugenommen, erreicht jedoch noch lange nicht die von C/C++-Code. Für erfolgreiche WebGL-Anwendungen der Zukunft ist deshalb die Optimierung der JavaScript-Performance sehr wichtig.

Im März diesen Jahres wurde von Fabric Engine eine vielversprechende Plattform mit dem gleichen Namen vorgestellt, die die Ausführung von JavaScript-Code auf das Niveau von *multithreaded* C++-Anwendungen senken soll (vgl. Lippert, 2012)[10]. Die Fabric Engine ist Open-Source und unter der GNU Affero General Public License erhältlich, dies entspricht auch dem Sinn eines freien Webs wie es von WebGL selbst auch verfolgt wird.

[9] http://learningwebgl.com/ – LearningWebGL von Giles Thomas.
[10] http://fabricengine.com/ – Offizielle Website der Fabric Engine.

3.2.1.2 Vertex-Shader

Ein Vertex ist ein Knotenpunkt eines grafischen Primitivs. In WebGL steht dem Entwickler / der Entwicklerin ausschließlich das Dreieck zur Verfügung. Dies trifft so zwar auch auf OpenGL, alle Derivate und andere APIs zu, jedoch gibt es dort vorkonfigurierte Körper wie Kugeln oder Zylinder – WebGL bietet diese nicht (vgl. Strohm, 2011).

Der Vertex-Shader implementiert eine universell programmierbare Methode zum Arbeiten mit Vertices (Plural von Vertex). Die Eingabe des Vertex-Shaders setzt sich wie folgt zusammen:

- **Attributes** – Per-Vertex-Daten (Positionsangaben, Farben, Transparenz, etc.) in Form eines Vertex-Arrays.
- **Uniforms** – Konstante Daten die vom Vertex-Shader verwendet werden.
- **Samplers** – Ein spezifischer Typ von Uniforms die Texturen repräsentieren die vom Fragment-Shader verwendet werden. Samplers sind optional im Vertex-Shader.

Die Ausgabe des Vertex-Shaders werden variierende Variablen (*Varyings*) genannt. In der Rasterisierungsphase werden die Varyings für jedes Fragment berechnet und als Eingabe an den Fragment-Shader übergeben. Bevor die Daten also an den Fragment-Shader weitergereicht werden können werden noch die Primitive zusammengestellt und Rasterisiert.

In der Zusammenstellungsphase der Primitive wird festgestellt ob sich ein Primitiv im sichtbaren Bereich des Frustums (die Region innerhalb eines 3D-Raumes der auf dem Bildschirm sichtbar ist) befindet, sollte das Primitiv sich nicht voll und ganz im Frustum befinden muss es möglicherweise abgeschnitten (*clipped*) werden. Sollte es komplett außerhalb sein wird es verworfen. Hier findet auch das sogenannte *Culling* statt, das zu Deutsch etwa so viel bedeutet wie „*aussortieren und töten*“. Hierbei werden die Seiten eines Primitivs verworfen die nicht sichtbar sind (also z. B. die

Rückseite eines Quaders der sich zwar im Frustum befindet, jedoch nur Front und die Seiten sind wirklich am Bildschirm sichtbar).

Nach dem *Clipping* und *Culling* wird in der Rasterisierungsphase das dreidimensionale Objekt in darstellbare zweidimensionale Fragmente zerlegt. Diese zweidimensionalen Fragmente repräsentieren Pixel die auf dem Bildschirm dargestellt werden können. Nach dieser Phase werden die Daten an den Fragment-Shader übergeben (vgl. Munshi, et al., 2008).

3.2.1.3 Fragment-Shader

Der Fragment-Shader implementiert eine universell programmierbare Methode zum Arbeiten mit Fragmenten. Der Fragment-Shader muss immer für jedes Fragment ausgeführt werden. Folgende Argumente werden vom Fragment-Shader entgegengenommen:

- **Varyings** – Ausgabe des Vertex-Shaders die durch die Rasterisierungsphase durch Interpolation generiert werden.
- **Uniforms** – Konstante Daten die vom Fragment-Shader verwendet werden.
- **Samplers** – Ein spezifischer Typ von Uniforms die Texturen repräsentieren die vom Fragment-Shader verwendet werden.

Der Fragment-Shader kann das Fragment verwerfen oder einen Farbwert generieren, dieser wird mit *gl_FragColor* referenziert. Die Farbe, Tiefe, Matrize und Bildschirmkoordinatenposition (x_w, y_w) die von der Rasterisierungsphase erzeugt wurden sind die Eingaben für die fragmentbezogenen Operationen (vgl. ebd.).

3.2.1.4 Fragmentbezogene Operationen

Nach dem Fragment-Shader folgen die fragmentbezogenen Operationen (*per-fragment operations*). Ein Fragment das von der Rasterisierungsphase an der Stelle (x_w, y_w) im Bildschirmkoordinatensystem produziert wurde, kann nur den Pixel an

der Stelle (x_w, y_w) im Frame-Buffer modifizieren. Um dies sicherzustellen werden während den fragmentbezogenen Operationen folgende Tests und Operationen durchgeführt:

- **Pixelbesitzertest** – Hier wird festgestellt ob der Pixel an der Stelle (x_w, y_w) im Frame-Buffer auch dem aktuellen WebGL-Kontext gehört. Dieser Test erlaubt es dem Window-System festzulegen welcher Pixel aktuell auch dem WebGL-Kontext zugeordnet ist. Wird z. B. ein Pixel im WebGL-Frame-Buffer Fenster von einem anderen Fenster momentan überlagert, kann das Window-System festlegen, dass dieser Pixel nicht dem WebGL-Kontext zugehörig ist und dieser wird an dieser Stelle dann möglicherweise verworfen.

- **Scissor Test** – Befindet sich das Fragment an der Stelle (x_w, y_w) außerhalb der Schnittmaske die von WebGL zuvor definiert wurde, wird das Fragment verworfen.

- **Stencil and Depth Test** – Anhand der eingehenden Matrize und Tiefe wird festgestellt ob das Fragment behalten oder verworfen wird.

- **Blending** – Hier wird die eingehende Farbe mit der Farbe an der Position (x_w, y_w) im Frame-Buffer kombiniert.

- **Dithering** – Verfahren zur Minimierung von Artefakten durch limitierte Präzision beim Speichern von Farbwerten im Frame-Buffer.

Am Ende all dieser Operationen und Tests wird ein Fragment verworfen oder die Werte werden an der Stelle (x_w, y_w) im Frame-Buffer geschrieben (vgl. ebd.).

3.3 WebGL Bibliotheken

Anhand der sehr kurzen Einführung in die WebGL-Grafik-Pipeline wird sofort klar, dass WebGL selbst hoch komplex ist. Für jede Szene müssen die Vertices neu berechnet und übergeben werden, so expandiert sich selbst ein einfacher sich rotierender Quader zu mehreren hundert Zeilen Code. So hat das komplette HTML-Dokument von Übung 5 bei Learning WebGL 379 Zeilen Code (vgl. Thomas, 2009),

natürlich könnte hier durch verschiedene Optimierungstechniken die Anzahl verringert werden, dies würde jedoch die Übersichtlichkeit stark verringern.

WebGL ist nun mal – wie OpenGL selbst – eine Low-Level-API für 3D-Computergrafiken. Noch bevor der finale WebGL-Standard vorgestellt wurde, gab es deshalb gleich mehrere Bibliotheken (respektive Frameworks) die sich diesem Umstand annahmen und eine höhere Abstraktionsebene definierten, wodurch es Entwicklern und Entwicklerinnen erleichtert werden soll 3D-Computergrafiken ins Web zu bringen. Im Folgenden sollen die wichtigsten Vertreter kurz vorgestellt werden.

3.3.1 GLGE

GLGE[11] (wofür die Abkürzung steht ist nicht bekannt) ist mitunter die älteste WebGL-Bibliothek und komplette 3D-Grafikengine für WebGL. Die 3D-Szene wird im JavaScript-Code mit einem GLGE-spezifischen XML-Dialekt, JSON (*JavaScript Object Notation*) oder einem Binärformat definiert. Des Weiteren werden COLLADA (*Collaborative Design Activity*), Wavefonts OBJ und MD2 (das Dateiformat für z. B. Quake II) unterstützt.

Anhand des Animationssystems können Models bewegt und animiert werden. Es existieren diverse Standardmaterialien die anhand des Texture-Layer-Systems konfiguriert werden können, Schatten und Reflexionen die von GLGE automatisch ermittelt werden. Die benötigten Shader werden über den internen Shader-Composer berechnet. Es gibt eine nicht dokumentierte Shader-Injection-Möglichkeit, mit dieser ist es möglich Teilfunktionen der Shader abzuändern.

Via GPU-Picking, das anhand einer *pick*-Methode zur Verfügung steht, lassen sich Objekte aus der Szene herausfinden, so z. B. welches Objekt sich gerade unterhalb des Mauszeigers befindet.

[11] http://www.glge.org/ – Offizielle GLGE Website.

JigLibJS[12] ist fester Bestandteil von GLGE. JigLibJS ist eine JavaScript-Portierung der bekannten Bullet C++-Physikengine[13]. Die Performance entspricht natürlich nicht ansatzweise der vom C++-Original, doch für einfache Physikdarstellungen reicht es aus.

GLGE orientiert sich an klassischen Game-Engines. Die Umsetzung ähnelt Großteiles sehr stark der eines C++-Programms und viel weniger der eines JavaScript-Programms. Trotzdem ist es eine sehr weit verbreitete WebGL-Bibliothek, die z. B. vom US-Militär eingesetzt wird. Ein großes Manko ist die schlechte Dokumentation, sehr viele Funktionen sind ausschließlich im Quelltext ersichtlich (vgl. Sons, 2012).

Abbildung 3 *– Die „Car Physics Demo" von GLGE demonstriert die Möglichkeiten von JigLibJS.*
Live-Demo: http://glge.org/demos/cardemo/
(eigene Darstellung)

3.3.2 three.js

Mit three.js[14] ist es ähnlich wie mit GLGE. Die Bibliothek besitzt einen sehr mächtigen Funktionsumfang, das System verfügt über Material-, Licht-, Schatten- und Reflexionssysteme. Häufig benötigte Shader sind auch bereits in der Bibliothek enthal-

[12] http://www.jiglibjs.org/ – Offizielle JigLibJS Website.
[13] http://bulletphysics.org/ – Offizielle Bullet Physics Website.
[14] https://github.com/mrdoob/three.js/ – Offizielle three.js Website.

ten, zusätzlich kann three.js um eigene Shader erweitert werden. Eine Shader-Injection wie bei GLGE gibt es nicht. Mit der Dokumentation verhält es sich gleich wie bei GLGE, der Großteil befindet sich im Quelltext.

Von der Bibliothek werden Konverter für Wavefront OBJ und Autodesks FBX bereitgestellt. COLLADA und mit *webgl-loader* komprimierte Meshes können direkt geladen werden. Three.js bietet als einzige Bibliothek einen Canvas Renderer an, das bedeutet, dass sich Szenen theoretisch auch ohne WebGL-Unterstützung rendern lassen, insofern der Webbrowser eine 2D-Canvas-API anbietet. In der Praxis funktioniert dies jedoch nur mit sehr einfachen 3D-Szenen, da bei komplexen Szenen die fehlende GPU-Unterstützung sofort ins Gewicht fällt und die Framerate in die Knie geht (vgl. ebd.).

