UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Ein Szenengraph auf Basis von WebGL zur Anwendung in sozialen Netzwerken

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von
Alexander Ehrhardt

Erstgutachter: Prof. Dr.-Ing. Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)

Zweitgutachter:

Koblenz, im Oktober 2011

Erklärung

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

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Diplomarbeit

Alexander Ehrhardt

(Matr.-Nr. 204 210 490)

Thema: **Ein Szenengraph auf Basis von WebGL zur Anwendung in sozialen Netzwerken**

Ziel der Diplomarbeit ist es, einen bereits teilweise entwickelten Szenengraph an die Anforderungen einer Anwendung für soziale Netzwerke anzupassen. Die Anwendung soll es dem Benutzer ermöglichen in die Rolle eines von ihm angepassten Avatars zu schlüpfen. Mit dem Avatar ist es dem Benutzer möglich durch eine dreidimensionale Welt zu navigieren und mit anderen Avataren Kontakt aufzunehmen (oder Spiele zu spielen).

Bei der Entwicklung liegt ein Schwerpunkt auf neuer und zum Teil noch experimenteller Web-Technologie. Diese wird auf ihre Brauchbarkeit hin untersucht. Abschließend soll erörtert werden, warum die schon seit langem vorhandene 3D-Web-Technologie bisher keinen nennenswerten Durchbruch erlangen konnte.

Schwerpunkte dieser Arbeit sind:

1. Festlegung des Applikationsszenarios der Anwendung
2. Erhebung der daraus resultierenden Anforderungen an den Szenengraph und die Anwendung
3. Recherche nach geeigneten Technologien
4. Anpassen des Szenengraphs
5. Implementieren und Testen der Anwendung
6. Evaluierung der Anwendung
7. Vergleich mit bestehender 3D-Web-Technologie und Erfolgsaussichten von WebGL

Koblenz, den 21. März 2011

S. Müller

Prof. Dr. Stefan Müller

1 Einleitung

Die Möglichkeiten einen Szenengraphen sinnvoll in ein soziales Netzwerk zu integrieren sind vielfältig. Das soziale Netzwerk mit einem Teil seiner Mittel in eine virtuelle Welt abzubilden, ist der Ansatz dieser Arbeit. Dabei liegt das Ziel zum einen in der Erörterung der technischen Machbarkeit und zum anderen in der Erschaffung eines Mehrwerts für den Benutzer. Um diesen Mehrwert zu erzeugen, wird versucht, die Eigenschaften der dritten Dimension auszunutzen. So lassen sich in einer virtuellen Welt im Vergleich zu einem sozialen Netzwerk gänzlich neue Sachverhalte, wie die Proxemik ausdrücken, oder aber auch existierende Bestandteile des Netzwerks ergänzen. Inhalte, die zuvor nur beschrieben oder auf Fotos dargestellt wurden, können nun im dreidimensionalen Raum betrachtet werden. Die Profilseite spiegelt sich in dieser Arbeit beispielsweise als austauschbarer Avatar, mit eigens eingerichtetem Raum wider.
Verschiedene, aber hierfür zwingend erforderliche 3D-Schnittstellen für Browser gibt es zwar schon seit längerem, einen nennenswerten Durchbruch in Form einer populären Anwendung, gab es bisher noch nicht. Dabei erhalten speziell virtuelle Welten, als plattformabhängiges, natives, eigenständiges Produkt regen Zulauf. So gibt es, in der wohl bekanntesten virtuellen Welt, Second Life selbst nach Jahren noch eine große, aktive Nutzerzahl[1].
Die erst im März 2011 von der einflussreichen Khronos Group[2] verabschiedete WebGL-Spezifikation, könnte den nötigen Umschwung bringen und den Zugang zum Web-3D für die breite Masse attraktiver machen. WebGL ist auch die 3D-Schnittstelle, die diese Arbeit verwendet. Diese Wahl hat sowohl den Szenengraph, als auch die restliche Technologie, die für eine virtuelle Welt benötigt wird, beeinflusst. Aus diesem Grund ist die Präsentation und der Stand der Wissenschaft der verschiedenen 3D-Schnittstellen, der Ausgangspunkt dieser Arbeit. Nach der Abhandlung der 3D-Schnittstellen, folgen zwei Abschnitte, die Anforderungen sowie deren Grundlagen erörtern. Welche weiteren Technologien, neben WebGL genutzt werden, ist Teil des Abschnitts 5. Aufgrund dieser Basis, werden der Entwurf und die Fähigkeiten des Szenengraphs beschrieben. Die Beschreibung der resultierenden Anwendung ist in Abschnitt 7 erklärt. Die beiden letzten Abschnitte evaluieren die Anwendung und geben einen Ausblick und ein Fazit.

[1] Ungefähr eine Millionen aktive Accounts in den letzten 30 Tagen[Pad11] (Stand 22.September 2011).

[2] Die Khronos Group wurde im Januar 2000 u.a. von 3Dlabs, ATI, Discreet, Evans & Sutherland, Intel, NVIDIA, SGI und Sun Microsystems gegründet [UNB11a]

2 Stand der Wissenschaft

2.1 3D-Schnittstellen der Browser

Zur Zeit ist es möglich durch mehrere verschiedene Technologien 3D-Inhalte in einer Webseite darzustellen. Bisher war es jedoch immer nötig, bei GPU-unterstütztem Rendering, entsprechende Plugins bzw. Erweiterungen je nach Browser und Betriebssystem zu installieren. Die Abhängigkeit zwischen Browser, Betriebssystem und Plugin ist zugleich durch die einhergehenden Inkompatibiltätsprobleme, der Nachteil dieses Ansatzes. Zudem wird durch die Installation eine weitere Angriffsfläche für Schadsoftware geschaffen, die Sicherheitslücken des betreffenden Plugins ausnutzen kann [BEJZ09]. In den folgenden Teilabschnitten wird auf die wichtigsten Technologien eingegangen. Zu WebGL werden außer einem kurzen Überblick, auch die wesentlichen Unterschiede zu OpenGL und OpenGL ES beschrieben, da diese Einfluss auf den Szenengraphen haben. Außerdem wird eine eine virtuelle Welt vorgestellt, die WebGL schon als Schnittstelle für die grafische Ausgabe nutzt. Der letzte Teil dieses Abschnitts versucht Erfolgsaussichten der verschiedenen 3D-Schnittstellen einzuschätzen.

2.1.1 VRML

Mit VRML ist 1994 die erste Technologie erschienen, die über ein Plugin 3D-Inhalte im Browser darstellen konnte. Der erhoffte Erfolg blieb jedoch aus. Über die Gründe für den nur mäßigen Erfolg, wurde viel diskutiert. Der Umstand, dass die Sprache nicht erweitert werden konnte und Modellierungstools nur einen unzureichenden VRML-Export boten, hat seinen Teil dazu beigetragen. In [BRDA11] wird als Hauptgrund die unzureichende Unterstützung der VRML-Grafik, durch die zu dieser Zeit vorhandenen Prozessoren und Netzwerke, genannt.

2.1.2 Adobe Flash Player

Das meistgenutzte Browser-Plugin ist der Adobe Flash Player. Nach eigenen Angaben ist dieser auf 99 Prozent der Desktop-Computer installiert [UNB11i]. Zudem ist davon auszugehen, dass Ende 2011 auch 36 Prozent aller Smartphones den Flash Player unterstützen werden [UNB11h]. Diese Anteile scheinen hoch, allerdings müssen sie ihn in Bezug auf die 3D-Fähigkeit des Players relativiert werden. So wird erst ab der, noch nicht erschienenen Version 11 des Players, nativ die Grafikkarte unterstützt. Vorherige Versionen ermöglichen die Darstellung von 3D Grafiken zwar ebenso, jedoch nur über Bibliotheken, die diese emulieren [Pau10].
Wird die neue Version des Flash Players jedoch ebenso schnell wie die vorherigen Versionen aufgenommen, kann davon ausgegangen werden, dass auch Version 11, Anfang 2012 bei einem Großteil der Internetnutzer installiert sein wird [LD11]. Die darin enthaltene GPU beschleunigten 3D API (Stage3D API) bietet eine hohe Kompatibilität, indem sie auf den verschiedenen Plattformen auf unterschiedliche 3D-Schnittstellen[3] zugreift. Dieser Ansatz beinhaltet ebenso einen Software-Renderer, der auf Geräten ohne Grafikchip oder mit nicht unterstützen Grafikchip als „Fallback“ dient [UNB11r].

2.1.3 Unity 3D

Unity 3D ist eine komplette Spiel-Engine, samt einer Entwicklungsumgebung und verschiedenen Tools. Es grenzt sich alleine dadurch schon von einer „Low-Level-API“ wie OpenGL ab. Im Vergleich zu einem von Grund auf neu geschriebenen Spiel mit OpenGL, braucht die Entwicklung eines vergleichbaren Spiels unter Unity 3D nur einen Bruchteil dieser Zeit. Die Entwicklung ist zwar nur

[3] OpenGL 1.3 auf Linux und MaxOS, OpenGL ES 2.0 auf mobilen Plattformen, DirectX 9 auf Windows [UNB11r]

auf Mac OSX und Microsoft Windows möglich, jedoch können die entwickelten Spiele für das iPad, iPhone, XBox, Playstation, die Android-Platform und einen Webplayer portiert werden. Dabei ist der Webplayer für den Internet Explorer, Safari und Mozilla-basierte Browsern lauffähig [UNB11n] und kann sich bereits auf über 60 Millionen Installationen stützen [UNB11o]. Unter Linux gibt es allerdings keinen lauffähigen Webplayer. Aber auch diese Lücke soll durch eine Umsetzung der Engine mit Googles Native Client[4] geschlossen werden. Interessanter ist jedoch die Ankündigung einer Umsetzung der Unity-Engine für den Flash Player, auf Basis, der bereits erwähnten, Stage3D API [UNB11s]. Diese Umsetzung würde die hohe Verfügbarkeit des Flash Players mit dem Entwicklungskomfort von Unity 3D verbinden.

2.1.4 Java Applets

Java bietet mit seinen Applets ebenso die Möglichkeit an, 3D-Grafiken nativ auf der Grafikkarte zu berechnen. Dazu nutzt das Applet das JNI (Java Nativ Interface), welches es ermöglicht betriebssystemspezifische Funktionen aufzurufen. Somit ist man in der Lage Funktionen aufzurufen, die die Grafikkarte direkt ansprechen. Die Bibliotheken JOGL (Java OpenGL) und LWJGL (Lightweight Java Game Library) bieten durch diese Technik eine Schnittstelle zu OpenGL an. Auf LWJGL baut mit der jMonkeyEngine sogar eine ganze Spiel-Engine auf, die auch Java Applets unterstützt und somit im Browser lauffähig ist. Die Verbreitung und Akzeptanz von Java Anwendungen und der nötigen Runtime Environment (JRE) ist im Vergleich zu der, des Flash Players, wesentlich geringer. Im Bezug auf 3D-Grafik spiegelt sich dieser Umstand auch darin wieder, dass Sun[5] den Support für die auf JOGL aufsetzende Szenengraphen-Bibliothek Java3D stoppte. Zudem ist in dem zu Flash konkurrierenden Framework JavaFX, welches ebenfalls von Sun stammt, keine offizielle 3D-Unterstützung enthalten [BEJZ09].

2.1.5 WebGL

Die derzeit aktuellste Schnittstelle für das Rendern von 3D Web-Inhalten, stellt WebGL (Web Graphics Library) dar. Bisher ist WebGL nur in den neueren Versionen von Chrome, Firefox, Safari und Opera verfügbar. Es grenzt sich aber von den bisher beschriebenen Möglichkeiten dadurch ab, dass es nicht als „Plugin-Fremdkörper“ in die HTML-Seite eingebettet wird. Stattdessen wird WebGL durch das HTML5 Canvas-Element, wie jedes andere HTML-Element nahtlos in den DOM-Baum integriert. WebGL nutzt dazu einen Context, der von dem Canvas erzeugt wird und rendert mittels JavaScript-Funktionen in dieses Canvas-Element. Somit können mit JavaScript sowohl die Seitenelemente, als auch die Darstellung der WebGL-Inhalte verwaltet werden. Eine extra definierte Schnittstelle[6], wie sie häufig benötigt, wird, wenn mit einem Plugin gerendert wird, fällt daher weg.
Hierbei dient das Canvas nicht nur als Leinwand, um die mit WebGL erzeugte Grafik darzustellen, das Canvas kann zudem genutzt werden, um Grafiken anderer JavaScript-APIs darzustellen. Aktuell kann auch mit einem Canvas-2D-Rendering-Context in das Canvas-Element gerendert werden. Dadurch wird die Möglichkeit geboten Ergebnisse eines 2D Render-Passes durch den 2D-Context, beispielsweise als Textur in WebGL weiterzuverwenden.
Die WebGL API basiert dabei auf der OpenGL ES (Embedded Systems) 2.0 API und stimmt mit ihr,

[4] Der Native Client stellt in Chrome eine Sandbox bereit, in der nativer Code ausgeführt werden kann. Dadurch kann Unity 3D ohne Plugin in Chrome auch auf Linux nativ ausgeführt werden [Sey11].

