Hochschule Mittweida (FH)
University of Applied Sciences

Fachbereich Medien

Stefan von der Krone

„Analytischer Vergleich der neuen dreidimensionalen Fähigkeiten des Flash Player 10 mit gängigen 3D-Engines für ActionScript 3.0 auf Basis des Flash Player 9“

– Bachelorarbeit –

Hochschule Mittweida –
University of Applied Science (FH)

Leipzig, 2009

Hochschule Mittweida (FH)
University of Applied Sciences

Fachbereich Medien

Stefan von der Krone

„Analytischer Vergleich der neuen dreidimensionalen Fähigkeiten des Flash Player 10 mit gängigen 3D-Engines für ActionScript 3.0 auf Basis des Flash Player 9“

– Eingereicht als Bachelorarbeit –

Hochschule Mittweida –
University of Applied Science (FH)

Erstprüfer: Prof. Dr.-Ing. Robert J. Wierzbicki
Zweitprüfer: Dipl.-Ing. (FH) Stefan Dittmar

Die vorgelegte Arbeit wurde eingereicht am:
12. Oktober 2009

Leipzig, 2009

Von der Krone, Stefan:

„Analytischer Vergleich der neuen dreidimensionalen Fähigkeiten des Flash Player 10 mit gängigen 3D-Engines für ActionScript 3.0 auf Basis des Flash Player 9“, 2008, 99 Seiten

Hochschule Mittweida – University of Applied Science (FH), Fachbereich Medien, Bachelorarbeit

Referat

Die vorliegende Bachelorarbeit beschäftigt sich mit den neuen nativen Transformationsmöglichkeiten des Flash Players in der Version 10. Zunächst wird ein näherer Blick auf die Geschichte der Technologie Flash und im Besonderen auf dessen Scriptsprache ActionScript geworfen. Im Anschluss werden die konkreten Untersuchungsbeispiele vorgestellt und die Untersuchungsmethoden erläutert. Auf Grundlage der erhobenen Daten erfolgt am Ende ein qualifiziertes Fazit.

Inhaltsverzeichnis

Verzeichnisse

I. Abbildungsverzeichnis

II. Tabellenverzeichnis

Vorwort

Das Thema dieser Bachelorarbeit wählte ich aufgrund meiner Faszination für die Technologie Flash, meiner Freude am Programmieren sowie der stetigen Entwicklung in der Wahrnehmung des Internets. Mit dem Erfolg des Web2.0s wurden die Benutzeroberflächen von Internetseiten immer plastischer. Der Logische Schritt ist somit auch der Gang in die dritte Dimension. An einer allgemeinen Spezifikation in Form von WebGL[1] wird seit geraumer Zeit gearbeitet, aber Flash ist auf diesem Gebiet schon etwas länger vertreten. Und mit dem Erscheinen der zehnten Version des Flash Players war es nur logisch, sich damit zu beschäftigen.

Außerordentlichen Dank gilt dabei all meinen Freunden, im Besonderen Thomas „Eddie“ Dietze und Clemens Petzold für die inhaltliche sowie Anja Ebert, Sarah Gerischer, Anja Langrock, Florian Reinke und Diane Scheffler für ihre persönliche Unterstützung. Meinen Kollegen von PLUSPOL interactive möchte ich für die enorme Unterstützung danken, besonders den Geschäftsführern Jörg Brückner, Thomas Lange und Stefan Dittmar, der sich auch als Zweitprüfer engagierte. Natürlich gilt auch mein Dank dem Fachbereich Medien der Fachhochschule Mittweida, ganz besonders Frau Sieglinde Klimant und Professor Wierzbicki für die Inspiration und ihr Vertrauen, das sie mir entgegenbrachten.

[1] http://www.khronos.org/news/press/releases/khronos-webgl-initiative-hardware-accelerated-3d-graphics-internet/

Diese Arbeit widme ich all diesen Menschen und auch den Flashern da draußen für ihren unermüdlichen Einsatz.

Stefan von der Krone, Oktober 2009

P.S.:

Fragen und Anregungen zur Bachelorarbeit können jederzeit per Mail an skrone@htwm.de gerichtet werden.

1. Einleitung

Am 15. März 2008 veröffentlichte die Adobe Systems Incorporated die erste Beta-Version des Flash Player 10.[2] [3] Mit ihr wurden auch gleich erste Beispiele zu den Fähigkeiten des neuen Releases auf einer eigens dafür eingerichteten Website auf Adobes Plattform labs.adobe.com[4] bereitgestellt. Als größte Neuerung wurden die erweiterten nativen Darstellungsmöglichkeiten angepriesen. Dazu zählte neben einer deutlich verbesserten Textrenderingengine[5] auch die Unterstützung für die dreidimensionale Darstellung von graphischen Objekten[6]. Letzteres scheint eine Reaktion seitens Adobe auf die wachsende Anzahl von 3D-Frameworks für den Flash Player 9 und ActionScript 3.0 zu sein. Bekannte Implementierungen dafür sind unter anderem Papervision3D[7] und Away3D[8], die unter einer Open-Source-Lizenz angeboten werden, sowie das kommerzielle Alternativa3D[9]. Diese beispielhaft genannten 3D-Engines haben die Möglichkeiten und Fähigkeiten des Flash Player 9 in Verbindung mit ActionScript 3.0 aufgezeigt, aber

---

[2] http://www.golem.de/showhigh2.php?file=/0805/59705.html

[3] http://www.heise.de/newsticker/Erste-Beta-des-Flash-Player-10-liegt-vor--/meldung/107905

[4] http://labs.adobe.com/technologies/flashplayer10/demos/

[5] http://www.adobe.com/devnet/logged_in/jchurch_flashplayer10.html

[6] http://labs.adobe.com/technologies/flashplayer10/demos/

[7] http://www.papervision3d.org

[8] http://www.away3d.com

[9] http://www.alternativaplatform.com/en/

auch deren Grenzen offenbart. Durch die enorm verbesserte Leistungsfähigkeit des Flash Players – hier speziell die komplett neue AVM2[10] – ermöglichte seit Erscheinen des Flash Player 9 eine rapide Entwicklung hin zu Präsentationen in 3D. Es gab zwar schon früher ähnliche Ansätze mit älteren Versionen des Flash Players, diese scheiterten aber an deren begrenzter Leistungsfähigkeit.

Hier kommt der neue Flash Player 10 ins Spiel, da er, wie eingangs beschrieben, Unterstützung für die beschleunigte dreidimensionale Darstellung graphischer Objekte bietet. Das heißt, dass im Idealfall kein komplizierter Algorithmus geschrieben, sondern einfach nur eine native Funktion oder Eigenschaft eines graphischen Objekts bemüht werden muss. Beispiele hierfür sind die Rotation um die drei Achsen sowie die Bewegung eines Objektes entlang der z-Achse mit korrekter perspektivischer Veränderung.

Ziel dieser wissenschaftlichen Arbeit ist der Vergleich der Leistungsfähigkeit von bekannten 3D-Frameworks für den Flash Player 9, mit der nativen 3D-Unterstützung des Flash Player 10 mittels verschiedener Kriterien. Angesichts des immerwährenden Fortschritts bei den Webtechnologien, ist diese Arbeit vor allem für Flashentwickler interessant, aber genauso auch für Spieleentwickler, die sich in jüngster Vergangenheit mit den bekannten 3D-Engines für ActionScript 3.0 befasst haben. Weiterhin wären neue Bedienkonzepte in Flash denkbar,

[10] ActionScript 3.0 Virtual Machine, die Virtuelle Maschine zur Ausführung von ActionScript-3.0-Code

die auf die native 3D-Beschleunigung aufbauen, genauso wie Produktpräsentationen in 3D[11]. Zur Untersuchung werden drei einfache Anwendungsbeispiele, die eine räumliche Darstellung ermöglichen, verwendet. Diese werden in separaten Kapiteln näher erläutert. Um eine Vergleichbarkeit zu gewährleisten, werden alle Anwendungsbeispiele in vier Versionen programmiert – eine für den Flash Player 10 sowie drei für den Flash Player 9. Letztere implementieren dabei die 3D-Frameworks Alternativa3D, Away3D sowie Papervision3D. Diese kleinen Programme werden dann anhand dreier Grundkriterien untersucht:

- Resourcenverbrauch (Höhe des Speicherverbrauchs sowie Auslastung des Systems)
- Aufwand (Programmieraufwand, Komplexität der Algorithmen)
- Qualität (Bildqualität, sonstige Auffälligkeiten bei der Darstellung)

Die einzelnen Kriterien werden im späteren Verlauf der Arbeit näher erläutert.

[11] http://www.adobe.com/devnet/logged_in/jchurch_flashplayer10.html

2. Zielsetzung

In der Online-Werbewirtschaft ist es im Zusammenhang mit der Technologie Flash essentiell, sich bei einem Projekt Gedanken über wichtige technische Aspekte des Endproduktes zu machen. Unter anderem stellen sich dabei gewöhnlich folgende Fragen:

- Welche Flash Player Version wird als minimale Voraussetzung benötigt?
- Wie aufwendig ist die grafische Darstellung?
- Wie hoch ist die maximal erlaubte Dateigröße des Flash-Films?

Die erste Frage nach der Version des benötigten Flash Players richtet sich hauptsächlich nach der angepeilten Zielgruppe. Allgemein nimmt man an, dass es einfach eine möglichst alte Flash Player Version sein sollte. Branchenüblich wäre derzeit wohl Version 8 aus dem Jahr 2005. In die Entscheidung nach der Version des Flash Players bezieht man für gewöhnlich auch die sogenannte „Flash Player Version Penetration“ mit ein. Diese trifft eine Aussage über die allgemeine Verbreitung des Flash Players im Internet. Dabei werden verschiedene Webstatistiken ausgewertet. Adobe stellt die Ergebnisse der Erhebung auf ihrer Website zur Verfügung[12]. Bei sogenannten Microsites,

[12] http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

also kleineren Websites komplett in Flash, nimmt man für gewöhnlich eine neuere Version.

Die zweite Frage entscheidet ebenfalls über die Flash Player Version. Bei einfachen grafischen Darstellungen spielt diese meist keine Rolle. Soll es aufwendiger werden, zum Beispiel mit Weichzeichnung oder Bitmap-Manipulation, dann ist Flash Player 8 als Minimum Pflicht. Geht das Ganze dann auch noch in die dritte Dimension, dann kommt man um ActionScript 3.0 und mindestens Flash Player 9 nicht mehr herum, da ältere Versionen einfach nicht leistungsfähig genug für die dazu erforderlichen Berechnungen sind. Über die genauen Fähigkeiten und Besonderheiten der Flash Player Versionen wird in einem späteren Kapitel ausführlich eingegangen.

Die dritte Frage betrifft eigentlich hauptsächlich Werbebanner. Im Allgemeinen beträgt die Dateigröße eines Werbebanners maximal 30 Kilobyte – aufwendige Grafiken dabei nicht mehr sinnvoll. Eine räumliche Darstellung ist deshalb eigentlich gar nicht mehr möglich, dies ginge dann nur noch über entsprechende zweidimensionale Verzerrungen. Microsites sind von einem Dateigrößen-Limit nur selten betroffen. Ein Art unverbindliches Limit liegt hier bei maximal drei bis fünf Megabyte. Deutlich höhere Volumen sind eher nicht ratsam. Gestreamte oder dynamisch nachgeladene Inhalte betreffen aber nicht.

Wie bereits in der Einleitung erwähnt, soll es nun möglich sein, grafische Objekte, sogenannte *DisplayObjects*, dreidimensional zu transformieren. Dazu ist kein spezielles 3D-

Framework wie Papervision3D notwendig, dies geschieht nunmehr nativ in der virtuellen Maschine des Flash Players. Der Ballast durch ein entsprechendes Framework entfällt dadurch, was in einer geringeren Dateigröße des Flash-Films resultiert. Nutzt man alle Möglichkeiten der aktuellen Beta-Version von Papervision3D, dann müsste man mit etwa 450 Kilobyte zusätzlichen Speicherverbrauchs rechnen, bei simplen 3D-Szenen wären es immer noch etwa 100 Kilobyte. Für Werbebanner ist das natürlich deutlich zu viel. Mit den nativen Möglichkeiten des neuen Flash Players wäre eine primitive 3D-Szene als Werbebanner ohne Weiteres möglich und platzsparend, was ohne Zweifel einen Vorteil für den neuen Flash Player 10 bedeutet.

Des Weiteren kann es bei aufwendigen Microsites zu enormen Leistungseinbrüchen unter anderem durch erhöhten Speicherverbrauch kommen. Ältere Computer sind davon meist aufgrund geringeren Arbeitsspeichers stärker betroffen. Anders könnte dies mit dem neuen Flash Player aussehen – zumindest theoretisch. An dieser Stelle sollte erwähnt werden, dass die Nutzung von 3D-Frameworks nicht zwangsläufig mit den oben genannten Problemen einhergehen muss, es besteht aber eine erhöhte Gefahr.

Hier setzt diese Bachelorarbeit an. Mit ihr soll untersucht werden, inwieweit die Neuerungen des neuesten Flash Players hier Besserung versprechen. Damit soll sie auch eine Hilfestellung für Flash-Entwickler sein, fundierte Entscheidungen und Empfehlungen für zukünftige Projekte machen zu können. Mit Hilfe der Untersuchung bestimmter Aspekte des

neuen Flash Player 10 soll eine Aussage über den effektiven Einsatz der neuen Möglichkeiten getroffen werden. Es stellt sich dabei die Frage, ob der Einsatz eines 3D-Frameworks in Verbindung mit dem Flash Player 9 lohnt oder ob man nicht doch auf den Flash Player 10 setzt, der eine eventuell bessere Leistung mit geringerem Speicherverbrauch bietet, auch wenn man dadurch eine eingeschränkte Reichweite durch die derzeitige Verbreitung von Version 10 hinnehmen muss.

3. Flash und ActionScript

Das nun folgende Kapitel beschäftigt sich ausführlicher mit der Technologie Flash und ihrer Geschichte. Einen besonderen Schwerpunkt dabei bildet ActionScript, dessen Entwicklung und Sprach-Elemente im Hinblick auf das Thema dieser Arbeit genauer betrachtet werden.

Flash ist eine plattformübergreifende Technologie zur Darstellung interaktiver und multimedialer Inhalte, die mittlerweile auf vielfältige Art und Weise im Einsatz ist. Ursprünglich als Plugin für den Netscape-Browser und später auch den Internet Explorer zur Darstellung animierter Vektorgrafiken entwickelt, kamen im Laufe der Zeit weitere Einsatzgebiete hinzu. So traten mit dem Erfolg des sogenannten Web2.0 die Video- und Audio-Funktionalitäten mehr und mehr in den Vordergrund, was besonders am enormen Erfolg von Youtube und MySpace erkennbar ist.[13]

Mittlerweile wird Flash auch auf mobilen Geräten in Form von FlashLite eingesetzt, einer abgespeckten Version des eigentlichen Flash Players. Aber auch in anderer Software ist Flash zu finden. So wurde im Spiel „Grand Theft Auto 4“ des amerikanischen Unternehmens Rockstar Games das Menü mit Hilfe von Flash realisiert, auch Interface-Elemente der „CryEngine“, einer 3D-Engine der deutschen Firma Crytek, nut-

[13] Create or Die – Video Revolution online, S. 65ff.

zen diese Technologie[14]. Der Vorteil dabei ist die kurze Entwicklungszeit der Elemente, wodurch dann mehr Ressourcen für kritischere Teile des Entwicklungsprozesses zur Verfügung stehen.

Abbildung 1, Interface-Beschreibung der "CryEngine", man beachte "DYNAMIC FLASH ASSETS"

Ein weiteres Einsatzgebiet ist der Desktop. Wurden Flash-Anwendungen dank Zusatztechnologien wie zum Beispiel Zinc[15] teilweise auch schon früher als Desktopprogramme genutzt, geht Adobe seit dem Februar 2008 den Weg über die Adobe Integrated Runtime (kurz AIR).[16] Diese Software ist mit der Java Laufzeitumgebung von Sun vergleichbar, die zum Ausführen von in Java entwickelten Programmen erforderlich ist. AIR unterstützt dabei nicht nur Flash, sondern auch HTML, JavaScript und CSS. Adobe setzt bereits entsprechende Anwendungen in den eigenen Produkten als Zusatz ein und bietet

[14] http://www.golem.de/showhigh2.php?file=/0908/69105.html

[15] http://www.multidmedia.com/software/zinc/

[16] http://www.golem.de/0802/57918.html

dazu auch eine Schnittstelle an. Ein Beispiel dafür ist Kuler[17], welches zur Erstellung von und zum Arbeiten mit Farbpaletten in diversen Programmen der Creative Suite 4, wie zum Beispiel Photoshop und Flash, zur Verfügung steht.

[17] http://kuler.adobe.com/

3.1. Die Geschichte von Flash

Die Geschichte von Flash beginnt etwa 1993 mit der Idee eines Zeichenprogramms, das speziell auf die Arbeit mit elektronischen Stiften auf dem Computer-Monitor ausgelegt sein soll. Das geplante Programm SmartSketch des jungen Startup-Unternehmens FutureWave Software betritt damit ein noch junges und wenig bedachtes Geschäftsfeld im Grafikbereich. SmartSketch basiert dabei auf ein ebenso junges Betriebssystem der Firma Go. 1994 wird Go von AT&T aufgekauft und die Entwicklung an deren Betriebssystem eingestellt. Damit steht FutureWave ohne Markt für SmartSketch da und muss sich umorientieren. Der nächste Schritt ist, die eigene Software an Windows und Macintosh anzupassen, wodurch man dann aber mit Größen wie FreeHand von Aldus (später von Macromedia übernommen) und Illustrator von Adobe konkurrieren muss.