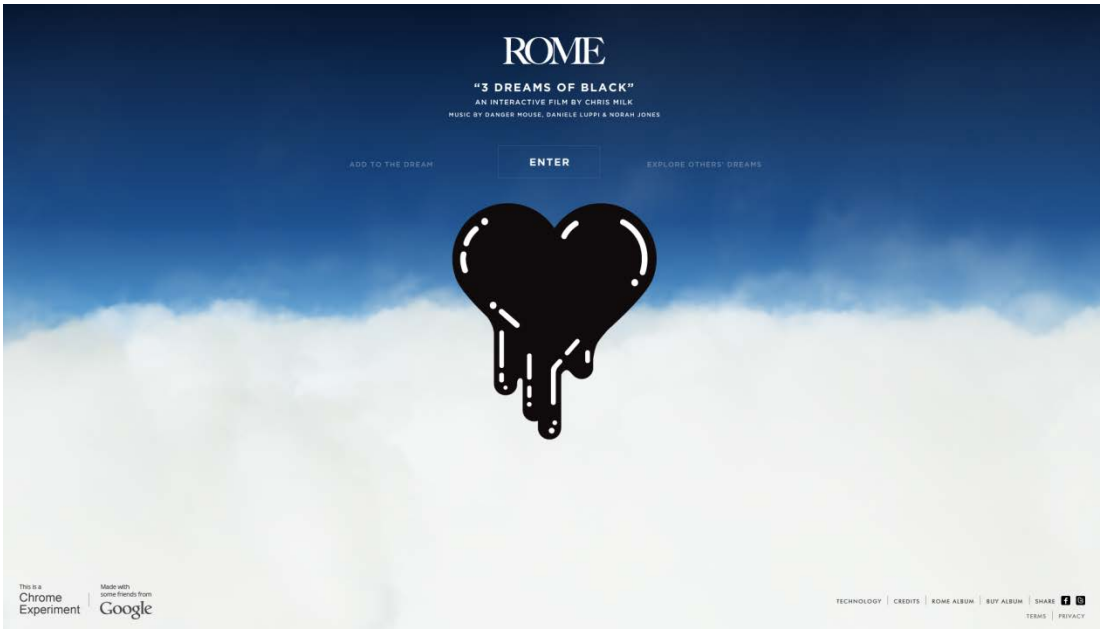

Abbildung 4 *– ROME ist die wohl bekannteste Technologie-Demo die mit three.js realisiert wurde.*
Weblink: http://www.ro.me/
(eigene Darstellung)

3.3.3 PhiloGL

Das Unternehmen Sencha ist bekannt für die JavaScript-Frameworks ExtJS[15] und Sencha Touch[16]. Dem WebGL-Framework PhiloGL[17] von Sencha ist die Erfahrung

[15] http://www.sencha.com/products/extjs/ – Offizielle Sencha ExtJS Webseite.

der Entwickler und Entwicklerinnen mit JavaScript-Frameworks anzumerken und auch die Dokumentation und die Tutorials sind sehr gut. PhiloGL für sich genommen ist jedoch keine vollwertige WebGL-Bibliothek. Vielmehr handelt es sich um eine Bibliothek die ganz knapp oberhalb von WebGL selbst angesiedelt ist und die wichtigsten Funktionen die bei der WebGL-Programmierung benötigt werden zusammenfasst. Auch bei der Unterstützung von 3D-Datenformaten kann PhiloGL nicht mit den anderen Bibliotheken mithalten.

Dafür ist PhiloGL modular aufgebaut und lässt sich um fertige Module erweitern, oder sehr einfach manuell erweitern. Hier besteht auch ein großer Unterschied zu den anderen Frameworks, die sich nicht so einfach an die eigenen Bedürfnisse anpassen lassen. PhiloGL bietet des Weiteren ein ausgereiftes Event-System, ähnlich den DOM-Events (vgl. ebd.).

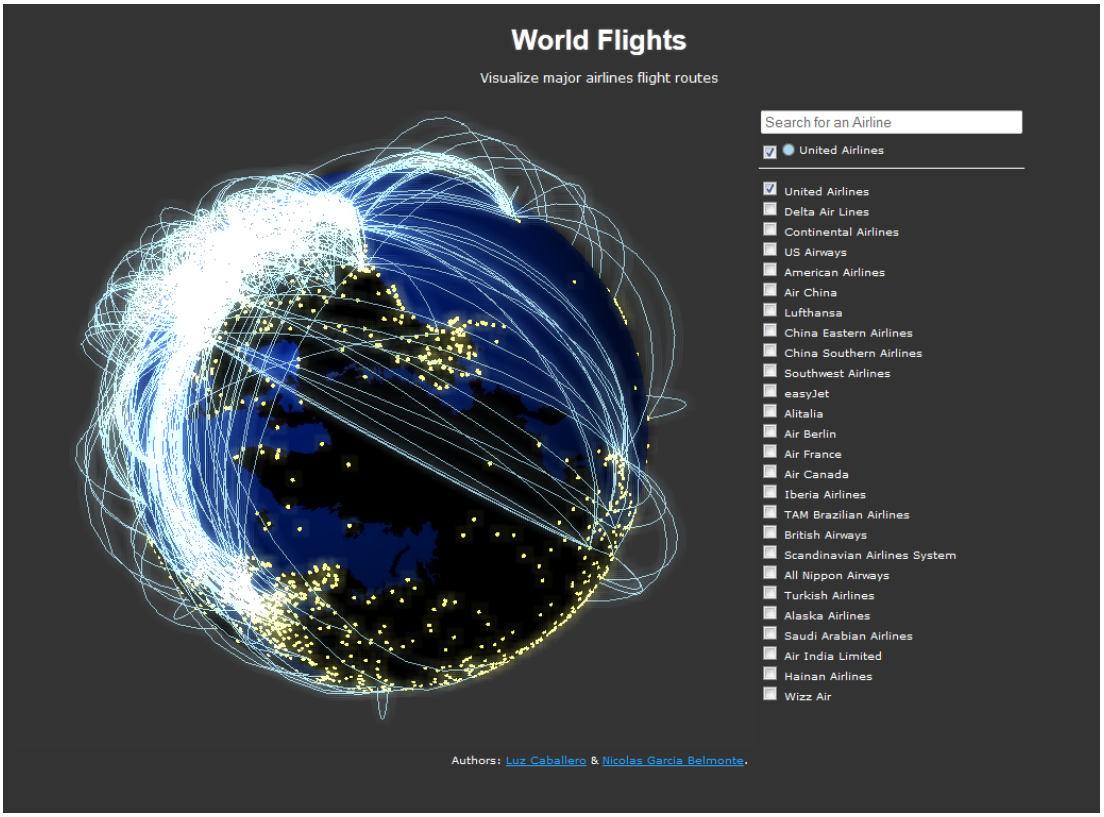

Abbildung 5 *– Die „World Flights"-Demo wurde mit dem PhiloGL-Framework umgesetzt.*
Live-Demo: http://www.senchalabs.org/philogl/PhiloGL/examples/worldFlights/
(eigene Darstellung)

[16] http://www.sencha.com/products/touch/ – Offizielle Sencha Touch Webseite.
[17] http://www.senchalabs.org/philogl/ – Offizielle Sencha PhiloGL Webseite.

3.3.4 SceneJS

Die SceneJS-API konzentriert sich sehr stark auf die Verwendung von JSON. Eine SceneJS-3D-Szene erhält dadurch einen fast schon deklarativen Charakter (mehr dazu im Kapitel 3.3.6 „X3DOM und XML3D“). Im Gegensatz zu X3DOM oder XML3D erhebt SceneJS nicht den Anspruch plattformunabhängig zu sein, sondern zielt ausschließlich auf eine Verwendung mit WebGL ab.

COLLADA- und OBJ-Dateien können über einen Konverter in JSON umgewandelt werden. Die Shader-Injection von SceneJS ist sehr ausgereift und werden SceneJS-Hooks genannt. Insgesamt gibt es 15 solcher Hooks die die Manipulation von Shadern ermöglichen. SceneJS kann auch eine sehr gute und ausgereifte Dokumentation vorweisen (vgl. ebd.).

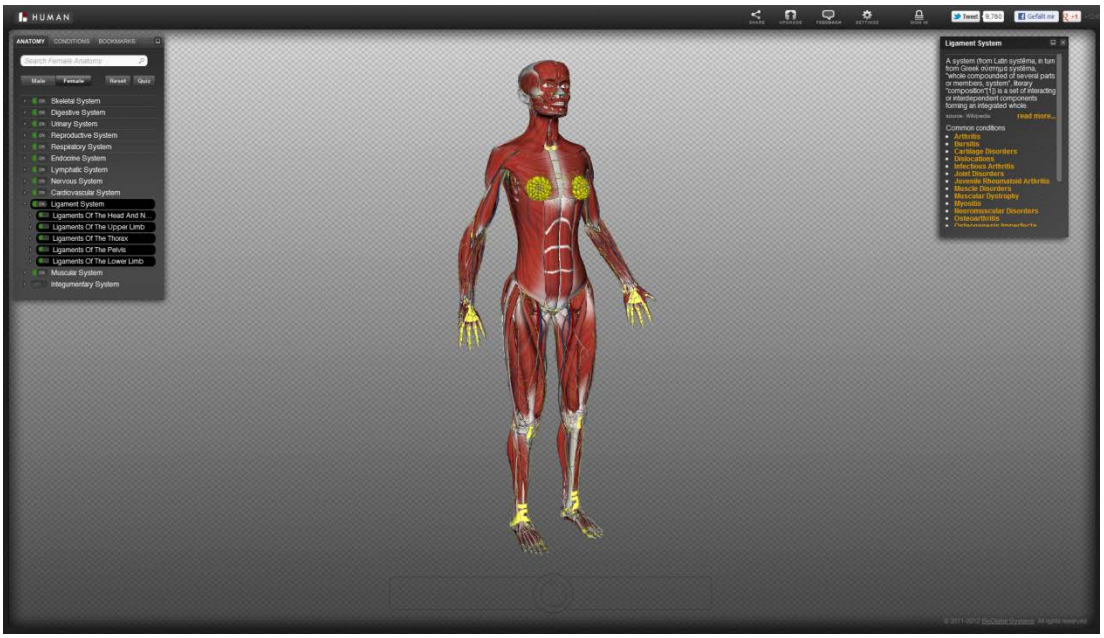

Abbildung 6 *– Extrem umfangreiche „Real-World“-Applikation die mit SceneJS umgesetzt wurde.*
Weblink: https://www.biodigitalhuman.com/
(eigene Darstellung)

3.3.5 CubicVR.js

CubicVR.js[18] ist die letzte Bibliothek die im Rahmen dieser Arbeit vorgestellt werden soll und zugleich eine JavaScript-Portierung der gleichnamigen C++-Rendering-Engine[18]. CubicVR.js bietet auch Material-, Schatten- und Lichtsysteme an. Daten werden über das bibliothekseigene JSON- oder XML-Format eingelesen. Eine Besonderheit ist die Unterstützung des GML-Formats (*Graffiti Markup Language*), diese Dateien werden anhand des Trackings einer Sprühdose erstellt und lassen sich dann in eine CubicVR.js-Szene importieren. CubicVR.js lässt sich durch weitere JavaScript-Bibliotheken erweitern, ist jedoch nicht modular aufgebaut. So bietet die Bibliothek Unterstützung für *ammo.js*[19] – eine Physiksimulation ähnlich JigLibJS, siehe Kapitel 3.3.1 „GLGE“ – und über *pdf.js*[20] können PDF-Dateien als Texturen verwendet werden (vgl. ebd.).