[5] Sun wurde 2010 von Oracle übernommen.

[6] Ohne eine solche Schnittstelle wäre es einem 3D-Plugin nicht möglich auf Seiteninhalte zuzugreifen, oder sie zu verändern.

bis auf wenige Ausnahmen, überein [HL11a]. Durch diesen Ansatz ist die Portabilität von WebGL Anwendungen auch auf mobile Geräte gewährleistet [UNB11u]. Zwischen dem „normalen“ OpenGL und OpenGL ES gibt es hingegen größere Unterschiede. Im folgenden Unterabschnitt wird daher kurz auf die Unterschiede zwischen OpenGL ES 2.0 und OpenGL, sowie auf Unterschiede zwischen WebGL und OpenGL ES 2.0 eingegangen. Außerdem wird in dem letzten Teil dieses WebGL-Abschnitts eine virtuelle Welt vorgestellt, die bereits WebGL nutzt.

2.1.5.1 Unterschiede zwischen WebGL, OpenGL ES 2.0 und OpenGl

Der Ursprung des Unterschieds zwischen OpenGL ES und OpenGL liegt hauptsächlich in den leistungsspezifischen Begrenzungen der zugrunde liegenden Hardware. Während mobile Geräte eine geringere Effizienz bzw. Performance bezüglich CPU, Speicher, Busbandbreite und Stromverbrauch bieten, fallen diese Grenzen bei Desktop-Systemen weitaus weniger ins Gewicht. Dies hat zur Folge, dass das auf mobile Endgeräte zugeschnittene OpenGL ES lediglich eine Untermenge der Befehle, genauer gesagt der effizientesten Befehle, von OpenGL ist [Grü11]. OpenGL ist beispielsweise in der Lage einen Punkt mittels Display-Listen, Vertex-Arrays oder im Immediate-Mode (d.h. zwischen `glBegin` und `glEnd`) zu zeichnen. Jeder dieser Wege hat zwar seine Vorteile, muss aber auch effizient auf der Hardware umgesetzt werden. So liegt es auf der Hand, dass es mit Schwierigkeiten verbunden ist die komplette, teilweise überladene OpenGL-Spezifikation auf der limitierten Hardware von mobilen Geräten bereitzustellen [WLH07].

2.1.5.2 KataSpace

Die einzige zum Zeitpunkt dieser Arbeit bekannte virtuelle Welt, die auf HTML5 und WebGL setzt, ist KataSpace [CP11] (siehe Abb1). In der virtuelle Welt von KataSpace sind die Benutzer gemeinsam in einer kleinen Welt und können miteinander chatten und interagieren. KataSpace baut dabei auf der Sirikata Plattform auf, welche als Grundlage für das Entwickeln virtueller Welten angesehen werden kann. Die Plattform bietet dafür eine Reihe von Bibliotheken und Protokollen an [UNB11q].

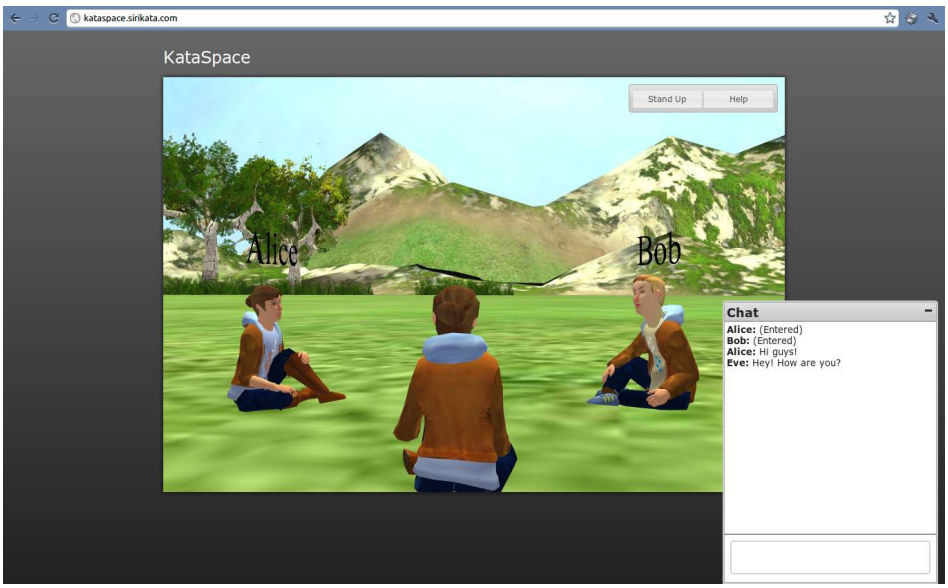


Abbildung 1: Screenshot der virtuellen Welt von KataSpace

2.1.6 Erfolgsaussichten

Welche der Technologien sich in der Zukunft durchsetzen wird bleibt abzuwarten. Der Fokus wird dabei auf Flash oder WebGL liegen, ein Durchbruch einer Java basierten Lösung wäre schon längst geschehen. Unity baut auf einer 3D-Schnittstelle auf und steht deshalb, wenn überhaupt, nur mit seinem Webplayer in direkter Konkurrenz. Daher ist eher davon auszugehen, dass Unity durch einen Erfolg der Unity-Flash-Umsetzung und somit direkt von einem Durchsetzen des Flash Players profitieren wird.
Schaut man sich den bisherigen Erfolgskurs von Flash an, so ist die große Popularität unter anderem Plattformen wie MySpace oder Youtube zu verdanken. Eine ebenso erfolgreiche Anwendung oder Plattform auf Basis einer der 3D-Technologien, könnte das gleiche für eben diese bedeuten [LD11].

3 Grundlage der Anforderungen

In diesem Abschnitt werden auf Grundlage verschiedener Quellen Kernpunkte erschlossen, die in die Umsetzung der virtuellen Welt einfließen sollen. Diese Punkte sind daher der Ausgangspunkt der Anforderungen aus Abschnitt 4. Dazu wird in einem ersten Teilabschnitt, der Begriff und die Wichtigkeit des Avatars in einer virtuellen Welt geklärt. In einem zweiten Teilabschnitt werden vier Patterns vorgestellt, die als Hilfe und Basis für Entwürfe gemeinsam genutzter 3D-Welten dienen sollen.

3.1 Der Avatar in einer virtuellen Welt

Ein Avatar ist kurz gefasst die Schnittstelle zwischen einem Benutzer und der virtueller Welt. Eine gute Definition für einen Avatar findet sich in [BPO08]: „Ein Avatar ist eine Darstellung eines Benutzers. Diese bezieht sich sowohl auf eine 2D-, als auch eine 3D-Darstellung, durch die der Benutzer auf die Umgebung (d.h eine Online-Welt) einwirken kann[..]“. Zusätzlich ist ein Avatar auch ein Mittel, um die Privatsphäre und Anonymität eines Benutzers zu bewahren [VJP07].
Eine der wichtigsten Eigenschaften, die einem Avatar gegeben werden kann, ist dessen Individualisierbarkeit. In Second-Life stellt das Individualisieren des Avatars sogar eine der wichtigsten Aktivitäten überhaupt dar [DWYW09]. Die Individualisierbarkeit erlaubt es, zumindest in Teilaspekten, eine Art Idealpersönlichkeit des Benutzers darzustellen. Die daraus resultierenden vielfältigen Avatare ermöglichen es, mit einer imaginären Identität zu experimentieren. Wird keine Möglichkeit zur Individualisierung geboten, besteht das Risiko, dass der Benutzer sich nicht mit dem Avatar identifizieren kann. Somit sinkt das Gefühl einer sozialen Präsenz, in einer synthetischen Welt [BPO08]. Das mögliche Erscheinungsbild, das durch die Individualisierung abgedeckt wird, kann gleichzeitig eine Gruppenzugehörigkeit signalisieren oder sogar Aussagen über das Verhalten in der virtuellen Welt machen.
In einer Studie von [BPO08] konnten drei Hauptgründe für die Wichtigkeit einzigartiger Avatare gefunden werden:

- Ausdruck der Persönlichkeit und Wiedererkennung
- Erzeugen von „Inhalten“ in einer virtuellen Welt
- Vertrauen und Ansehen

Speziell im Zusammenhang mit sozialen Netzwerken ist ein Avatar in der Lage in gewisser Weise die Identität und Aktivitäten eines Benutzers in die virtuelle Welt zu projizieren.

3.2 Patterns für gemeinsam genutzte 3D-Räume

Neben der Realisierung eines individualisierbaren Avatars orientiert sich die Umsetzung u.a. an vier Patterns[7] aus [BGP09], welche auf eine langfristige Bindung der Benutzer, sowie eine gute Umsetzung der Präsenz [8] in virtuellen Welten abzielen.
Dabei ist anzumerken, dass vor allem die Patterns, die sich zum Ziel eine langfristige Bindung des Benutzers gesetzt haben, schon in der Planungsphase Probleme bereiten. Diese Probleme basieren darauf, dass diese Patterns als Mittel für eine zeitintensive Beschäftigung die Erfüllung von zum

[7] Diese Pattern sollen als Richtlinie für Funktionen und Designentscheidungen während der Entwicklungsphase von virtuellen Welten dienen.

[8] Mit der Präsenz ist das Gefühl gemeint, sich „in“ der virtuellen Welt zu befinden.

Teil komplexen Aufgaben durch einen oder mehrere Benutzer in der virtuellen Welt heranziehen. Da sich jedoch, jede Aufgabe als Programmcode niederschlagen würde und neue Aufgaben somit nur durch ein Update der gesamten App oder durch eine Implementierung einer eigenen Script-Sprache umsetzen ließe, würde es den Umfang dieser Arbeit sprengen. Der Inhalt des fünften Patterns zielt ausschließlich auf Aufgaben ab, die in Benutzergruppen gelöst werden sollen und ist aufgrund der erwähnten Problematik nicht Teil der Arbeit. Deshalb wird im folgenden nur ein Überblick über vier, der insgesamt fünf Pattern gegeben. Die vier Patterns bilden damit die restliche Grundlage für die in Abschnitt 4 beschriebenen Anforderungen.

- Pattern 1: „Wo bin ich?“
  Der Ausgangspunkt dieses Patterns ist ein Benutzer, der auf einen Bildschirm blickt, auf dessen zweidimensionaler Oberfläche die dreidimensionale virtuelle Welt dargestellt wird. Das Ziel muss es jedoch sein, den Benutzer in der virtuellen Welt „versinken“ zu lassen, d.h. er muss das Gefühl bekommen Teil der virtuellen Welt zu sein. Um dieses Gefühl optimal zu vermitteln, sollte die virtuelle Welt einen großen Bezug zur Realen aufweisen. Für die Darstellung der Welt wäre eine hochauflösende detaillierte Grafik, die die reale Welt genau darstellt, eine mögliche Lösung. Diese Lösung würde allerdings zu hohen Ansprüche an die Hardware, als auch an die 3D-Modelle der Welt führen und wäre daher schwer umsetzbar. Des Weiteren hat sich gezeigt, dass eine hoch detaillierte Grafik alleine nicht zwingend zu dem gewünschten Ziel führt. So sind wie in [BGP09] beschrieben, viele der sehr realistisch modellierten Umgebungen aus Second Life kaum besucht. Dies liegt zum Teil an der mangelnden Interaktion, die an diesen Orten geboten wird.
  Sinnvoller ist es eine virtuelle Welt zu erschaffen, die sich zwar einfacher Objekte und Texturen bedient, die sich jedoch durch einen hohen Wiedererkennungswert auszeichnen. Zudem sollten, um den räumlichen Eindruck für den Benutzer zu erhöhen, geometrische Formen, konvergierende Linien, Licht und Schatten eingesetzt werden.
  Um ressourcenschonend eine andere Umgebungen darzustellen, muss nicht unbedingt eine komplett neue Geometrie geladen werden. Es kann auch die vorhandene Geometrie durch den Austausch von Texturen „verändert“ werden. Außerdem sollte darauf geachtet werden, dass die Welt so konzipiert ist, dass der Benutzer möglichst viel interagieren muss, d.h., dass er beispielsweise die Welt erkunden muss, um sich einen Überblick zu verschaffen oder auch gezwungen ist mit anderen Avataren zu sprechen.