1995 ist FutureWave dann auf der SIGGRAPH[18] vertreten und präsentiert SmartSketch einem professionellen Publikum. Dabei erhält man hauptsächlich die Bitte, daraus doch ein Animationsprogramm zu entwickeln. Zeitgleich registriert man den bemerkenswerten Erfolg des noch jungen Internets und erkennt früh dessen Chancen. Der erste Schritt ist nun, SmartSketch mit Funktionalitäten für Animationen zu erweitern. Da die damalig einzige Möglichkeit zur Darstellung von

[18] SIGGRAPH: Special Interest Group on GRAPHics and Interactive Techniques – Eine jährliche international Konferenz zum Thema Computer-Grafik

animierten Vektorgrafiken in Java besteht, entwickelte man ein erstes Java-Programm. Dessen Leistungsfähigkeit liegt aber noch weit unter dem, was man sich vorstellt. Ende 1995 veröffentlicht dann Netscape seine Programmschnittstelle, die es möglich machen soll, den Browser Netscape Navigator mit Plugins zu erweitern. Darin sieht FutureWave eine Chance, entwickelt ein erstes Plugin und veröffentlicht im Zuge dessen im Mai 1996 den FutureSplash Animator. Erste große Interessenten finden sich schnell in Microsoft und Disney, die die neue Technologie in ihre Webauftritte sowie andere Produkte integrieren. Ende 1996 wird dann auch Macromedia aufmerksam, kauft FutureWave Software und fügt den FutureSplash Animator der eigenen Produktpalette hinzu – Macromedia Flash 1.0 ist geboren.[19]

Macromedia Flash entwickelt sich zu einem ernstzunehmenden Produkt und steigt in der Bedeutung für das Internet mit dessen Entwicklung und Erfolg. Die Webdesigner erkennen schnell die Möglichkeiten und bald gibt es erste größere Projekte und Animationen im Internet. Im Folgenden werden wichtige Meilensteine in der Entwicklung von Flash aufgezählt:

- 1998: Unterstützung für MP3-Audio-Kompression in Flash 3
- 1999: MP3-Streaming in Flash 4
- 2000: ActionScript 1.0 in Flash 5
- 2002: Video-Kompression mit Sorenson Spark in Flash 6 (MX)

[19] http://www.adobe.com/macromedia/events/john_gay/

- 2003: ActionScript 2.0 sowie das FLV-Videoformat in Flash 7 (MX 2004)
- 2005: Neuer Video-Codec On2 VP6, Bitmap-Verarbeitung mit ActionScript in Flash 8 sowie Übernahme von Macromedia, Inc durch die Adobe Systems, Inc.
- 2006: ActionScript 3.0, neue virtuelle Maschine AVM2 für den Flash Player in Flash 9 (CS3)
- 2007: Unterstützung für H264-Video- und AAC-Audio-Kompression in Flash Player 9 Update 3
- 2008: Erweiterungen für ActionScript 3.0, erweitere 3D-Transformationen, neue Textrenderingengine und Inverse Kinematik in Flash 10 (CS4)

Laut Adobe liegt die Verbreitung des Flash Players bei etwa 98 bis 99 % weltweit[20], tatsächlich dürfte diese aber ein wenig niedriger ausfallen, entsprechende Erhebungen sind leider mit Vorsicht zu genießen, da niemand aufgrund der Größe des Internets und der Menge an Benutzern für eine Repräsentativität der Daten sorgen kann. Dennoch ist Flash damit ein De-facto-Standard für interaktive Grafiken im Internet – und das ohne wirklich ernstzunehmenden Konkurrenten. Mit Silverlight hat zum Beispiel Microsoft ein Konkurrenzprodukt anzubieten, dessen Erfolg sich aber in Grenzen hält. Laut StatOwl.com lag die Verbreitung im Juni 2009 bei unter 30

[20] http://www.adobe.com/products/player_census/flashplayer/version_penetration.html

Prozent[21], wobei dieser Wert ebenso mangels Repräsentativität mit Vorsicht zu genießen ist. Es ist möglich, dass die Verbreitung deutlich höher ist, da Microsoft sein Konkurrenzprodukt über das Windows-eigene Update-System anbietet. Dennoch ist Silverlight bisher recht unbedeutend für den Markt.

Eine weitere Alternative stellt der freie Standard SVG[22] dar. Die Spezifikation des World Wide Web Consortiums (W3C) beschreibt die Darstellung von Vektorgrafiken auf Basis von XML. Eine besonders starke Verbreitung genießt SVG in mobilen Geräten sowie unter dem freien Betriebssystem Linux. Unter den Browsern ist das Bild sehr uneinheitlich. Besonders der Internet Explorer fällt mit komplett fehlender Unterstützung auf. Abhilfe schafft da ein Plugin von Adobe[23], das aber nicht mehr weiterentwickelt wird[24]. Im Web hat SVG momentan im Vergleich zu Flash eine eher nachrangige Bedeutung.[25]

---

[21] http://www.statowl.com/custom_ria_market_penetration.php

[22] Scalable Vector Graphics, http://www.w3.org/TR/SVG11/

[23] http://www.adobe.com/svg/viewer/install/mainframed.html

[24] http://www.adobe.com/svg/eol.html

[25] Petzold, 2008

3.2. ActionScript

Die Geschichte von ActionScript reicht zwar nicht ganz so weit zurück, wie die von Flash an sich, aber dennoch ist die Scriptsprache ein wesentlicher Bestandteil davon und ohne ihr lässt sich der Erfolg des Flash Players nicht vollständig erfassen. Ohne die damit denkbaren Interaktionsmöglichkeiten, hätte Flash wahrscheinlich nicht die Bedeutung erlangt, die es heute hat.

3.2.1. Die Anfänge

Erste rudimentäre Implementierungen einer Scriptsprache in Form von Zeitleistensteuerung und simplen Ereignissen bieten Flash 2 und 3. Flash 4 implementiert erstmals auch bedingte Anweisungen und Schleifen. Man kann Eingabefelder verwenden und über CGI-gestützten[26] Datenaustausch sind nun auch Formulare in Flash möglich.

Mit Flash 5 wird die eigene Scriptsprache komplett neu nach dem Standard ECMA-262[27] entwickelt, worauf auch JavaScript basiert. Es entsteht mit ActionScript 1.0 eine prototypische Scriptsprache, die nun weitaus mächtiger ist als die Version in Flash 4. Prototypisches Programmieren ist dabei eine Art Objektorientierung, wie man sie zum Beispiel von Java

[26] CGI - Common Gateway Interface: Standard zur Kommunikation mit einem Webserver

[27] http://www.ecma-international.org/publications/standards/Ecma-262.htm

kennt, wobei sie nicht so strikt sondern vielmehr sehr dynamisch funktioniert. Neue Objekte werden quasi zur Laufzeit erzeugt, um bestehende zu erweitern. Im Folgenden soll ein Beispiel eine solche prototypenbasierte Programmierung verdeutlichen.

```
var baseObject = {
    num : 1,
    baseText : "Hallo Welt!"
}

function subObject() {
    this.num = 2;
    this.subText = "Hello World!";
};

subObject.prototype = baseObject;;

var instanceOfSubObject = new subObject();

trace( baseObject.num );
// Ausgabe: "1"
trace( baseObject.baseText );
// Ausgabe: "Hallo Welt!"
trace( instanceOfSubObject.num );
// Ausgabe: "2"
trace( instanceOfSubObject.subText )
// Ausgabe: "Hello World!"
```

Abbildung 2, Beispiel einer prototypenbasierte Programmierung

Der Vorteil dieser Programmierung liegt in der flexibleren Implementierung neuer Datentypen und Anweisungen zur Laufzeit. Der Nachteil ist dabei aber, dass der Quelltext bei umfangreicheren Projekten schwerer zu warten ist. Der Compiler ist zudem auch – aufgrund der Flexibilität und Erweiterbarkeit zur Laufzeit – nicht in der Lage, den Maschinencode zu optimieren, was im Endeffekt zu Leistungsengpässen führen kann.

Flash 6, auch Flash MX genannt, bringt Optimierungen in der Scriptsprache und weitere Neuerungen mit sich. Die Auffälligste ist die Drawing-API, die Programmschnittstelle zum Zeichnen von Vektorformen zur Laufzeit. Per Script kann man nun zum Beispiel ein Malprogramm schreiben – was das klassi-

sche Anwendungsbeispiel dazu darstellt. Weiterhin sind damit zur Laufzeit Veränderungen in Vektorformen möglich, zum Beispiel aufgrund von Nutzereingaben oder ähnlichen veränderlichen Erhebungen.

Eine weitere Neuerung ist die *LocalConnection*[28]. Dieser neue Datentyp erlaubt die Kommunikation zwischen zwei oder mehr physisch voneinander getrennten Flashfilmen auf einer Website oder gar auf zwei oder mehr unterschiedlichen Webseiten. Die *LocalConnection* wird oft in Werbebanner verwendet, wenn sie voneinander abhängige Animationen enthalten. Dabei müssen die Filme über die lokale Verbindung synchronisiert werden. Mit der Einführung von ActionScript 3.0 und der neuen AVM2 gibt es nun auch ein weiteres Einsatzgebiet: die Kommunikation zwischen AVM1- (bis Flash 8 und ActionScript 2.0) und AVM2-Filmen (ab Flash 9 und ActionScript 3.0). Es ist zwar mit ActionScript 3.0 möglich, ältere Flash-Movies zu laden und anzuzeigen, aber man kann kaum mit ihnen arbeiten. Das geht dann mit Hilfe der *LocalConnection*. Dieser Weg macht aber nur dann Sinn, wenn die Portierung der bestehenden Anwendung auf ActionScript 3.0 deutlich mehr Zeit in Anspruch nehmen würde, als eine Implementierung der *LocalConnection*.

[28] „local connection“: lokale Verbindung, die nur Flash-Movies auf dem Rechner und nicht darüber hinaus verbindet

3.2.2. ActionScript 2.0

Flash 7 – oder MX 2004 – bietet die bis dahin für Programmierer interessanteste Neuerung: ActionScript 2.0. Zwar nur ein Aufsatz auf die bisherige prototypische Objektstruktur, ist die neue Version dennoch eine sinnvolle Erweiterung, die die Entwicklung erheblich vereinfacht und optimiert, denn nun kann man die gesamte Code-Basis in eine Klassenstruktur ähnlich wie in Java packen. Damit einher gehen die weiteren Vorteile der klassenbasierten objektorientierten Programmierung. Polymorphie ermöglicht es zum Beispiel, unterschiedliche Datentypen beispielsweise über ein Interface[29] anzusprechen, ohne dass der eigentliche Datentyp bekannt sein muss. Das Gleiche gilt auch für Subklassen, die alle über Funktionen ihrer gemeinsamen Basisklasse angesprochen werden können. Diese Eigenschaft nennt man Vererbung. Dabei erbt eine neue Klasse die Funktionen und Eigenschaften ihrer Basisklasse und überschreibt oder erweitert sie bei Bedarf um neue Funktionen oder Eigenschaften. Eine weitere wichtige Eigenschaft der klassenbasierten Objektorientierung ist die sogenannte Datenkapselung. Schlüsselwörter wie „public“ oder „private“ erlauben es, Zugriffe auf Methoden und Variablen freizugeben bzw. zu sperren. Damit lassen sich die Schnittstellen der Klassen von außen auf das Nötigste reduzieren und interne Algorithmen nach außen verbergen. Auch in diesem Fall können Interfaces helfen.

[29] Interface: Schnittstelle, die Funktionen definiert, die von Klassen, welche das Interface implementieren, angeboten werden müssen

```actionscript
/**
 *  Klasse Child (Datei Child.as)
 **/

class Child {

    public var num : Number = 10;

    public function Child() {
        // Konstruktor
        // initialisiert die Instanz
    }

    public function setText( text : String ) : Void {
        trace( text );
    }

}

/**
 *  Initialisierung (zum Beispiel über die Flash-
 *  Entwicklungsumgebung)
 **/

import Child;
var child : Child = new Child();
trace( child.num ); // Ausgabe: "10"
child.setText( "Hallo Welt!" ); // Ausgabe: "Hallo Welt!"
```

Abbildung 3, Beispiel einer klassenbasierten Programmierung

Mit Flash 8 wird die Anzeigefähigkeiten von ActionScript mehr bedacht. So gibt es nun mehrere neue Klassen, die die geometrische Programmierung in Flash vereinfachen. Die Matrix- und die Transform-Klasse arbeiten dabei eng zusammen und ermöglichen Verzerrungen und andere Transformationen mit ActionScript. Rudimentäre 3D-Frameworks und Pseudo-3D-Effekte[30] sind damit nun einfacher.

Der bedeutendste Neuzugang zu ActionScript sind aber die Bitmap-Filter sowie die neue *BitmapData*-Klasse. Die Filter erlauben nun Veränderungen an MovieClips und Bildern, wie man sie sonst nur aus Bildbearbeitungsprogrammen wie Photoshop kennt. Somit sind besonders leicht für das Web2.0 typische Effekte wie Schatten, Glühen oder Weichzeichnen möglich.

[30] http://www.flashsandy.org/

Die Neuerungen sind sehr weitreichend und erlauben unterschiedliche optische Anpassungen zur Laufzeit.

Die *BitmapData*-Klasse lässt weitere Effekte zu. Mit ihr lassen sich zum Beispiel mehrere MovieClips in eine Bitmap-Grafik zeichnen, wobei sich diese dann bequem über ihre Alpha- und Positionseigenschaften animieren lässt. Dadurch lassen mitunter Leistungsengpässe durch übermäßig viele Vektorelemente vermeiden. Die *BitmapData*-Klasse ist auch für 3D-Effekte notwendig. Dabei dient sie als Textur eines Dreiecks, das je nach Positionierung und Ausrichtung im dreidimensionalen Raum entsprechend verzerrt wird. In Kombination mit weiteren Dreiecken lassen sich daraus komplexe Körper bilden.[31]

3.2.3. ActionScript 3.0

Im Zuge der Übernahme von Macromedia durch Adobe Ende 2005[32] und der Entwicklung an Flex 2 wird eine neue Scriptsprache im Hinblick auf die gestiegenen Anforderungen an Flash entwickelt: ActionScript 3.0. Diese neue Version bringt wesentliche Neuerungen mit sich, die die Entwicklung von Flash-Projekten deutlich fehlerresistenter, standardkonformer und strikter machen soll.

---

[31] http://help.adobe.com/en_US/AS2LCR/Flash_10.0/

[32] http://www.adobe.com/aboutadobe/pressroom/pressreleases/200512/120505AdobeAcquiresMacromedia.html

Als Unterbau für ActionScript 3.0 kommt die neue Virtuelle Maschine AVM2 hinzu, die eine deutliche höhere Leistung und schnellere Ausführung von ActionScript-Code verspricht. Geschwindigkeitsvergleiche[33] und Frameworks für die unterschiedlichsten Ansprüche[34] [35] [36] [37] [38] bestätigen schnell diesen Zuwachs an Leistung. Damit einhergehend wir die Scriptsprache basierend auf den ECMA-Standard ECMAScript 4[39] [40] neu konzipiert. Die Syntax wurde wesentlich angepasst und gleicht nunmehr Java. Es gibt Packages, in denen die Klassen organisiert sind, und der objektorientierte Ansatz wurde deutlicher gemacht. Die Datenkapselung ist nun vielfältiger, mit dem Schlüsselwort „private“ versehene Bezeichner sind nur noch klassen-intern zugreifbar, mit „protected“ gekennzeichnete Variablen und Funktionen können auch von den Subklassen verarbeitet werden und mit „internal“ erlaubt man den Zugriff nur noch package-intern. Mit Hilfe der neu hinzugekommenen Namespaces kann man diese Kapselung noch weiter aufschlüsseln. Bei XML-Verarbeitung ist die Arbeit mit Namespaces besonders wichtig, wenn man zum Beispiel mit XHTML-Dateien arbeitet. Das gleiche trifft natürlich auch auf

---

[33] http://blog.greensock.com/tweening-speed-test/
[34] http://www.papervision3d.org/
[35] http://away3d.com/
[36] http://nochump.com/blog/?p=15
[37] http://box2dflash.sourceforge.net/
[38] http://code.google.com/p/as3corelib/
[39] http://www.ecmascript.org/es4/spec/overview.pdf
[40] http://www.golem.de/0808/61775.html

andere XML-basierte Standards zu, die mit eigenen Namensräumen arbeiten.

Die AVM2 überprüft auch zur Laufzeit die Datentypen und wirft Fehler (Exceptions im Java-Jargon), die entsprechend behandelt werden müssen. Zum Beispiel wird ein Fehler geworfen, wenn eine Variable mit dem Datentyp Number in eine Variable mit dem Datentyp String gespeichert wird, da beide nicht kompatibel zueinander sind. Dadurch wird man zu einer strikteren Programmierung verleitet, die zudem die Wartung besonders in großen Projekten deutlich erleichtert. Es ist auch möglich, eigene Fehlertypen zu erstellen und einzusetzen. Das erleichtert besonders die Fehlersuche.

Eine weitere neue Implementierung ist E4X, ECMAScript for XML[41], eine Spezifikation, die beschreibt, wie XML-Daten verarbeitet werden. Im Vergleich zum XML-Datentyp in ActionScript 2.0, in dem über generalisierte Methoden und Eigenschaften auf die Inhalte einer XML zugegriffen werden konnte, lassen sich die Daten nun objektorientiert und damit wesentlich einfacher auslesen, besonders wenn man mit dem Aufbau der XML-Datei vertraut ist. Ein Beispiel soll den Unterschied verdeutlichen:

[41] http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-357.pdf

```actionscript
/**
 *  XML-Verarbeitung in ActionScript 2.0
 **/

var xmlString : String = "<rootnode><childnode id=\"10\"><textnode>Hallo
    Welt!</textnode></childnode></rootnode>";
var xml : XML = new XML( xmlString );
xml.ignoreWhite = true;
trace( xml.firstChild.childNodes[ 0 ].childNodes[ 0 ].childNodes[ 0 ] );
// Ausgabe: "Hallo Welt!"
trace( xml.firstChild.childNodes[ 0 ].attributes.id );
// Ausgabe: "10"

/**
 *  XML-Verarbeitung in ActionScript 3.0
 **/

var xml : XML = <rootnode>
                    <childnode id='10'>
                        <textnode>Hallo Welt!</textnode>
                    </childnode>
                </rootnode>;

trace( xml.childnode.textnode );
// Ausgabe: "Hallo Welt!"
trace( xml.childnode.@id );
// Ausgabe: "10"
```

Abbildung 4, Beispiel einer XML-Verarbeitung in ActionScript 2.0 und 3.0

Eine der weitreichendsten Neuerungen betrifft wohl die Display-Liste, einer hierarchischen Ansammlung von Display-Objekten. In ActionScript 2.0 wird alles mit Hilfe von MovieClips dargestellt, mit ihnen lassen sich neue erstellen, Objekte aus der Bibliothek auf die Bühne holen und Textfelder erzeugen. Das Stage-Objekt, die globale Bühne im Flash Player, ist eine Klasse mit statischen Eigenschaften.

In ActionScript 3.0 tritt die Klasse *MovieClip* deutlich in den Hintergrund und erweitert die mächtige *Sprite*-Klasse lediglich um die zusätzlichen Funktionen für die Zeitleistensteuerung. MovieClips lassen sich nun mit Hilfe des Operators „new“ erstellen und einem *Sprite* oder anderem DisplayObjectContainer hinzufügen. Das Gleiche trifft auch auf die *TextField*-Klasse zu. Die globale Bühne des Flash Players ist nun eine Instanz der *Stage*-Klasse, auf die nur noch über Umwege global zugegriffen werden kann. Die einzelnen

DisplayObjects wie *Sprite*, *MovieClip* oder *TextField* können erst auf ihr *stage*-Objekt zugreifen, wenn sie zur Display-Liste hinzugefügt wurden. Die folgende Abbildung soll die Display-Hierarchie der Klassen zeigen:

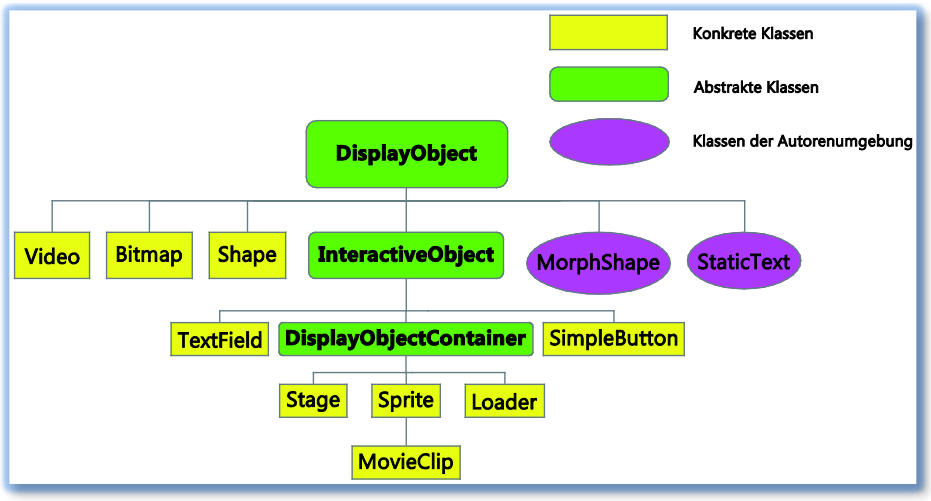

Abbildung 5, Hierarchie der Display-Klassen[42]

Die Display-Hierarchie zum Beispiel in Papervision3D ist ähnlich aufgebaut, wobei die einzelnen Klassen nicht von einer Flash-eigenen Displayklasse erben. Der Aufbau von Papervision3D wird in einem separaten Kapitel erläutert.