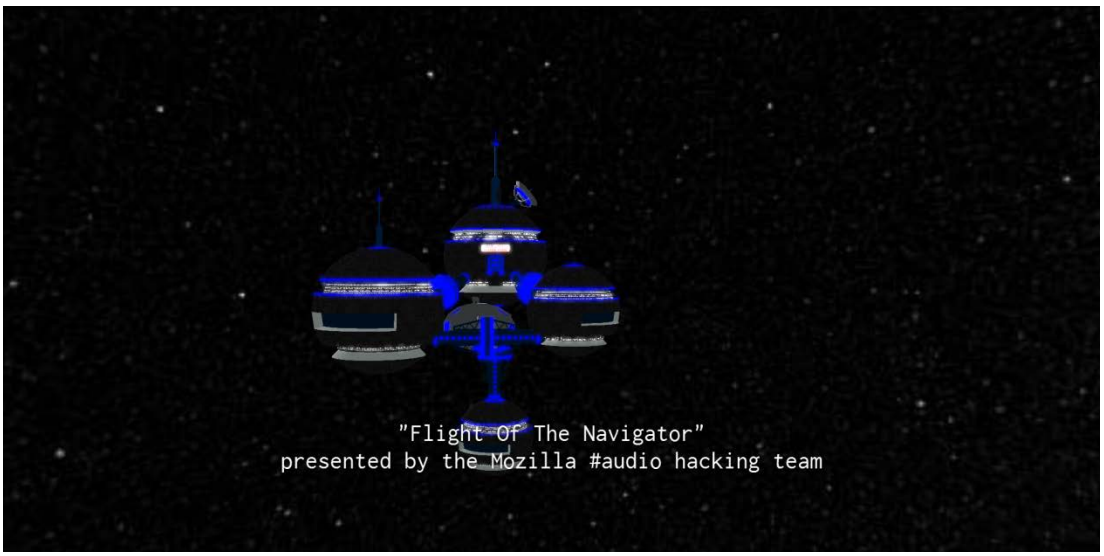

***Abbildung 7** – „Flight Of The Navigator“ ist der wohl bekannteste Einsatz von CubicVR.js. Diese Technologie-Demo wurde auch verwendet um Firefox 4 zu bewerben.*
Weblink: http://videos.mozilla.org/serv/mozhacks/flight-of-the-navigator/
(eigene Darstellung)

[18] http://www.cubicvr.org/ – Offizielle CubicVR.js und CubicVR Website.
[19] https://github.com/kripken/ammo.js/ – Offizielles ammo.js Github-Repository.
[20] https://github.com/mozilla/pdf.js – Offizielles pdf.js Github Repository.

3.3.6 Google Open 3D (O3D)

Ursprünglich begann die Entwicklung von O3D als Browser-Erweiterung die es ermöglichten sollte 3D-Computergrafiken ins Web zu bringen und zielte von vornherein auf echte Computerspiele und andere Anwendungen die Wert auf gute 3D-Qualität setzen (z. B. CAD) ab (vgl. Braun, 2009). Als der WebGL-Standard jedoch Form annahm änderte Google seinen Kurs für das ursprüngliche O3D-Projekt. O3D soll in Zukunft als JavaScript-Frontend auf WebGL aufbauen und erweitert die offene API mit diversen Wrapper-Funktionen die es Entwicklern und Entwicklerinnen erleichtern soll opulente 3D-Anwendungen zu entwickeln (vgl. Papakipos, 2010).

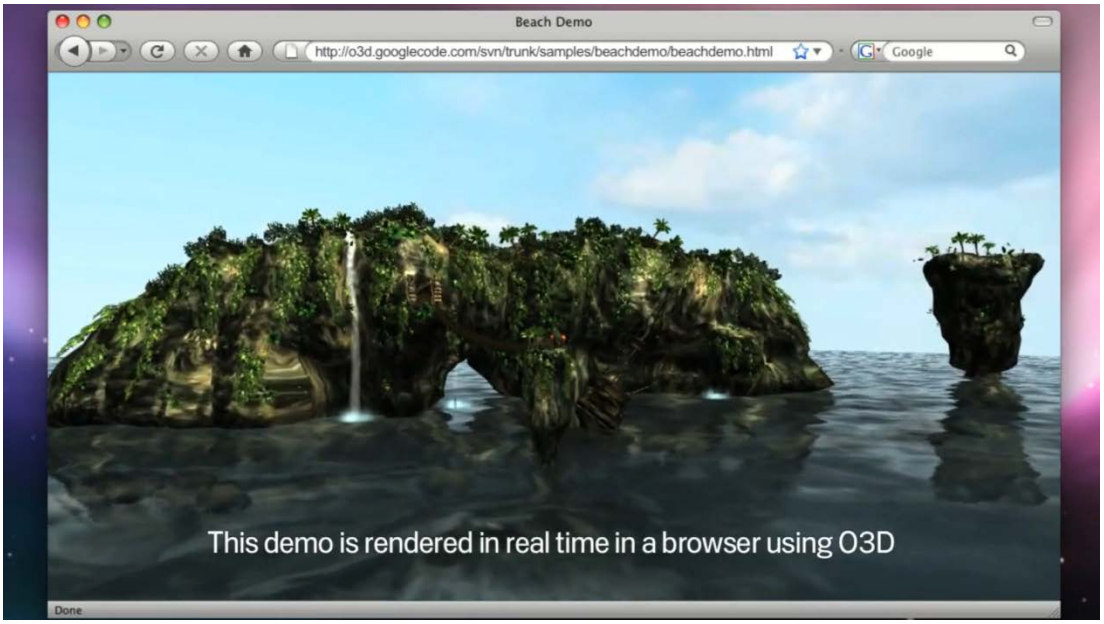

Abbildung 8 *– Bildschirmfoto von Googles O3D Präsentationsvideo.*
Weblink: http://www.youtube.com/watch?v=uofWfXOzX-g
(eigene Darstellung)

3.4 Deklaratives 3D

Um Computergrafiken am Bildschirm darzustellen, gibt es zwei verschiedene Herangehensweisen, die Imperative und die Deklarative. WebGL zählt hierbei zu den Imperativen, mit WebGL wird folglich anhand einer prozeduralen API beschrieben wie die Computergrafik zu zeichnen ist (im Kapitel 3.2 „Funktionsweise“ wird näher

darauf eingegangen). Hingegen wird bei der deklarativen Weise angegeben was auf dem Bild zu sehen ist. Web-Entwickler / -Entwicklerinnen sind mit letzterem weitaus besser vertraut, da diese Herangehensweise denen anderer etablierter Web-Sprachen wie HTML und SVG (Scalable Vector Graphics) entspricht. Ein Punkt dem sowohl X3DOM als auch XML3D Rechnung tragen (vgl. Völkl, 2012).

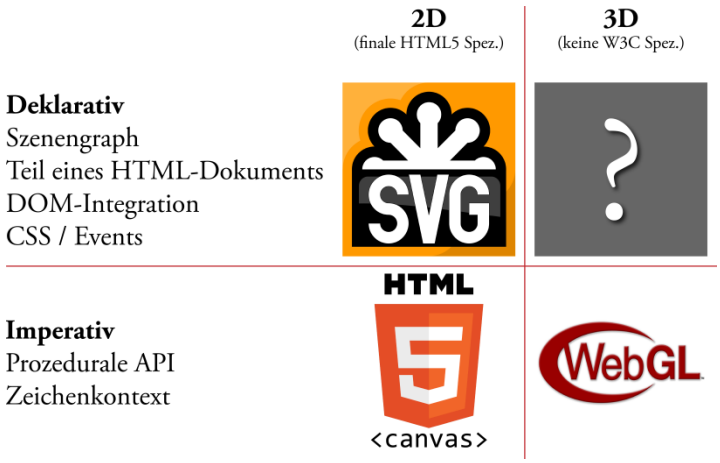

Abbildung 9 *– Beziehung von deklarativer und imperativer Herangehensweisen illustriert anhand von 2D- und 3D-Kontext (vgl. Frauenhofer IGD, 2012).*
(eigene Darstellung[21] [22] [23])

Der Krux an WebGL und allen 3D-Engines ist die Tatsache, dass nur Spezialisten / Spezialistinnen fähig sind Anwendungen damit zu erstellen. Dies trifft so z. B. auch auf VRML und X3D zu. Damit 3D-Computergrafiken ihren Siegeszug im Web antreten können, muss die Einstiegshürde jedoch stark verringert werden. 3D-Computergrafiken werden in der Gegenwart hauptsächlich in Computer- und Videospielen eingesetzt. Größere Entwickler- / Entwicklerinnenteams bestehend aus besagten Spezialisten / Spezialistinnen und arbeiten an speziellen Engines die für die zu erstellende Anwendung optimiert wurden. Die immensen Kosten werden daraufhin durch die Verkäufe wieder gedeckt.

[21] Das SVG-Logo stammt von Wikimedia Commons [http://commons.wikimedia.org/wiki/File:SVG_logo.svg] und steht unter der Creative Commons-Lizenz Namensnennung-Weitergabe unter gleichen Bedingungen 3.0 Unported (CC BY-SA 3.0) zur Verfügung. Namensnennung (Angabe entsprechend Lizenz): *€*, *AMK1211*, *Lucideer*, *Victormoz*, Richard Fussenegger (*Fleshgrinder*)

[22] Das HTML5 Logo stammt von Wikimedia Commons [http://commons.wikimedia.org/wiki/File:HTML5-logo.svg] und steht unter der Creative Commons-Lizenz Namensnennung 3.0 Unported (CC BY 3.0) zur Verfügung. Namensnennung (Angabe entsprechend Linzenz): W3C

[23] Das WebGL Logo stammt von der offiziellen Khronos Group Website [http://www.khronos.org/news/logos]. Das WebGL Logo und die Wortmarke sind registrierte Warenzeichen der Khronos Group und werden hier im Rahmen des UrhG § 51 Absatz 1. zur Erläuterung des Inhaltes verwendet.

Bei den sogenannten „Casual Games“ – wie es jedes Browser-Game nun mal ist – muss die Entwicklung jedoch viel schneller vonstattengehen. Neue Ideen müssen schnellstmöglich umgesetzt und auf den Markt gebracht werden. Das *alte*, traditionelle System funktioniert hier bereits nicht mehr.

Noch viel weniger im industriellen Bereich. CAD-Techniker z. B. erhalten nicht die Möglichkeit an 3D-Engines zu feilen, sie müssen schnell und effizient die geforderten Objekte umsetzen und dem Vorgesetzen vorzeigen. Nicht jeder Vorgesetzte besitzt ein CAD-Programm, jedoch jeder besitzt in der Regel einen Webbrowser. So könnte der CAD-Techniker das fertige Model exportieren, am Server hochladen und einfach die URL zum Model via E-Mail versenden.

Beide Ansätze, X3DOM und XML3D, zielen genau darauf ab. Aufbauend auf HTML5 soll der bestehende DOM um 3D erweitert werden. Dies macht auch Sinn, ist doch der DOM bereits ein deklarativer 2D-Szenengraph der dargestellten Webseite. Entsprechend wäre es nicht sehr zielführend neue Konzepte und eine neue API zu definieren, die Verwendung der alten, bekannten DOM-API bietet auch noch weitere Vorteile. So lassen sich 3D-Computergrafiken direkt im DOM selbst darstellen (siehe auch Abbildung 6).

X3DOM und XML3D sind die zwei einzigen Open-Source-Projekte weltweit die eben diesen Ansatz auf verschiedene Art und Weise verfolgen. Am 30. Juni 2011 wurde im Rahmen der W3C *Community & Business Groups* die *Declarative 3D for the Web Architecture* Arbeitsgruppe ins Leben gerufen (vgl. Sons, 2011). Sie setzt sich aus den Mitgliedern von X3DOM und XML3D zusammen. Ihre Aufgabe ist es, einen Konsens zu finden, der dann vom W3C in den HTML5-Standard für deklaratives 3D im Web überführt werden kann (vgl. Sons, et al., 2011).

Abbildung 10 *– DOM der FH JOANNEUM Startseite in 3D dargestellt. Hier ist gut ersichtlich, dass es sich auch beim DOM um eine Art Szenengraph handelt.*
(eigene Darstellung)

3.4.1 X3DOM

X3DOM (*Extensible 3th Dimension Document Object Model*, ausgesprochen *X-Freedom*) wurde vom Frauenhofer IGD (*Frauenhofer-Institut für Graphische Datenverarbeitung*)[24] in Darmstadt ins Leben gerufen und baut auf dem zuvor bereits vorgestellten X3D-Standard auf. Ziel des Frauenhofer IGD ist es, ein deklaratives Pendant zu WebGL als Teil des HTML-Standards zu etablieren. Die Portierung von X3D zu X3DOM erfolgt dabei ausschließlich unter der Verwendung von den freien Standards des W3C und ohne Webbrowser Plug-Ins – wie bisher das bereits in Kapitel 2.3 „Extensible 3D (X3D)“ vorgestellte SAI.

X3DOM wurde zum ersten Mal auf der W3C TPAC 2009 (*W3C Technical Plenary / Advisory Committee Meetings Week 2009*) in Santa Clara, USA, vorgestellt. Seit

[24] http://www.igd.fraunhofer.de/ – Offizielle Website des Frauenhofer IGD

Ende 2009 ist es verfügbar und wird in verschiedenen Projekten bereits erfolgreich eingesetzt.