- Pattern 2: „Nimmt mich die 3D-Welt wahr?“
  Das Ziel dieses Patterns ist es dem Benutzer den Eindruck zu vermitteln, dass er von der virtuellen Welt wahrgenommen wird. Damit ist gemeint, dass die Umwelt, in der sich sein Avatar befindet, Reaktionen auf sein Verhalten zeigt. Dies sollte unter anderem die gewohnten physikalischen Gesetzte, wie beispielsweise die Schwerkraft beinhalten. Die Umwelt darf sich aber auch durchaus „reaktiver“, als die reale Welt zeigen. So können beispielsweise Informationen über den momentanen Ort in der virtuellen Welt angezeigt werden oder jegliche Interaktionen zusätzlich visualisiert werden.

- Pattern 3: „Was sind unsere Superkräfte?“
  Mit „Superkräften“ sind in diesem Pattern die Fähigkeiten gemeint, die nur in einer virtuellen Welt ermöglicht werden können. Beispielsweise das Teleportieren, Fliegen oder ein Blickwinkel auf die Welt, der nicht der Ego-Perspektive entspricht. Da diese Kräfte einen großen Einfluss auf den Umgang mit der virtuellen Welt haben, müssen sie wohlüberlegt eingesetzt werden. Würde man es beispielsweise ermöglichen, von einem Ort in der Welt alles zu steuern, so dass

sich der Avatar nicht mehr bewegen müsste, würde der Anreiz, die Umgebung zu erkunden und jegliche nonverbale Kommunikation, verloren gehen. Aus diesem Grund sollten Superkräfte nur eingesetzt werden, wenn sie wirklich nötig sind.

- Pattern 4: „Was sollen wir eigentlich tun?“
  Dieses Pattern zielt darauf ab, die Aktivitäten in der virtuellen Welt so zu gestalten, dass Benutzer die Anwendung langfristig, wiederholt und freiwillig nutzen.
  Nachdem die Spannung von dem „Neuen“ abgeflaut ist, muss dem Benutzer Ansprechendes und Interessantes geboten werden. Alleine die soziale Interaktion reicht nicht aus. Selbst Gespräche mit Fremden bieten auf lange Sicht nicht den erforderlichen Anreiz. Damit über eine längere Zeitspanne die Motivation die Anwendung zu benutzen, nicht verloren geht, sollten Aufgaben angeboten werden. Benutzer müssen daran interessiert sein, die Ziele dieser Aufgaben zu erreichen. Dabei macht es Sinn die Aufgaben in Teilaktionen aufzuteilen. Diese sollten schnell bewältigt werden können und weiterhin den Raum lassen, mit anderen Benutzern zu interagieren. Um das Interesse aufrecht zu erhalten, können Aufgaben so gestaltet werden, dass unter den Benutzern eine Art Wettbewerb hervorgerufen wird. Dabei sollte die Aufgaben im Schwierigkeitsgrad den Benutzer nicht entmutigen.

4 Anforderungen an den Szenengraphen

Aus den in Abschnitt 3 beschriebenen Grundlagen, werden in diesem Abschnitt die Anforderungen hergeleitet.
Die Umsetzung dieser Anforderungen, soll dabei ausschließlich die Basis einer virtuelle Welt darstellen. Aus den erwähnten Pattern lassen sich weitaus mehr Anforderungen herleiten, als die nachfolgende Liste. Dies würde jedoch den Umfang der Arbeit übersteigen. Dennoch soll diese Anforderungsliste ausreichend genug sein, um Aussagen über den Mehrwert und die technische Machbarkeit einer virtuellen Welt mittels WebGL zu machen.

1. Wahl zwischen unterschiedlichen Avataren
2. Erstellung eigener Avatare
    - Anmerkung: Hoch detaillierte 3D-Avatare sind dabei nicht zwingend erforderlich, was sich unter anderem auch in der Beliebtheit der Wii-Avatare zum Ausdruck bringt [BPO08]. Die Umsetzung des Avatars schließt natürlich auch dessen Bewegung ein. Die Bewegung durch die virtuelle Welt ist eine der grundlegendsten Aktivitäten überhaupt. Alleine durch die Position des Avatars können schon Sachverhalte wie Freundschaft bzw. Zugehörigkeit vermittelt werden [BPO08].
3. Verbesserung des Raumgefühl (Pattern 1):
    (a) einfache Schattierung der Objekt
    (b) Kacheln als Bodentextur
4. Förderung der Interaktion und der Erkundung des Raums (Pattern 1):
    (a) Kommunizieren über Sprechblasen, die nur lokal über dem Avatar dargestellt werden
5. reaktive Interaktion (Pattern 2):
    (a) Darstellung einer Bewegungstrajektorie, wenn der eigene Avatar bewegt wird
    (b) Anzeige des Raumnamens beim Betreten eines neuen Raums
6. Verbesserung des Überblicks beim Navigieren (Pattern 3):
    (a) Third-Person-Perspektive auf die Welt
7. dynamisches Nachladen der virtuellen Welt vermeiden (Pattern 3):
    (a) Teleportation in andere Räume
8. Anbieten einfacher Aufgaben (Pattern 4):
    (a) Gestalten des eigenen Raums
    (b) Besuchen anderer Räume (Räume der Freunde)

5 Verwendete Technologien

Damit die erhobenen Anforderungen und der Szenengraph umgesetzt werden können, bedarf es außer dem schon präsentierten WebGL, weiterer Technologien. Diese werden vor dem eigentlichen Szenengraph vorgestellt, da sie zum Teil die Grundlage für ihn darstellen.
Im ersten und zweiten Unterabschnitt geht es um das Google Web Toolkit, die Google App Engine und die dazugehörigen Bibliotheken. Beide Frameworks bilden die Grundlage für die Programmierung auf Client- und Serverseite. Im letzten Abschnitten wird Facebook als die Wahl des sozialen Netzwerks vorgestellt und die Umsetzung der Anwendung als Facebook-App erläutert.

5.1 Google Web Toolkit

Das Google Web Toolkit, im folgenden GWT genannt, ist ein Framework, mit dem komplexe Web-Anwendungen auf Basis von HTML und JavaScript entwickelt werden können. Es nutzt dabei AJAX[9], um mit dem Server zu kommunizieren. AJAX bezeichnet ein grobes Konzept, welches mittels JavaScript eine Anfrage an den Server stellt und zu einem späteren Zeitpunkt asynchron eine Antwort über eine Callback-Funktion erhält. Während der „Wartezeit“ auf die Antwort des Servers, können durch das Script auf der Clientseite andere Aufgaben abgearbeitet werden. Dieser Umstand ist deshalb von größerer Bedeutung, da sich erst mit dieser Technik reaktive Applikationen im Web umsetzen lassen. Es wird dadurch auch eine Art Multithreading ermöglicht, weil sich eine Aufgabe auf Client und Server aufteilen lässt. Betrachtet man darüber hinaus den Gesichtspunkt, dass JavaScript lediglich durch einen einzigen Thread ausgeführt wird, fällt das Mutlithreading besonders ins Gewicht. Rechenintensive Aufgaben, sollten daher geschickt auf Client Server aufgeteilt werden. Kommt es dazu, dass JavaScript durch eine Aufgabe ausgelastet ist, so ist die Benutzeroberfläche stark verzögert oder gar nicht mehr ansprechbar. Bevor AJAX in der Webentwicklung Einzug erhielt, war es üblich, dass man eine Veränderung der Webseite nur durch ein komplettes Neuladen bewerkstelligen konnte.
Neben AJAX macht sich das GWT einen Java-zu-JavaScript-Compiler zu nutze. Dieser stellt das Herzstück des GWTs dar und nimmt großen Einfluss auf die Entwicklung mit dem GWT. In dem folgenden Teilabschnitt wird er daher gesondert behandelt. Auf diesem Wissen aufbauend, wird im letzten Abschnitt zum GWT, die konkrete Umsetzung von AJAX detailliert erklärt, da diese Einfluss auf die Übertragung der 3D-Daten nimmt.

5.1.1 Java-zu-JavaScript-Compiler

Hauptmerkmal des GWTs ist der Java-zu-JavaScript-Compiler. Das heißt eine Anwendung wird nicht mehr mit JavaScript, sondern mit Java geschrieben und durch einen Compiler zu JavaScript umgewandelt [10]. Die Schnittstelle, die der Compiler nutzt, um aus Java-Code JavaScript-Code zu erzeugen, nennt sich JavaScript Native Interface (JNSI). Dieses Konzept ähnelt dem, des Java Native Interfaces, das schon in Abschnitt 2.1.4 beschrieben wurde. Ein einfaches Beispiel zeigt Listing 1.

```
1  public static native void alert(String msg) /*-{
2    $wnd.alert(msg);
3  }-*/;
```

Listing 1: Definition einer JNSI Methode [UNB11d].

[9] AJAX ist eine Abkürzung und steht für „Asynchronous JavaScript and XML“.
[10] Genauer gesagt zu einem großen Teil JavaScript und einem kleineren Teil HTML und XML

Wie man sieht entspricht die Methodensignatur bis auf das `native`, der einer gewöhnlichen Javamethode. Der Rumpf der Methode enthält jedoch JavaScript-Code, welcher durch den Compiler benutzt wird, um aus dem Java- JavaScript-Code zu erzeugen. Solche Methoden müssen aber im Normalfall nicht geschrieben werden[11], da vieles in den schon vorhandenen GWT-Bibliotheken abgedeckt ist [UNB11d]. Im Szenengraph wird beispielsweise ein WebGL-Binding (`http://code.google.com/p/gwt-g3d/`) genutzt, um die nativen JavaScript-WebGL-Methoden aufzurufen.

Der Ansatz, Java-Code zu programmieren und Java-Script Code zu erzeugen, birgt viele Vorteile, im Vergleich zur direkten Programmierung mit JavaScript:

1. man hat die Vorteile einer objektorientierten Sprache:
   (a) Generalisierung und Spezialisierung wird ermöglicht
   (b) Design-Patterns können angewendet werden [Dew09]
   (c) weniger duplizierten Code durch Wiederverwendung
   (d) weniger Fehler durch statische Typisierung im Vergleich zur dynamischen Typisierung
   (e) weniger Memory Leaks (dies jedoch nur aufgrund von Java) [Dew09]
2. durch die Verwendung von Eclipse als Entwicklungsumgebung und dem Google-Eclipse-Plugin, ist effizienteres Entwickeln möglich:
   (a) JUnit-Tests können geschrieben und verwendet werden
   (b) Refactoring-Werkzeuge stehen zur Verfügung [Dew09]
   (c) Kompilierfehler werden während dem Schreiben des Codes angezeigt [Dew09]
   (d) Debug-Werkzeuge stehen zur Verfügung
3. der kompilierte JavaScript-Code ist optimiert
4. der Code ist weniger anfällig für Cross-Site-Scripting [GU10]
5. der Code kann bei Bedarf verschleiert[12] werden
6. es können automatisch JavaScript-Code-Permutation für die verschiedenen Browsertypen und Versionen erzeugt werden

Gerade letzter Punkt führt in der normalen Web-Entwicklung mit JavaScript dazu, dass manuell ein und die selbe JavaScript-Funktion je nach Browsertyp und -version mehrmals implementiert werden muss. Ferner müssen die verschiedenen Permutationen dieser Funktionen zusätzlich von jedem Browser geladen werden, unabhängig davon ob sie während der Ausführung gebraucht werden oder nicht. Die Ursache dafür liegt in den Eigenarten der Browser und deren inkonsistente Ausführung mancher JavaScript-Funktionen bzw. durch das Fehlen neuer JavaScript-Funktionen in älteren Browsern. Dieses Verhalten der Browser ist auch das Hauptproblem, welches durch das GWT versucht wird zu lösen und auch zum großen Teil schon gelöst wurde. Umgesetzt wird es durch ein Verfahren namens „Deferred Binding“. Man kann sich „Deffered-Binding“ als „dynamisches Class-Loading während der Kompilierzeit“ vorstellen, d.h. es wird beispielsweise durch das Kompilieren für jeden

[11] In dem über 16.000 Zeilen langen Code dieser Arbeit existiert lediglich eine einzige JNSI-Methode

[12] Durch eine Verschleierung soll Quellcode schwieriger nachvollzogen werden können, es werden beispielsweise die Variablen- und Funktionsnamen in ein einfaches Zeichen umbenannt.