Eine ähnlich hierarchisch aufgebaute Neuimplementierung in ActionScript 3.0 ist das Event-System, welches nun ebenfalls ECMA-konform ist. Die Basis bildet der sogenannte EventDispatcher, eine Basisklasse, von der alle Klassen erben, die Events auslösen sollen. In ActionScript 2.0 wird vorwiegend mit Funktionsverweisen wie zum Beispiel *onEnterFrame()* oder *onRelease()* des *MovieClips* gearbeitet. Diese Verweise werden mit eigenen Funktionen, sogenannten Event-Handlern über-

[42] vgl. Moock 2007, S. 459

schrieben, um auf die Ereignisse zu hören. Der Nachteil besteht darin, dass jeweils nur ein Event-Handler auf ein Ereignis hören kann. Um diese Grenzen zu erweitern, kann man auch einen eigenen rudimentären EventDispatcher in ActionSript 2.0 entwickeln.

In ActionScript 3.0 dient die Funktion *addEventListener()* zum Registrieren eines Event-Handlers auf ein bestimmtes Ereignis, es können dabei beliebig viele registriert werden. Dem Event-Handler wird dann bei einem Ereignis ein Event-Objekt übergeben, das er bei Bedarf weiterverarbeiten kann. Ein Beispiel soll die Funktionsweise grob erläutern und vergleichend einen Fall in ActionScript 2.0 bieten:

```
1   /**
2    *  Event-Handling in ActionScript 3.0
3    **/
4
5   import flash.events.Event;
6   import flash.display.MovieClip;
7
8   var mc : MovieClip = new MovieClip();
9
10  mc.addEventListener( Event.ENTER_FRAME, enterframeHandler );
11  addChild( mc );
12
13  function enterframeHandler( evt : Event ) : void {
14      // Anweisungen
15  }
16
17  /**
18   *  Event-Handling in ActionScript 2.0
19   **/
20
21  var mc : MovieClip = _root.createEmptyMovieClip( "mc", 0 );
22
23  mc.onEnterFrame = function() : Void {
24      // Anweisungen
25  }
```

Abbildung 6, Beispiele des Event-Handlings in ActionScript 3.0 und 2.0

Der hierarchische Aspekt des neuen Event-Systems lässt sich ganz einfach mit Hilfe der Display-Liste erläutern: Registriert man zum Beispiel auf einen Button einen Event-Handler für das *CLICK*-Event, wird dieses Ereignis direkt vom

Button ausgelöst, sobald er geklickt wird. Es macht also Sinn, das Ereignis auf den Button zu registrieren. Es ist aber auch möglich, das Ereignis auf jeden *DisplayObjectContainer* zu registrieren, in dem sich der Button befindet. Das kann ein *MovieClip*, ein *Sprite* oder gar das *stage*-Objekt des Buttons sein. Das *CLICK*-Ereignis wird einfach die gesamte Display-Liste bis hoch zum *stage*-Objekt ausgelöst. Diese Eigenschaft eines Events nennt man Bubbling[43], das heißt, das Ereignis „blubbert" nach oben, bis zum höchsten Objekt – in dem Fall die globale Bühne des Flash Players.

Flash 10 erweitert die Anzeigefähigkeiten um die Möglichkeiten der dreidimensionalen Darstellung und einige andere Eigenschaften. Einer der wichtigsten Punkte dabei wäre wohl die zusätzlichen Attribute der Display-Klassen wie *z* für die Translation in der Tiefe des Raums sowie *rotationX*, *rotationY* und *rotationZ* für die Rotation um die drei Achsen. Diese vier Attribute bilden die Grundlage für die dreidimensionale Darstellung von Display-Objekten.

Neue Klassen wie *Matrix3D*, *Vector3D* und *PerspectiveProjection* unterstützen dabei die Programmierung für die dritte Dimension. Sie bieten native Funktionen zur erweiterten Transformation im Raum und ermöglichen es, global die Brennweite sowie das Sichtfeld einzustellen.

Die neuen *Shader*- beziehungsweise *ShaderFilter*-Klassen erweitern die Bitmap-Filter und erlauben individuelle

[43] to bubble (engl.): blubbern, Blasen bilden

und zugleich native Effekte auf Display-Objekten. Diese bieten sich auch für 3D-Frameworks auf Basis des Flash Player 10 an. Das PixelBender Toolkit von Adobe dient der Erstellung der Shader-Programme, die vom Flash Player nativ ausgeführt werden. Das heißt, sie laufen direkt in CPU-naher Maschinensprache und nicht in der virtuellen Maschine des Flash Players.[44] [45]

[44] http://help.adobe.com/en_US/AS3LCR/Flash_10.0/

[45] http://labs.adobe.com/technologies/pixelbender/

4. 3D-Programmierung in ActionScript 3.0

Dieses Kapitel soll die Grundlagen der Programmierung dreidimensionaler Inhalte in ActionScript 3.0 umreißen und damit einen vereinfachten Einblick in die komplexe Arbeit einer 3D-Engine geben. Für detaillierte Ausführungen sei auf die einschlägige Literatur[46] [47] zu diesem Thema verwiesen. Das gleiche gilt für die mathematischen Grundlagen, worauf auch nur am Rande eingegangen wird.

[46] Hawkins, Astle 2001, S. 63ff.

[47] Finney 2004, S. 89ff.

4.1. Grundlagen

Das Ziel der Programmierung von dreidimensionalen Inhalten ist es, auf einer zweidimensionalen Oberfläche wie zum Beispiel dem Bildschirm einen räumlichen Eindruck zu erwecken. Dies erreicht man, indem ein Tiefeneindruck mit Hilfe eines Fluchtpunktes erzeugt wird. Wie in der Realität erscheinen weiter entfernt befindliche Objekte dadurch kleiner als solche, die sich näher am Betrachter befinden. Die Lichtverhältnisse, Schattenwürfe und die Materialbeschaffenheit der angezeigten Objekte tragen ihr Übriges zum räumlichen Eindruck bei.

Um nun die dreidimensionalen Objekte auf einer zweidimensionalen Ebene abbilden zu können, sind mehrere Schritte erforderlich. Zunächst müssen die Objekte in räumlichen Daten aufbereitet werden, das heißt, ihnen werden Koordinaten über räumliche Vektoren zugeordnet. Aus diesen Vektoren baut sich das Objekt dann zusammen, sie werden Vertices[48] genannt und bilden die kleinste räumliche Einheit. Eine Fläche zum Beispiel eines Quaders besteht dabei aus vier Vertices, die zusammen eine Fläche bilden. Da sich in der 3D-Programmierung die Arbeit mit Dreiecken, in der Fachsprache Triangles genannt, durchgesetzt hat, würde diese Fläche dementsprechend aus mindestens zwei Dreiecken bestehen. Diese werden auch als Polygone gezeichnet. Eine Kombination aus

[48] vertex (engl.): Eckpunkt

Polygonen, wie zum Beispiel die Fläche des Quaders oder auch der gesamte Quader als solcher bildet ein sogenanntes Mesh[49].

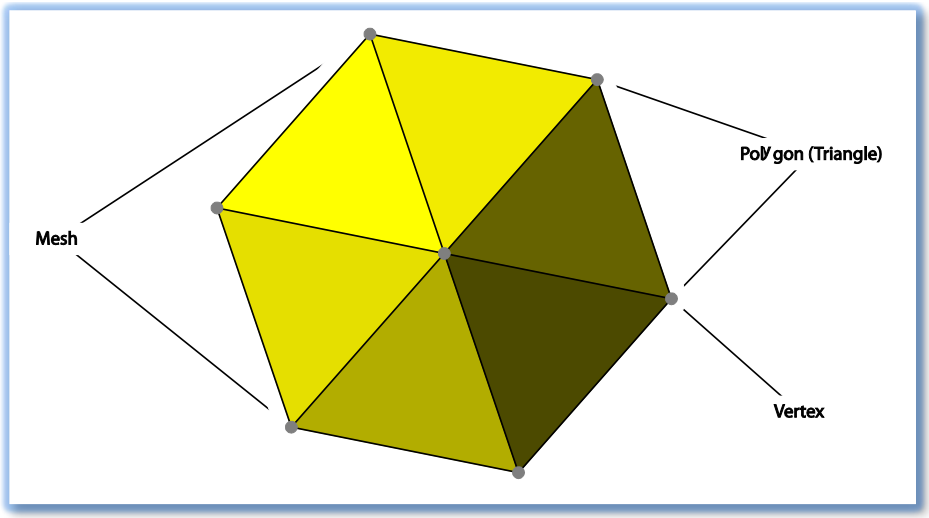

Abbildung 7, Darstellung von Vertex, Polygon und Mesh

Eine Alternative zur Arbeit mit Polygonen stellt die Voxelgrafik[50] dar, bei der die Objekte als eine Menge von Pixeln abgebildet werden, die immer deutlich kleiner sind, als auf dem Bildschirm darstellbar. Aufgrund des bisher enormen Rechenaufwands hat sich die Voxelgrafik nicht durchsetzen können, lediglich in der Medizin findet sich ein großes Anwendungsfeld. Die Vorteile liegen besonders in der Darstellung starrer und enorm detailreicher Landschaften und Oberflächen.[51]

Das dreidimensionale Koordinatensystem, für gewöhnlich das kartesische Koordinatensystem, bildet die Grundlage für die Positionierung dreidimensionaler Objekte im Raum. Da-

[49] mesh (engl.): Netz
[50] Voxel: Volumenpixel
[51] c't 2009, Heft 4, S. 182ff.

bei unterscheidet man zwischen einem globalen und beliebig vielen lokalen Koordinatensystemen für die einzelnen Körper. Diese werden meist mittig darin positioniert, was besonders die Rotation vereinfacht. Für die Darstellung im globalen Koordinatensystem werden die Positionen der Vertices mit Hilfe von 4x4-Matrizen aufbereitet. In Flash 10 wäre dafür die Matrix3D-Klasse zuständig.

Die Kamera und der der sogenannte Viewport[52] dienen der Bestimmung, welcher Ausschnitt einer Szene wie dargestellt wird. Die Kamera ist dabei ein virtuelles Objekt im Raum, dessen Position selbst über Koordinaten bestimmt wird. Des Weiteren verfügt sie über Eigenschaften, wie die Brennweite und das Sichtfeld – im Fachjargon als „focal length" und „field of view" bezeichnet. Beide Eigenschaften bestimmten dabei, wie die Körper im Raum transformiert werden. Eine niedrige Brennweite zum Beispiel lässt die Distanzen zwischen Körpern in der Tiefe größer erscheinen. Je größer die Brennweite, desto geringer erscheint dabei diese Distanz. Dies entspricht der Perspektivprojektion, bei der es einen sogenannten Fluchtpunkt gibt. Eine unendlich große Brennweite würde identische Körper unabhängig von ihrer Ausrichtung in der Tiefe gleichgroß erscheinen lassen. Diese Darstellung wäre dann eine Parallelprojektion, die keinen Fluchtpunkt besitzt.

Um die dreidimensionalen Daten in eine zweidimensionale Darstellung zu projizieren, wird das sogenannte Rende-

[52] viewport (engl.): Darstellungsfeld

ring[53] verwendet. Dabei wird die sogenannte Render-Pipeline durchlaufen, welche sich grob in drei Abschnitte aufteilen lässt.

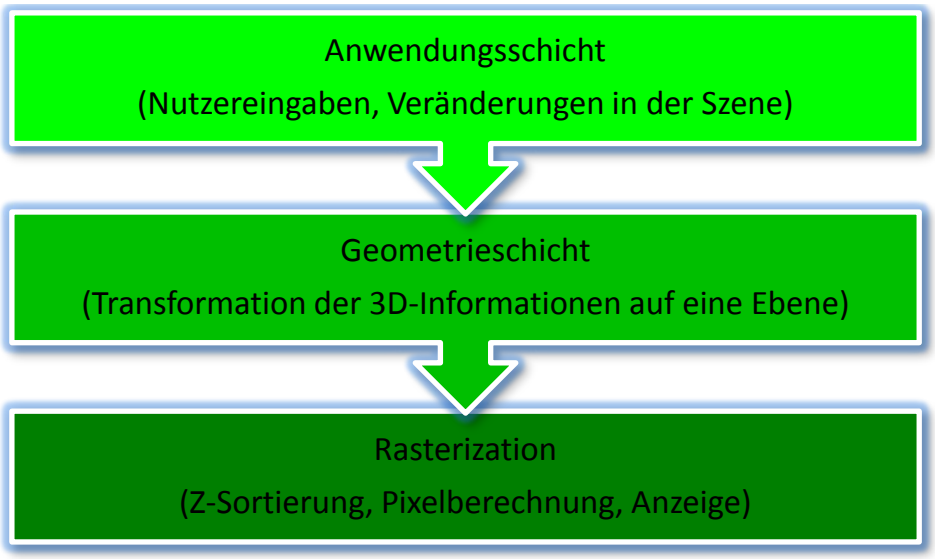

Abbildung 8, Ablauf der Renderpipeline[54]

Die Anwendungsschicht verarbeitet die Nutzereingaben, weitere Interaktionen und externe Resourcen, wie zum Beispiel Sound oder Netzwerksignale bei Mehrspieler-Partien. In dieser Schicht werden auch gegebenenfalls Anpassungen an den Polygonen einzelner Körper vorgenommen und die Kollisions-erkennung durchgeführt.

In der Geometrieschicht werden anhand der Polygon- und Vertexdaten die Transformationen der einzelnen Körper durchgeführt und die lokalen Koordinaten der Polygone in das globale Koordinatensystem überführt. Gegebenenfalls werden hier auch die Beleuchtung und der Schattenwurf berechnet.

[53] to render (engl.): wiedergeben, übersetzen
[54] Vgl. Lehmann 2009, S. 34

Die Rasterization ist dann für das Ergebnis, welches auf dem Bildschirm dargestellt werden soll, verantwortlich. In diesem Schritt erfolgen unter anderem die sogenannte Z-Sortierung und die Pixelberechnung. Die Z-Sortierung ist ein Algorithmus, der die einzelnen Polygone der Tiefe nach sortiert, so dass sich entfernte Polygone hinter nahen befinden. Die Pixelberechnung bestimmt die Farbwerte der einzelnen Pixel aufgrund der Textur auf den Polygonen.[55]

[55] Lehmann 2009, S. 24ff.

4.2. Aufbau und Funktionsweise bisheriger 3D-Frameworks am Beispiel von Papervision3D

4.2.1. Aufbau

Wie bereits weiter oben erwähnt, ist die Klassenhierarchie der Display-Objekte ähnliche wie die nativen Display-Objekte von Flash aufgebaut. Die folgende Abbildung soll dies genauer beleuchten.

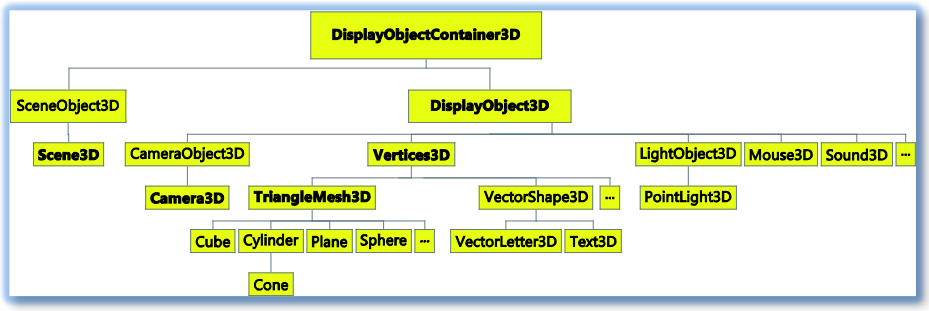

Abbildung 9, Display-Klassenhierarchie von Papervision3D (gekürzt)

Die Klasse *DisplayObjectContainer3D* hat eine zentrale Funktion im Papervision3D-Framework. Aus ihr leiten sich die Klassen *Scene3D*, *Camera3D* und alle wichtigen Grundkörper ab. Die *Scene3D*-Klasse ist dabei mit der *Stage*-Klasse vergleichbar, da sie der globale Container aller Display-Objekte ist. Im Unterschied zur *Stage*-Klasse lassen sich aber mehrere Szenen, also *Scene3D*-Instanzen, verwenden, wohingegen derzeit nur eine *Stage*-Instanz verwendet werden kann. Die *Camera3D*-Klasse beschreibt die eigentliche Kamera der Engine. Es können beliebig viele Kameras verwendet werden.

Eine weitere Klasse, die von *DisplayObjectContainer3D* erbt, ist die *DisplayObject3D*-Klasse. Sie stellt die Basis für alle

Körper in Papervision3D dar. Sie enthält zwei wesentliche Eigenschaften, die für den Rendervorgang wichtig sind: *material* bzw *materialList*, die Instanzen vom Typ *MaterialObject3D* enthalten, und *geometry*, das eine Instanz vom Typ *GeometryObject3D* bereithält. *MaterialObject3D* ist dabei die Basis-Klasse für alle Materialien wie Texturen. Das können Bitmap-, Vektor-Grafiken oder gar Animationen oder Videos sein. Die *GeometryObject3D*-Klasse enthält alle wichtigen Geometriedaten, wie Vertices, Polygone oder komplette Meshes. Sie stellt eine speicherschonende Kapselung der Daten zum Beispiel zahlreicher Instanzen ein und desselben Körpers dar.

Die *Viewport3D*-Klasse ist die Schnittstelle zur Flash-eigenen Display-Liste, da sie von der *Sprite*-Klasse erbt, und dient der Darstellung des fertigen Renderings. Sie ist auch für die Interaktion zuständig und fängt Maus- und Tastatur-Ereignisse ab, die sie an die eigentlichen Display-Objekte weiterleitet.

Die Instanzen all dieser Klassen werden für jeden Rendervorgang in eine Instanz vom Typ *RenderSessionData* zwischengespeichert. Dieses Objekt wird an die eigentliche Render-Engine, zum Beispiel die *BasicRenderEngine*, weitergeleitet. Diese ist der Einstiegspunkt in die Renderpipeline.

4.2.2. Rendervorgang

Der konkrete Rendervorgang wird durch die Funktion *renderScene()* der Render-Engine, zum Beispiel die

BasicRenderEngine, initiiert. Das Rendering erfolgt dann anhand der Daten von *Viewport3D*, *Camera3D* und *Scene3D*, welche in einer Instanz vom Typ *RenderSessionData* gebündelt werden. Im Anschluss wird die *ProjectionPipeline* gestartet, die für jedes DisplayObject3D und dessen Kindelemente die implementierte Methode *project()* aufgerufen. Dabei werden die dreidimensionalen Koordinaten der Polygone in Relation zur *Camera3D*-Transformation auf eine zweidimensionale Ebene projiziert. Alle sichtbaren Polygone werden als *RenderTriangle* zur Rendering-Liste hinzugefügt. Die Klasse *RenderTriangle* implementiert die Methode *render()*, die letztendlich das eigentliche Rendering des Polygons übernimmt. Dabei wird das Dreieck mit Hilfe des Flash-eigenen *Graphics*-Objekts in ein Display-Objekt gezeichnet und mit der Textur vom Typ *MaterialObject3D* gefüllt.