Die Idee X3D in HTML komplett zu integrieren ist jedoch nicht neu. So war X3D zu Beginn im HTML5-Standard bereits als deklarative Sprache referenziert, wurde jedoch zwischenzeitlich im Rahmen einer allgemeinen Überarbeitung aus dem Standard ausgeschlossen. Innerhalb des Web3D-Konsortiums wurde jedoch mittlerweile eine HTML5/X3D-Interessensgruppe angesiedelt die basierend auf dem X3DOM Integrationsmodell nun bereits Version 1.3 des Standards im Dezember 2011 veröffentlicht hat (vgl. x3dom, 2011).

Dabei wird mit X3DOM gar nicht darauf abgezielt den gesamten X3D-Standard zu Portieren. Vielmehr wird ein konkretes HTML-Profil definiert das die wichtigsten Szenegraphen enthält (z. B. <*shape*>), andere, komplexere Konzepte werden außen vorgelassen. X3DOM erweitert somit, wie X3D durch SAI, das HTML-Dokument durch weitere Knotenpunkte, wodurch zur Laufzeit dynamisch und interaktiv die 3D-Szene beeinflusst werden kann (vgl. Behr, et al., 2012).

Es folgt ein einfaches Beispiel, das zeigen soll wie sich X3DOM in der Praxis anfühlt. Die Syntax entspricht erwartungsgemäß der von HTML bzw. XML.

```
<html>
<head>
  <title>X3DOM Quader</title>
  <link rel="stylesheet" href="x3dom.css">
  <script src="x3dom.js"></script>
</head>
<body>
  <h1>X3DOM Quader</h1>
  <x3d width="500" height="400"
  xmlns="http://www.web3d.org/specifications/x3d-namespace">
    <scene>
      <background skyColor="0 0 1"></background>
      <shape>
        <appearance>
          <imageTexture url="texture.png"></imageTexture>
        </appearance>
        <box></box>
      </shape>
    </scene>
  </x3d>
</body>
</html>
```

Quelltext 1 *– Einfacher Quader mit deklarativem X3DOM direkt im HTML-Dokument eingebettet*
(eigener Code)

Abbildung 11 *– Ergebnis des obigen X3DOM-HTML-Dokuments, der Quader ist durch die Verwendung vom X3DOM JavaScript sofort interaktiv und kann mit Hilfe der Maus bewegt werden. Live-Weblink: http://test.bobdo.net/bac1/x3dom.html*
(eigene Darstellung)

X3DOM ist als JavaScript-Frontend umgesetzt, wie im Quelltext auf Seite 41 durch die <*script*>-Referenzierung ersichtlich, das durch ein Fallback-Modell zur Darstellung der 3D-Szene verschiedene 3D-Backends verwenden kann. Derzeit werden vier verschiedene Backends unterstützt:

1. **Modifizierter Browser** – Das X3DOM-Skript überprüft ob der Webbrowser das *x3d*-Element unterstützt. Ist dies der Fall wird die Ausführung sofort abgebrochen, da der Browser die X3D-Angaben im HTML-Dokument selbst umsetzen kann. Hierzu muss jedoch die Codebasis des Webbrowsers modifiziert werden, was diverse Sicherheitsprobleme mit sich bringt und auch dazu führt, dass Endbenutzer einen speziellen Browser installieren müssen. Weshalb das Frauenhofer IGD bisher nur einen modifizierten Webkit-Browser für Apples iPhone zu demonstrationszwecken erstellt hat.

2. **X3D-Plug-In** – Als nächstes wird vom X3DOM-Skript überprüft ob ein X3D-Plug-In installiert ist das SAI unterstützt, z. B. der Instant Player[25] vom Frauenhofer IGD selbst. Das Plug-In übernimmt dann die Darstellung der 3D-Szene, nachdem das Plug-In vom X3DOM-Skript initialisiert wurde. Nun können vom X3DOM-Skript DOM-Veränderung über die SAI-Schnittstelle des Plug-Ins an den Szenengraphen übertragen werden, die Überwachung des DOMs erfolgt dabei anhand von DOM-Level-2-Events[26]. Diese Variante impliziert natürlich wieder den Nachteil für den Endbenutzer, dass er ein spezielles Plug-In installieren muss.

3. **WebGL** – Im dritten Schritt wird überprüft ob ein WebGL-Kontext erstellt werden kann und der Webbrowser somit WebGL direkt unterstützt. Ist dies der Fall wird ein X3D-Szenengraph instanziiert und die Synchronisation von DOM und Szenengraph erfolgt wiederum via DOM-Level-2-Events. Diese Variante hat den Vorteil, dass sie von sehr vielen modernen Plattformen unterstützt wird und völlig ohne die Installation von zusätzlicher oder modifizierter Software auskommt. Ein klarer Nachteil ist, dass JavaScript nicht die Performanz bietet die native C/C++-Anwendungen bieten und WebGL ledig-

[25] http://www.instantreality.org/ – Offizielle Website
[26] http://www.w3.org/TR/DOM-Level-2-Events/ – DOM-Level-2-Events W3C Standard

lich das Shader-Model 2.0 unterstützt (verglichen mit der DirectX-9-Unterstützung von Adobes Stage 3D, mehr dazu im Kapitel 4.1 „Adobe Stage 3D“).

4. **Adobe Flash** – Als letzte Fallback-Variante steht Adobe Flash 11 mit Stage 3D zur Verfügung. Auf Stage 3D und seine Fähigkeiten wird im Kapitel 4.1 „Adobe Stage 3D“ genauer eingegangen.

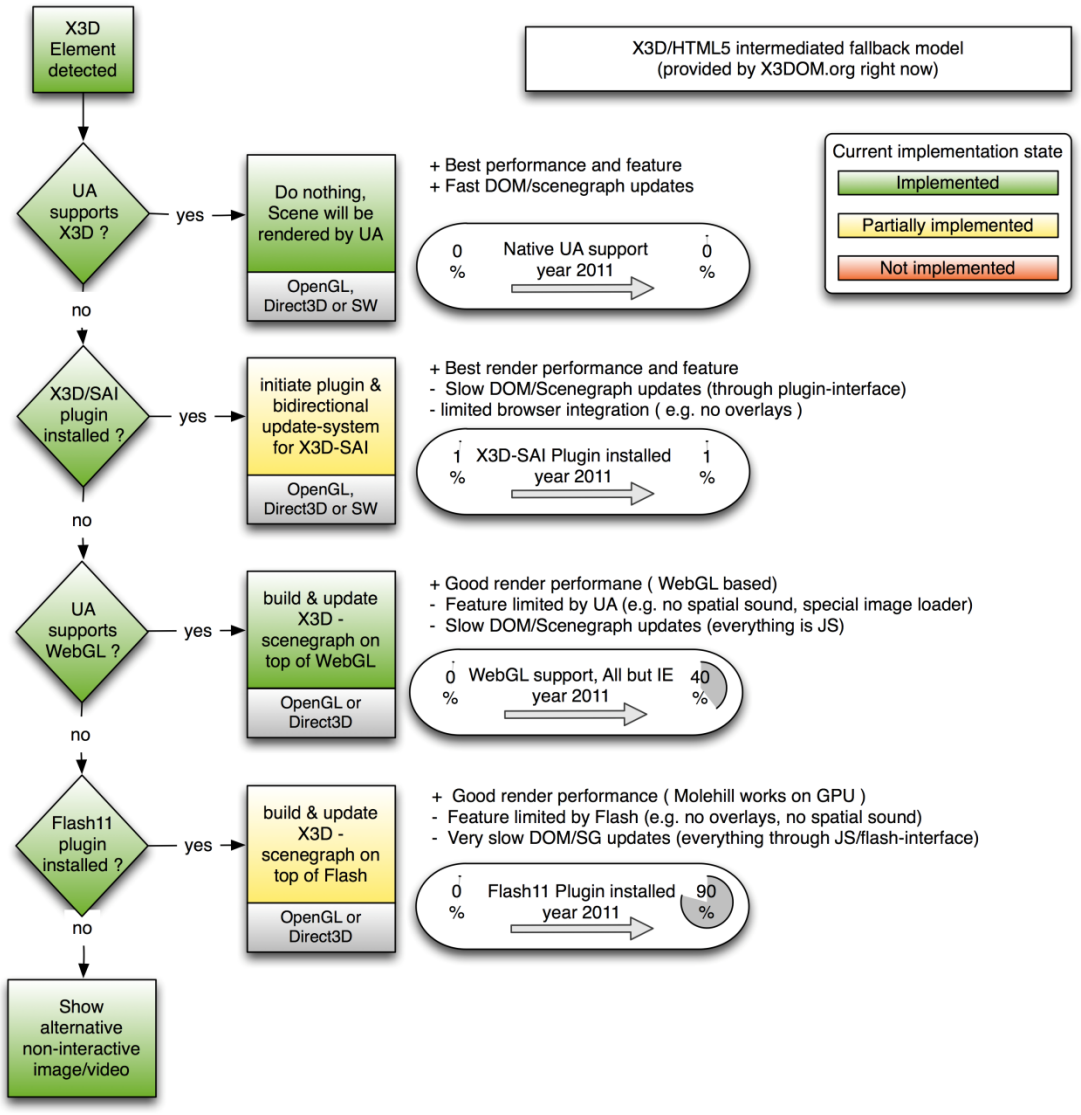

Abbildung 12 *– Das X3DOM-Fallback-Modell im Detail, mit genauen Erläuterungen welche Vor- und Nachteile die jeweilige Variante der Umsetzung des X3D-Szenengraphen mit sich bringt.*
(vgl. Frenguelli, 2012)

3.4.2 XML3D

XML3D (*Extensible Markup Language 3th Dimension*) wird gemeinsam mit Intel vom DFKI (Deutsche Forschungszentrum für Künstliche Intelligenz)[27] und Intel VCI (Intel Visual Computing Institute)[28] entwickelt und geht einen ähnlichen Weg wie X3DOM. Die Umsetzung baut jedoch nicht auf einem bereits etablierten Standard wie X3D auf, es wird lediglich der HTML5-Standard an den Stellen für 3D erweitert, wo es notwendig ist. Dies bringt die Vorteile mit, dass die Entwickler / Entwicklerinnen, außer den durch HTML5 vorgegebenen, keine fremden Konzepte oder Techniken übernehmen müssen.

XML3D wurde zum ersten Mal erfolgreich auf der CeBIT 2010 (Akronym für Centrum für Büroautomation, Informationstechnologie und Telekommunikation) vorgestellt. Dort zeigten die Entwickler / Entwicklerinnen auch einige beeindruckende Demos, die zeigten was mit 3D in einem Webbrowser alles möglich ist. Passend zu Intels Raytracing Forschungen[29,30,31,32,33] wurde eine 3D-Szene gezeigt, die via Echtzeit-Raytracing im Webbrowser gerendert wurde (vgl. Golem.de, 2010).

Anders als bei X3DOM besitzt XML3D kein Fallback-Modell. Die Entwickler / Entwicklerinnen setzen voll und ganz auf modifizierte Webbrowser, die sie auf der Projekt-Website zum Download anbieten.[34] Das XML3D-Team hätte auch auf ein Webbrowser Plug-In setzen können. Dies hätte jedoch zur Folge gehabt, dass sie die existierenden CSS-Eigenschaften nicht erweitern hätten können. Dasselbe trifft auch auf JavaScript und WebGL zu, da die Implementierungen nicht durch ein Plug-In modifiziert werden können.