Browsertyp und jede Browserversion entsprechender JavaScript-Code erzeugt. Die durch Defered-Binding erzeugten Permutationen des Codes, können nicht nur von Typ und Version des Browsers abhängig gemacht werden, sondern auch flexibel in anderen Kontexten angewandt werden. Zum Beispiel ist es möglich für jede Fremdsprache, die von der Anwendung unterstützt werden soll, eine Permutation zu erzeugen. Will man zum Beispiel eine Anwendung für Firefox, Chrome und den Internet Explorer auf Deutsch, Englisch, Spanisch und Französisch erstellen, so wird der Compiler des GWTs 3 x 4, also 12 Permutationen des Programmcodes erzeugen. Die richtige Permutation wird dabei zu Beginn, beim Besuch der Seite, durch den sogenannten „Bootstrapping"-Prozess ausgewählt, nachgeladen und verwendet. Dieser Prozess wird, durch ein ein kleines „Vorlade-Script" und eine Lookup-Tabelle umgesetzt [UNB11f].
Ein weiterer großer Vorteil der GWT-Entwicklung sind die Debug-Werkzeuge (Punkt 2d). Diese ermöglichen es, den Client- und Servercode auch ohne vorheriges Kompilieren im Browser zu testen. Dazu wird ein Plugin im Browser installiert, welches den erstellten Java-Bytecode des Eclipse-Debuggers dynamische in JavaScript umwandelt. Die Ausführungsgeschwindigkeit ist zwar um ein vielfaches langsamer, als die der später kompilierten Anwendung, jedoch kann durch diesen Ansatz der Eclipse-Debugger benutzt werden, der direkt den Java Code debuggen kann. Gleichzeitig kann im Browser das dynamisch übersetzte JavaScript durch Debugger, wie FireBug[13] oder den WebGL Inspector[14] zusätzlich debuggt werden.
Natürlich hat das GWT auch Grenzen. So werden nicht alle Klassen aus der Java-Runtime-Environment emuliert. Es wird zwar ein Großteil der Klassen aus den Paketen `java.lang` und `java.util` unterstützt, aus dem Paket `java.io` jedoch nur das Interface `Serializable`. Dies macht auch Sinn, da das java.io-Paket hauptsächlich dazu dient, mit Input- und Outputstreams beispielsweise Dateien in das lokale Dateisystem zu schreiben. JavaScript hingegen hat aus Sicherheitsgründen keinen Zugriff auf das lokale Dateisystem. Genauso wenig werden, die in Java enthaltenen Threads durch das GWT unterstützt, da JavaScript nur durch einen Thread ausgeführt wird. Eine vollständige Liste der emulierten Klassen ist auf [UNB11k] nachzulesen.
Aus diesen Gründen sollte beim Entwickeln immer daran gedacht werden, dass der geschriebene Code und auch die darin benutzten Bibliotheken letztendlich in JavaScript übersetzt werden müssen.

5.1.2 Remote Procedure Call

Das GWT setzt das beschriebene AJAX-Konzept mit Remote Procedure Calls (RPC) um. Unter einem RPC kann man sich eine Methode vorstellen, die der Client mit beliebigen Parametern aufruft und der Server mit einem Rückgabewert beantwortet. Die RPCs machen dabei einen Großteil der Kommunikation zwischen Client und Server der entwickelten Anwendung aus. Ihre Funktionsweise hat Auswirkung auf die Wahl und das Format der zu übertragenden Daten. Daher werden sie in diesem Teilabschnitt ausführlich vorgestellt.
Um einen RPC zu implementieren, muss wie in Listing 2 als erstes ein Interface definiert werden, das die RPC-Methode deklariert. Die Deklaration lässt erahnen, dass die Methode einem Client eine Liste mit Zufallszahlen, die auf dem Server generiert wurden, zurückgeben wird.

Zusätzlich zu dem genannten Interface, muss ein weiteres Interface erstellt werden[15]. Dieses In-

[13] https://addons.mozilla.org/de/firefox/addon/firebug/

[14] Der WebGL Inspector ermöglicht es WebGL-Anwendungen während der Laufzeit zu untersuchen. Man kann sich beispielsweise alle geladenen Texturen samt deren Eigenschaften anzeigen lassen (`http://benvanik.github.com/WebGL-Inspector/`).

[15] Wenn Eclipse und das Google-Eclipse-Plugin verwendet wird, wird dieses zweite Interface sogar generiert und muss

```
1 @RemoteServiceRelativePath("randomnumber")
2 public interface RandomNumberService extends RemoteService {
3   List<Integer> getRandomNumbers(int size) throws IllegalArgumentException;
4 }
```

Listing 2: Interface, das eine RPC-Methode deklariert.

terface deklariert die gleiche Methode, deren Deklaration sich nur durch einen leeren Rückgabewert (vom Typ `void`) und einen weiteren Parameter (vom Typ `AsyncCallback`) unterscheidet (siehe Listing 3 Zeile 2). Dieser weitere Parameter ist das Objekt, das sich um den asynchronen Callback, wie er bei diesem Konzept für den Aufruf auf der Clientseite gebraucht wird, kümmert.

```
1 public interface RandomNumberServiceAsync {
2   void getRandomNumbers(int size, AsyncCallback<List<Integer>> callback);
3 }
```

Listing 3: „Asynchrones Interface“, das eine RPC-Methode deklariert.

Nachdem die Methode mit den beiden Interfaces deklariert ist, muss sie nur noch auf Serverseite definiert werden und auf Clientseite aufgerufen werden. Dazu muss, wie in Listing 4 gezeigt, auf der Serverseite das erste Interface (Listing 2) implementiert werden.

```
1  public class RandomNumberServiceImpl extends RemoteServiceServlet implements
2      RandomNumberService {
3    @Override
4    public List<Integer> getRandomNumbers(int size) throws IllegalArgumentException {
5      if(size < 0)
6        throw new IllegalArgumentException("Size must be greater than zero.");
7      List<Integer> randomNumbers = new ArrayList<Integer>();
8      for (int i = 0; i < size; i++)
9        randomNumbers.add(new Integer((int)(Math.random() * 10)));
10     return randomNumbers;
11   }
12 }
```

Listing 4: Definition der RPC-Methode, durch die Implementierung des Interfaces auf dem Servlet

Auf der Clientseite muss logischerweise das zweite Interface mit dem asynchronen Callback-Objekt aufgerufen werden (siehe hierzu Listing 5 Zeile 2). Ein Java-Entwickler wird sich an dieser Stelle fragen, wie man ein Interface aufruft. Dies wird durch das beschriebene Deffered-Binding(Abschnitt 5.1.1) gelöst. Es gibt so die Möglichkeit zur Instanziierung des Interfaces (zu sehen in Zeile 1 von Listing 5). Deffered-Binding ist an dieser Stelle nötig, da die Datenübertragung, je nach Browser, durch den nativen JavaScript-Code des RPCs, anders abgewickelt werden muss.

Der letzte Schritt, um einen RPC zu realisieren, ist die „Verknüpfung“ zwischen dem ersten Interface und der Implementierung der Methode dieses Interfaces auf Serverseite in der Datei „web.xml“ (siehe hierzu Listing 6). Dies ist notwendig, damit beim Aufruf eines RPCs sichergestellt ist, welches Servlet dieses Interface implementiert hat und deshalb auch die betreffende Methode beantworten kann.

---

nicht von Hand geschrieben werden.

```
1   RandomNumberServiceAsync randomNumberService = GWT.create(RandomNumberService.class);
2   randomNumberService.getRandomNumbers(5, new AsyncCallback<List<Integer>>() {
3     @Override
4     public void onSuccess(List<Integer> randomNumbers) {
5       for (Integer randomNumber : randomNumbers)
6         System.out.println("Random number: " + randomNumber);
7     }
8
9     @Override
10    public void onFailure(Throwable caught) {
11      Window.alert("Failure while calling random number service.");
12    }
13  });
```

Listing 5: Aufruf der RPC-Methode auf der Clientseite.

Dazu wird über den Pfad des betreffenden Servlets das Servlet definiert (Listing 6 Zeile 1-4) und über ein Mapping von dieser Servlet-Definition und einem URL-Pattern, welches im Interface von Listing 2 in Zeile 1 definiert ist, verknüpft.

Listing 6: „Verknüpfung“ zwischen dem Interface und dem Servlet, das es implementiert in der „web.xml“.

```
1   <servlet>
2     <servlet-name>greetServlet</servlet-name>
3     <servlet-class>
4       de.uniko.gwt.randomnumberexample.server.RandomNumberServiceImpl
5     </servlet-class>
6   </servlet>
7
8   <servlet-mapping>
9     <servlet-name>greetServlet</servlet-name>
10    <url-pattern>/randomnumberexample/randomnumber</url-pattern>
11  </servlet-mapping>
```

Die Schritte einen einzigen RPC zu realisieren sind zwar komplex, relativieren sich jedoch, wenn mehrere Methoden in ein und demselben Interface bwz. Servlet deklariert und definiert werden. Des Weiteren bietet dieses Verfahren eine automatische String-Serialisierung, der zu übertragenden Daten an, so dass beliebige Objekte übertragen werden können. Die Serialisierung ist unumgänglich, da im Normalfall zwischen Client und Server mittels HTTP-Requests lediglich Strings übertragen werden können[16]. Im beschriebenen Beispiel wird eine Liste aus Integern (damit sind die Zufallszahlen gemeint) auf dem Server automatisch serialisiert und auf dem Client wieder deserialisiert. Beides läuft für den Entwickler verdeckt ab, trotzdem sollte man sich dessen bewusst sein.
Neben den RPCs stellt das GWT weitere Mechanismen zur Verfügung Daten zwischen Client und Server zu übertragen. Auf diese soll jedoch nicht weiter eingegangen werden, da sie in der Arbeit keine Verwendung finden.

5.1.3 Local Storage und Session Storage

Die Übertragung der 3D-Modelle und Texturen nimmt einen großen Teil der Ladezeit in der App in Anspruch. Damit diese Daten nicht mehrfach übertragen werden müssen, kann man den Local-Storage oder Session-Storage in Anspruch nehmen. Der Local Storage hat eine Kapazität von 5MB

[16] Es gibt auch Möglichkeiten Binärdaten zu übertragen: `https://developer.mozilla.org/En/XMLHttpRequest/Using_XMLHttpRequest#Handling_binary_data`. Diese sind jedoch noch nicht im GWT implementiert.

und speichert die abgelegten Daten sessionübergreifend, jedoch nicht browserübergreifend. Der Session Storage hingegen, speichert seine Daten nur über die Dauer einer Session und bietet unbegrenzte Kapazität [UNB11g]. Beide Speicher sind eine Art Hash-Map, die in der Lage ist zu einem Schlüssel einen Wert abzuspeichern. Allerdings muss sowohl der Schlüssel als auch dessen Wert ein String sein. Aus diesem Grund ist es nötig die abzuspeichernden Daten in einen String zu serialisieren. Die Serialisierungsfunktionen, die benutzt werden, um Objekte eines RPCs zu serialisieren, können hierzu nicht benutzt werden, da diese Serialisierung asymmetrisch[17] ist. Eine einfache Serialisierung ist mit der Bibliothek gwt-jsonmaker (`http://code.google.com/p/gwt-jsonmaker/`) möglich. Diese kann Objekte als String in das JSON-Format[18] serialisieren und deserialisieren.
Bei der Verwendung des Session Storages mit Chrome ist außerdem ein Bug zu beachten. Dieser äußert sich durch einen JavaScript-Fehler, sobald der Session Storage, der laut [UNB11g] eine unbegrenzte Kapazität hat, eine Kapazität von 5MB überschreitet. Ein einfaches Workaround für diesen Bug ist in Listing 7 zu sehen.

```
1 private void setSessionStorageItem(String key, String data){
2   try{
3     sessionStorage.setItem(key, data);
4   }catch(Exception e){
5     return;
6   }
7 }
```

Listing 7: Hinzufügen eines Tasks zu der Default-Task-Queue.

Um die beiden Storage-Typen effektiv einzubinden, werden sie in der Anwendung in Zusammenarbeit mit einem Connection-Manager genutzt. Die Aufgabe des Connection-Managers besteht darin jeder anderen Klasse auf dem Client einen einfachen Zugriff auf die Daten der verschiedenen Servlets zu geben. Der Connection-Manager ist daher als Singleton umgesetzt und bündelt alle RPCs, die die Anwendung tätigen muss. Somit ist es möglich Daten, die über den Connection-Manager angefragt werden und sich nur selten oder gar nicht verändern, im Session-Storage zwischenzuspeichern und bei einer erneuten Anfrage aus diesem wieder auszulesen. Des Weiteren kann an einer Stelle erfasst werden, welche Daten am häufigsten gebraucht werden, so dass diese in den Local-Storage geschrieben werden. In einer neuen Session stehen somit Daten, die häufig genutzt werden direkt zur Verfügung.