Um Darstellungsfehler zu vermeiden, wird zusätzlich eine Z-Sortierung durchgeführt. Diese ist in der entsprechenden Render-Engine implementiert. Die *BasicRenderEngine* sortiert dabei die Polygone ihrer Tiefe nach (siehe 4.1). Der Nachteil besteht darin, dass Darstellungsfehler nicht komplett vermieden werden können, da sich mehrere Polygone überlagern oder gar überschneiden können. Weiterhin werden auch komplett verdeckte Polygone gezeichnet, was unnötig Ressourcen verbraucht. Die Alternative ist die *QuadrantRenderEngine*, die einen komplexeren, aber auch effektiveren Algorithmus implementiert.

Der komplette Rendervorgang ist insgesamt enorm rechenaufwendig und sollte im Idealfall nur initiiert werden, wenn auch wirklich eine Veränderung in der Szene zu verzeichnen ist. Das heißt, dass ein üblicher Aufruf der *renderScene()*-Funktion mit jedem neuen Frame kaum praktikabel ist und deshalb vermieden werden sollte. Sinnvoll ist es hingegen, global eine Art Beobachter zu definieren, der Veränderungen erkennt und daraufhin den Rendervorgang startet.[56]

```
1   package {
2       import org.papervision3d.cameras.Camera3D;
3       import org.papervision3d.objects.primitives.Sphere;
4       import org.papervision3d.render.BasicRenderEngine;
5       import org.papervision3d.scenes.Scene3D;
6       import org.papervision3d.view.Viewport3D;
7
8       import flash.display.Sprite;
9
10      public class Papervision3DExample extends Sprite {
11
12          private var _sphere : Sphere;
13          private var _scene : Scene3D;
14          private var _camera : Camera3D;
15          private var _viewport : Viewport3D;
16          private var _renderEngine : BasicRenderEngine;
17
18          public function Papervision3DExample() {
19              _sphere = new Sphere( );
20              _scene = new Scene3D( );
21              _scene.addChild( _sphere );
22              _camera = new Camera3D( );
23              _viewport = new Viewport3D( );
24              addChild( _viewport );
25              _renderEngine = new BasicRenderEngine( );
26              _renderEngine.renderScene( _scene, _camera, _viewport );
27          }
28      }
29  }
```

Abbildung 10, Beispiel-Klasse mit einer simplen Papervision3D-Implementierung

[56] Lehmann 2009, S. 36ff.

4.3. 3D-Programmierung für Flash Player 10

Der Rendervorgang im Flash Player 10 ist nicht so komplex in ActionScript implementiert, da die komplette Anzeige weitestgehend nativ berechnet wird. Zwangsläufig wird in jedem neuen Frame die Anzeige komplett neu gezeichnet, entsprechend der eingestellten und dynamisch veränderbaren Bildrate oder spätestens nach Vollendung eines Scripts. Mit den neuen Eigenschaften der *DisplayObject*-Klasse lässt sich ganz bequem die Ausrichtung, Positionierung und Transformation einer Grafik verändern.

Weiterführende Berechnungen und neue dreidimensionale Körper müssen aber extra erstellt werden. Das betrifft unter anderem auch den Z-Sortieralgorithmus, der nicht nativ implementiert ist. Körper müssen aus einer Vielzahl von Display-Objekten wie *Shapes*, *Sprites*, *MovieClips* oder *Bitmaps* erzeugt werden. Im Idealfall besteht dann ein solcher Körper wie üblich aus dreieckigen Polygonen, aber im Fall eines Quaders reichen auch rechteckige Flächen. Licht- und Schattenberechnungen müssen auch zusätzlich programmiert werden. Die *Shader*-Klasse und das entsprechende PixelBender-Toolkit können aber dabei sehr hilfreich sein.

Natürlich sind dahingehend spezialisierte Frameworks hilfreich, weshalb es auch schon erste Versionen von Papervision3D, Away3D und Alternativa3D mit Unterstützung für den Flash Player 10 gibt.

5. Forschungsobjekte – Die Applikationen

Dieses Kapitel soll die drei für diese Bachelorarbeite ausgewählten Anwendungsbeispiele vorstellen und ihre Besonderheiten näher vorstellen. Zum Aufbau der Programme sei einleitend noch gesagt, dass sie zuerst für den Flash Player 10 programmiert wurden. Darauf aufbauend wurden sie auf das Papervision3D-Framework portiert. Anschließend wurde davon jeweils eine Version basierend auf Away3D und Alternativa3D entwickelt. Diese Vielfalt soll die unterschiedlichen internen Algorithmen der Frameworks und die damit einhergehenden Leistungsunterschiede berücksichtigen. Zudem wurde diese Reihenfolge der Entwicklung gewählt um die Programme intern – unabhängig vom jeweiligen 3D-Framework – vergleichbar zu halten.

Die eigentlichen Applikationen wurden aufgrund ihrer vielfältigen Anforderungen sowie unterschiedlicher technischer Implementierungen gewählt. Cube3D erfasst dabei die Darstellung von Körpern, CoverFlow zeigt die räumliche Verzerrung von Flächen und HeightMap stellt eine aufwendigere Partikelanimation dar.

5.1. Cube3D – Eine Wand aus Würfeln

Cube3D ist ein simples Beispiel des einfachsten dreidimensionalen Körpers: dem Würfel. Um den Flash Player für einen angemessenen Vergleich ausreichend auszulasten, wird in dieser Applikation eine Art Wand aus Würfeln erstellt und diese dann per Zufall um ihre drei Achsen rotiert. Gleichzeitig werden alle Würfel dieser Wand um ihre lokalen drei Achsen rotiert.

In diesem Fall ist die Z-Sortierung außerordentlich wichtig, da hier nicht nur die einzelnen Seiten der Würfel korrekt dargestellt werden müssen. Es ist auch erforderlich, dass die Würfel in der korrekten Reihenfolge dargestellt werden. Die 3D-Frameworks erledigen dies natürlich von alleine und haben damit bis auf kleinere Darstellungsfehlern bei sich überlagernden Würfeln keine Probleme. In der Flash Player 10 Version muss ein solcher Algorithmus erst implementiert werden, da eine Z-Sortierung nativ nicht unterstützt wird.

In der Version für den Flash Player 10 besitzt jede Klasse, die ein dreidimensionales Objekt darstellt, die Funktion *renderView()*, welche die Z-Sortierung vornimmt. Für einen einzelnen Würfel bedeutet dies, dass alle sechs Seiten und die Vektoren ihrer Mittelpunkte in synchronisierten Listen vorgehalten werden. Ihre Vektoren werden dann mit Hilfe der Methode *getTransformedVector3D()* die komplette Display-Liste bis nach oben mit jedem Eltern-Container transformiert, so dass am Ende ein Vektor herauskommt, der die globale Positi-

on der entsprechenden Würfelseite widerspiegelt. Dies entspricht im Grunde der Übertragung von Koordinaten eines lokalen Koordinatensystems (der Würfel) in ein globales Koordinatensystem (die Bühne des Flash Players). Im Anschluss wird die Distanz zum Mittelpunkt berechnet und in einer entsprechenden Liste vorgehalten, welche im Anschluss sortiert wird. Am Ende werden die Würfelseiten der Reihenfolge dieser Liste entsprechend sortiert. Die folgende Abbildung stellt die entsprechenden Funktionen *renderView()* und *getTransformedVector3D()* dar:

```
 1   package de.stefanvonderkrone.as3.cube {
 2       import de.stefanvonderkrone.as3.material.Material3D;
 3       import de.stefanvonderkrone.utils.IndexedValueObject;
   ...
11       public class Cube3D extends Sprite3D {
   ...
22           private var _sideVector3Ds : Vector.<Vector3D>;
23           private var _sides : Vector.<Material3D>;
   ...
124          override public function renderView() : void {
125              var distances : Array = [];
126              var currentVector3D : Vector3D;
127              var distance : Number;
128              while ( numChildren > 0 )
129                  removeChildAt( 0 );
130              for ( var i : int = 0; i < _sideVector3Ds.length ; i++ ) {
131                  currentVector3D = getTransformedVector3D( _sideVector3Ds[ i ], this );
132                  distance = Math.sqrt( Math.pow( currentVector3D.x, 2 ) +
                       Math.pow( currentVector3D.y, 2 ) +
                       Math.pow( (-currentVector3D.z - focalLength ), 2 ) );
133                  distances.push( new IndexedValueObject( distance, i ) );
134              }
135              distances.sortOn( "value", Array.NUMERIC && Array.DESCENDING );
136              for ( var k : int = 0; k < _sideVector3Ds.length ; k++ ) {
137                  addChild( _sides[ distances[k].index ] );
138              }
139          }
140      }
141  }
142
```

Abbildung 11, in der Methode *renderView()* werden alle Seiten des Würfels in der korrekten Reihenfolge auf der Bühne platziert

```
1   package de.stefanvonderkrone.as3.cube {
2       import flash.display.DisplayObject;
3       import flash.geom.Vector3D;
  ...
10      public class Sprite3D extends Sprite {
  ...
32          protected function getTransformedVector3D( vector3D : Vector3D, container :
              DisplayObject ) : Vector3D {
33              var returnVector3D : Vector3D = null;
34              if ( container.transform.matrix3D != null ) {
35                  returnVector3D = container.transform.matrix3D.clone().deltaTransformVector(
                      vector3D );
36                  if ( container.parent != null ) returnVector3D = getTransformedVector3D(
                      returnVector3D, container.parent );
37              }
38              else returnVector3D = vector3D;
39              return returnVector3D;
40          }
```

Abbildung 12, die Methode *getTransformedVector3D()* überträgt die lokalen Koordinaten eines Vektors in das globale Koordinatensystem

Es hat sich bei der Entwicklung herausgestellt, dass es besser ist, alle Würfelseiten zunächst von der Display-Liste zu entfernen und dann in der richtigen Reihenfolge wieder hinzuzufügen. Der Weg über die Methoden *setChildIndex()*,*swapChildren()* oder *swapChildrenAt()* der *DisplayObjectContainer*-Klasse hat sich nicht bewährt, da ein unkontrolliertes Flimmern verursachten wird.[57]

Eine weitere interessante Erkenntnis betrifft das *matrix3D*-Objekt. Die *Matrix3D*-Klasse bestimmt, wie ein *DisplayObject* im Raum dargestellt wird. Um aber damit arbeiten zu können, muss zunächst eine dreidimensionale Transformation vorgenommen werden. Dies kann zum Beispiel das simple Setzen der *z*-Koordinate auf 0 sein. Erst dann wird die *Matrix3D*-Klasse instanziiert und steht für die entsprechenden Berechnungen zur Verfügung.

In den einzelnen Portierungen für die jeweiligen 3D-Frameworks gibt es lediglich eine globale *renderView()*-

[57] http://www.flashandmath.com/advanced/p10cube/

Funktion, die den entsprechenden Render-Prozess anwirft. Besonderheiten gibt es bei beim Away3D- und beim Alternativa3D-Framework. In allen drei Frameworks werden die Körper über Materialien texturiert. Bei Away3D wird unter anderem auch die entsprechende Klasse ColorMaterial genutzt. Bei der Instanziierung wird ihr lediglich eine Farbe als Zahl übergeben. Da aber der Konstruktor der Klasse keinen bestimmten Typ verlangt, muss die Zahl unbedingt eine Ganzzahl sein.

```
 1   package away3d.materials
 2   {
  ...
18       public class ColorMaterial extends EventDispatcher implements
           ITriangleMaterial, IFogMaterial, IBillboardMaterial
19       {
  ...
103          public function ColorMaterial(color:* = null, init:Object = null)
104          {
105              if (color == null)
106                  color = "random";
107
108              this.color = Cast.trycolor(color);
109
110              ini = Init.parse(init);
111
112              _alpha = ini.getNumber("alpha", 1, {min:0, max:1});
113          }
```

Abbildung 13, Konstruktor der *ColorMaterial*-Klasse – der Parameter *color* wird von der *Cast*-Klasse geparst

```
 1   package away3d.core.utils {
  ...
11       public class Cast
12       {
  ...
60
61           public static function trycolor(data:*):uint
62           {
63               if (data is uint)
64                   return data as uint;
65
66               if (data is int)
67                   return data as uint;
68
69               if (data is String)
70               {
71                   if (data == "random")
72                       return uint(Math.random()*0x1000000);
73
74                   if (colornames == null)
75                   {
76                       colornames = new Dictionary();
77                       colornames["steelblue"] = 0x4682B4;
78                       colornames["royalblue"] = 0x041690;
  ...
217                      colornames["transparent"] = 0xFF000000;
218                  }
219
220                  if (colornames[data] != null)
221                      return colornames[data];
222
223                  if (((data as String).length == 6) && hexstring(data))
224                      return parseInt("0x"+data);
225              }
226
227              return 0xFFFFFF;
228          }
```

Abbildung 14, hier wird deutlich, dass dem Konstruktor der *ColorMaterial*-Klasse übergebene Farbe nicht auf den Typ *Number* geprüft wird, es muss also eine ganze Zahl oder der Name der Farbe übergeben werden

Deshalb wird in der entsprechenden Implementierung die Zufallszahl gerundet:

```
material = new ColorMaterial( Math.round( Math.random() * 0xFFFFFF ) );
```

Abbildung 15, Instanziierung der *ColorMaterial*-Klasse mit einem Farbwert als ganze Zahl.

Das Alternativa3D-Framework arbeitet bei der Rotation ganz anders als der Flash Player oder die übrigen beiden Frameworks. Anstatt die Rotation in Grad anzugeben beziehungs-

weise anzunehmen, arbeitet Alternativa3D intern immer in der Maßeinheit Radiant. Entsprechend müssen alle Rotationen umgerechnet werden.

5.2. CoverFlow – Adaption von Apples iTunes Cover Flow

Die CoverFlow-Applikation versucht als konkretes Anwendungsbeispiel den Cover Flow Effekt von Apples iTunes nachzuahmen. Zusätzlich wird dabei die Google-Suche implementiert, um nach Bildern zu suchen, die dann auch angezeigt werden. Ein rudimentäres Scrolling, also die Navigation durch die Bilder, geschieht über die Pfeiltasten. Aus technischer Sicht stellt CoverFlow eine Möglichkeit dar, die Frameworks und den Flash Player 10 auf korrekter dreidimensionaler Verzerrung der Texturen zu untersuchen.

Die Suche arbeitet über eine Schnittstelle mit der Google-Suche zusammen. Diese Schnittstelle basiert dabei auf der Flash-eigenen *HTTPService*-Klasse, welche bei einer konkreten Suche eine Anfrage an Google sendet und auf eine entsprechende Antwort wartet. Der *HTTPService* wird von der *Searcher*-Klasse verwaltet, die sämtliche Suchanfragen weiterleitet und alle Antworten der Applikation zur Verfügung stellt. Die Antworten vom Google-Server sind dabei JSON[58]-kodiert, werden dekodiert und anschließend über die *SearchHandler*-Klasse als *SearchDataItem* bereitgestellt. Jedes *SearchDataItem* enthält die entsprechende URL zu einem Bild, das geladen wird.

Verwaltet und angezeigt wird das eigentliche CoverFlow durch die *ASCoverFlow*-Klasse. Diese bekommt sämtliche Er-

[58] JSON: JavaScript Object Notation – ein für Mensch und Maschine leicht lesbares Datenformat, ähnlich wie XML

gebnisse und instanziiert für jedes ein neues CoverFlow-Objekt vom Typ *ASCoverFlowItem*. Jedes Objekt für sich lädt das entsprechende Bild, zeigt es an und erzeugt eine Spiegelung von sich. Dazu dient die *ReflectionSprite*-Klasse.

Die Version für den Flash Player 10 enthält keine spezielle Methode, um den Render-Vorgang anzuwerfen. Die Z-Sortierung läuft in diesem Fall über den simplen Weg, dass lediglich das *ASCoverFlowItem* nach vorne geholt wird, welches auch wirklich zentral angezeigt werden soll. Dies geschieht über die Methode *setHighestChildIndex()*:

```
 1   package de.stefanvonderkrone.ascoverflow {
  ...
10       import de.stefanvonderkrone.ascoverflow.ASCoverFlowItem;
  ...
14       public class ASCoverFlow extends Sprite {
15
16           private var _coverFlowItemList : Vector.<ASCoverFlowItem>;
  ...
115          private function setHighestChildIndex( child : ASCoverFlowItem ) : void {
116              setChildIndex( child, (_coverFlowItemList.length - 1) );
117          }
```

Abbildung 16, die Methode *setHighestChildIndex()*

Die Implementierungen der 3D-Frameworks enthalten zusätzlich die *ASCoverFlowPlane*-Klasse, die vom *ASCoverFlowItem* erzeugt wird und letztendlich den Körper darstellt, der von der 3D-Engine angezeigt werden soll. Die *ASCoverFlow*-Klasse dient dabei als 3D-Container, der zusätzlich bei jeder Änderung ein Event wirft, das dafür sorgt, dass der Render-Prozess gestartet wird. Dieser wird nicht zwangsläufig in jedem neuen Frame gestartet, sondern nur, wenn es auch wirklich erforderlich ist.

5.3. Heightmap – Darstellung eines Graustufenbitmaps als Höhenkarte

Die Heightmap-Applikation ist ein Experiment, bei dem ein zufällig generiertes Graustufen-Bitmap, welches sich mit jedem neuen Frame stetig ändert, als Höheninformation für die y-Koordinate mehrerer Objekte im Raum dient. Vergleichbar ist dies mit speziellen Shader-Programmen für moderne Grafikkarten (ab DirectX 8), die ebene Texturen anhand der Lichtverhältnisse in der Szene derart verändern, dass sie plastisch erscheinen – als wären sie dreidimensional. Der Fachbegriff für diesen Effekt lautet Normal-Mapping. Die Entwickler des Alternativa3D-Frameworks konnten bereits erfolgreich einen solchen Effekt für den Flash Player entwickeln[59].

Ein verwandter Begriff ist das Height-Mapping, der die Funktionalität der Heightmap-Applikation genauer erklärt. Viele 3D-Engines bilden ganze Landschaften nicht mehr anhand von unzähligen Polygon-Daten ab, sondern nutzen dabei Graustufen-Texturen, bei der der Farbwert jedes Pixels die Höhe eines Punktes in der Landschaft darstellt. Anhand der Punkte berechnet die Engine im Anschluss selbstständig die benötigten Polygone.

Zusätzlich ist die Heightmap ein Beispiel für Partikel-Animation, besonders von Flüssigkeiten. Durch die Erhöhung der Menge an Punkten und dem Hinzufügen von Lichtinforma-

[59] http://blog.alternativaplatform.com/en/2007/12/30/spot-normal-mapping/

tionen, kann dabei der Eindruck einer sich stetig ändernden dreidimensionalen Oberfläche erweckt werden.

Technisch basiert die Applikation auf ein *BitmapData*-Objekt, das sich wie oben beschrieben stetig ändert. Basierend darauf werden in jedem neuen Frame die *y*-Positionen der Partikel aktualisiert. Um die Leistungsfähigkeit stärker auf die Probe zu stellen, kommt zusätzlich noch eine Rotation des Containers um die *y*-Achse hinzu. Mit dem Mausrad ist es zudem möglich, eine Art Tiefenschärfe-Effekt anzupassen, in dem der Schärfehorizont in der Tiefe des Raumes verändert werden kann.