Welche Möglichkeiten sich durch die Verwendung von Standard CSS-Eigenschaften zur Manipulation der 3D-Computergrafik ergeben, ist sehr gut in der iX-Demo *3D on*

[27] http://www.dfki.de/ – Offizielle Website des DFKI
[28] http://www.intel-vci.uni-saarland.de/ – Offizielle Website des Intel VCI
[29] http://www.computerbase.de/artikel/grafikkarten/2007/bericht-raytracing-in-spielen/
[30] http://www.computerbase.de/artikel/grafikkarten/2008/bericht-raytracing-in-spielen-2.0/
[31] http://www.computerbase.de/artikel/grafikkarten/2009/bericht-quake-wars-mit-raytracing/
[32] http://www.computerbase.de/artikel/grafikkarten/2011/bericht-ray-tracing-4.0/
[33] http://www.computerbase.de/artikel/grafikkarten/2012/bericht-ray-tracing-5.0/
[34] http://www.xml3d.org/downloads/ – Download-Seite für die modifizierten Webbrowser

Top of HTML demonstriert, wie in Abbildung 5 ersichtlich. Den besten Eindruck erhält der Leser / die Leserin jedoch von einer Live-Demo. Diese ist online erreichbar unter http://graphics.cs.uni-sb.de/fileadmin/cguds/projects/xml3d/iX/integration.xhtml.

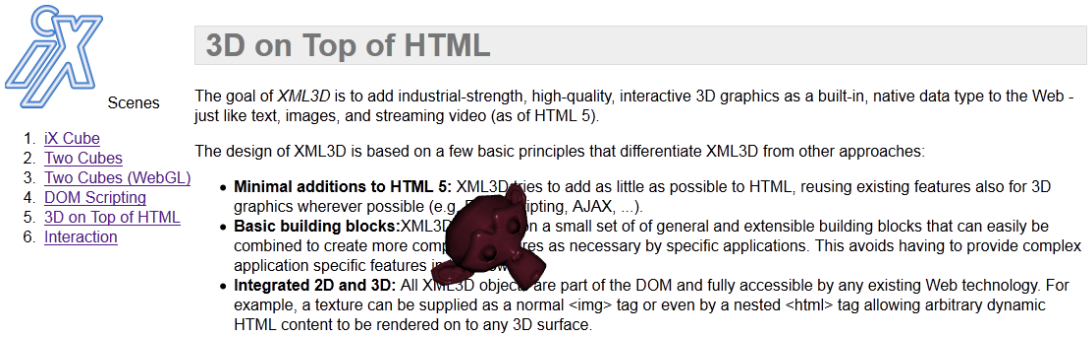

***Abbildung 13** – Die iX-Demo „3D on Top of HTML“ zeigt eindrucksvoll was mit XML3D möglich ist. Der Affe Suzanne stammt aus dem Blender DCC-Package[35] und wird über dem HTML-Dokument dargestellt.*
(eigene Darstellung)

3.5 WebGL-Sicherheit

Am 16. Juli 2011 gab die Abteilung *Security Research & Defense* von Microsoft in einem Blogeintrag bekannt, dass Microsoft WebGL in absehbarer Zeit nicht unterstützten werde. Dies begründete die Microsoft Abteilung mit den folgenden drei Argumenten (vgl. Security Research & Defense, 2011):

- Hardwarefunktionen würden zu freizügig übers Web bereitgestellt. Die Sicherheit hänge dadurch sehr stark vom Grafikkartentreiber ab. Angriffe auf die Grafikeinheiten können dank WebGL über das Internet aus der Ferne ausgenutzt werden. Die Microsoft Abteilung räumte ein, dass es möglich sei diese Angriffe einzuschränken, hält die Angriffsfläche jedoch für zu groß.

- Die Sicherheitsvorkehrungen im Webbrowser seien zu stark abhängig von Drittanbietern. Nicht immer öffnen sich Angriffsvektoren in der WebGL-API selbst, sondern auch in OEM- oder Systemkomponenten. Als unsicher eingestufte Hardwarekonfigurationen zu blocken hält die Microsoft Abteilung da-

[35] http://www.ogre3d.org/tikiwiki/DCC+Tools – Blender DCC Tools bei Ogre3D.

bei nicht für zielführend, da Benutzer diese Blockaden umgehen könnten um die von ihnen gewünschten WebGL-Inhalte trotzdem sehen zu können. Dabei wird im speziellen auf Grafikkartentreiber eingegangen, für die ein Updateverfahren ähnlich dem von Windows-Update fehle. Zudem würden manche Treiber nur einmal im Jahr aktualisiert oder die Treiberinstallation vom System geblockt.

- Als dritten und letzten Punkt führt die Microsoft Abteilung bestimmte Denial-of-Service-Szenarien (DoS) an. Betriebssysteme und Grafikinfrastruktur hätten keine Methoden um solche Angriffe die über Shader- und Geometriefunktionen hereinkommen abzuwehren. Das Risiko für den Rechner sei hier immens groß.

Die Sorgen die von der Microsoft Abteilung in diesem Blogeintrag geäußert wurden waren zu diesem Zeitpunkt jedoch nicht neu. Context Information Security veröffentlichte bereits zwei Artikel zu diesem Thema (auf die bezog sich die Microsoft Abteilung auch in ihrem Blogeintrag) (vgl. Forshaw, 2011) (vgl. Forshaw, et al., 2011) (vgl. Forshaw, 2011) und die Khronos Group selbst befasste sich bereits mit genau denselben Fragen (vgl. Khronos Group, 2011).

Die Ankündigung Microsofts führte trotzdem zu sehr viel Aufmerksamkeit seitens der Endbenutzer und anderer Webbrowser-Hersteller die bereits WebGL-Unterstützung anboten. In weiterer Folge wurde Microsoft vorgeworfen, dass diese Sicherheitsbedenken nur eine Ausrede seien, da die XNA 3D API im hauseigenen Silverlight 5 (siehe auch Kapitel 4.2 „Microsoft XNA 3D“) genau dieselben Angriffsvektoren aufweise (vgl. Muizelaar, 2011) (vgl. Ihlenfeld, 2011) (vgl. Kirsch, 2011) (vgl. Ihlenfeld, 2011).

4. WebGL-Konkurrenz

WebGL ist, wie bereits festgestellt wurde, nicht der erste Versuch 3D-Computergrafiken ins freie Web zu bringen. Aber auch in der Gegenwart ist WebGL nicht ohne Konkurrenz. An dieser Stelle sollen die größten Konkurrenten von WebGL kurz vorgestellt werden.

4.1 Adobe Stage 3D

Im Kapitel 2.4.1 „Adobe Flash“ wurde bereits auf die Entwicklungsgeschichte von Adobe Flash und den 3D-Fähigkeiten eingegangen. Am 21. September 2011 kündigte Adobe die neue Adobe Flash Version 11 an. Bereits in der Ankündigung wurde von Adobe explizit auf eine neue 3D-API mit direktem Zugriff auf die Hardware gesprochen (vgl. Auel, 2011).

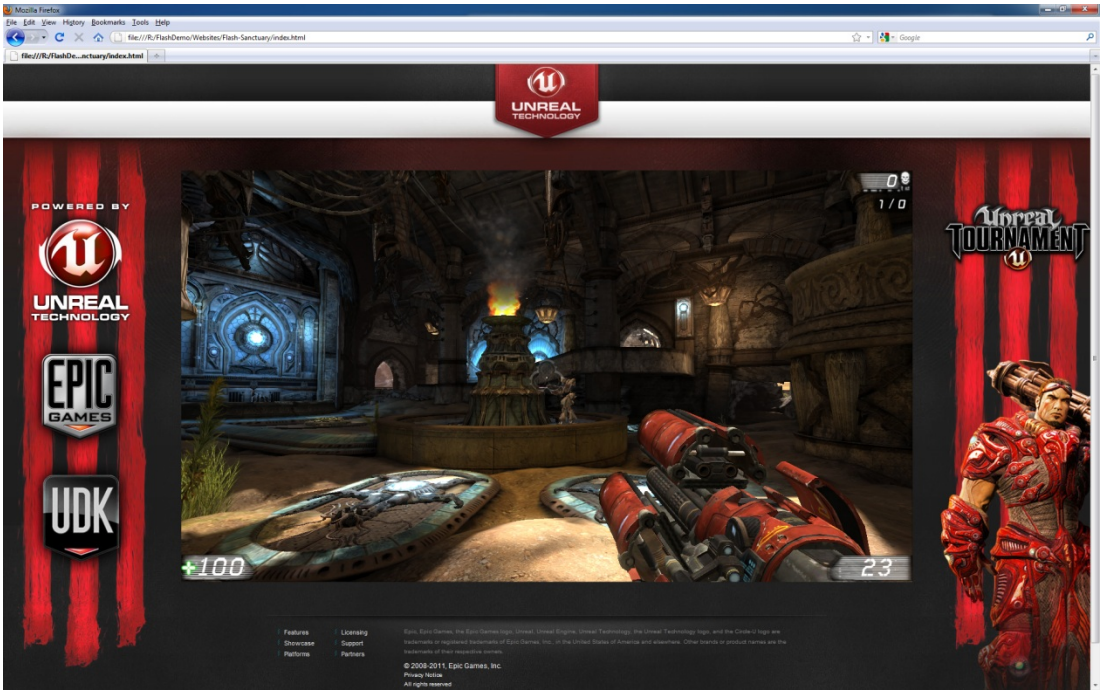

***Abbildung 14** – Bildschirmfoto der UT3 Sanctuary Technologie-Demo die von Epic Games auf der Adobe MAX 2011 vorgestellt wurde.*
(vgl. Huang, 2011)

Wenige Tage später – am 4. Oktober 2011 – standen die finalen Versionen von Adobe Flash Player 11 und Adobe Air 3 bereits im Rahmen der hauseigenen Entwickler-

/ Entwicklerinnenkonferenz Adobe MAX[36] zum Download bereit (vgl. König, 2011). Der Gründer von Epic Games – einer der bekanntesten Hersteller von 3D-Engines der Welt – Tim Sweeney präsentierte am selben Tag noch eine lauffähige Flash-Portierung[37,38] der hauseigenen Unreal Engine 3 (UE3) (vgl. Epic Games, 2011).

Mit weiteren Updates für den Adobe Flash Player und Adobe Air zeigte Adobe, dass das Unternehmen Flash zu einer vollwertigen Spieleplattform ausbauen will. Nachdem Flash als Plattform für RIA-Inhalte (*Rich Internet Application*) immer mehr von HTML5, CSS3 und JavaScript verdrängt wird verwundert die Politikwende von Adobe nur wenig (vgl. Neumann, 2012).

Mit der neuen Stage 3D genannten 3D-API führte Adobe sehr viele Techniken ein, die bis dahin so nur von DirectX-Applikationen unter Windows bekannt waren. Unter Windows spricht Stage 3D die Hardware auch mit DirectX 9 an, aktuell ist im Moment DirectX 11, im Vergleich zu WebGL ist dies trotzdem ein immenser grafischer Unterschied. WebGL unterstützt lediglich das Shader Model 2.0 und Stage 3D, insofern DirectX 9 zur Verfügung steht, bereits das Shader Model 3.0. Unter Linux kommt OpenGL 1.3 zum Einsatz und auf Mobilgeräten das bereits bekannte OpenGL ES 2.0 (vgl. Fischer, 2011).

4.2 Microsoft XNA 3D

Im Kapitel 2.4.2 „Microsoft Silverlight“ wurde bereits auf die Entwicklungsgeschichte von Microsoft Silverlight und den 3D-Fähigkeiten eingegangen. Auf der MIX11 gab Microsoft bekannt, dass die von der XBOX 360 bekannte XNA 3D-Engine auf Silverlight 5 abgewandelt Einzug finden wird. Dies soll es ermöglichen, Computerspiele einfach und mit wenig Aufwand zu portieren (vgl. Oneal, 2011).

Wie bereits in Kapitel 2.4.2 „Microsoft Silverlight“ erwähnt fehlt Microsoft die Unterstützung aus der Spieleindustrie. Anders als Adobe, die bei ihrer Präsentation von

[36] http://max.adobe.com/ – Offizielle Website zu Adobe MAX.
[37] http://youtu.be/UQiUP2Hd60Y – UE3-Flash-Support-Video 1
[38] http://youtu.be/xzyCTt5KLKU – UE3-Flash-Support-Video 2

Stage 3D mit Industrieschwergewichten wie Epic Games oder einer Flash-Portierung eines der beliebtesten *casual games* überhaupt („Angry Birds“) aufwarten konnten.