5.2 Google App Engine

Die Google App Engine (GAE) ist sowohl ein Framework, das die Entwicklung von Web-Anwendungen unterstützt, als auch die Plattform zum Hosten der entwickelten Anwendung in der Cloud [GU10]. Sie gehört damit zu den „Platform as a Service“ Produkten und unterstützt nativ Java, Python und in den neueren Versionen auch experimentell Go. In dieser Arbeit ist Java die gewählte Programmiersprache. Somit kommt es zu keinem Bruch zwischen der clientseitigen Entwicklung mit dem GWT und der serverseitigen Entwicklung mit der GAE. Außerdem ist die Entwicklung einer GWT-GAE-Anwendung durch das Google-Eclipse-Plugin gut aufeinander abgestimmt.
Einen weiteren Vorteil bietet die GAE durch eine automatische Skalierung der Serverinstanzen und eine hohe Performance. Automatische Skalierbarkeit bedeutet, dass bei einer hohen Auslastung der

[17] Asymmetrisch bedeutet, dass Objekte, die auf dem Client serialisiert bzw. deserialisiert werden, nur durch Serialisierungsmethoden, die es ausschließlich auf dem Server gibt, deserialisiert bzw. serialisiert werden können.

[18] JSON steht für JavaScript Object Notation und wird in JavaScript häufig für die Objektserialisierung genutzt, um Objekte zwischen Client und Server auszutauschen.

Web-Anwendung ohne Zutun neue Servlet-Instanzen gestartet werden. Dadurch ist es möglich die Auslastung auf mehrere Servlet-Instanzen zu verteilen, so dass die Anwendung weiterhin erreichbar und nutzbar bleibt. Aus diesem Grund eignet sich die App Engine auch für Anwendungen mit sehr großen Nutzerzahlen. Wird die App eine Zeit nicht genutzt, wird sogar die letzte Instanz „entladen". Danach hat die App beim Aufruf eine geringfügig längere Ladezeit aufgrund der neuen Instanziierungen.

5.2.1 Datenbank der Google App Engine

Über das SDK der App Engine ist auch ein Zugriff auf eine Datenbank, welche Teil der Plattform ist, möglich. Weder die Datenbank noch der Server muss während der Benutzung gewartet oder explizit eingerichtet werden. Hierbei sollte jedoch erwähnt werden, dass der direkte Zugriffe auf die Datenbank über die Bibliotheken der GAE einige Schwierigkeiten mit sich bringt [UNB11p]:

- der DatastoreService speichert nur spezielle GAE-spezifische Objekte und keine POJOs[19] ab
- Schlüssel[20] des DatastoreServices spezifizieren nicht den Datentyp des dazugehörigen Datenbankobjekts, was zu Fehlern bei der Verwendung führen kann
- der DatastoreService besitzt eine „maschinenfreundliche" aber kein „menschenfreundliche" Schnittstelle
- Transaktionen mit dem DatastoreService sind kompliziert

Um diese Schwierigkeiten zu umgehen, empfiehlt es sich eine Bibliothek zu benutzen, die den Umgang mit der Datenbank vereinfacht. In dieser Arbeit wird dazu Objectify [UNB11p] als Schnittstelle zur Datenbank verwendet. Objectifiy hat den Vorteil, dass Objekte (d.h. POJOs) aus der Datenbank ohne Konvertierung clientseitig auch durch das GWT verwendet werden können. Dazu müssen sie nur durch das Servlet angefragt werden und über einen RPC wie in Abschnitt 2 beschrieben, als Antwort weitergegeben werden. Listing 8 zeigt ein Servlet, das mit Objectify eine Klasse in der Datenbank registriert und über zwei Funktionen einen Lese- und Schreibzugriff für Objekte dieser Klasse bietet. In diesem speziellen Fall handelt es sich um den Zugriff auf ein Benutzerkonto. Zugriffe dieser Art, verbunden mit RPCs sind ein grundlegender und wiederkehrender Teil der Arbeit. Dieses Vorgehen ermöglicht es erst, Daten der Benutzer zu verwalten und persistent zu speichern.

Wie man in Zeile 10 des Listing 8 sieht, muss man sich die Datenbank als eine Art Hash-Map vorstellen. Man ruft einfach mit einem Schlüssel einen Wert ab, der zurückgegeben wird. Beziehungen zwischen Datenbankobjekten werden durch Schlüssel als Feldvariablen realisiert (siehe Listing 9). Die Datenbank unterscheidet sich mit diesem Ansatz von Relationalen Datenbanken [UNB11e].

Oftmals macht es auch Sinn den Zugriff nicht direkt über den Schlüssel eines Objekts, sondern über seine ID und die Klasse wie in Listing 10 Zeile 19 und 20 zu sehen ist, zu bewerkstelligen. Die ID ist hierbei eine einmalige Nummer (Long) oder eine Zeichenkette (String), die in jeder Datenbankklasse als Feldvariable vorhanden sein muss (siehe Listing 9 Zeile 3). Dieser Ansatz bietet den Vorteil, IDs im Sessionobjekt des Servlets abzulegen. Das Sessionobjekt ist ebenfalls eine Hashmap,

[19] POJO steht für:„Plain Old Java Object" und bezeichnet Java Objekte die keinen Beschränkungen unterliegen

[20] Schlüssel werden benutzt, um ein Objekt aus der Datenbank eindeutig zu identifizieren und darauf zuzugreifen

```
1  public class UserAccountServiceServlet extends RemoteServiceServlet implements
       UserAccountService{
2
3    static {ObjectifyService.register(UserAccountDto.class);}
4
5    @Override
6    public UserAccountDto getUserAccount(Key<UserAccountDto> userAccountKey){
7      Objectify objectify = ObjectifyService.begin();
8      return objectify.get(userAccountKey);
9    }
10
11   @Override
12   public Key<UserAccountDto> putUserAccount(UserAccountDto userAccount) {
13     Objectify objectify = ObjectifyService.begin();
14     return objectify.put(userAccount);
15   }
16 }
```

Listing 8: Servlet, das ein Benutzerkonto aus der Datenbank abfragt und als Ergebnis eines RPCs zurückgibt.

```
1 public class Employee
2 {
3   @Id String name;
4   Key<Employee> manager;
5 }
```

Listing 9: Klasse, die in einer Datenbank abgelegt werden kann und eine Many-To-One-Beziehung ausdrückt

die ausschließlich Schlüssel und Werte vom Typ String abspeichert. Diese Hashmap ist allerdings nicht global, sondern wird für jeden Nutzer und jede Session neu angelegt. So kann man beispielsweise eine einmalige Benutzer-ID zu einem Benutzerkonto verwalten. Dazu wird bei der erstmaligen Anfrage von einem Benutzer im Servlet-Sessionobjekt die Benutzer ID abgelegt (siehe Listing 10 Zeile 11). Bei jeder weiteren Anfrage des Benutzers kann die ID wieder aus dem Sessionobjekt ermittelt werden. Mit der ID wiederum kann das Benutzerkonto identifiziert werden (siehe Zeile 17). Listing 10 zeigt darüber hinaus, wie über das erlangte Benutzerkonto, ein Modell zu diesem Benutzer aus der Datenbank angefragt wird. Kommt es bei diesem Ansatz dazu, dass das Modell nicht mehr vorhanden ist[21], kann die ID des Modells aus dem Benutzeraccount ebenso gelöscht, oder ersetzt werden. Benutzt man das Sessionobjekt nicht, müsste für jede Anfrage die Benutzer- und Modell-ID mitgeschickt werden und die Anwendung wäre durch eine Manipulation der IDs auf der Clientseite, sehr leicht angreifbar.

5.2.2 Blobstore-API

Die Blobstore-API der GAE ist dazu gedacht, große Dateien hochzuladen, zu verarbeiten und anzubieten. Diese API ist notwendig, da es die App Engine, aufgrund ihrer Sandbox, in der sie jeden serverseitigen Code ausführt, verbietet in das lokale Dateisystem zu schreiben [UNB11w]. Die API eignet sich daher ideal um Benutzerinhalte hochzuladen und mit anderen Nutzern zu teilen. Jeder erstellte Blob wird über einen festen Schlüssel, genannt Blob-Key, ähnlich wie bei der Datenbank, identifiziert. Ein Blob-Key kann dabei in einen String umgewandelt werden und als Feld in einem POJO abgelegt werden. Das POJO kann wiederum im Datastore abgelegt werden, so dass ein persistenter Zugriff auf die Blob-Keys und somit auch deren Inhalte gewährleistet ist (siehe Abb. 2).

In der Arbeit kommt die Blobstore-API zum Einsatz Avatarmodelle, Modelle für die Innenein-

[21] Benutzerspezifische Inhalte können beispielsweise gelöscht werden.

```java
public class ModelServiceServlet extends RemoteServiceServlet implements UserAccountService{

  static {
    ObjectifyService.register(ModelDto.class);
    ObjectifyService.register(UserAccountDto.class);
    }

  @Override
  public void startSession(String userId){
    HttpSession session = getThreadLocalRequest().getSession(true);
    session.setAttribute("userId", userId);
  }

  @Override
  public ModelDto getUserModel(){
    HttpSession session = getThreadLocalRequest().getSession(true);
    String userId = (String) session.getAttribute("userId");
    Objectify objectify = ObjectifyService.begin();
    UserAccountDto userAccount = objectify.find(UserAccountDto.class, userId);
    ModelDto model = objectify.find(ModelDto.class, userAccount.getModelId());
    if(model == null){
      // Model is deleted -> replace the model ID in "userAccount"
      userAccount.setModelId(getExistingModelId());
      return getUserModel();
    }
    return model;
  }
}
```

Listing 10: Servlet, das zu einem Benutzerkonto aus der Datenbank ein Model abfragt und es als Ergebnis eines RPCs zurückgibt.

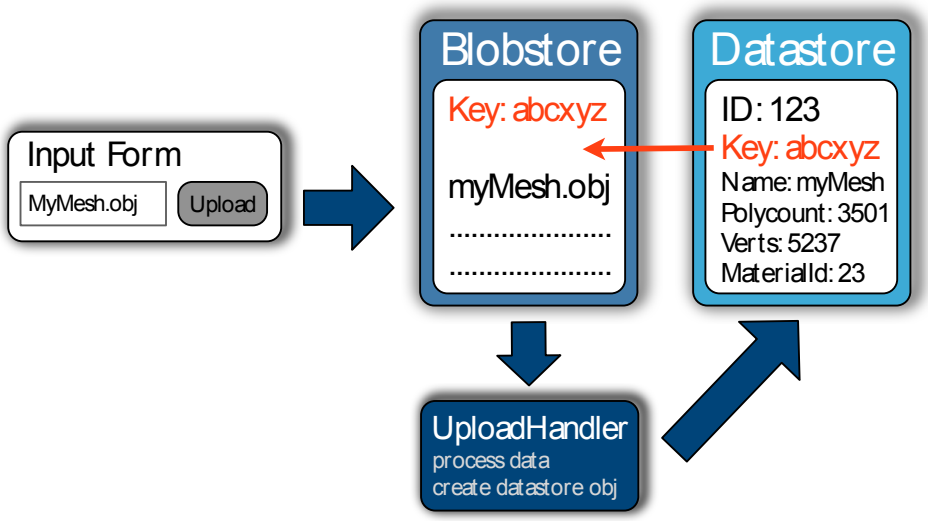


Abbildung 2: Verwaltung der Blobs mit dem Datastore.

richtung und Texturen hochzuladen und zu speichern. Da es sich bei einem Avatarmodell, als auch bei einem Modell für die Innenausstattung um mehrere Dateien handelt, werden Zip-Archive hochgeladen. Somit muss nur ein einziger Uploadvorgang vollzogen werden. Dieser profitiert zusätzlich von dem, durch die Komprimierung bedingten, geringeren Datenaufkommen.
Allerdings handelt es sich bei Teilen der Blobstore-API um experimentelle Technologie, wodurch es in der Arbeit an mehreren Stellen zu Problemen gekommen ist. Der Ablauf um eine Datei hochzuladen wird daher in den folgenden drei Punkten ausführlich beschrieben:

1. Anfrage der Upload-URL auf dem Server:
   Der Client fragt per RPC den Server nach einer Upload-URL. Die Upload-URL dient als eindeutiger Pfad zum Blobstore für die nächste Datei, die hochgeladen wird. Listing 11 zeigt die Methode des RPCs auf dem Server, die die Upload-URL über die Blobstore-Api ermittelt. Bei der Erstellung der URL (Zeile 6) muss als Parameter eine Callback-URL angegeben werden, die aufgerufen wird, sobald die Datei, die diese Upload-URL benutzt, vollständig hochgeladen ist. Hierbei sollte unbedingt beachtet werden, dass es im Debug-Modus zu einem Bug bei der Erstellung der URL kommen kann: Fälschlicherweise wird anstatt der Local-Host-IP (`127.0.0.1`), die IP `0.0.0.0` in der Upload-URL eingetragen. Das Workaround (Zeile 7 und 8) basiert darauf diese falsche IP, durch die Local-Host-IP zu ersetzen. Dies darf jedoch nur im Debug-Modus geschehen, da der Blobstore in diesem Fall lokal emuliert wird.

```
1  private final BlobstoreService blobstoreService = BlobstoreServiceFactory.
       getBlobstoreService();
2    private final static boolean IS_DEVELOPMENT_MODE = (SystemProperty.environment.value() ==
         SystemProperty.Environment.Value.Development);
3
4    @Override
5    public String getBlobUploadUrl() {
6      String uploadUrl = blobstoreService.createUploadUrl("/blobuploaded");
7      if(IS_DEVELOPMENT_MODE)
8        uploadUrl = uploadUrl.replaceFirst("http://0.0.0.0:8888", "http://127.0.0.1:8888");
9      return uploadUrl;
10   }
```

Listing 11: RPC Methode, um an eine Upload-URL zu gelangen.