In der Version für den Flash Player 10 wird nicht einfach der Container der Partikel rotiert. Es hat sich herausgestellt, dass es stattdessen weitaus performanter ist, die Partikel anhand der globalen Rotation und ihrer eigenen Position neu zu positionieren. Die folgende Abbildung soll die Implementierung dazu zeigen:

```
  1   package de.stefanvonderkrone.effects {
  ...
 18       public class Bitmap3DEffect extends Sprite {
  ...
 32           private var _yRotation : Number = 0;
  ...
201
202           private function rotateByAngle( yRotation : Number ) : void {
203               var newAngle : Number = Angle.toRadian( yRotation );
204               var radius : Number;
205               var oldAngle : Number;
206               var angle : Number;
207               for ( var i : int = 0; i < _xNum ; i++ ) {
208                   var _zItems : Vector.<Sprite> = _xItems[ i ];
209                   for ( var k : int = 0; k < _zNum ; k++ ) {
210                       radius = Math.sqrt( Math.pow( _zItems[ k ].x, 2 ) +
                            Math.pow( _zItems[ k ].z, 2 ) );
211                       oldAngle = Math.acos( _zItems[ k ].x / radius );
212                       if ( Math.asin( _zItems[ k ].z / radius ) < 0 )
                            oldAngle = -oldAngle;
213                       angle = oldAngle + ( _yRotation - newAngle );
214                       _zItems[ k ].x = radius * Math.cos( angle );
215                       _zItems[ k ].z = radius * Math.sin( angle );
216                   }
217               }
218               _yRotation = newAngle;
219           }
```

Abbildung 17, Rotation der Partikel um den Mittelpunkt ihres Containers

Die Portierungen auf die jeweiligen 3D-Frameworks arbeiten genauso. Es haben sich zwar beim Entwickeln keine derart großen Leistungsunterschiede wie bei der Version für den Flash Player 10 ergeben, aber zum besseren Vergleich wurde es hierbei belassen.

Eine Besonderheit betrifft zusätzlich die Portierung für das Alternativa3D-Framework. Hier ist der Tiefenschärfe-Effekt nicht möglich, da das Framework noch keine Filter, wie zum Beispiel ein Weichzeichnungsfilter, für 3D-Objekte unterstützt.

6. Untersuchungsmethodik

In diesem Kapitel soll die Untersuchungsmethodik genau beleuchtet werden, welche sich dabei auf drei wesentliche Bereiche erstreckt:

- Performance (der Einfluss der jeweiligen Applikation auf die Leistungsfähigkeit des Testrechners sowie des Flash Players)
- Qualität der Optik (gibt es Verzerrungen oder sonstige optische Auffälligkeiten, die die Qualität mindern oder gar erhöhen)
- Quantität (Dateigröße der jeweiligen Applikation, zusätzlich der Aufwand ihrer Umsetzung, also der Umfang des Quelltextes)

6.1. Das Testsystem

Zuallererst sollten einige Worte zum verwendeten Testsystem verloren werden. Für die Untersuchung der Performance wird ein etwa fünf Jahre alter Rechner mit Windows XP SP3 benutzt. Der Rechner besteht aus einem Prozessor vom Typ AMD Athlon X2 3800+mit 2 Gigahertz, zwei Gigabyte Arbeitsspeicher und einer Grafikkarte vom Typ Nvidia 8800 GTS 512MB. Abgesehen von der Grafikkarte entspricht der Rechner in der Leistungsfähigkeit der unteren Mittelklasse. Als repräsentativ kann man dieses System natürlich keineswegs bezeichnen, es hilft aber, die Ergebnisse besser in den Kontext der Leistungsfähigkeit aktueller Rechnersysteme einzuordnen. Abgesehen von Neuentwicklungen wie den sogenannten NetBooks und NetTops, kann man davon ausgehen, dass neuere Rechnersysteme mindestens die Leistung des verwendeten Testsystems aufweisen.

Die Fähigkeiten des neuen Flash Players, die Grafikkarte zum Rendern hinzuzuziehen wird nicht benutzt, da davon auszugehen ist, dass dies zunächst noch selten Anwendung findet. Seitens Adobe gibt es auch keine Garantie, dass die Verwendung der Grafikkarte einen Leistungsgewinn mit sich bringt. Es kann sogar dazu führen, dass die Flash-Applikation schlechter läuft.[60] Alle Applikationen sind als Release kompiliert worden, sie enthalten deshalb auch keinerlei Debug-Informationen, welche die Dateigröße verfälschen würden. Zu-

[60] http://www.kaourantin.net/2008/05/what-does-gpu-acceleration-mean.html

dem wird auf dem Testsystem der Flash Player in der ausführbaren Version[61] verwendet, nicht in der Plugin[62]- oder ActiveX[63]-Version.

Bildschirmfotos der Test-Applikationen wurden auf einem deutlich jüngerem Notebook mit Windows 7 RC Build 7100 angefertigt. Sie sollten aber aufgrund der Plattformunabhängigkeit des Flash Players keine Unterschiede zu Windows XP SP3 aufweisen.

[61] Unter anderem unter http://www.adobe.com/support/flashplayer/downloads.html zu finden

[62] Für Mozilla Firefox, Apple Safari, Google Chrome oder Opera

[63] Für den Microsoft Internet Explorer

6.2. Performance

Die Untersuchung des Einflusses auf die Leistungsfähigkeit des Testrechners bzw. des Flash Players ist die bei weitem Umfangreichste. Hier werden mit Hilfe zweier kleiner Programme vier Aspekte beobachtet und aufgezeichnet:

- Bilder pro Sekunde
- Flash-interner Speicherverbrauch
- CPU-Auslastung
- RAM-Auslastung

Die ersten beiden Punkte werden durch das eigens entwickelte Flash-Programm *LocalConnectionProtocoller* begutachtet und aufgezeichnet. Die letzten beiden Punkte übernimmt ein kleines Werkzeug namens *ProcessLogger*[64], das in C# geschrieben wurde und somit auf Microsofts .NET-Plattform basiert. Dadurch ist es möglich, die systemnahen Resourcen eines Prozesses zu untersuchen, was mit Flash-spezifischen Möglichkeiten nicht machbar ist. Beide Programme zeichnen lediglich die Rohdaten auf, da die Untersuchung nicht durch intensive Berechnungen beeinflusst werden soll. Zusätzlich sind sie quasi über einen einheitlichen Zeitstempel synchronisiert. Dies kann leider nicht mit absoluter Genauigkeit geschehen, da beide Programme nicht absolut zeitgleich gestartet werden können. Aber die zeitliche Abweichung wäre maximal nur die Hälfte des Intervalls, mit dem die Daten regelmäßig erfasst werden.

[64] Clemens Petzold, 2009

Der *LocalConnectionProtocoller* arbeitet intern mit einer erweiterten und somit namensgebenden *LocalConnection* und ist mit der jeweiligen Test-Applikation verbunden. Nachdem der Protokoll-Vorgang gestartet wurde, sendet die Test-Applikation mit Hilfe der internen *Protokoller*-Klasse im vorgegebenen Intervall die geforderten Daten an den *LocalConnectionProtocoller*, der die Daten zwischenspeichert. Am Ende des Vorgangs können die Daten in eine CSV[65]-Datei gespeichert werden, die später in Microsoft Excel ausgewertet wird. Darin enthalten sind dann drei wesentliche Datentypen:

- Der Zeitstempel in Millisekunden
- Die Anzahl der verstrichenen Frames seit dem letzten Intervall
- Der aktuelle Flash-interne Speicherverbrauch

Der *ProcessLogger* wird manuell vor dem Protokoll-Vorgang gestartet und danach wieder beendet. Auch er speichert die Daten in eine CSV-Datei. Wichtige Daten sind hier der *Zeitindex* des Testsystems, die *Gesamtprozessorzeit* des Prozesses sowie *die Auslastung des physischen Speichers* durch den jeweiligen Prozess[66]. Es muss aber angemerkt werden, dass es Diskrepanzen zwischen der Zeitrechnung in C# und der im Flash Player. In C# wird der Zeitstempel in zehntel Millisekunden seit dem 1. Januar 0001 um 12 Uhr berechnet[67]. Der

---

[65] CSV: Abkürzung für Comma Seperated Values, weit verbreitetes Format für die Darstellung tabellarischer Daten

[66] In diesem Fall der Flash Player

[67] http://msdn.microsoft.com/de-de/library/system.datetime.ticks.aspx

Flash Player hingegen berechnet den Zeitstempel in Millisekunden seit Mitternacht des 1. Januars 1970. Um diese Diskrepanz auszugleichen, wurde der *ProcessLogger* entsprechend angepasst und rechnet deshalb den *Zeitindex* in eine Flash-konforme Zeit um.

6.2.1. Messabweichungen

Die Messung der Performance ist ein kritischer Punkt in dieser Untersuchung, weil sie unter Bedingungen durchgeführt wird, die keine absolute Genauigkeit garantieren. Aus diesem Grund muss das Ziel sein, diese Genauigkeit so hoch wie möglich zu halten, um im Endeffekt zu einer hieb- und stichfesten Schlussfolgerung aus den Messergebnissen zu kommen.

Da der *ProcessLogger* sehr systemnah arbeitet, kann davon ausgegangen werden, dass dieser keinen großen Einfluss auf die Messungen haben wird. Eine kurze Messung hat dies abgesehen von vereinzelten kleinen Ausbrüchen bestätigt. Im Durchschnitt belastet der *ProcessLogger* die CPU mit weniger als zwei Prozent nur unwesentlich. Auch der Arbeitsspeicherverbrauch von maximal 15 Megabyte fällt beim Testsystem mit 2 Gigabyte nicht weiter ins Gewicht.

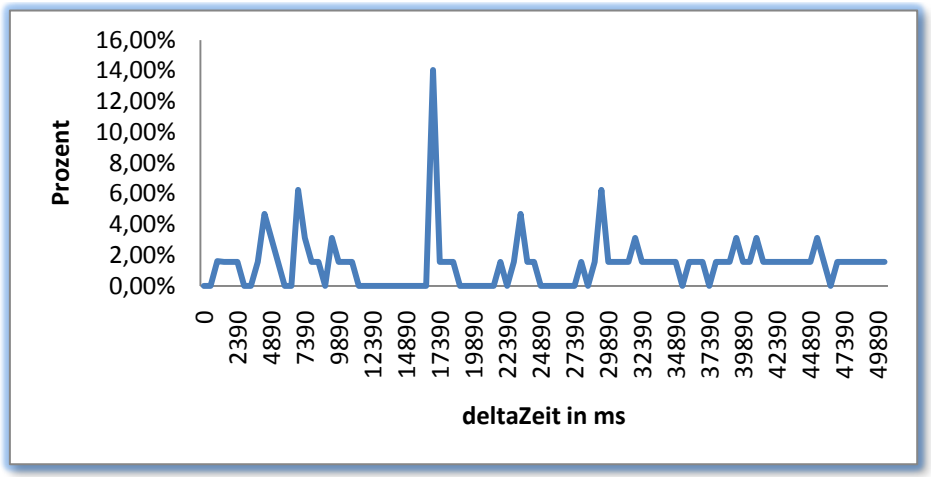

Abbildung 18, Verlauf der CPU-Auslastung des *ProcessLoggers*

Anders sieht dies bei den Daten aus, die durch den *LocalConnectionProtocoller* erhoben werden. Aufgrund der plattformunabhängigen Architektur des Flash Players ist er unter hohen Anforderungen bei weitem nicht so leistungsfähig wie native beziehungsweise plattformspezifische Anwendungen. Deshalb ist eine Prüfung des Einflusses des *LocalConnectionProtocollers* auf die Messdaten verpflichtend. Aus diesem Grund wurde eine entsprechende Messreihe am Beispiel der Heightmap-Applikation in der Version für den Flash Player 10 durchgeführt. Die Messreihe bestand dabei aus drei Messungen mit unterschiedlichen Intervallen:

- 1000 Millisekunden
- 500 Millisekunden
- 100 Millisekunden

Im Folgenden sind die Verläufe der Bilder pro Sekunde für alle drei Intervalle in Diagrammen aufgeführt. Die erhobenen Rohdaten sind auf der beiliegenden CD beziehungsweise im Internet unter folgender Adresse einsehbar:

http://www.stefanvonderkrone.de/bachelorarbeit/

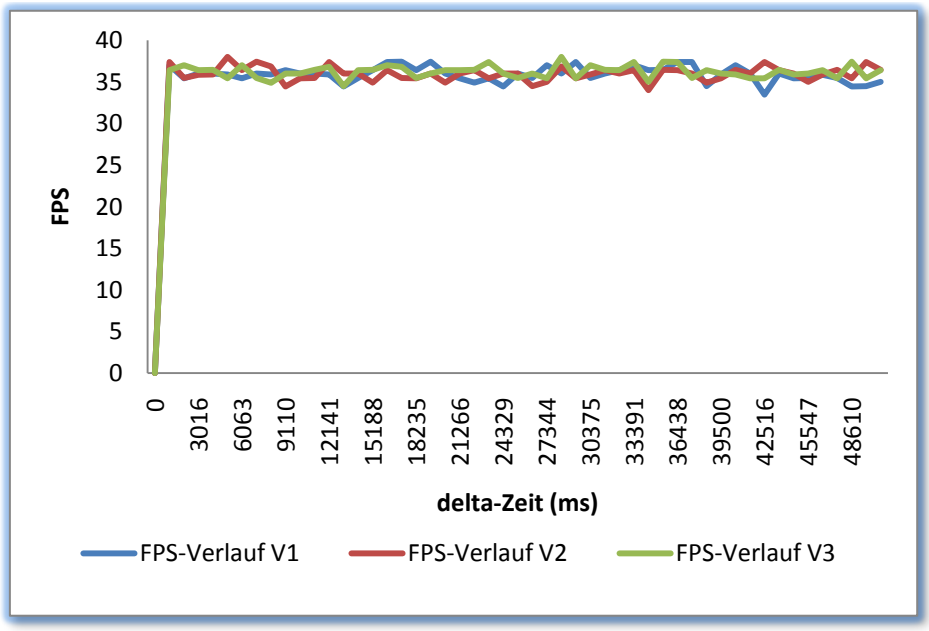

Abbildung 19, FPS[68]-Verlauf für 1000ms

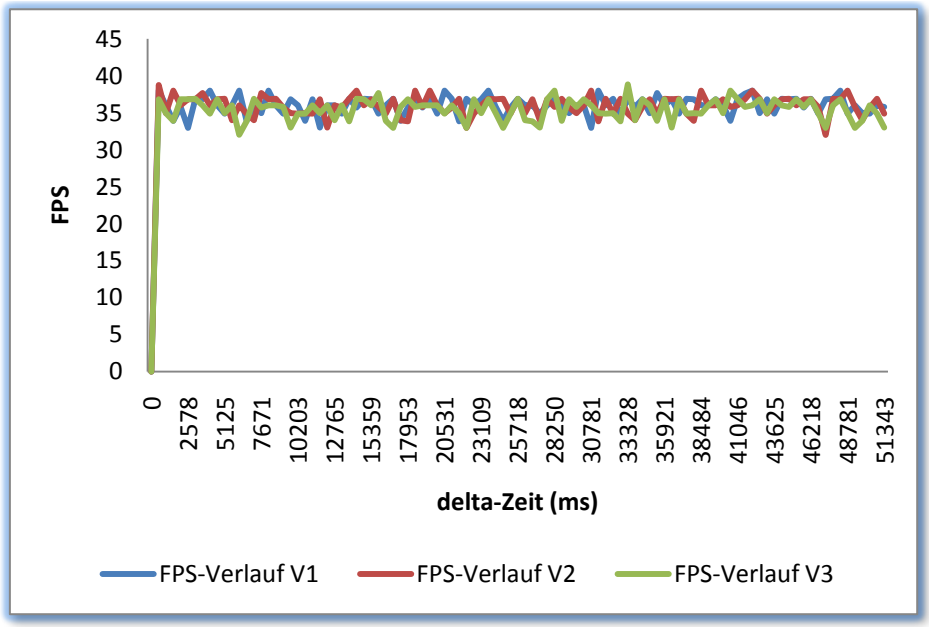

Abbildung 20, FPS-Verlauf für 500ms

[68] FPS (Frames Per Seconds): Bilder pro Sekunde

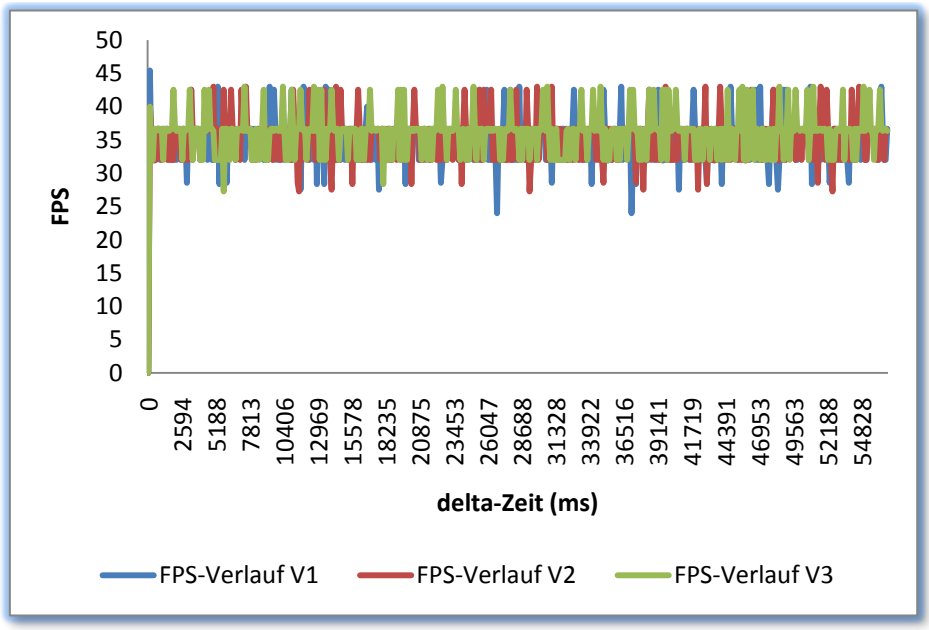

Abbildung 21, FPS-Verlauf für 100ms

Auf der Ordinate ist der jeweilige Wert für die Bilder pro Sekunde seit dem letzten Intervall verzeichnet. Die Abszisse bildet den zeitlichen Ablauf ab. Für jedes Intervall wurden vier Messvorgänge durchgeführt, wobei der Erste wieder verworfen wurde.