Abbildung 15 *– Bildschirmfoto der XNA 3D „Babylon“ Technologie-Demo umgesetzt mit Silverlight 5.*
Weblink: http://david.blob.core.windows.net/babylon/Babylon.html
(eigene Darstellung)

4.3 Unity

Unity[39] ist eine Spiele-Engine die von Unity Technology entwickelt und angeboten wird. Ähnlich wie bei Adobe kann der Unity Web Player wie der Adobe Flash Player von Endbenutzern kostenlos verwendet werden. Zur Erstellung von Computerspielen stehen zwei Editoren zur Verfügung: Unity Free und Unity Pro. Wie die Namen bereits suggerieren ist Unity Free kostenlos erhältlich, jedoch stark beschnitten im Funktionsumfang, und Unity Pro kostenpflichtig.

Unity ist eine sehr ausgereifte Spielplattform die für die unterschiedlichsten Systeme bereit steht. Neben Desktop und Konsolen werden auch mobile Endgeräte und eben Webbrowser unterstützt. Laut einer Umfrage, die auf eine über zehnjährige Geschichte zurückblicken kann, von Mark Deloura aus dem Jahr 2011 wird Unity bei

[39] http://unity3d.com/ – Offizielle Unity Website.

61,8 % von traditionellen Studios eingesetzt und bei 68,4 % von *casual gaming* Studios. „*This is a big shift from two years ago, when Unreal was the clear winner,*“ (zu Deutsch: „Dies ist ein großer Umschwung von vor zwei Jahren, als Unreal noch der klare Gewinner war.“) ergänzt Deloura in seinem Artikel zu den Ergebnissen der Umfrage. Unreal belegte in beiden Kategorien (Traditionell und Casual) nun jeweils Platz 2. Bei den traditionellen Studios war der Anteil mit 60,7 % immer noch sehr hoch, bei den *casual gaming* Studios lag die Engine mit 43,7 % jedoch bereits stark abgeschlagen auf Platz 2 (vgl. Deloura, 2011).

Es existieren bereits diverse Spiele am Markt die komplett mit Unity umgesetzt wurden, so z. B. das erst kürzlich online gegangene Auto Club Revolution (ACR) von Eutechnyx[40]. Für die Rennen wird ein angepasster Client mit Unity eingesetzt, der Rest des Spiels – Event- und Wagenauswahl, Fahrzeugmanagement und Community – läuft komplett im Webbrowser ab (vgl. Steinlechner, 2012).

Abbildung 16 *– Bildschirmfoto eines Time-Trail-Rennens bei ACR basierend auf der Unity Engine.*
(eigene Darstellung)

[40] http://autoclubrevolution.com/ – Offizielle Auto Club Revolution Website.

5. Resümee

WebGL hat Zukunft, so viel steht fest. Dies ergibt sich bereits durch die breite Unterstützung aus der Industrie. Große Unternehmen wie nVIDIA, AMD, Intel, Google und Mozilla neben den großen Konsortien Khronos Group, W3C und Web3D, um nur die wichtigsten zu nennen. Trotzdem bin ich der Meinung, dass WebGL ein Schattendasein führen wird. Der Erfolg einer 3D-API ist stark an die dafür verfügbaren Applikationen gekoppelt. Im Moment gibt es unzählige Demos[41], jedoch kaum wirklich nützliche Real-World-Applikationen.

Der wohl wichtigste Erfolgsfaktor für eine 3D-API war, ist und bleibt jedoch die Unterstützung für die Spieleentwicklung. Böse Zungen behaupten, dass es Microsoft Windows ohne DirectX und die gesamte Spielerschaft rund um den Globus gar nicht mehr geben würde. Dies sei einmal dahingestellt, dass die Spieleindustrie und DirectX zum enormen Erfolg von Microsoft Windows beigetragen hat ist jedoch unumstritten. Hier haben Adobe und Unity im Moment die besten Karten. Während Unity im Webbereich noch nicht so stark vertreten ist, hat Adobe bereits vor vielen Jahren durch die Verwendung der Flash-Technologie durch YouTube die Grundsteine gelegt bekommen um der absolute Marktführer in diesem Segment zu werden. Es gibt abertausende Websites die Browser-Games auf Flash Basis anbieten, Umgangssprachlich wird oft schon von Flash-Games statt von Browser-Games gesprochen. Unity hat dafür den Vorteil für sehr viele Plattformen verfügbar zu sein. Ein Faktor der bei der ständig steigenden Plattformauswahl nicht außeracht gelassen werden darf. Sehr viele Spielekonzerne gaben in Interviews mehrfach an, dass sich ausschließlich Multiplattformentwicklungen rentieren in der heutigen Zeit. Nachdem Apple Adobes Flash komplett boykottiert und Konsolen keine Flash-Unterstützung bieten, steht hier Unity definitiv besser da. Microsoft spielt mit Silverlight und XNA bisher keine größere Rolle. Zwar bietet Microsoft alles was es für einen Erfolg benötigt: gute Grafikleistung, hohe Sicherheit, weite Verbreitung und plattformübergreifende Entwicklung. Trotzdem können sie die Gunst der Spieleindustrie nicht für sich gewinnen.

[41] http://www.chromeexperiments.com/webgl – Website von Google die WebGL-Demos sammelt.

Mit Stage 3D hat Adobe trotzdem eine sehr potente 3D-API eingeführt und Updates kommen in schnellen Zügen. Sicherheitsprobleme kennt Adobe bereits zur Genüge, zählen doch dessen Produkte zu den größten Sicherheitslücken der meisten Systeme (vgl. Eikenberg, 2011). Dem Unternehmen hat es bisher ausschließlich einen schlechten Ruf unter Fachleuten eingebracht, den Endbenutzer scheint es nicht sonderlich zu stören.

Ein weiterer wichtiger bestimmender Faktor in diesem Bereich der Industrie ist der Schutz der verwendeten Assets. Mit WebGL und JavaScript-Code ist dies nur schwer möglich. Der Code lässt sich höchstens durch die Verwendung von Obfuscators schwer lesbar machen, trotzdem steht jede Ressource via URL direkt für Interessierte (und die Konkurrenz) zum Download bereit. Dies muss auch so sein, um dem Gedanken des freien Webs gerecht zu werden. Technologien wie Adobe Flash, Microsoft Silverlight oder Unity ermöglichen es hingegen alle Bestandteile der Software in Binärform auszuliefern. Dadurch ist das geistige Eigentum eines Unternehmens geschützt und es lässt sich auch auf lange Sicht Geld damit verdienen.

Abschließend lässt sich sagen: „*Es geht heiß her in der dritten Dimension des Webs*." Nach über 15 Jahren stellt sich langsam aber sicher der Siegeszug – zumindest für Computerspiele – ein.

Literaturverzeichnis

(1) Kersten Auel: *X3D: 3D-Grafikstandard mit XML.* In: *heise online*. Heise Zeitschriften Verlag, 16. Februar 1999, abgerufen am 1. April 2012.
Weblink: http://heise.de/-14747

(2) Kersten Auel: *Adobe kündigt Flash 11 und AIR 3 an.* In: *heise online*. Heise Zeitschriften Verlag, 21. September 2011, abgerufen am 29. April 2012.
Weblink: http://heise.de/-1347115

(3) Johannes Behr, Yvonne Jung, Philipp Slusallek, Kristian Sons: *Dritte Dimension – X3DOM & XML3D: Deklaratives 3D in HTML5.* In: *iX Kompakt 2/2012 – Webdesign*. Verlag Heinz Heise, Hannover 8. Februar 2012, ISBN/ISSN: 9783936931952, S. 113–114.

(4) Gavin Bell, Anthony S. Parisi, Mark D. Pesce: *Version 1.0 Specification.* In: *The Virtual Reality Modeling Language*. Web3D Konsortium, 25. Januar 1996, abgerufen am 1. April 2012 (Englisch).
Weblink: http://www.web3d.org/x3d/specifications/vrml/VRML1.0/index.html

(5) Herbert Braun: *Google-Plug-in für 3D im Browser.* In: *c't News*. Heise Zeitschriften Verlag, 22. März 2009, abgerufen am 27. April 2012.
Weblink: http://heise.de/-214868

(6) Herbert Braun: *Microsoft stellt Silverlight 3 und Blend 3 vor.* In: *c't news*. Heise Zeitschriften Verlag, 18. März 2009, abgerufen am 26. April 2012.
Weblink: http://heise.de/-207648

(7) Herbert Braun: *Microsoft veröffentlicht Silverlight 3.* In: *heise online*. Heise Zeitschriften Verlag, 9. Juli 2009, abgerufen am 26. April 2012.
Weblink: http://heise.de/-6536

(8) Rachel Chalmers: *VRML Versus Chromeeffects: Microsoft Replies.* In: *CBR Online News*. Progressive Media Group, 16. September 1998, abgerufen am 1. Arpil 2012 (Englisch): „That Microsoft has lost faith in VRML is one of the industry's worst-kept secrets.“
Weblink: http://www.cbronline.com/news/vrml_versus_chromeffects_microsoft_replies

(9) Rachel Chalmers: *VRML VERSUS CHROMEFFECTS: MICROSOFT REPLIES.* In: *CBR Online News*. Progressive Media Group, 16. September 1998, abgerufen am 1. Arpil 2012 (Englisch): „That Microsoft has lost faith in VRML is one of the industry's worst-kept secrets.“
Weblink: http://www.cbronline.com/news/vrml_versus_chromeffects_microsoft_replies

(10) Mark Deloura: *Game Engine Survey 2011.* In: *Game Developer Magazine*. United Business Media, San Francisco 2011, S. 7–12.

(11) Diverse: *XTech 2006 Presentations.* In: *Mozilla Developer Network*. The Mozilla Foundation, 25. Mai 2006, abgerufen am 26. April 2012 (Englisch).
Weblink: https://developer.mozilla.org/en/XTech_2006_Presentations

(12) Diverse: In: *The WHATWG Wiki*. WHATWG, abgerufen am 26. April 2012 (Englisch). Weblink: http://wiki.whatwg.org/wiki/FAQ#What_does_.22Living_Standard.22_mean.3F

(13) Ronald Eikenberg: *Kaspersky-Studie: Adobe-Software größtes Sicherheitsrisiko.* In: *heise Security*. Heise Zeitschriften Verlag, 16. August 2011, abgerufen am 30. April 2012. Weblink: http://heise.de/-1323423

(14) Epic Games, *Unreal Engine 3 Support for Adobe Flash Player Announced.* Epic Games, 4. Oktober 2011, abgerufen am 29. April 2012 (Englisch). Weblink: http://epicgames.com/community/2011/10/unreal-engine-3-support-for-adobe-flash-player-announced/

(15) Paul Festa: *Microsoft shelves Chromeffects.* In: *CNET News*. CBS Interactive, 12. November 1998, abgerufen am 1. April 2012 (Englisch). Weblink: http://news.cnet.com/Microsoft-shelves-Chromeffects/2100-1023_3-217885.html

(16) Martin Fischer: *Adobe Stage 3D: Top-Spielegrafik im Internet-Browser.* In: *heise online*. Heise Zeitschriften Verlag, 5. Oktober 2011, abgerufen am 30. April 2012. Weblink: http://heise.de/-1355142

(17) James Forshaw: *WebGL - A New Dimension for Browser Exploitation.* In: *Blog*. Context Information Security, 11. Mai 2011, abgerufen am 29. April 2012 (Englisch). Weblink: http://contextis.com/resources/blog/webgl/

(18) James Forshaw: *WebGL - A New Dimension for Browser Exploitation - FAQ.* In: *Blog*. Context Information Security, abgerufen am 29. April 2012 (Englisch). Weblink: http://contextis.com/resources/blog/webgl/faq/

(19) James Forshaw, Paul Stone, Michael Jordon: *WebGL – More WebGL Security Flaws.* In: *Blog*. Context Information Security, 16. Juni 2011, abgerufen am 29. April 2012 (Englisch). Weblink: http://www.contextis.com/resources/blog/webgl2/

(20) Frauenhofer IGD, *About.* In: *x3dom Instant 3D the HTML way!*. Frauenhofer IGD, abgerufen am 27. April 2012 (Englisch). Weblink: http://www.x3dom.org/?page_id=2

(21) Federico Frenguelli: *X3D/HTML5 intermediated fallback model.* In: *X3DOM – Instant 3D the HTML way!*. Frauenhofer IGD, archiviert vom Original unter http://medea.googlecode.com/hg-history/db761b67461734cd76fe88ce2494cd4e87dc1272/imgs/x3dom-fallback-model.png am 6. September 2011, abgerufen am 27. April 2012 (PNG, Englisch). Weblink: http://www.x3dom.org/wp-content/uploads/2009/10/x3dom-fallback-Release-1.2.png

(22) Puneet Goel: *Flash Platform, "Molehill" and Gaming Updates.* In: *Adobe Flash Platform Blog*. Adobe Systems Incorporated, 1. März 2011, abgerufen am 26. April 2012 (Englisch). Weblink: http://blogs.adobe.com/flashplatform/2011/03/flash-platform-molehill-and-gaming-updates.html

(23) Golem.de, *Demonstration - XML3D, Echtzeit-Raytracing im Browser auf der Cebit 2010.* In: *Video*. Golem.de, 5. März 2010, abgerufen am 27. April 2010 (Flash-Video).