2. Bereitstellen der Upload-Widgets auf dem Client:
   Sobald eine Upload-URL auf dem Client verfügbar ist, kann eine Datei mit einem `FileUpload`-Widget, einem `FormPanel` und einem `Button` an diese URL hochgeladen werden (siehe Listing 12). Das `FileUpload`-Widget dient dazu einen Dateibrowser zu öffnen und wird über die `setWidget`-Methode des `FormPanels`, mit eben diesem Panel verbunden. Das `FormPanel` wiederum ist ein unsichtbares Panel, das nur dazu dient ein anderes Widget, in diesem Fall das `FileUpload`-Widget, an eine Zieladresse eines Servers zu schicken. Da das `FileUpload`-Widget den Pfad zu der Datei enthält, verschickt das `FormPanel` sozusagen den Pfad der Datei an die zuvor angefragte Upload-URL des Blobstores. Der Server des Blobstores wiederum veranlasst beim Eintreffen dieser Nachricht den Verbindungsaufbau für den Upload. Wichtig ist es bei der Erstellung des `fileUpload`-Widgets einen Namen für dieses Widget zu vergeben (siehe Zeile 4). Nur über diesen Namen wird es in Punkt 3 möglich sein, den Blobkey der hochgeladenen Datei zu ermitteln.
   Dieses Listing soll als Beispiel dienen und zeigt deshalb die Erstellung und Einstellung der Widgets. Im Normalfall werden die Widgets nicht für jede hochzuladende Datei neu erstellt, da

es ausreicht dem `FormPanel` eine neue Upload-URL für jede hochzuladende Datei zu setzen (Zeile 7).

```
1
2  private initUploadWidgets(String uploadUrl){
3    // Create a file browsing widget
4    FileUpload fileUpload = new FileUpload();
5    fileUpload.setName("zipFile");
6
7    // Create a form panel to submit the file upload widget to the server
8    FormPanel formPanel = new FormPanel();
9    formPanel.setAction(uploadUrl);
10   formPanel.setEncoding(FormPanel.ENCODING_MULTIPART);
11   formPanel.setMethod(FormPanel.METHOD_POST);
12   formPanel.setWidget(fileUpload);
13
14   // Create a button to start the upload
15   Button uploadButton = new Button("upload");
16   uploadButton.addClickHandler(new ClickHandler() {
17       @Override
18       public void onClick(ClickEvent event) {
19         formPanel.submit();
20       }
21   });
22
23   // Add the widgets to the web site
24   RootPanel rootPanel = RootPanel.get();
25   rootPanel.add(formPanel);
26   rootPanel.add(uploadButton);
27 }
```

Listing 12: Widgets, die es erlauben eine Datei vom Client in den Blobstore hochzuladen.

3. Zugriff auf den Blob:
   Wurde eine Datei vollständig hochgeladen, wird der Blobstoreserver die Post-Methode des Servlets aufrufen, dessen Pfad bei der Erzeugung der Upload-URL in Punkt 1 übergeben wurde. Listing 13 zeigt, wie in der Post-Methode der Blobkey der hochgeladenen Datei ermittelt werden kann. Dort wird über den Namen, der auf dem Client für das `FileUpload`-Widget angegebenen wurde, der betreffende Blobkey erfasst.

```
1
2  @Override
3  protected final void doPost(HttpServletRequest req, HttpServletResponse resp)
4      throws ServletException, IOException {
5
6    Map<String, BlobKey> blobs = blobstoreService.getUploadedBlobs(req);
7    BlobKey blobKey = blobs.get("zipFile");
8  }
```

Listing 13: Ermitteln des gerade hochgeladenen Blobs.

Mit dem Blobkey ist es möglich auf die Inhalte des Blobs zuzugreifen. Allerdings kommt es auch hier, wahrscheinlich aufgrund des experimentellen Status' zu Problemen. So kann ein Blob nicht immer unmittelbar, nachdem er erstellt wurde, ausgelesen werden. Abhilfe schafft es, vor einem Lesezugriff die Verfügbarkeit des Blobs über eine Zeitspanne hinweg mehrmals zu überprüfen. Listing 14 zeigt die Methoden, die diesen Workaround implementieren. Die erste Methode (`readBlob`) ist die Hauptmethode, sie überprüft mit der `isBlobAvailable`-Methode über drei Sekunden zehn mal die Verfügbarkeit des Blobs. Erst, wenn der Blob verfügbar ist, wird damit begonnen ihn auszulesen. Dabei sollte beachtet werden, dass der Blob in

mehreren und nicht zu großen Teilen (`bufferSize = 1024`) ausgelesen wird.

```
1  private static final FileService fileService = FileServiceFactory.getFileService();
2  private static final BlobstoreService blobstoreService = BlobstoreServiceFactory.
       getBlobstoreService();
3  private static final BlobInfoFactory blobInfoFactory = new BlobInfoFactory();
4
5  public static byte[] readBlob(BlobKey blobKey) throws BlobServiceException{
6    if(!isBlobAvailable(blobKey, 10, 300))
7      throw new BlobServiceException("File with blob key: " + blobKey.getKeyString() + ". is
           not available.");
8
9    FileObject fileObject = new FileObject("");
10   long bufferSize = 1024;
11   long offset = 0;
12   BlobInfo blobInfo = blobInfoFactory.loadBlobInfo(blobKey);
13   long fileSize = blobInfo.getSize();
14   while (offset < fileSize) {
15     if(offset + bufferSize >= fileSize)
16       bufferSize = fileSize - offset;
17     byte[] bytesFetched = blobstoreService.fetchData(blobKey, offset, offset + bufferSize);
18     fileObject.appendData(bytesFetched, (int)bufferSize);
19     offset += bufferSize;
20   }
21   return fileObject.getData();
22 }
23
24 private static boolean isBlobAvailable(BlobKey blobKey, int numberOfRetries, long timeOut){
25   int i = 0;
26   while(i < numberOfRetries){
27     try {
28       if(fileService.getBlobFile(blobKey) != null)
29         return true;
30       else{
31         sleep(timeOut);
32         i++;
33         continue;
34       }
35     } catch (FileNotFoundException e) {
36       sleep(timeOut);
37       i++;
38       continue;
39     }
40   }
41   return false;
42 }
43
44 private static void sleep(long millis){
45   try {
46     Thread.sleep(millis);
47   } catch (InterruptedException e) {
48     e.printStackTrace();
49   }
50 }
```

Listing 14: Zugriffsmethoden auf den Blob.

Im Gegensatz zum Lesen von Blobs, bereitet das Schreiben von Blobs keine Probleme. Die Beispiele der Dokumentation [UNB11x] sind daher ausreichend.

5.2.3 Task-Queue

Aufgrund der Beschränkung, dass keine neuen Threads auf der GAE gestartet und ausgeführt werden können, gibt es die Möglichkeit mit der GAE Tasks zu erstellen. Ein Task ermöglicht es, während einer Anfrage an ein Servlet, eine weitere Anfrage an ein anderes Servlet zu stellen, die im Hintergrund unabhängig von der ersten Anfrage abläuft. Dazu wird eine Task-Queue über die GAE API angefragt, der man Task-Optionen hinzugefügt. Die Optionen bestehen dabei aus einem Pfad zu dem besagten

anderen Servlet, beliebigen URL-Parametern und weiteren Einstellungen. Sobald die Optionen der Task-Queue hinzugefügt wurden, wird eine Anfrage an die Post-Methode des Servlets mit dem zuvor angegebenen Pfad geschickt. Das Servlet wird diese Anfrage erhalten, sobald alle vorherigen Tasks der Queue abgearbeitet sind. Listing 15 zeigt in einem einfachen Beispiel, wie man der Default-Queue einen Task mit einem Parameter hinzufügt.

```
1 // Set the servlet path
2 TaskOptions taskOptions = TaskOptions.Builder.withUrl("/taskrunner");
3 // Add a parameter
4 taskOptions = taskOptions.param("foo", "bar");
5 // Add the options to the queue
6 QueueFactory.getDefaultQueue().add(taskOptions);
```

Listing 15: Hinzufügen eines Tasks zu der Default-Task-Queue.

In der Arbeit werden Tasks dazu benötigt, Archive mit 3D-Modellen oder Texturen, die vom Benutzer hochgeladen werden, zu verarbeiten. Des Weiteren kümmert sich ein „Säuberungstask“ darum, einen Benutzer aus sämtlichen Räumen der virtuellen Welt zu entfernen, sobald die Verbindung zu ihm unterbrochen wird.
Da sich bei mehreren Tasks, wie es in der App der Fall ist, die einzelnen Tasks nur anhand ihrer übermittelten URL-Parameter im taskausführenden Servlet unterscheiden. Macht es Sinn die Tasks sinnvoller zu kapseln. In der Arbeit ist dies durch eine abstrakte Task-Klasse gelöst, die in der Lage ist sich selbst als String zu serialisieren. Durch diese Funktionalität können Instanzen dieser Klasse oder ihrer Subklassen, sich selbst serialisieren und sich als URL-Parameter in den Task-Optionen einbetten. Das Servlet, das nun die Anfrage bearbeitet, kann den Parameter deserialisieren und erhält dadurch die Instanz wieder. Listing 16 zeigt diese abstrakte Task-Klasse. Wie man sieht, besitzt die Klasse neben der `addToTaskQueue`- und der Serialisierungsmethode eine weitere Methode (`process`). Durch diese Methode kann, in jeder von Task abgeleiteten Klasse, die Aufgaben, die der konkrete Task übernehmen soll, aufgrund der Polymorphie ausgeführt werden.

Die Ausführung eines solchen Tasks ist wie in Listing 17 dementsprechend einfach. Durch diesen Ansatz wird nicht nur die explizite Unterscheidung zwischen unterschiedlichen Tasks im ausführenden Servlet überflüssig, es wird auch ermöglicht beliebig viele Eingabedaten, die in Form von Feldern der Klassen festgelegt werden, zu übermitteln. Denn durch die Serialisierung werden diese Felder ebenso mitserialisiert und sind somit auch im taskausführenden Servlet vorhanden.