Die Diagramme lassen erkennen, wie sich die FPS-Verläufe mit unterschiedlichen Intervallen verhalten. So ist die Streuung bei einem Intervall von 100 Millisekunden deutlich größer als bei 500 Millisekunden. Und dessen Streuung wiederum ist merklich größer als bei 1000 Millisekunden. Dies lässt sich sehr leicht dadurch erklären, dass die Berechnung des FPS-Wertes von der Größe des Intervalls abhängig ist und eigentlich nur den Mittelwert für die Anzahl der Bilder pro Sekunde innerhalb des entsprechenden Intervalls darstellt. Daraus lässt sich ableiten, dass mit länger werdendem Intervall

der FPS-Verlauf glatter wird. Ein Indiz für eine Beeinflussung der Messdaten durch den *LocalConnectionProtocoller* stellen die Diagramme aber noch nicht ganz dar. Erst mit den Mittelwerten für alle Intervalle wird die Aussage der Daten eindeutiger:

Tabelle 1, Mittelwerte sowie Maxima und Minima über drei Messungen für jedes Intervall

Durchschnitte der drei Intervalle:	1000ms	500ms	100ms
Zeit:	50661,67	51406,00	56344,00
Frames:	1826,33	1841,67	1994,00
Frames pro Sekunde:	36,05	35,83	35,39
FPS - Maximum gesamt:	38,00	38,83	45,45
FPS - Minimum gesamt:	33,46	32,00	24,00
Speicher (Byte):	6580290,93	6595979,41	6598132,76
Speicher - Maximum gesamt (Byte):	7041024,00	7077888,00	7065600,00
Speicher - Minimum gesamt (Byte):	6094848,00	6119424,00	6131712,00

Wie bereits erwähnt sind die Daten für das 1000 Millisekunden große Intervall wenig auffällig, der Abstand zwischen FPS-Maximum und FPS-Minimum ist deutlich kleiner als bei den anderen beiden Intervallen. Und der durchschnittliche FPS-Wert liegt hier am höchsten. Bei 500 Millisekunden ist der Abstand zwischen Maximum und Minimum größer. Bei 100 Millisekunden merkt man die starke Streuung sehr deutlich. Hier wird auch zeitweise der höchste FPS-Wert gemessen – aber auch der Niedrigste. Im Speicherverbrauch verhält es sich annähernd analog, aber die Unterschiede sind immer deutlich

niedriger als ein Prozent – insofern sind diese Werte in der Wahl des Intervalls zu vernachlässigen.

Anhand der FPS-Werte steht eindeutig fest, dass ein Intervall von 100 Millisekunden nicht praktikabel ist, da es das Ergebnis wesentlich beeinflusst. Anders sieht dies bei 500 Millisekunden aus, hier gibt es zwar einen messbaren Einfluss, dieser liegt im Schnitt bei unter einem Prozent. Deshalb ist die Wahl des finalen Intervalls eine eher pragmatische Entscheidung. Aufgrund der stärkeren Glättung der FPS-Werte bei 1000 Millisekunden wäre das 500 Millisekunden Intervall deutlich praxistauglicher, da es starke Ausbrüche beziehungsweise Einbrüche in den zu messenden Daten weniger durch eine Glättung verschleiert. Dies ist deshalb von Bedeutung, da zum Beispiel kurze Leistungsengpässe dennoch vom Betrachter bemerkt werden. Bei einem 1000 Millisekunden Intervall würden sie aber nicht so deutlich in den Daten auffallen, als bei einem Intervall von 500 Millisekunden. Die Abweichung der Daten ist dabei immer noch geringer als ein Prozent und stellt somit einen Kompromiss aus Messabweichung und Aussagekraft dar.

6.3. Qualität

Natürlich ist die Qualität der Darstellung dreidimensionaler Szenen sehr wichtig, besonders, da auch ein räumlicher Eindruck erweckt werden soll. Diesem Aspekt widmet sich die Untersuchung ebenfalls. Dabei gibt es aber keine genauen Messvorrichtungen, es werden deshalb Screenshots verglichen und danach überprüft, ob es wesentliche Unterschiede gibt. Es geht hierbei aber mehr um Fehler oder sonstige Auffälligkeiten in der Darstellung der Szenen. Unterschiede, die sich durch verschiedene Brennweiten oder Sichtfeldern ergeben, werden hierbei aber nicht erfasst, da sie die Qualität nicht mindern.

Konkrete Beispiele für Darstellungsfehler wären schlechte Überlagerungen von 3D-Körpern, wenn sie ineinander positioniert werden oder unnatürlich wirkende Verzerrungen der Texturen.

6.4. Quantität

Unter dem Gesichtspunkt der Quantität werden zwei Eigenschaften der Applikationen betrachtet. Das sind zum einen die Dateigröße der resultierenden SWF-Datei und zum anderen der Umfang des reinen Quelltextes, der für die Applikation geschrieben wurde.

Der erste Punkt ist insofern von Bedeutung, da es für den Benutzer am Ende unangenehm werden kann, wenn er sehr lange auf die eigentliche Applikation warten muss, da sie erst noch geladen werden muss. Nun betrifft das sicher nur Besitzer einer eher schwachen Netzverbindung – zum Beispiel Modem, ISDN[69], GPRS[70] sowie schwaches oder überlastetes WLAN[71]. Weiterhin stellt sich auch die Frage, ob sich ein Zusatz von circa 50 bis 100 Kilobyte für eine kleine 3D-Animation lohnen würde.

Der Umfang des reinen Quelltextes stellt dabei ein Indiz für den Aufwand der Programmierung der Applikation sowie der Implementierung des 3D-Frameworks dar. Dieser Aspekt ist aber nicht repräsentativ für eine allgemeine bzw. generelle Implementierung der verwendeten Frameworks zu verstehen, da jeder Programmierer einen eigenen Stil besitzt und jede neue Applikation andere Herangehensweisen erfordert. Die

---

[69] ISDN (Integrated Services Digital Network) – internationaler Standard für die digitale Telekommunikation

[70] GPRS (General Packet Radio Service) – Paketorientierter Dienst zur Datenübertragung, findet in Mobilfunknetzen Verwendung

[71] WLAN (Wireless Local Area Network) – lokales Funknetzwerk nach dem Standard IEEE 802.11

Suche nach der optimalen Implementierung für das jeweilige Framework hätte jeden Aufwand gesprengt, deshalb beschränkt sich die Programmierung auf einheitliche – und auch besser vergleichbare – Implementierungen.

Der Umfang des Quelltextes wird dabei folgendermaßen erfasst: Alle Kommentare, ob einzeilig oder mehrzeilig, sowie alle leeren Zeilen werden entfernt. Öffnende geschweifte Klammern kommen immer am Ende der dazugehörigen Deklaration, also am Ende einer *package*-, *class*-, *interface*- oder *function*-Deklaration. Das schließende Pendant dazu befindet sich immer auf einer eigenen Zeile. Ansonsten endet jede Zeile mit einem abschließenden Semikolon, vorher kommt kein Umbruch. In die Zählung werden nur eigene Klassen einbezogen. Externe Bibliotheken anderer Programmierer werden nicht mitgezählt. Diese sind meist aber auch nur als SWC[72]-Datei eingebunden. Zusätzlich werden die *Protokoller*-Klasse und dessen Implementierung komplett aus der Erfassung herausgenommen, da sie nur für die Untersuchung der Performance von Belang ist im Endprodukt aber keine Rolle spielt.

[72] SWC – Flash-Komponentendatei, enthält Komponenten oder vorkompilierte Klassen

7. Auswertung

Im folgenden Kapitel werden die Untersuchungsergebnisse ausgewertet. Diagramme dienen dabei der Veranschaulichung der Ergebnisse. Eine komplette Auflistung der Diagramme befindet sich auf der beigefügten CD. Nähere Infos befinden sich im Anhang.

7.1. Performance

7.1.1. Cube3D

Den Anfang macht mit „Cube3D“ die einzige Applikation, in der die 3D-Frameworks einen deutlichen Leistungsvorsprung gegenüber dem reinen Flash Player 10 aufweisen. Das Cube3D-Szenario hat sich damit als prädestiniertes Einsatzgebiet für die 3D-Frameworks erwiesen.

Am schnellsten läuft dabei die Away3D-Implementierung mit durchschnittlich 96 Bildern pro Sekunde. Erwähnenswert dabei ist, dass die Framerate teilweise über 100 Bildern pro Sekunde liegt, obwohl die SWF-Datei lediglich für 100 Bilder pro Sekunde kompiliert wurde. Vermutlich liegt dies intern an einer dynamischen Anpassung der Framerate oder der effizienten Nutzung der Methode *updateAfterEvent()* der Klasse *TimerEvent*.

Im Mittelfeld bewegen sich die Implementierungen der Alternativa3D- und Papervision3D-Frameworks mit jeweils etwa 55 beziehungsweise 45 Bildern pro Sekunde. Papervision3D zeigt hier trotz der recht hohen Framerate die stärksten Schwankungen mit Werten von eins bis 82. Die Bewegungen sind stark ruckelnd und es zeigen sich oft kurze Phasen des kompletten Stillstands. Wie das zustande kommt, lässt sich nicht erklären, da die anderen beiden Implementierungen des Papervision3D-Frameworks deutlich schwankungsärmer laufen. Vermutlich steht das im Zusammenhang mit der noch

recht intensiven Entwicklung an Papervision3D, das sich in der aktuellen Version 2.0 noch im Beta-Stadium befindet.

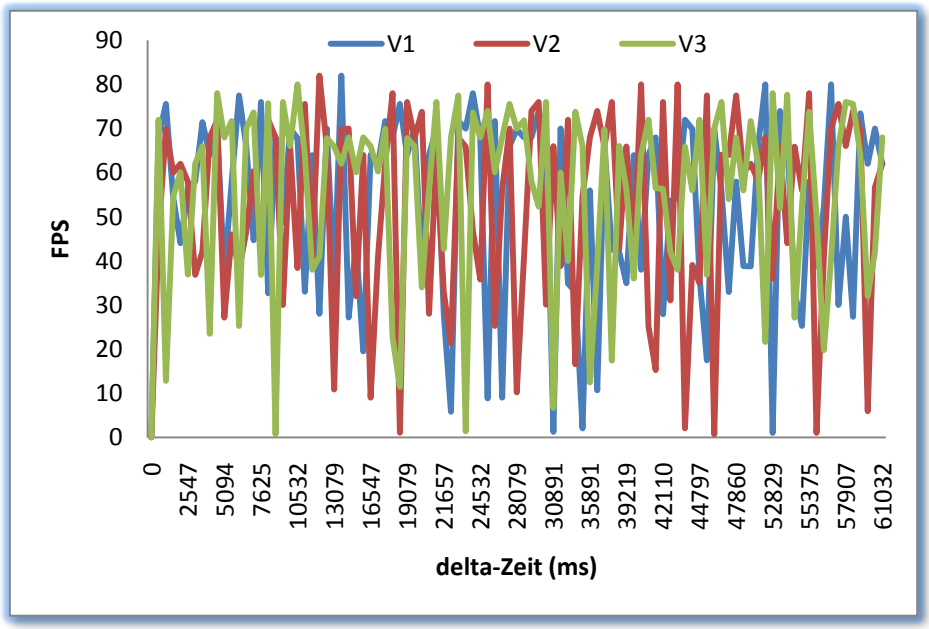

Abbildung 22, FPS-Verlauf Cub3D mit Papervision3D

Der Flash-interne Speicherverbrauch zeigt keine derart extremen Unterschiede, lediglich die Implementierung für den Flash Player 10 weist mit bis zu 17 Megabyte einen höheren Wert auf. Die restlichen Versionen bleiben im Bereich um maximal sechs bis sieben Megabyte. Besonders auffällig sind dabei die Speicher-Verläufe der Flash Player 10 und der Away3D-Version, bei denen die Grafen aller drei jeweiligen Versuche annähernd identisch aufeinander abbildbar sind.

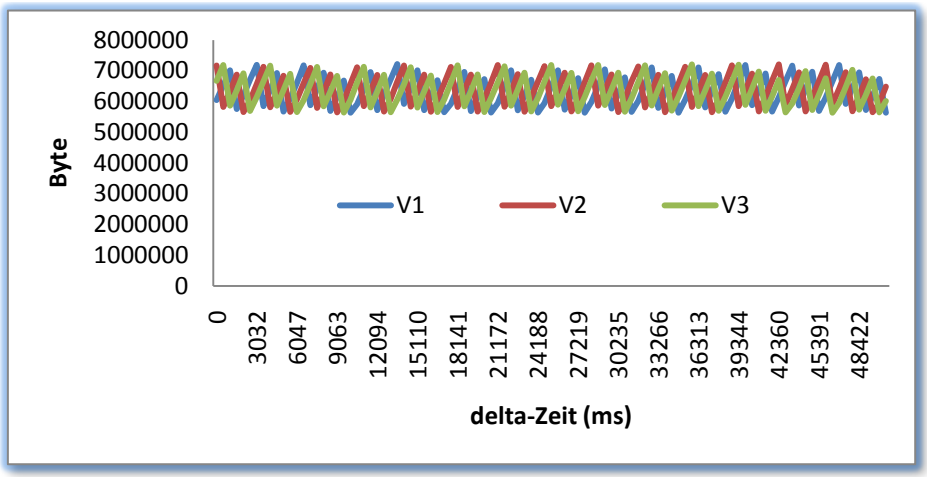

Abbildung 23, Speicher-Verlauf Cube3D mit Away3D

Die CPU-Auslastung liegt im Schnitt unter 70 Prozent, am genügsamsten war dabei Alternativa3D, welche die CPU durchschnittlich mit 60 Prozent auslastet. Zusammen mit der Version für den Flash Player 10 steigt diese nie über 77 Prozent. Papervision3D lässt die Last teilweise bis auf 92 Prozent ansteigen.

Der RAM-Verbrauch verhält sich analog zum Flash-internen Speicherverbrauch. Die 3D-Frameworks verursachen einen Verbrauch von maximal 19 Megabyte. Die Implementierung des Flash Player 10 lasten den Arbeitsspeicher mit bis zu 29 Megabyte aus.

Tabelle 2, Detailinformationen

	Alternativa3D	Away3D	Flash Player 10	Papervision3D
Frames pro Sekunde	54,58	95,84	18,45	45,19
Maximum	66,00	104,00	22,60	82,00
Minimum	44,57	80,00	14,63	0,69
durchschnittlicher interner Speicher (Megabyte)	5,34	6,06	16,57	5,38
Maximum (Megabyte)	6,25	6,89	16,98	6,01
Minimum (Megabyte)	4,63	5,38	16,18	4,75
durchschnittliche CPU-Auslastung (Prozent)	59,70	69,35	67,85	67,28
Maximum (Prozent)	76,56	84,38	76,56	92,19
Minimum (Prozent)	42,19	56,25	54,69	41,56
durchschnittlicher RAM-Auslastung (Megabyte)	18,07	18,63	28,60	17,81
Maximum (Megabyte)	18,13	18,67	28,62	17,82
Minimum (Megabyte)	17,84	18,49	28,42	17,54

7.1.2. CoverFlow

Mit der Applikation „CoverFlow“ zeigen sich die enormen Anforderungen, mit denen die 3D-Frameworks konfrontiert werden können. Alle drei Frameworks laufen im Schnitt mit 13 bis 15 Bildern pro Sekunde deutlich langsamer als die Version für den Flash Player 10, welche mit durchschnittlich 63 Bildern pro Sekunde arbeitet. Besonders auffällig ist bei allen eine Berg-Tal-Struktur im FPS-Diagramm. Das lässt sich damit erklären, dass zu den jeweiligen Zeitpunkten der Erfassung unterschiedlich viele Bilder abhängig von der Position in der Liste angezeigt werden. Am Anfang und Ende der Liste wird

jeweils immer nur knapp die Hälfte der Bildschirmbreite mit Bildern gefüllt, was die FPS-Werte aufgrund weniger umfangreicher Berechnungen in die Höhe treibt. Je mehr Bilder angezeigt werden müssen, desto niedriger fällt die Bildrate aus.

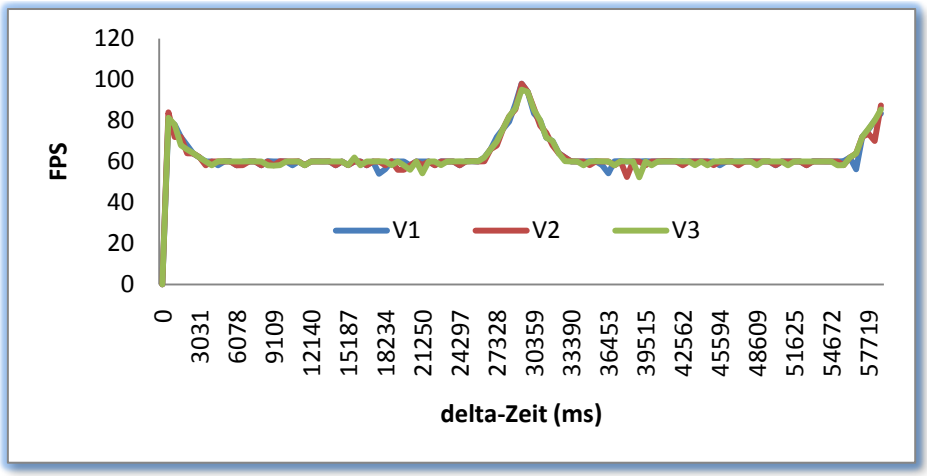

Abbildung 24, FPS-Verlauf CoverFlow für den Flash Player 10

Beim Flash-internen Speicherverbrauch liegen alle vier Versionen mit etwa 44 bis 49 Megabyte recht nah beieinander. In den Diagrammen durch einen abrupten Abfall der Kurve sehr gut erkennbar sind die Eingriffe der Garbage Collection, die den internen Speicher soweit wie möglich bereinigt. Leider lässt sich der Einsatz der Garbage Collection weder steuern noch sonst irgendwie vorhersagen, was sich durch die unterschiedlichen Momente des Eingreifens zeigt. Am besten ist dies im entsprechenden Diagramm für das PaperVsion3D-Framework erkennbar. Mit jedem Eingriff werden im Schnitt in allen Programmen immer etwa 5 Megabyte frei.

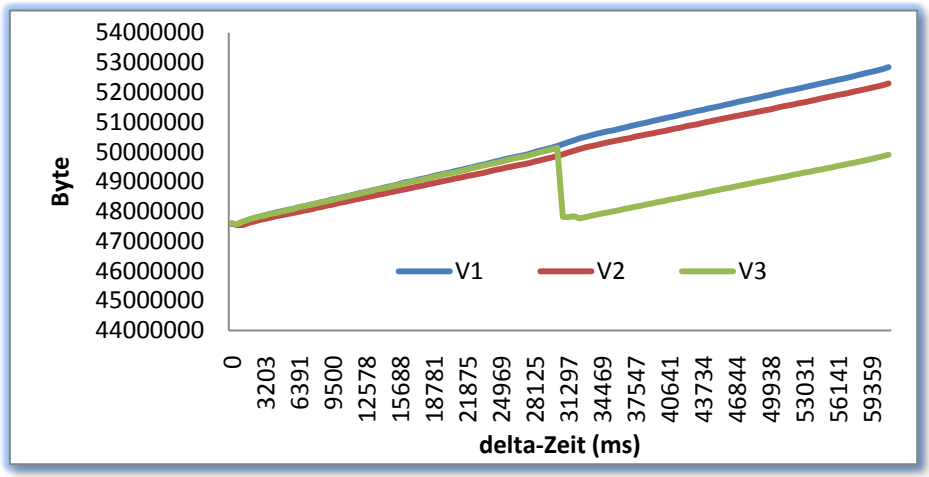

Abbildung 25, einmaliger Eingriff der Garbage Collection im Beispiel des Papervision3D-Frameworks

Ein ähnliches Bild zeigt sich bei der CPU-Auslastung, die im Schnitt bei etwa 77 bis 82 Prozent recht hoch ausfällt. Dabei schwankt die Auslastung immer um etwa 20 Prozentpunkte. Die Ausschläge sind beim Flash Player 10 mit 59 bis 96 Prozent am Höchsten. Bei den 3D-Frameworks übersteigt die Maximal-Auslastung niemals 91 Prozent, dennoch bedeuten diese Werte enorme Einbußen für ein flüssiges Arbeiten mit dem Testsystem.