Weblink: http://video.golem.de/desktop-applikationen/2911/demonstration-xml3d-echtzeit-raytracing-im-browser-auf-der-cebit-2010.html

(24) Tobias Hager: *(T)Räume online – Was man mit VRML machen kann*. In: *iX*. Heise Zeitschriften Verlag, Hannover Mai 2000, S. 180 (Weblink: http://heise.de/-505804. 16. April 2000, abgerufen am 1. April 2012).

(25) Ian Hickson: In: *The WHATWG Blog*. WHATWG, 19. Jänner 2011, abgerufen am 26. April 2012 (Englisch).
Weblink: http://blog.whatwg.org/html-is-the-new-html5

(26) Emmy Huang: *Flash Games Showcased at Adobe MAX- Rovio's Angry Birds & Epic Games.* In: *Adobe Flash Platform Blog*. Adobe Systems Incorporated, 4. Oktober 2011, abgerufen am 30. April 2012 (Englisch, Direktlink zum Bild: http://blogs.adobe.com/flashplatform/files/2011/10/UT3_Sanctuary_FlashInBrowser_061.jpg).
Weblink: http://blogs.adobe.com/flashplatform/2011/10/flash-games-showcased-at-adobe-max-rovio's-angry-birds-epic-games.html

(27) Jens Ihlenfeld: *3D-API fürs Web.* Khronos und Mozilla entwickeln 3D-Schnittstelle für Browser auf Basis von OpenGL. In: *News*. Golem.de, 25. März 2009, abgerufen am 27. April 2012.
Weblink: http://www.golem.de/0903/66105.html

(28) Jens Ihlenfeld: *Kritik an Microsoft aus den eigenen Reihen.* Golem.de, 21. Juni 2011, abgerufen am 29. April 2012.
Weblink: http://www.golem.de/1106/84352.html

(29) Jens Ihlenfeld: *WebGL 1.0 ist fertig.* 3D im Browser. In: *News*. Golem.de, 4. März 2011, abgerufen am 27. April 2012.
Weblink: http://www.golem.de/1103/81890.html

(30) Jens Ihlenfeld: *WebGL ist gefährlich.* Golem.de, 17. Juni 2011, abgerufen am 29. April 2012.
Weblink: http://www.golem.de/1106/84275.html

(31) Margarete Jahrmann: *Von VRML97 zu Web3D.* In: *Telepolis*. Heise Zeitschriften Verlag, 26. März 1999, abgerufen am 1. April 2012.
Weblink: http://www.heise.de/tp/artikel/3/3357/1.html

(32) Joyent, *Node's goal is to provide an easy way to build scalable network programs.* In: *node.js*. Joyent, Inc., abgerufen am 27. April 2012 (Englisch).
Weblink: http://nodejs.org/about/

(33) Khronos Group, *Khronos Releases Final WebGL 1.0 Specification.* In: *Press*. Khronos Group, 3. März 2011, abgerufen am 27. April 2012 (Englisch, Kein Direktlink möglich.).
Weblink: http://www.khronos.org/news/press/2011/03

(34) Khronos Group, *WebGL Security.* In: *News*. Khronos Group, 9. Mai 2011, abgerufen am 29. April 2012 (Englisch).
Weblink: http://www.khronos.org/news/permalink/webgl-security

(35) Christian Kirsch: *Microsoft hält WebGL für zu unsicher*. In: *iX News-Meldung*. Heise Zeitschriften Verlag, 17. Juni 2011, abgerufen am 29. April 2012.
Weblink: http://heise.de/-1262166

(36) Kai König: *Kais bewegtes Web: Flash Player 11 und AIR 3*. In: *heise Developer*. Heise Zeitschriften Verlag, 4. Oktober 2011, abgerufen am 29. April 2012.
Weblink: http://www.heise.de/developer/artikel/Flash-Player-11-und-AIR-3-1354338.html

(37) Alex Lash, Michael Kanellos: *Microsoft buffs its Chrome*. In: *CNET News*. CBS Interactive, 26. März 1998, abgerufen am 1. April 2012 (Englisch).
Weblink: http://news.cnet.com/Microsoft-buffs-its-Chrome/2100-1033_3-209559.html

(38) Robert Lippert: *Fabric Engine bringt JavaScript auf C++-Niveau*. In: *heise Developer*. Heise Zeitschriften Verlag, 30. März 2012, abgerufen am 27. April 2012.
Weblink: http://www.heise.de/developer/meldung/Fabric-Engine-bringt-JavaScript-auf-C-Niveau-1495054.html

(39) Chris Marrin: *4.3 Supported GLSL Constructs*. In: *WebGL Specification*. Khronos Group, 27. April 2012, abgerufen am 29. April 2012 (Englisch).
Weblink: http://www.khronos.org/registry/webgl/specs/latest/#4.3

(40) Carlos U. Matesanz: *History*. In: *Blog*. Papervision3D.org, 2006, abgerufen am 26. April 2012 (Englisch).
Weblink: http://blog.papervision3d.org/about/

(41) Jeff Muizelaar: Jeff Muizelaar, 16. Juni 2011, abgerufen am 29. April 2012 (Englisch, Jeff Muizelaar ist Entwickler bei The Mozilla Foundation).
Weblink: http://muizelaar.blogspot.com/2011/06/webgl-considered-harmful.html

(42) Aaftab Munshi, Daniel Ginsburg, David Shreiner: 2008. *OpenGL ES 2.0 Programming Guide*. 1. Hrsg. Amsterdam Addison-Wesley Longman,

(43) Alexander Neumann: *GDC: 3D im Browser – WebGL 1.0 ist fertig*. In: *heise online*. Heise Zeitschriften Verlag, 4. März 2011, abgerufen am 27. April 2012.
Weblink: http://heise.de/-1201976

(44) Alexander Neumann: *Flash Player 11.2 mit Premium-Funktionen für Spieleprogrammierer*. In: *heise Developer*. Heise Zeitschriften Verlag, 28. März 2012, abgerufen am 29. April 2012.
Weblink: http://heise.de/-1485286

(45) Victor Nicollet: *Adobe Flash 10 : 3D in your browser*. In: *Nicollet.Net*. 10. Dezember 2008, abgerufen am 26. April 2012 (Englisch).
Weblink: http://www.nicollet.net/2008/12/flash-10-3d/

(46) Aaron Oneal: *Graphics & 3D with Silverlight 5*. In: *MIX11*. Microsoft, 13. April 2011, abgerufen am 30. April 2012 (Video, verschiedene Formate, Englisch).
Weblink: http://channel9.msdn.com/Events/MIX/MIX11/MED06

(47) Matt Papakipos: *The future of O3D*. In: *The Chromium Blog*. Google, Inc., 7. Mai 2010, abgerufen am 29. April 2012 (Englisch).
Weblink: http://blog.chromium.org/2010/05/future-of-o3d.html

(48) Marc D. Pesce: *ADMIN: BOF SIGGRAPH Meeting Notes.* In: *www-vrml mailing list.* 25. Juli 1994, archiviert vom Original unter http://1997.webhistory.org/www.lists/www-vrml.1994/0385.html, abgerufen am 31. März 2012 (Englisch).

(49) Marc D. Pesce: *MISC: Labyrinth at SIGKIDS '94 Report.* In: *www-vrml mailing list.* 2. August 1994, archiviert vom Original unter http://1997.webhistory.org/www.lists/www-vrml.1994/0397.html, abgerufen am 1. April 2012 (Englisch, Inklusive Quelltext der ersten VRML-Demo).

(50) Marc D. Pesce, Brian Behlendorf: *www-vrml mailing list.* 28. März 1995, archiviert vom Original unter http://www.w3.org/Mail/WWW_VRML.html, abgerufen am 1. April 2012 (Englisch).

(51) Marc D. Pesce, Peter Kennard, S. A. Parisi: *Cyberspace.* In: *Proceedings of The First International Conference on The World-Wide Web*. Labyrinth Group, archiviert vom Original unter http://web.archive.org/web/19970217210541/http://www.cern.ch/PapersWWW94/mpesce.ps am 17. Februar 1997, abgerufen am 1. April 2012 (PostScript, Englisch).

(52) Dave Raggett: *Extending WWW to support Platform Independent Virtual Reality.* Web3D Consortium, 1. Quartal 1994, abgerufen am 31. März 2012 (Englisch).
Weblink: http://www.w3.org/People/Raggett/vrml/vrml.html

(53) Arun Ranganathan: *WebGL Draft Released Today.* In: *hacks.mozilla.org Articles*. The Mozilla Foundation, 10. Dezember 2009, abgerufen am 27. April 2012 (Englisch).
Weblink: http://hacks.mozilla.org/2009/12/webgl-draft-released-today/

(54) Holger Schwichtenberg: *.NET neu erfunden in Native Code und COM.* Windows 8 Apps benötigen neue Windows Runtime. In: *heise Developer*. Heise Zeitschriften Verlag, 15. September 2011, abgerufen am 26. April 2012.
Weblink: http://www.heise.de/developer/artikel/Windows-8-Apps-benoetigen-neue-Windows-Runtime-1344071.html

(55) Security Research & Defense, *WebGL Considered Harmful.* In: *TechNet Blogs*. Microsoft Corporation, 16. Juni 2011, abgerufen am 29. April 2012 (Englisch).
Weblink: http://blogs.technet.com/b/srd/archive/2011/06/16/webgl-considered-harmful.aspx

(56) Kristian Sons: *Welcome to the Declarative 3D Community Group.* In: *W3C Community and Business Groups*. Declarative 3D for the Web Architecture Community Group, 30. Juni 2011, abgerufen am 30. April 2012 (Englisch).
Weblink: http://www.w3.org/community/declarative3d/2011/06/30/welcome-to-the-declarative-3d-community-group/

(57) Kristian Sons: *Mit Licht und Schatten: JavaScript-Frameworks für WebGL.* In: *iX Kompakt 2/2012 – Webdesign*. Verlag Heinz Heise, Hannover 8. Februar 2012, ISBN/ISSN: 9783936931952, S. 125–131.

(58) Sons, K. et al., *Declarative 3D for the Web Architecture Community Group Charter*. In: *W3C Community*. W3C, 30. Juni 2011, abgerufen am 30. April 2012 (Englisch).
Weblink: http://www.dfki.de/~sons/charter/

(59) Peter Steinlechner: *Virtueller Motorsportverein im offenen Betatest.* Golem.de, 30. März 2012, abgerufen am 30. April 2012.
Weblink: http://www.golem.de/1203/90867.html

(60) Peter Strohm: *Quellcode-Matroschka – WebGL-Tutorial, Teil 1: Grundlagen.* In: *iX 5/2011*. Heinz Zeitschriften Verlag, Hannover Mai 2011, S. 117–121.