5.2.4 Channel API

Der Austausch von Daten zwischen mehreren Benutzern, sei es zur Kommunikation oder zur Bewegung des Avatars, ist eine Grundvoraussetzung für eine virtuelle Welt. Die Übertragung von Daten zwischen einer Web-Anwendung wird jedoch im Normalfall über HTTP abgewickelt. HTTP ist ein Protokoll, das es nicht auf direktem Weg zulässt, Daten zwischen mehreren Clients auszutauschen. Es arbeitet nach einem Anfrage-Antwort-Prinzip zwischen Client und Server, welches durch die RPCs des GWTs aus Abschnitt 5.1.2 abgedeckt wird. Aus diesem Grund ist es nicht möglich, dass ein Client zu einem anderen Client Daten überträgt oder ein Server ohne vorherige Anfrage des Clients, Daten zu diesem schickt. Einen Ausweg für dieses Problem bieten Server-Push-Verfahren, welche Daten mittels Polling von Server an Client übermitteln. Polling bezeichnet ein Verfahren, das darauf basiert,

```
1
2  public abstract class AbstractTask implements Serializable{
3    private static final long serialVersionUID = 1L;
4    public static final String SERIALIZED_TASK = "serializedTask";
5    private static final String TASK_RUNNER_SERVLET_PATH = "/taskrunner";
6    protected transient TaskOptions taskOptions = TaskOptions.Builder.withUrl(
         TASK_RUNNER_SERVLET_PATH);
7    protected transient Queue queue = QueueFactory.getDefaultQueue();
8
9    public abstract void process();
10
11   public final void addToTaskQueue(){
12     taskOptions.param(SERIALIZED_TASK, serializeBase64Task());
13     taskOptions.countdownMillis(0);
14     queue.add(taskOptions);
15   }
16
17   protected String serializeBase64Task(){
18     byte[] array = null;
19     try{
20       ByteArrayOutputStream byteOutputStream = new ByteArrayOutputStream();
21       ObjectOutputStream objectOutputStream  =   new ObjectOutputStream( byteOutputStream );
22       objectOutputStream.writeObject(this);
23       objectOutputStream.close();
24       array = byteOutputStream.toByteArray();
25     }
26     catch(Exception e){
27       e.printStackTrace();
28     }
29     return new String(Base64Coder.encode(array));
30   }
31 }
```

Listing 16: Abstrakte Task-Klasse, die sich selbst serialisieren und an ein Task-Runner-Servlet verschicken kann.

```
1  public class TaskRunnerServiceServlet extends HttpServlet{
2    @Override
3    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
4        throws ServletException, IOException {
5      String base64SerializedTask = req.getParameter(AbstractTask.SERIALIZED_TASK);
6      AbstractTask abstractTask = (AbstractTask) deserializeBase64Task(base64SerializedTask);
7      abstractTask.process();
8    }
9
10   private Object deserializeBase64Task(String base64SerializedTask){
11     Object object = null;
12     byte[] serializedTask = Base64Coder.decode(base64SerializedTask);
13     try{
14       ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(serializedTask);
15       ObjectInputStream objectInputStream = new ObjectInputStream(byteArrayInputStream);
16       object = objectInputStream.readObject();
17     }
18     catch(Exception e){
19       e.printStackTrace();
20     }
21     return object;
22   }
23 }
```

Listing 17: Das Task-Runner-Servlet kann eintreffende Tasks, die von der AbstractTask-Klasse abgeleitet wurden, deserialisieren und ausführen.

dass der Client wiederholt Anfragen an den Server stellt und der Server diese erst beantwortet, wenn die Daten, die übertragen werden sollen, vorhanden sind. Der direkte Datenaustausch zwischen mehreren Clients ist auch mit dieser Technik nicht möglich. Er kann aber nun durch eine Weiterleitung des Servers abgewickelt werden.
Die GAE stellt ein solches Server-Push-Verfahren bereits über die Channel API [UNB11c] zur Verfügung. Ein Polling-Verfahren muss daher nicht eigenständig implementiert werden. Die Channel API erlaubt es über eine einmalige ID, die beim Verbindungsaufbau zwischen Client und Server ausgetauscht wird, Strings vom Server an den Client zu schicken. Die ID dient dabei zur Unterscheidung der verschiedenen Clients auf dem Server, d.h. eine Nachricht kann nur über diese ID an den jeweiligen Client übertragen werden. Clientseitig gibt es für diese Bibliothek offiziell nur eine JavaScript-Schnittstelle. Ein GWT-Binding für diese Schnittstelle wird jedoch extern angeboten [H.11]. Da es nur möglich ist Strings von Server zu Client zu schicken, ist es wieder notwendig eine effektive Serialisierung eigenständig zu implementieren. Hierbei kann die asymmetrische und hoch performante RPC-Serialisierung genutzt werden. Die Benutzung dieser Serialisierung ist eigentlich nicht für Entwickler gedacht und aus diesen Gründen nicht dokumentiert. Wie die Serialisierung trotzdem genutzt werden kann, lässt sich im Quellcode zu [Sen11] finden. Listing 18 zeigt die daraus abgeleitete Deserialisierung auf dem Client.

```
1
2  private SerializationStreamFactory serializationStreamFactory = GWT.create(
       IsSerializableService.class);
3  // These dummy interfaces are needed to deserialize the objects
4  public interface IsSerializableService extends RemoteService {
5    IsSerializable getMessage(IsSerializable object);
6  }
7  public interface IsSerializableServiceAsync {
8    void getMessage(IsSerializable object, AsyncCallback<IsSerializable> callback);
9  }
10
11 public IsSerializable deserialize(String serializedString) {
12   try {
13     return (IsSerializable)serializationStreamFactory.createStreamReader(serializedString).
           readObject();
14     } catch (SerializationException e) {
15       e.printStackTrace();
16   }
17 }
```

Listing 18: Manuelle RPC-Deserialisierung auf dem Client

Die beiden Dummy-RemoteService-Interfaces am Anfang des Listings (Zeile 3-8) sind notwendig, damit die zu serialisierenden und deserialisierenden Objekte in einer White-List aufgenommen werden. Diese Liste fungiert als internes Sicherheitskonzept des GWTs und gibt normalerweise an, welche Objekte durch einen RPC versendet und empfangen werden dürfen. Erst durch diesen Trick lässt sich die interne Serialisierung, die für die RPCs verwendet wird, nutzen.
Da die `getMessage`-Methode (Zeile 4) ein Objekt vom Typ `IsSerializable` als Argument sowie als Rückgabewert hat, wird in die White-List `IsSerializable` zum serialisieren und deserialisieren aufgenommen. `IsSerializable` stellt hierbei ein Interface dar. Deshalb können alle Objekte, die dieses Interface implementieren, mit den RPC-Serialisierungsmethoden serialisiert und deserialisiert werden.
Nach der Deserialisierung auf dem Client, fehlt nur die Serialisierung auf dem Server. Listing 19 zeigt die serverseitige Serialisierung, welche ohne weitere Kniffe auskommt.

```
1  public String serialize(IsSerializable object)
2  {
3    String serializedObjectString = null;
4    ServerSerializationStreamWriter streamWriter = new ServerSerializationStreamWriter(RPC.
         getDefaultSerializationPolicy());
5    streamWriter.prepareToWrite();
6    try {
7      streamWriter.writeObject(object);
8    } catch (SerializationException e) {
9      e.printStackTrace();
10   }
11   return streamWriter.toString();
12 }
```

Listing 19: Manuelle RPC-Serialisierung auf dem Server

5.2.4.1 WebSocket API

Als effektive alternative zu den Server-Push-Verfahren, gibt es seit kurzer Zeit die WebSocket API. Diese erlaubt eine bidirektionale Verbindung zwischen Client und Server. Der Vorteil dieser Verbindung, liegt darin, dass es nicht mehr zu einem ständigen Verbindungsaufbau und Abbau kommt. Dies ist nämlich bei HTTP der Fall. Durch den Wegfall des Auf- und Abbaus entsteht ein kleinerer Overhead, der wesentlich geringere Latenzen zwischen Client und Server verursacht.
Während der Implementierungsphase dieser Arbeit wurde die WebSocket API, jedoch ausschließlich von Chrome unterstützt, da sie in Firefox aufgrund von Sicherheitsbedenken deaktiviert worden ist. Die in Abschnitt 2.1.5 beschriebene virtuelle Welt KataSpace, ist aus diesem Grund ausschließlich in Chrome lauffähig.
Überdies gibt es nur wenige Server-Implementierungen dieser API. Das schließt auch die Server der GAE ein. Die Benutzung eines WebSocket-fähigen Servers, hätte daher zur Folge gehabt, dass sämtliche Vorteile der App Engine, im speziellen die komfortable und automatische Einrichtung der ganzen Serverinfrastruktur, nicht mehr genutzt werden könnten.

5.3 Facebook

Die Arbeit als Facebook-App umzusetzen, hat vor allem den pragmatischen Grund, dass es das meistbesuchte soziale Netzwerk ist. Dem verursachten Netzwerkverkehr nach [Int11] rangiert es auf Platz zwei hinter `www.google.com`. Des Weiteren bietet Facebook für die App-Entwicklung eine umfangreiche Dokumentation.
Da die Anbindung an Facebook einen integralen Bestandteil dieser Arbeit darstellt, soll in den beiden folgenden Unterabschnitten, die App-Registrierung, -Autorisierung, -Autentifizieren, die Kommunikation zwischen der App und Facebook und auf die einhergehenden Schwierigkeiten eingegangen werden. Dazu werden die nötigen Schritte beschrieben, um auf Facebook eine App anbieten zu können, wie man diese debuggen kann und wie die Kommunikation zwischen der App und Facebook abläuft. Grundlage für die daraus entstandenen Bibliothek und Teile, der hier beschriebenen Schritte, bot das Tutorial von [ere11].

5.3.1 Registrierung der App

Unter der Registrierung muss man sich vereinfacht gesagt, das Bereitstellen einer App-URL innerhalb der Facebook-Domain vorstellen. Wird diese URL aufgerufen, so sieht die Webseite aus, wie eine herkömmlich Facebook-Seite mit der bekannten Menüleiste, jedoch mit dem Unterschied, dass im Zentrum dieser Seite ein `iframe` mit einer Breite von 760 Pixeln eingebettet ist (siehe Abb. 3).

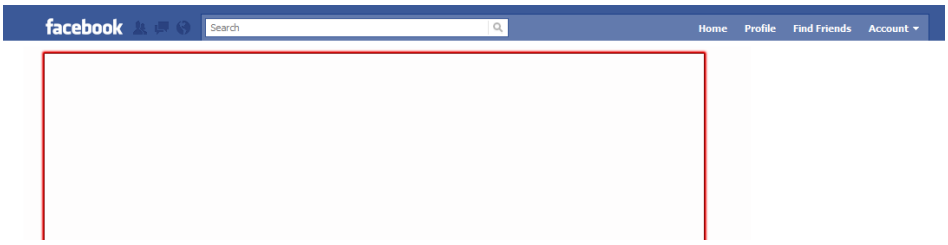


Abbildung 3: Leere Facebook-App-Seite (Der rote Rahmen dient nur zur Kenntlichmachung und hebt das `iframe` hervor).

Dieser Aufbau ermöglicht es erst die App, die in das `iframe` geladen wird, von Facebook zu trennen, was zur Folge hat, dass die eigentliche App auf einem eigenen Server liegen muss. Während der Registrierung braucht man daher lediglich die URL zu dem Server, auf dem die eigene App gehostet wird, eine noch freie wählbare Facebook-App-URL und einen frei wählbaren App-Namen (siehe Abb.4). Durch die Registrierung erhält man eine „Application ID“ und ein „Application Secret“, der Zweck dieser beiden Zeichenketten wird in Abschnitt 5.3.3 beschrieben. Die Registrierung der App ist damit jedoch abgeschlossen.

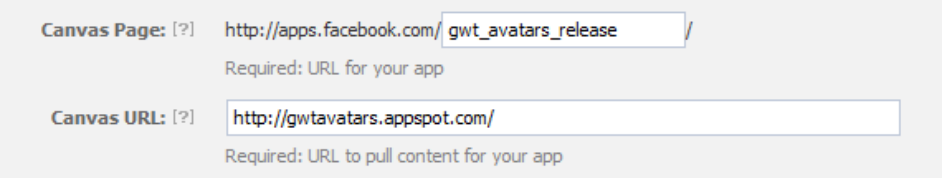


Abbildung 4: Canvas Page muss eine noch „freie“ URL auf Facebook sein. Canvas URL ist die URL auf der die App eigentlich liegt

5.3.2 Debuggen einer Facebook GWT-GAE-App

Um die Entwicklungszyklen einer GWT-GAE-App, die speziell für Facebook entwickelt wird, möglichst effektiv zu gestalten, wird in diesem Abschnitt erklärt, wie die App lokal gehostet werden kann und trotzdem auf Facebook eingebettet wird. Dieser Ansatz hat den Vorteil, dass die App ohne zu deployen, im Debug-Modus getestet werden kann. Somit bleibt die App unter voller Kontrolle der Eclipse-Debugwerkzeuge. Ebenso entfällt die Zeit, die benötigt wird, um die App zu deployen, welche bei kleineren Apps schon mehrere Minuten betragen kann. Dazu muss als erstes wie im vorherigen Abschnitt beschrieben eine weitere Registrierung einer Facebook-App durchgeführt werden. Diese bekommt anstatt der Adresse auf die, die App später deployet wird, eine Localhost-URL. Genauer gesagt, die Adresse, die das Google-Eclipse-Plugin im Development-Mode nach dem starten der App angibt. Das alleine reicht jedoch nicht, da es im Development-Mode fälschlicherweise zu einer Verletzung der „Same Origin Policy“ (im Folgenden SOP genannt) durch „Cross Site Scripting“[22] kommt[Ste11]. Der Browser blockiert daher die Facebook-App-Seite im Developmentmodus. Dieser Fehler kann jedoch durch die Änderung einer Zeile in der generierten Datei `hosted.html` aus dem

[22] Cross-Site Scripting bezeichnet das Nachladen von Inhalten, die auf einem anderen Host liegen oder einen anderen Port bzw. Protokoll erfordern. Wird ein solches Verhalten von dem Browser entdeckt, kommt es zu einer Verletzung der „Same Origin Policy“ und in Folge dessen wird die betroffene Seite blockiert, da hierdurch ein potentieller Angriff ermöglicht wird (vgl. [Rud11])

App-Verzeichnis behoben werden[23] (siehe hierzu Listing 20).