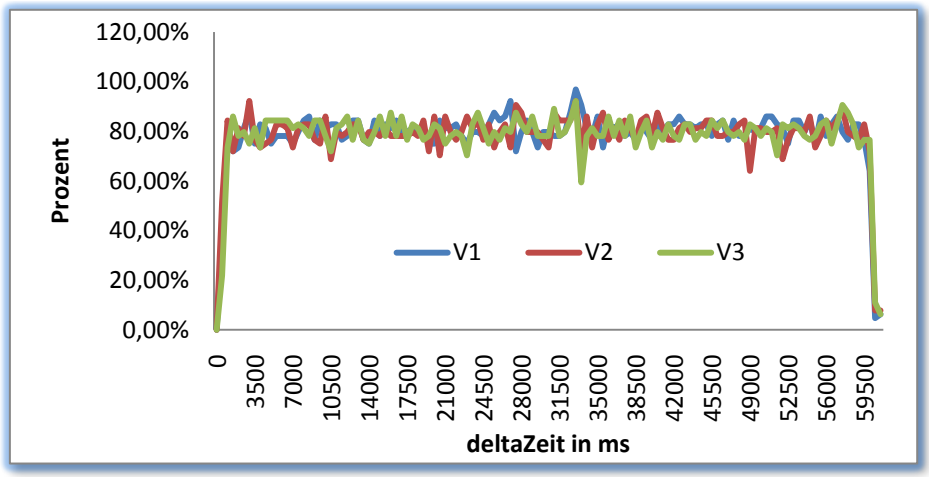

Abbildung 26, CPU-Auslastung CoverFlow für den Flash Player 10

Der RAM-Verbrauch offenbart wieder etwas größere Unterschiede. So sind die Versionen mit Alternativa3D- und Papervision3D-Framework mit bis zu 75 beziehungsweise 76 Megabyte deutlich genügsamer. Flash Player 10 genehmigt sich mit bis zu 88 Megabyte den meisten Anteil am RAM.

Tabelle 3, Detailinformationen

	Alternativa3D	Away3D	Flash Player 10	Papervision3D
Frames pro Sekunde	13,20	15,06	63,40	12,86
Maximum	25,24	27,18	98,00	22,00
Minimum	10,38	11,63	52,33	10,66
durchschnittlicher interner Speicher (Megabyte)	46,27	48,70	44,19	47,36
Maximum (Megabyte)	49,31	53,43	46,58	50,46
Minimum (Megabyte)	43,92	45,46	41,20	45,34
durchschnittliche CPU-Auslastung (Prozent)	76,91	81,53	78,82	78,38
Maximum (Prozent)	90,63	89,06	96,88	89,06
Minimum (Prozent)	0,00	0,00	4,69	0,00
durchschnittlicher RAM-Auslastung (Megabyte)	72,12	79,14	83,21	73,45
Maximum (Megabyte)	75,78	84,91	88,01	75,05
Minimum (Megabyte)	66,33	70,08	65,79	70,92

7.1.3. Heightmap

Die „Heightmap“-Applikation zeichnet ein ähnliches Bild, wie die „CoverFlow“-Applikation – nur deutlich intensiver. So schaffen die Implementierungen des Away3D- und des Papervision3D-Frameworks im Schnitt zwar noch etwa 13 Bilder pro Sekunde, das Alternativa3D-Framework hingegen

bricht dabei auf einen Wert von fünf ein. Eine Besonderheit ist zusätzlich, dass keine Filter unterstützt werden und die enorm niedrige Performance somit umso erstaunlicher ist. Die Version für den Flash Player schafft auch nur noch einen Wert von etwa 36 Bildern pro Sekunde.

Der Flash-interne Speicherverbrauch schwankt unter den Implementierungen von maximal sieben (Flash Player 10) bis 36 Megabyte (Papervision3D). Das Papervision3D-Framework schwankt dabei in sich am stärksten um etwa 25 Megabyte.

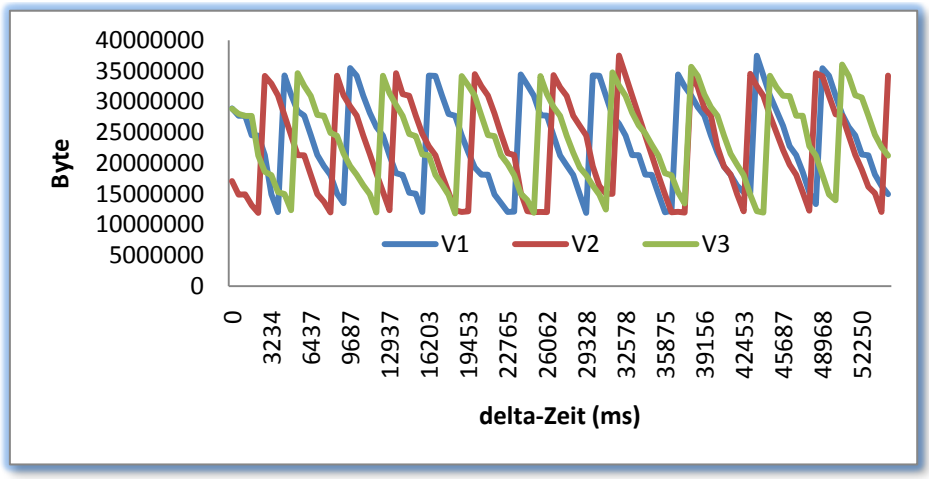

Abbildung 27, interner Speicherbverbrauch HeightMap mit Papervision3D

Die CPU-Auslastung ist dabei erstaunlich gering und steigt bei den 3D-Frameworks nie über 57 Prozent. Im Schnitt liegt der Wert bei knapp unter 50 Prozent. Mit der Version für den Flash Player 10 steigt der Wert auf bis zu 67 Prozent an. Dennoch liegen diese Werte weit unter denen der anderen beiden Applikationen Cube3D und CoverFlow.

Bei der RAM-Auslastung gibt es deutlichere Unterschiede zu verzeichnen. Mit lediglich knapp 18 Megabyte arbeitet die Flash Player 10 Version dabei am effizientesten. Im Mittelfeld liegen Alternativa3D mit 22 Megabyte und Away3D mit etwa 32 Megabyte. Papervision3D ist mit 48 Megabyte dabei am hungrigsten.

Tabelle 4, Detailinformationen

	Alternativa3D	Away3D	Flash Player 10	Papervision3D
Frames pro Sekunde	5,24	12,54	36,00	12,91
Maximum	5,65	13,59	38,83	14,00
Minimum	4,80	9,69	32,95	10,66
durchschnittlicher interner Speicher (Megabyte)	9,15	18,26	6,29	22,20
Maximum (Megabyte)	10,68	20,55	6,75	35,78
Minimum (Megabyte)	7,91	15,77	5,83	11,29
durchschnittliche CPU-Auslastung (Prozent)	49,39	48,45	56,18	49,83
Maximum (Prozent)	53,13	54,69	67,19	56,25
Minimum (Prozent)	43,75	42,19	45,31	40,63
durchschnittlicher RAM-Auslastung (Megabyte)	21,63	32,17	17,96	47,82
Maximum (Megabyte)	21,95	32,25	17,98	48,74
Minimum (Megabyte)	20,27	32,01	17,84	46,46

7.2. Quantität

7.2.1. Zeilenanzahl

Bis auf Cube3D ergibt der Aufwand in Form der Zeilenanzahl für jede Version einer Applikation keine auffällig großen Unterschiede. Die Positionen für die meisten beziehungsweise wenigsten Zeilen wechseln jeweils zwischen Away3D und der Implementierung für den Flash Player 10. Die restlichen beiden Frameworks liegen immer im Mittelfeld. Im Schnitt kann man aber feststellen, dass der Flash Player 10 den höchsten Aufwand erfordert und mit Papervision3D dabei am effizientesten programmiert werden kann. Diese Aussage trifft aber lediglich auf die Zeilenanzahl zu.

Tabelle 5, Zeilenanzahlen

	Alternativa3D	Away3D	Flash Player 10	Papervision3D
CUBE3D	308	283	638	289
COVERFLOW	988	999	830	945
HEIGHTMAP	319	307	351	317

7.2.2. Dateigrößen

Die Dateigrößenunterschiede zeichnen ein sehr klares Bild und machen deutlich, mit welchen Kosten in Form von Bytes man bei der jeweiligen Implementierung rechnen kann. Die absoluten Unterschiede zwischen den Versionen sind immer annähernd identisch. Away3D erreicht dabei immer die höchste Dateigröße. Am effizientesten ist jeweils immer die Version für den Flash Player 10, was insofern logisch ist, als dass sie kein zusätzliches Framework benötigt.

Tabelle 6, Dateigrößen

	Alternativa3D	Away3D	Flash Player 10	Papervision3D
CUBE3D	59665	134880	19991	78147
COVERFLOW	148917	225309	107040	168627
HEIGHTMAP	47885	122277	7210	67074

7.3. Qualität

7.3.1. CoverFlow

Mit CoverFlow zeigen sich zwei Probleme mit denen anscheinend die 3D-Frameworks zu kämpfen haben. Die Kanten der planaren Körper weisen eine auffällige Wellenbildung auf und in der Wechselanimation sind unterschiedliche Geschwindigkeiten der einzelnen Körper zueinander bemerkbar, obwohl die Animationsdauer jeweils immer dieselbe ist. Ersteres lässt sich durch die Unterteilung der 3D-Körper in Polygone erklären, was bei der 3D-Transformation des Flash Player 10 nicht geschieht – oder eben komplett nativ, was offensichtlich zu einer besseren Qualität führt.

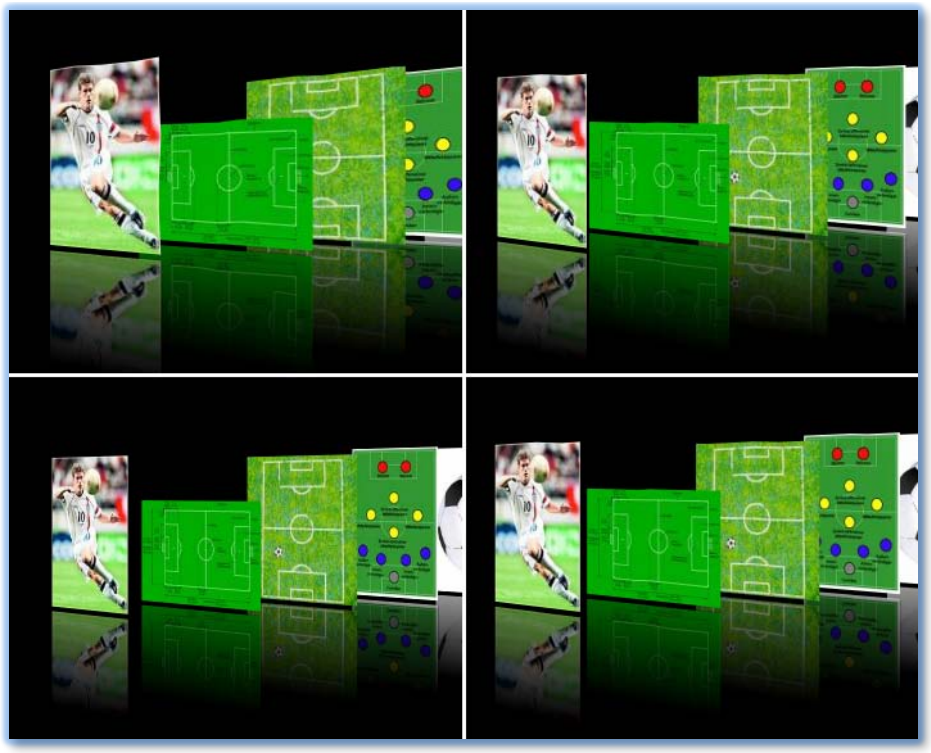

Tabelle 7, Vergleichsbilder CoverFlow in folgender Reihenfolge (zeilenweise): Alternativa3D, Away3D, Flash Player 10, Papervision3D

Das Problem mit der Animationsgeschwindigkeit lässt sich nicht so ohne Weiteres erklären. Vermutlich ist der enorme Aufwand der Berechnungen eine Ursache. Das mag auch daran liegen, dass die Animation für jeden Körper einzeln initiiert wird. Eine Optimierung wäre eine globale Animation, an der alle Körper abhängig von ihrer Position in der Liste gebunden sind. Da sich dieses Problem in einer Animation manifestiert, wäre eine entsprechende Abbildung wenig aussagekräftig, weshalb auf eine selbstständige Ausführung der CoverFlow-Applikation verwiesen wird.

7.3.2. Cube3D

Mit Cube3D zeigt sich, wie erfolgreich die einzelnen Frameworks mit Überlagerungen von Körpern umgehen. Hier sticht Alternativa3D positiv hervor, bei der die sich berührenden Körper mit korrekter Schnittkante dargestellt werden. Ein kleines Manko dessen ist ein sehr dünner Schnitt in der Textur entlang dieser Kanten. Die anderen beiden Frameworks verursachen deutliche Fehler in der Darstellung, die durch die korrekte Z-Sortierung der Polygone verursacht werden, aber nicht abgefangen werden. Dadurch passiert es, das ein Polygon des einen Körpers plötzlich hinter einem Polygon eines anderen Körpers erscheint. Die Schnittkante kann man dabei nur erahnen. Aufgrund der Z-Sortierung in der Flash Player 10 Version auf Basis zusammenhängender Körper entstehen dabei keine Schnittkanten.

Tabelle 8, Vergleichsbilder Cube3D in folgender Reihenfolge (zeilenweise): Alternativa3D, Away3D, Flash Player 10, Papervision3D

7.3.3. HeightMap

Die größten Auffälligkeiten in der HeightMap-Applikation sind die fehlenden Filter-Möglichkeit des Alternativa3D-Frameworks sowie die deutlichen Größenunterschiede der dargestellten Partikel. Letzteres ist besonders bei Away3D ersichtlich, hier sind die Partikel fast gar nicht zu erkennen. Papervision3D kommt dabei dem Flash Player 10 am nächsten.

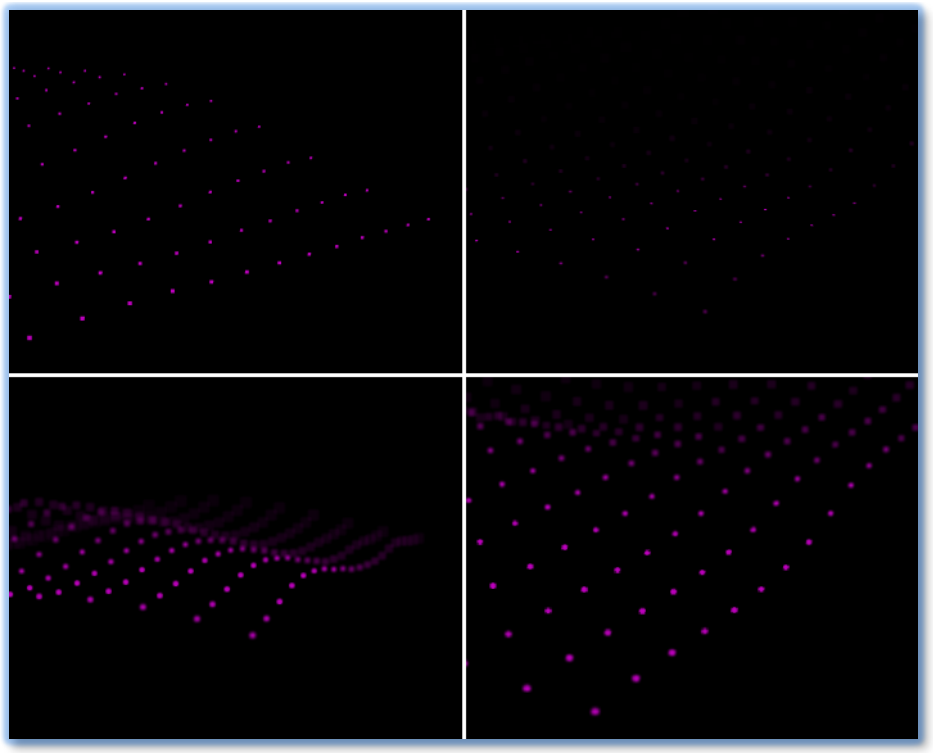

Tabelle 9, Vergleichsbilder HeightMap in folgender Reihenfolge (zeilenweise): Alternativa3D, Away3D, Flash Player 10, Papervision3D

8. Fazit

Die vorangegangenen Kapitel beschäftigten sich unter anderem mit Flash als Technologie, im Besonderen mit der Scriptsprache ActionScript. Es wurden Grundlagen für die 3D-Programmierung vermittelt und weiterführend am Beispiel von Papervision3D erläutert. Anhand konkreter Anwendungsbeispiele wurden die Möglichkeiten dreidimensionaler Inhalte in Flash vorgestellt und einer empirischen Untersuchung unterzogen.

Im Ergebnis dieser Untersuchung ist festzustellen, dass es keine absolute und eindeutig zu bevorzugende Lösung gibt, die die 3D-Programmierung in ActionScript 3.0 erleichtert oder immer zu einem leistungsfähigen Produkt führt – weder mit einem der genutzten Frameworks noch mit den Möglichkeiten des Flash Player 10. Zwei Szenarien, wie sie in dieser Bachelorarbeit betrachtet wurden, liefen optimaler mit dem Flash Player 10 (CoverFlow, HeightMap) und lediglich eines war prädestiniert für die 3D-Frameworks (Cube3D). Aber gerade Letzteres zeigte die eigentlichen Stärken der 3D-Frameworks: die Darstellung von Körpern im Raum.

Gerade hier zeigte sich für den Flash Player 10, dass es alleine mit korrekter dreidimensionaler Verzerrung von *Sprites* nicht getan ist. Der Programmierer muss einen gewissen zusätzlichen Aufwand bewältigen (zum Beispiel Z-Sortierung), um die gleichen Ergebnisse zu erreichen, wie sie mit den 3D-Frameworks möglich sind. Das liegt aber mitunter daran, dass

die neuen Möglichkeiten der 3D-Transformation lediglich Hilfsmittel sind, die durch native Berechnungen einen Geschwindigkeitszuwachs ermöglichen können. Das muss aber nicht zwangsläufig der Fall sein, wie am Beispiel von Cube3D deutlich wird. Diese nativen Schnittstellen bedeuten noch lange kein Framework im Sinne von Papervision3D, Away3D oder Alternativa3D, solch eines muss zunächst entwickelt und optimiert werden.

Von Alternativa3D[73] und Papervision3D[74] gibt es bereits Versionen für den Flash Player 10. Alternativa3D erreicht dabei ebenfalls eine höhere Leistung gegenüber der Version für den Flash Player 9. Papervision3D für den Flash Player 10 befindet sich noch in einem sehr frühen Stadium, kurze Tests zeigten sogar eine niedrigere Leistung gegenüber der Ur-Version. Anscheinend wird diese Version aber seit Ende September 2008 nicht mehr weiterentwickelt. Das Team von Away3D veröffentlichte im Verlauf dieser Arbeit mit Away3DLite eine sehr schlanke, nach eigenen Angaben maximal 25 Kilobyte beanspruchende Bibliothek, die von Grund auf neu entwickelt wurde und in Demonstrationen eine erstaunliche Leistung zeigt[75].