(61) Daniel Terdiman: *Cool Web front-end for multiple virtual world entry.* In: *CNET News*. CBS Interactive, 11. Oktober 2007, abgerufen am 1. April 2012 (Englisch).
Weblink: http://news.cnet.com/8301-13772_3-9796393-52.html

(62) The Khronos Group, *Khronos Launches Initiative to Create Open Royalty Free Standard for Accelerated 3D on the Web.* Open call for industry participation and contributions; Project initiated by Mozilla. In: *Press*. The Khronos Group, 24. März 2009, abgerufen am 27. April 2012 (Englisch, Direktlink nicht möglich).
Weblink: http://www.khronos.org/news/press/2009/03

(63) Arthur J. Thomas: *please let me introduce myself.* In: *www-vrml mailing list*. 9. Juni 1994, archiviert vom Original unter http://1997.webhistory.org/www.lists/www-vrml.1994/0000.html, abgerufen am 31. März 2012 (Englisch): „I attended the workshop on Virtual Reality at WWW94 in Geneva, and can confirm the excitement that people felt [...]“

(64) Giles Thomas: *Lesson 5.* In: *Learning WebGL*. Giles Thomas, 2. November 2009, abgerufen am 29. April 2012 (Englisch).
Weblink: http://learningwebgl.com/lessons/lesson05/index.html

(65) TIOBE, *The JavaScript Programming Language.* TIOBE Software BV, abgerufen am 27. April 2012 (Englisch).
Weblink: http://www.tiobe.com/index.php/paperinfo/tpci/JavaScript.html

(66) Gerhard Völkl: *3D-Darstellungen im Webbrowser: Zielgerichtet.* In: *iX Kompakt 2/2012 – Webdesign*. Verlag Heinz Heise, Hannover 8. Februar 2012, ISBN/ISSN: 9783936931952, S. 110–111 (Weblink: http://heise.de/-1108267. 20. Oktober 2011, abgerufen am 1. April 2012).

(67) Vladimir Vukićević: *Canvas 3D: GL power, web-style.* In: *Words*. Vladimir Vukićević, 26. November 2007, abgerufen am 27. April 2012 (Englisch).
Weblink: http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style/

(68) Vladimir Vukićević: *Canvas 3D Extension Update.* In: *Words*. Vladimir Vukićević, 28. März 2009, abgerufen am 27. April 2012 (Englisch).
Weblink: http://blog.vlad1.com/2009/03/28/canvas-3d-extension-update-2/

(69) W3C, *Results of Questionnaire Shall we Adopt HTML5 as our specification text for review?.* W3C, 4. Mai 2007, abgerufen am 26. April 2012 (Englisch).
Weblink: http://www.w3.org/2002/09/wbs/40318/htmlbg/results

(70) James A. Wagner: *Movable Metaverse: Japanese company creates 2nd Web-based Second Life browser*. In: *New World Notes*. 3. Oktober 2007, abgerufen am 1. April 2012 (Englisch).
Weblink: http://nwn.blogs.com/nwn/2007/10/second-move-jap.html

(71) Web3D Consortium, *Extensible 3D (X3D), ISO/IEC 19775-2:2004.* Part 2: Scene access interface (SAI). In: *X3D*. Web3D Consortium, abgerufen am 1. April 2012 (Englisch).
Weblink: http://www.web3d.org/x3d/specifications/OLD/ISO-IEC-19775-X3DAbstractSpecification_+_Amendment1_to_Part1/Part02/SAI.html

(72) Web3D Konsortium, *The Virtual Reality Modeling Language.* abgerufen am 1. April 2012 (Englisch).
Weblink: http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/

(73) Randy Weston: *Microsoft debuts Chromeffects.* In: *CNET News*. CBS Interactive, 21. Julie 1998, abgerufen am 1. April 2012 (Englisch).
Weblink: http://news.cnet.com/2100-1001-213566.html

(74) WHATWG, *WHAT open mailing list announcement.* WHATWG, 4. Juni 2004, abgerufen am 26. April 2012 (Englisch).
Weblink: http://www.whatwg.org/news/start

(75) x3dom, *1.3 Release.* In: *Releases*. x3dom – Instant 3D the HTML way!, 23. Dezember 2011, abgerufen am 28. April 2012 (Englisch).
Weblink: http://www.x3dom.org/?p=2677

(76) Alexander Zadorozhny: *Away3D flash engine.* 2009, abgerufen am 26. April 2012 (Englisch).
Weblink: http://away.kiev.ua/away3d/

Glossar

(1) **API** – Application Programming Interface, zu Deutsch Schnittstelle zur Anwendungsprogrammierung, generell als Programmierschnittstelle bezeichnet. Hierbei handelt es sich um einen Programmteil, der auf Quelltextebene anderen Programmen vorgibt wie diese sich in das Softwaresystem integrieren können.

(2) **CPU** – Central Processing Unit, zu Deutsch Hauptprozessor, generell als Prozessor bezeichnet. Dabei handelt es sich um die zentrale Verarbeitungseinheit (ZVE) eines Computers. Programme und Berechnungen werden hier ausgeführt.

(3) **CSS** – Cascading Style Sheets, deklarative Formatierungssprache zur Formatierung von Inhalten im World Wide Web.

(4) **Direct3D** – Siehe DirectX.

(5) **DirectX** – Sammlung von Programmierschnittstellen (APIs) für multimediale Anwendungen von Microsoft. DirectX umfasst u. a. Direct3D, DirectSound, DirectMusic, DirectInput, etc.

(6) **GLSL** – OpenGL Shading Language (auch GLslang genannt), ist eine höhere Programmiersprache die eine C-ähnliche Syntax bietet und es OpenGL basierten Applikationen ermöglicht einer GPU direkt Anweisungen über die Rendering-Pipeline zu übergeben, ohne dabei auf proprietäre Sprachen oder Assembler zurückgreifen zu müssen (siehe auch Kapitel 3.2 „Funktionsweise“).

(7) **OpenGL** – Open Graphics Library, eine freie, quelloffene und plattformunabhängige Spezifikation zur Implementierung von Programmierschnittstellen (APIs) zur Realisierung von 3D-Computergrafiken (siehe auch Kapitel 3.2 „Funktionsweise“).

(8) **OpenGL ES** – Wie OpenGL, jedoch mit abgezielt auf eingebettete Systeme (siehe auch Kapitel 3.2 „Funktionsweise“).

(9) **GPU** – Graphics Processing Unit, zu Deutsch Grafikprozessor. Hierbei handelt es sich um einen Mikroprozessor der die Berechnungen zur Darstellung des Bildes auf dem Bildschirm übernimmt. Es gibt verschiedenste Ausbauformen (Integrated Graphics Processor (IGP), Accelerated Processing Unit (ALU) und als Erweiterungssteckkarte).

(10) **HTML** – Hypertext Markup Language, textbasierte Auszeichnungssprache zur Strukturierung von multimedialen Inhalten im World Wide Web. HTML ist eine Untermenge von SGML.

(11) **HTTP** – Hypertext Transfer Protocol, Protokoll zur Übertragung von Daten in einem Netzwerk.

(12) **IEC** – International Electrotechnical Commission, zu Deutsch International Electrotechnical Commission. Internationale Vereinigung von Normungsorganisationen die internationale Normen im Bereich der Elektrotechnik festlegen [www.iec.ch].

(13) **ISO** – International Organization for Standardization, zu Deutsch Internationale Organisation für Normung. Internationale Vereinigung von Normungsorganisationen die internationale Normen festlegen [www.iso.org].

(14) **O3D** – Google Open Third Dimension, ursprünglich als Webbrowser-Plug-In geplant wird das Projekt jetzt als JavaScript-Framework für WebGL weiterentwickelt und soll Web-Entwicklern dabei Unterstützen einfach 3D-Computergrafiken im World Wide Web zu realisieren (siehe auch Kapitel 3.3.6 „Google Open 3D (O3D)“).

(15) **SAI** – Scene Access Interface, eine Schnittstelle die es ermöglicht zur Laufzeit eine X3D-Szene zu modifizieren (siehe auch Kapitel 2.3 „Extensible 3D (X3D)“).

(16) **SGML** – Standard Generalized Markup Language, ist eine Metasprache mit deren Hilfe sich Auszeichnungssprachen für Dokumente definieren lassen.

(17) **SGI** – Silicon Graphics International, Hersteller von Computer, Workstations und Grafiklösungen.

(18) **SIGGRAPH** – Special Interest Group on Graphics and Interactive Techniques, Gruppe der Association for Computing Machinery (ACM) die sich im Rahmen einer Tagungsreihe die seit 1974 stattfindet jährlich trifft.

(19) **SVG** – Scalable Vector Graphics, ist eine deklarative, textbasierte Auszeichnungssprache zur Erstellung von vektorbasierten Grafiken im World Wide Web.

(20) **URL** – Uniform Resource Locator, identifiziert eine Ressource eindeutig innerhalb eines gegebenen Netzwerkes (z. B. HTTP, FTP, etc.).

(21) **VAG** – VRML Architecture Group, Arbeitsgruppe zur Standardisierung von VRML.

(22) **VRML** – Virtual Reality Modeling Language, vor 1995 Virtual Reality Markup Language, erster Versuch 3D-Computergrafiken ins Web zu überführen (siehe auch Kapitel 2.1 „Virtual Reality Modeling Language“).

(23) **VRML2** – Siehe VRML97.

(24) **VRML97** – Der erste anerkannte VRML-Standard, zertifiziert durch die ISO und festgehalten im Standard ISO/IEC 14772.

(25) **VRML99** – Der letzte Kongress der unter dem Namen VRML abgehalten wurde. Interessante Randnotiz: Der Österreicher Christian Bauer [www.chrisbauer.com] zeichnete sich dafür verantwortlich, dass diese Veranstaltung in Europa stattfand statt in Kalifornien (USA), wie bisher.

(26) **W3C** – World Wide Web Consortium, Konsortium, geleitet von Tim Berners-Lee, gegründet zur Standardisierung offener Standards im World Wide Web.

(27) **Web** – Abkürzung von World Wide Web.

(28) **Web3D** – World Wide Web Third Dimension, Konsortium gegründet zur Standardisierung offener Standards für 3D-Computergrafiken im World Wide Web.

(29) **WebGL** – World Wide Web Open Graphics Library, freie, quelloffene und plattformunabhängige Spezifikation zur Implementierung von Programmierschnittstellen (APIs) zur Realisierung von 3D-Computergrafiken in Webbrowsern ohne Plug-Ins über das HTML *canvas*-Element. Die Umsetzung basiert dabei auf JavaScript (respektive ECMAScript) und der Shader-Sprache GLSL ES (siehe auch Kapitel 3 „WebGL“).

(30) **WHATWG** – Web Hypertext Application Technology Working Group, ist eine Arbeitsgruppe die von den führenden Webbrowser-Herstellern Mozilla und Opera ins Leben gerufen wurde nachdem die Unternehmen mit der Politik des W3Cs unzufrieden waren. Ziel der Arbeitsgruppe ist es, Vorschläge für neue Web-Standards zu definieren und diese dem W3C vorzulegen um sie in einen gültigen Standard für das World Wide Web zu überführen.

(31) **WWW** – Abkürzung von World Wide Web.

(32) **X3D** – Extensible Third Dimension, ein offener Standard für 3D-Computergrafiken im Web des Web3D-Konsortiums. X3D wurde von VRML abgeleitet (siehe auch Kapitel 2.3 „Extensible 3D (X3D)“).

(33) **X3DOM** – Extensible Third Dimension Object Model, Initiative zur Definition eines deklarativen Standards zur Erstellung von 3D-Computergrafiken im World Wide Web (siehe auch Kapitel 3.4 „Deklaratives 3D“).

(34) **XHTML** – Extensible Hypertext Markup Language, siehe HTML, im Kontrast zu HTML wurde XHTML anhand von XML definiert, welches eine strengere Untermenge von SGML darstellt. Dadurch ist XML strikter in der Anwendung, jedoch einfacher zu Parsen.

(35) **XML** – Extensible Markup Language, Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten. XML stellt eine Untermenge von SGML dar.

(36) **XML3D** – Extensible Markup Language Third Dimension, Initiative zur Definition eines deklarativen Standards zur Erstellung von 3D-Computergrafiken im World Wide Web (siehe auch Kapitel 3.4 „Deklaratives 3D“).