Allgemein stellen sich je nach Einsatzzweck und Anforderung zwei Implementierungen als vorteilhaft heraus: Away3D und Flash Player 10. Gerade Cube3D hat die enorme Leis-

---

[73] http://alternativaplatform.com/en/alternativa3d/

[74] http://code.google.com/p/papervision3d/source/browse/#svn/trunk/branches/cs4/src

[75] http://away3d.com/away3d-lite-v1-0-fastest-and-smallest-3d-engine-in-flash

tungsfähigkeit des Away3D-Frameworks herausgestellt. Auf der Projektseite finden sich einige interessante Referenzen, die diesen Eindruck bestätigen[76]. Als prädestinierter Einsatzzweck von Away3D würden sich größere Projekte wie Spiele, Produktpräsentationen oder umfangreiche Microsites anbieten. Ist die Dateigröße und eine langsamere Internetverbindungen nicht wichtig, kämen sicher auch kleinere Projekte in Frage.

Der Flash Player 10 bietet schon von Grund auf eine enorme Leistungsfähigkeit bei der 3D-Transformation. Seine Stärken liegen bei der Animation kleiner und kleinster Partikel, in der Qualität der Darstellung sowie in einer niedrigen resultierenden Dateigröße, was gerade langsame Internetverbindungen zu Gute kommen kann. In Zeiten des mobilen Internets ist das ein Faktor der nicht unterschätzt werden sollte. Denn auch wenn viele Mobiltelefone bisher nur FlashLite unterstützen, gibt es dennoch viele mobile Notebook-Nutzer, die mit über 90 prozentiger Wahrscheinlichkeit das Flash Player Plugin ab Version 9 installiert haben. In diesem Zusammenhang wären Werbebanner mit 3D-Effekten ein geeignetes Einsatzgebiet für den Flash Player 10.

Alternativa3D und Papervision3D fallen im Vergleich dazu etwas ab. Ersteres beeindruckt zwar mit einer sehr guten und annähernd fehlerfreien Darstellung, ist aber aufgrund fehlender Filter-Unterstützung nur begrenzt nutzbar. Und besonders bei Partikel-Animationen leidet Alternativa3D unter einem enormen Leistungseinbruch. Was noch hinzukommt, ist der

[76] http://away3d.com/away3d-fwa-winners-and-more

lizenzrechtliche Gedanke, auch wenn er in dieser Untersuchung keine Rolle spielt. Dennoch sollte man bei der Verwendung des Alternativa3D-Frameworks beachten, dass bei einer kommerziellen Nutzung Lizenzgebühren anfallen.

Papervision3D genießt zwar dem Anschein nach einen guten Ruf sowie eine starke Verbreitung unter Flash-Entwicklern, aber dies täuscht nicht darüber hinweg, dass sich das Framework noch in einem Beta-Stadium befindet. Gerade anhand der Applikation Cube3D werden die Schwächen deutlich, die sich in einer extrem stark schwankenden Bildrate manifestieren. Wie Away3D bieten sich auch PaperVision3D und Alternativa3D für größere Projekte an, aber immer unter Berücksichtigung der oben genannten Schwächen.

Als finales Fazit ist festzustellen, dass die bestehenden Frameworks keineswegs durch die neuen Fähigkeiten des Flash Player 10 ersetzt werden können. Die Entscheidung über den Einsatz der beispielhaft untersuchten 3D-Frameworks oder des Flash Player 10 muss dabei immer mit dem letztendlichen Einsatzzweck einher gehen. Dass dabei eine Symbiose aus den Frameworks mit den nativen Fähigkeiten des Flash Player 10 mehr Leistung verspricht, kann als sehr realistisch erachtet werden. Auch was zukünftige Versionen des Flash Players betrifft, so ist von Adobe noch Einiges zu erwarten. Besonders im Zusammenhang mit GPU-, 64bit- und MultiCore-Unterstützung wäre ein enormer Zuwachs an Leistung und vielfältigerer Möglichkeiten denkbar. Dieser Ausblick in die Zukunft ist aber immer mit Vorsicht zu genießen.

9. Schlussbemerkungen

Abschließend sollte erwähnt werden, dass diese Arbeit nicht zum Ziel hatte, hoch optimierte Programme zu entwickeln und zu testen. Es ging grundsätzlich um eine unter den Implementierungen vergleichbare Untersuchung. Dennoch kann diese Arbeit als Inspiration und Ausgangspunkt für weitere Untersuchung dienen, da mit ihr nicht alle Aspekte betrachtet werden konnten. Zum Beispiel könnte ein nächster Schritt darin bestehen, die Flash Player 10 Versionen der 3D-Frameworks miteinander zu vergleichen – vielleicht auch mit ihren Pendants für den Flash Player 9.

Die 3D-Transformationen sind nicht die einzigen Neuerungen des Flash Player 10, eine der Interessantesten dürfte mit Sicherheit der Shader sein. In Form der Klassen *Shader*, *ShaderFilter* und *ShaderJob* stehen einige Mittel zur Verfügung, die die Leistung vieler Applikationen deutlich erhöhen können. Die sehr hardwarenah ausführbaren Shader-Programme, die mit dem PixelBender ToolKit entwickelt werden, sind hauptsächlich für die Bildverarbeitung gedacht. Mittlerweile gibt es aber erste Anwednungsbeispiele, die diese Möglichkeiten auf andere Berechnungen ausweiten[77].

[77] http://www.flashmagazine.com/tutorials/detail/using_pixel_bender_to_calculate_information/

Quellenverzeichnis

a. Printbasierte Quellen

- CENGAGE LEARNING (Hrsg.), 2001: Kevin Hawkins, Dave Astle, OpenGL game programming, S. 63ff.
- CENGAGE LEARNING (Hrsg.), 2004: Kenneth C. Finney, 3D game programming all in one, S. 89ff.
- FRANZIS VERLAG GMBH (Hrsg.), 2006: Caroline & Matthias Kannengiesser, Flash 8 - Studienausgabe (2. überarbeitete und stark erweiterte Auflage), S. 23ff.
- HEISE ZEITSCHRIFTEN VERLAG GMBH & CO. KG (Hrsg.), 2007: Manfred Bertuch, Klötzchenwelten - Fotorealistische Computergrafik mit Voxel-Raycasting, c't 2009, Heft 4, S. 182ff.
- HOCHSCHULE MITTWEIDA (FH) UNIVERSITY OF APPLIED SCIENCES (Hrsg.), März 2009: Nick Lehmann, Entwurf, Umsetzung und Optimierung einer webbasierenden und interaktiven First-Person-Engine unter Benutzung von Papervision3D, S. 24ff.
- HOCHSCHULE MITTWEIDA (FH) UNIVERSITY OF APPLIED SCIENCES (Hrsg.), August 2008: Clemens Petzold, SVG im Internet – Gegenwärtige und zukünftige Entwicklung., S. 15 u. 18
- O'REILLY MEDIA, INC. (Hrsg.), August 2007: Colin Moock, Essential ActionScript 3.0, S. 457ff.

- SOFTWARE & SUPPORT VERLAG GMBH (Hrsg.), 2007: Frank Puscher, Flash-Video - Porträt einer Karriere, CREATE OR DIE > Video Revolution online, S. 65ff.

b. Webbasierte Quellen

- ADOBE SYSTEMS INC., THE MOZILLA FOUNDATION, OPERA SOFTWARE ASA, AND OTHERS (Hrsg.)23. Oktober 2007: Adobe Systems Inc., The Mozilla Foundation, Opera Software ASA, and others, Proposed ECMAScript 4th Edition – Language Overview, http://www.ecmascript.org/es4/spec/overview.pdf, Gesichtet: 26. Juli 2009 14:18 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)5. Dezember 2005: Holly Campbell, Adobe Completes Acquisition of Macromedia, http://www.adobe.com/aboutadobe/pressroom/pressreleases/200512/120505AdobeAcquiresMacromedia.html, Gesichtet: 25. Juli 2009 12:56 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)August 2009: Adobe Labs, Pixel Bender, http://labs.adobe.com/technologies/pixelbender/, Gesichtet: 4. August 2009 12:48 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)15. März 2008: Adobe Labs, Flash Player 10, http://labs.adobe.com/technologies/flashplayer10/, Gesichtet: 21. September 2008 21:48 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.): Adobe Systems Incorporated, Adobe SVG Viewer download area,

http://www.adobe.com/svg/viewer/install/mainframed.html, Gesichtet: 3. Oktober 2009 18:22 Uhr

- ADOBE SYSTEMS INCORPORATED (Hrsg.): Adobe Systems Incorporated, Adobe to Discontinue Adobe SVG Viewer, http://www.adobe.com/svg/eol.html, Gesichtet: 3. Oktober 2009 18:23 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)15. Mai 2008: Adobe Labs, Flash Player 10 Feature Demos and Videos, http://labs.adobe.com/technologies/flashplayer10/demos/, Gesichtet: 17. Oktober 2008 00:43 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)Juli 2009: Adobe Systems Incorporated, Kuler, http://kuler.adobe.com/, Gesichtet: 2. Juli 2009 0:27 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)Juni 2009: Adobe Systems Incorporated, Flash CS4 Professional ActionScript 3.0 Language Reference, http://help.adobe.com/en_US/AS3LCR/Flash_10.0/, Gesichtet: 25. Juli 2009 12:24 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)November 2008: Adobe Systems Incorporated, Flash CS4 Professional ActionScript 2.0, http://help.adobe.com/en_US/AS2LCR/Flash_10.0/, Gesichtet: 25. Juli 2009 12:28 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)11. August 2008: Justin Everett-Church, Introducing Adobe Flash Player 10,

http://www.adobe.com/devnet/logged_in/jchurch_flashplayer10.html, Gesichtet: 17. Oktober 2008 00:54 Uhr

- ADOBE SYSTEMS INCORPORATED (Hrsg.)18. August 2008: Lee Brimelow, Six reasons to use ActionScript 3.0, http://www.adobe.com/devnet/actionscript/articles/six_reasons_as3.html, Gesichtet: 30. Juni 2009 22:49 Uhr
- ADOBE SYSTEMS INCORPORATED (Hrsg.)März 2009: Adobe Systems Incorporated, Adobe Flash Player Version Penetration, http://www.adobe.com/products/player_census/flashplayer/version_penetration.html, Gesichtet: 9. Juni 2009 22:55 Uhr
- ALTERNATIVA LLC. (Hrsg.)4. März 2008: Alternativa LLC., Alternativa Platform, http://alternativaplatform.com/en/, Gesichtet: 17. Oktober 2008 00:47 Uhr
- ALTERNATIVA LLC. (Hrsg.): Alternativa LLC., Alternativa3D — browser 3D-engine based on Adobe Flash, http://alternativaplatform.com/en/alternativa3d/, Gesichtet: 1. Oktober 2009 14:52 Uhr
- ALTERNATIVA LLC. (Hrsg.)30. Dezember 2007: Anton Volkov, "Bump" light from spotlight source, http://blog.alternativaplatform.com/en/2007/12/30/spot-normal-mapping/, Gesichtet: 11. August 2009 16:44 Uhr

- CHAMBERS, DURA, CANTRELL (Hrsg.)Juli 2009: Mike Chambers, Daniel Dura, Christian Cantrell, as3corelib - ActionScript 3.0 library for several basic utilities., http://code.google.com/p/as3corelib/, Gesichtet: 26. Juli 2009 13:42 Uhr
- CHANG (Hrsg.)22. November 2008: David Chang, AS3 Zip Library Release, http://nochump.com/blog/?p=15, Gesichtet: 26. Juli 2009 13:29 Uhr
- DOYLE (Hrsg.)17. März 2009: Jack Doyle, Speed Test - Tweening Engine Comparison Tool, http://blog.greensock.com/tweening-speed-test/, Gesichtet: 26. Juli 2009 13:12 Uhr
- ECMA INTERNATIONAL (Hrsg.)Dezember 2005: John Schneider, Rok Yu, Jeff Dyer, u.a., Standard ECMA-357 ECMAScript for XML (E4X) Specification (2nd Edition), http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-357.pdf, Gesichtet: 30. Juni 2009 22:49 Uhr
- ENSLEY, KASKOSZ (Hrsg.)Oktober 2008: Doug Ensley, Barbara Kaskosz, Interactive Bitmap Cube in Flash Player 10, http://www.flashandmath.com/advanced/p10cube/, Gesichtet: 11. August 2009 16:41 Uhr
- GOLEM.DE (Hrsg.)15. März 2008: Ingo Pakalski, Flash Player 10 mit interaktiven 3D-Animationen - Neuer Flash Player unterstützt Hardware-Beschleunigung,

http://www.golem.de/showhigh2.php?file=/0805/59705.html, Gesichtet: 21. September 2008 21:37 Uhr

- GOLEM.DE (Hrsg.)25. Februar 2008: Jens Ihlenfeld, Adobe veröffentlicht AIR 1.0 und Flex 3, http://www.golem.de/0802/57918.html, Gesichtet: 2. Juli 2009 0:09 Uhr
- GOLEM.DE (Hrsg.)17. August 2009: Peter Steinlechner, Crytek-Chef stellt Sparse-Voxel-Grafik für Crysis 2 vor, http://www.golem.de/showhigh2.php?file=/0908/69105.html, Gesichtet: 3. Oktober 2009 16:56 Uhr
- GOLEM.DE (Hrsg.)18. August 2008: Alexander Merz, JavaScript-Entwicklung wieder vereint, http://www.golem.de/0808/61775.html, Gesichtet: 26. Juli 2009 14:22 Uhr
- GOOGLE CODE (Hrsg.)26. September 2008: Carlos Ulloa, John Grden, Ralph Hauwert, Tim Knip, Andy Zupko, papervision3d - Open Source realtime 3D engine for Flash, http://code.google.com/p/papervision3d/source/browse/#svn/trunk/branches/cs4/src, Gesichtet: 1. Oktober 2009 14:55 Uhr
- KHRONOS GROUP (Hrsg.)4. August 2009: Jonathan Hirshon, Khronos Details WebGL Initiative to Bring Hardware-Accelerated 3D Graphics to the Internet, http://www.khronos.org/news/press/releases/khronos-webgl-initiative-hardware-accelerated-3d-

graphics-internet/, Gesichtet: 5. Oktober 2009 20:26 Uhr

- MACROMEDIA, INC (Hrsg.)2001: Jonathan Gay, The History of Flash, http://www.adobe.com/macromedia/events/john_gay/, Gesichtet: 5. Juli 2009 21:53 Uhr
- MICROSOFT CORPORATION (Hrsg.)November 2007: Microsoft Corporation, .NET Framework-Klassenbibliothek - DateTime.Ticks-Eigenschaft, http://msdn.microsoft.com/de-de/library/system.datetime.ticks.aspx, Gesichtet: 6. September 2009 17:45 Uhr
- MULTIDMEDIA LIMITED (Hrsg.)Juli 2009: Multidmedia Limited, Zinc™ 3.0 - Rapid Application Development for Adobe® Flash, http://www.multidmedia.com/software/zinc/, Gesichtet: 2. Juli 2009 0:04 Uhr
- NORTHWAY (Hrsg.)22. November 2008: Colin Northway, Box2DFlashAS3 2.0.1, http://box2dflash.sourceforge.net/, Gesichtet: 26. Juli 2009 13:32 Uhr
- PFEIFFER (Hrsg.)Juli 2009: Thomas Pfeiffer, Sandy 3D engine (AS3 & AS2) for Adobe Flash/about, http://www.flashsandy.org/, Gesichtet: 22. Juli 2009 23:42 Uhr
- STATOWL.COM (Hrsg.)Juni 2009: StatOwl.com, Rich Internet Application Market Share,

http://www.statowl.com/custom_ria_market_penetration.php, Gesichtet: 5. Juli 2009 22:27 Uhr

- TOMORROW FOCUS AG INVESTOR & PUBLIC RELATIONS (Hrsg.): TOMORROW FOCUS AG Investor & Public Relations, TECHNISCHE SPEZIFIKATIONEN, http://pickup.tomorrow-ag.de/_adtech/techspecs/2006/, Gesichtet: 8. Juni 2009 23:22 Uhr
- ULLOA (Hrsg.)29. Oktober 2006: Carlos Ulloa, John Grden, Ralph Hauwert, Tim Knip, Andy Zupko, Papervision3D.org, http://www.papervision3d.org/, Gesichtet: 17. Oktober 2008 00:18 Uhr
- URO (Hrsg.)16. Mai 2008: Tinic Uro, What does GPU acceleration mean?, http://www.kaourantin.net/2008/05/what-does-gpu-acceleration-mean.html, Gesichtet: 6. September 2009 16:34 Uhr
- WORLD WIDE WEB CONSORTIUM (W3C) (Hrsg.)30. April 2009: World Wide Web Consortium (W3C), Scalable Vector Graphics (SVG) 1.1 Specification, http://www.w3.org/TR/SVG11/, Gesichtet: 5. Oktober 2009 23:06 Uhr
- ZADOROZHNY (Hrsg.)2. Juni 2007: Alexander Zadorozhny, Rob Bateman, Fabrice Closier, Peter Kapelyan, Greg Caldwell, Jalava, Andreas Engstrom, Katopz, Away3D Flash Engine, http://away3d.com/, Gesichtet: 17. Oktober 2008 00:32 Uhr

- ZADOROZHNY (Hrsg.)11. September 2009: Rob Bateman, Away3D Lite v1.0: fastest and smallest 3d engine in Flash., http://away3d.com/away3d-lite-v1-0-fastest-and-smallest-3d-engine-in-flash, Gesichtet: 1. Oktober 2009 14:27 Uhr
- DEHAAN, MAYHEW (FLASHTHUSIAST) (Hrsg.)19. Oktober 2008: Jen deHaan, Go download Flash Player 10 update for Flash CS4, http://flashthusiast.com/2008/10/15/go-download-flash-player-10-update-for-flash-cs4/, Gesichtet: 11. August 2009 16:44 Uhr
- FLASHMAGAZINE.COM (Hrsg.)13. August 2009: Elad Elrom, Using Pixel Bender to calculate information, http://www.flashmagazine.com/tutorials/detail/using_pixel_bender_to_calculate_information/, Gesichtet: 1. Oktober 2009 15:51 Uhr
- HEISE ONLINE (Hrsg.)15. März 2008: Christian Hirsch (c't Magazin), Erste Beta des Flash Player 10 liegt vor, http://www.heise.de/newsticker/Erste-Beta-des-Flash-Player-10-liegt-vor--/meldung/107905, Gesichtet: 21. September 2008 21:40 Uhr

Anhang

Der gesamte Anhang ist auf der beiliegenden CD und im Internet unter folgender Adresse zu finden:

http://www.stefanvonderkrone.de/bachelorarbeit/

Die Daten liegen unter anderem im CSV-, Office-2007- und ODF-Format vor. Des Weiteren sind dort alle wichtigen Entwicklungsdaten sowie die Projektdateien für die Entwicklungsumgebungen FDT[78], FlashDevelop[79] sowie Visual Studio 2008[80] zu finden. Letztere betrifft lediglich den *ProcessLogger*.

Online ist es möglich durch die Dateien zu navigieren, sie können aber auch als ISO-Abbild zum eigenständigen Brennen auf einen CD-Rohling sowie als ZIP-Archiv heruntergeladen werden. Für die Arbeit mit den Flash-Applikationen wir ein FlexSDK[81] mindestens in der Version 3.3 benötigt.

Fragen und Anregungen zum Anhang kann man jederzeit per Mail an skrone@htwm.de richten.

---

[78] http://www.fdt.powerflasher.com/

[79] http://www.flashdevelop.org/community/viewforum.php?f=11

[80] http://www.microsoft.com/germany/Express/product/visualcsharpexpress.aspx

[81] http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+3

Erklärung zur selbstständigen Anfertigung der Arbeit

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Leipzig, der 6. Oktober 2009

_____

Stefan von der Krone