Listing 20: Änderungen zur Vorbeugung einer SOP-Verletzung(vgl. [Ste11])

```
1  gwtOnLoad = function(errFn, modName, modBase){
2    // ....
3    var topWin = window.top; // << Betreffende Zeile
4    // ...
5  }
6
7
8  // Muss geändert werden zu:
9  gwtOnLoad = function(errFn, modName, modBase){
10   // ....
11   var topWin = window;
12   // ...
13 }
```

5.3.3 Autorisierung und Autentifizierung durch das OAuth-Protokoll

Nachdem beschrieben wurde, wie die App registriert und debuggt wird, soll in diesem Abschnitt gezeigt werden, wie die Autorisierung und Authentifizierung zwischen Benutzer und App umgesetzt wird. Man unterscheidet dabei zwischen App- und Benutzer-Authentifizierung, sowie der App-Autorisierung. Die Benutzer-Authentifizierung garantiert, dass der Benutzer derjenige ist, für den er sich ausgibt[UNB11b]. Die App-Authentifizierung gewährleistet, dass die Benutzerinformationen auch wirklich an die besagte App und nicht an eine andere App weitergegeben werden können[UNB11b]. Die Autorisierung, genauer gesagt, die App-Autorisierung stellt sicher, dass der Benutzer zum einen weiß, welche Daten er mit der App teilt und zum anderen, dass ausschließlich diese Daten und Informationen von der App abgefragt werden können[UNB11b].
Facebook setzt dabei auf das OAuth 2.0 Protokoll[UNB11l]. Dieses Protokoll dient dazu, benutzerspezifische Inhalte, die üblicherweise an eine Webseite, einen Benutzernamen und ein Passwort gebunden sind, einer anderen Webseite bzw. App zur weiterzugeben. Dabei wird der zweiten Seite oder App weder der Benutzernamen noch das Passwort übermittelt[HL11b].

5.3.4 Autorisierung und Authentifizierung aus der Sicht des Benutzers

Der Ablauf der Autorisierung und Autentifizierung gliedert sich aus Sicht des Benutzers in drei Punkte und wird im Folgenden kurz beschrieben:

1. Der Benutzer ruft die App-Seite auf
2. Ist der Benutzer zu diesem Zeitpunkt nicht in Facebook eingeloggt, wird er darum gebeten sich nun einzuloggen (siehe Abb.5).
3. Nach dem Einloggen wird er, falls er die App noch nicht autorisiert hat, zur App-Autorisierungsseite weitergeleitet (siehe Abb.6)
4. Autorisiert der Benutzer die App, steht sie ihm danach zur Verfügung.

[23] Die Datei „hosted.html“ hat folgenden Pfad: „MeinGwtProjekt/war/meingwtprojekt/hosted.html“

Abbildung 5: Einloggen auf Facebook.

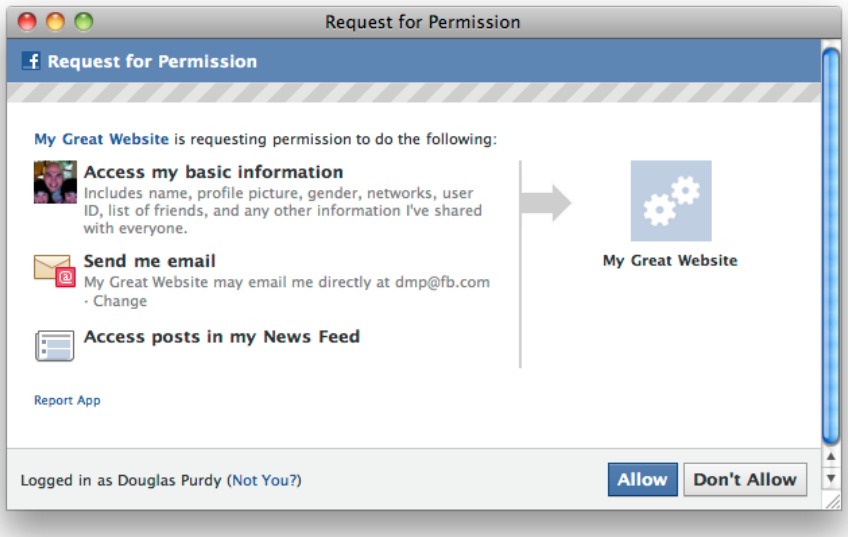


Abbildung 6: Autorisierung der App. Der Benutzer sieht an dieser Stelle welche Informationen die App von seinem Facebook-Account beziehen will und kann sich entscheiden der App den Zugriff zu diesen Informationen zu gewähren.

5.3.5 Autorisierung und Authentifizierung aus der Sicht eines GWT-GAE-App-Entwicklers

Ziel der Autorisierung und Authentifizierung aus Sicht des Entwicklers ist das Erlangen eines Access-Token. Dieses Token kann als Schlüssel angesehen werden, um an die Benutzerinformationen, die im Autorisierungsdialog angegeben sind, zu gelangen. Das Accesstoken ist dabei nichts anderes als eine Zeichenkette, welches ausschließlich einer autorisierten App ermöglicht über Anfragen an einen Facebook-Server an die genannten Benutzerinformationen zu kommen.
Das nötige Vorgehen, um an das Token zu gelangen, lässt sich am besten anhand eines Szenarios erklären. Dabei wird angenommen, dass der Benutzer weder bei Facebook eingeloggt ist, noch dass er die App autorisiert hat. Dieser Vorzustand deckt alle Fälle der Autorisierung und Authentifizierung ab.
Zum Zeitpunkt des ersten Aufrufs der App-Seite, weiß weder das clientseitig ausgeführte JavaScript noch der App-Server, um welchen Benutzer es sich handelt. Dieser Zustand wird Clientseitig dadurch erfasst, dass die URL in der Browseradressleiste auf Parameter[24] untersucht werden. Hierbei wird auf die Parameter „code“ und „error_reason“ untersucht(siehe hierzu Listing 21). Diese beiden Parameter sind, unter der Annahme des oben beschriebenen Vorzustands, nicht vorhanden. Aus diesem Grund veranlasst das clientseitig ausgeführte JavaScript eine Weiterleitung. Die URL zur Weiterleitung richtet sich an: `https://www.facebook.com/dialog/oauth?` und beinhaltet die folgenden Parameter:

- „method“ erhält den Wert „permissions.request“. Mit diesem Parameter wird mitgeteilt, dass die App Zugriff auf die Benutzerdaten des Benutzers, der gerade die Seite der App geöffnet hat, haben will.
- „client_id“, ihr Wert muss die in 5.3.1 beschriebene „Application ID“ sein, damit der Facebook-Server unsere App eindeutig identifizieren kann[25].
- „redirect_uri“ bekommt die URL zu der Facebook-App-Seite, welche in dem `iframe` die App einbindet. Durch diese URL kann der Facebook-Server den Benutzer nach der Authentifizierung und Autorisierung wieder auf die App-Seite weitergeleiten.
- „scope“ der Wert von Scope ist abhängig von den Benutzerinformationen auf die unsere App Zugriff erlangen will[26]. Diese Informationen werden auch im schon erwähnten Autorisierungsdialog (siehe Abb. 6) angezeigt.

```
1     String authorizationCode = Location.getParameter("code");
2     String errorReason = Location.getParameter("error_reason");
3     if((errorReason == null || errorReason.equals("")) &&
4        (authorizationCode  == null || authorizationCode.equals(""))){
5       redirect(getAuthorizeUrl());
6     }
```

Listing 21: Überprüfung der Parameter

Die Weiterleitung hat zur Folge, dass sich der Benutzer als erstes mit seinem Benutzernamen und Passwort über ein Login-Dialog auf einem Facebook-Server einloggt (siehe Abb. 5)[27]. Nach

[24] URL-Parameter sind Zusätze in einer URL, die auf Clientseite und Serverseite abgefragt werden können. Sie haben einen Namen und einen Wert. Z.B http://www.parameter.com/index.html?parameter1=wert1.

[25] Die App-ID ist öffentlich zugänglich. Dieser Schritt könnte daher auch von Dritten vollzogen werden.

[26] Die möglichen Werte sind auf [UNB11m] aufgelistet und beschrieben.

[27] Hat der Benutzer sich zuvor eingeloggt, wird ihn der Facebook-Server anhand eines Login-Cookies [UNB11b] identifizieren und ihn direkt den App-Autorisierungsdialog angezeigen.

einer erfolgreichen Validierung von Benutzername und Passwort seitens Facebook, ist die Benutzer-Authentifizerung abgeschlossen. Unmittelbar danach bekommt der Benutzer den App-Autorisierungsdialog angezeigt. Lehnt der Benutzer die Autorisierung ab, so wird er wieder von dem Facbook-Server zurück auf die in 5.3.5 angegebene URL weitergeleitet. Diesmal enthält die URL allerdings den in Listing 21 angegebenen Fehlercode[28]. Stimmt er der Autorisierung zu, wird er ebenso weitergeleitet, jedoch enthält die URL den in Listing 21 angegebenen Autorisierungs-Code, welcher nicht mit dem Access-Token zu verwechseln ist. Somit ist neben der Benutzer-Authentifizierung auch die App-Autorisierung abgeschlossen worden. Der letzte Schritt ist die App-Authentifizierung, d.h. ist die App wirklich die App, als die sie sich mit der öffentlich zugänglichen App-ID ausgibt. Dazu wird der, in der URL enthaltene, Autorisierungs-Code an den App-Server geschickt. Dieser fragt mit diesem Autorisierungs-Code, den in 5.3.1 beschriebenen App-Secret [29], der Redirection-URL und der App-ID einen Facebook-Server nach dem Access-Token. Die Anfrage wird validiert und nur bei erfolgreicher Validierung wird ein Access-Token, für die vom Benutzer autorisierten Inhalte zurückgegeben. Abbildung 7 fast die Schritte der Autorisierung und Authentifizierung in einem Sequenzdiagramm nochmals zusammen.

5.3.6 Kommunikation zwischen App und Facebook

Nach dem Erhalt des Access-Token kann auf die, bei der App-Autorisierung angegebenen Daten zugegriffen werden. Die Schnittstelle für den Zugriff bietet die Graph API. Die Graph API greift dabei auf den sozialen Graph von Facebook zu. Der Graph repräsentiert, die im Netzwerk vorhandenen Objekte[30] und deren Beziehungen [31] [UNB11j].
Für die virtuelle Welt braucht man Zugriff auf den Benutzeraccount und auf die Liste der Freunde des Benutzers, damit der Avatar seinen Raum und die Räume seiner Freunde besuchen kann. Ziel ist es die Daten clientseitig abzufragen. Dazu wird per RPC eine Anfrage an den Server gestellt. Als Parameter wird der bereits erlangte Access-Token übergeben. Der Server ruft nach der dem Eingang der Anfrage, wie in Listing 22 exemplarisch beschrieben, eine an die Graph API gerichtete URL auf. Der Aufruf durch `fetchUrl` in Zeile 3 ist nichts anderes als ein HTTP-Request mit einer HTTP-Response (siehe hierzu [UNB11t]).

```
1  public FacebookUserDto getCurrentUser(String accessTokenString) throws FacebookExceptionDto
       {
2    String urlString = "https://graph.facebook.com/me?" + "access_token=" + encodeUrl(
         accessTokenString);
3    String jsonString = fetchUrl(urlString);
4
5    // Throws an exception if there was an invalid access token used
6    checkForAccessTokenError(jsonString, accessTokenString);
7
8    FacebookUserDto currentUser = new Gson().fromJson(jsonString, FacebookUserDto.class);
9    return currentUser;
10 }
```

Listing 22: Abfragen der Benutzerinformationen über der Graph API

[28] Dem Benutzer kann an dieser Stelle eine Fehlermeldung angezeigt werden, dass er die App nur benutzen kann, wenn er sie autorisiert.

[29] Das Secret sollte nur auf dem Server verfügbar sein. Somit ist es an dieser Stelle unmöglich für Dritte, sich als die eigene App auszugeben.

[30] Mit Objekten sind: Benutzer, Fotos, Seiten und Veranstaltungen gemeint.

[31] Mit Beziehungen sind: Freundschaftsbeziehungen zwischen Benutzern, publizierte Inhalte und Foto-Tags gemeint.

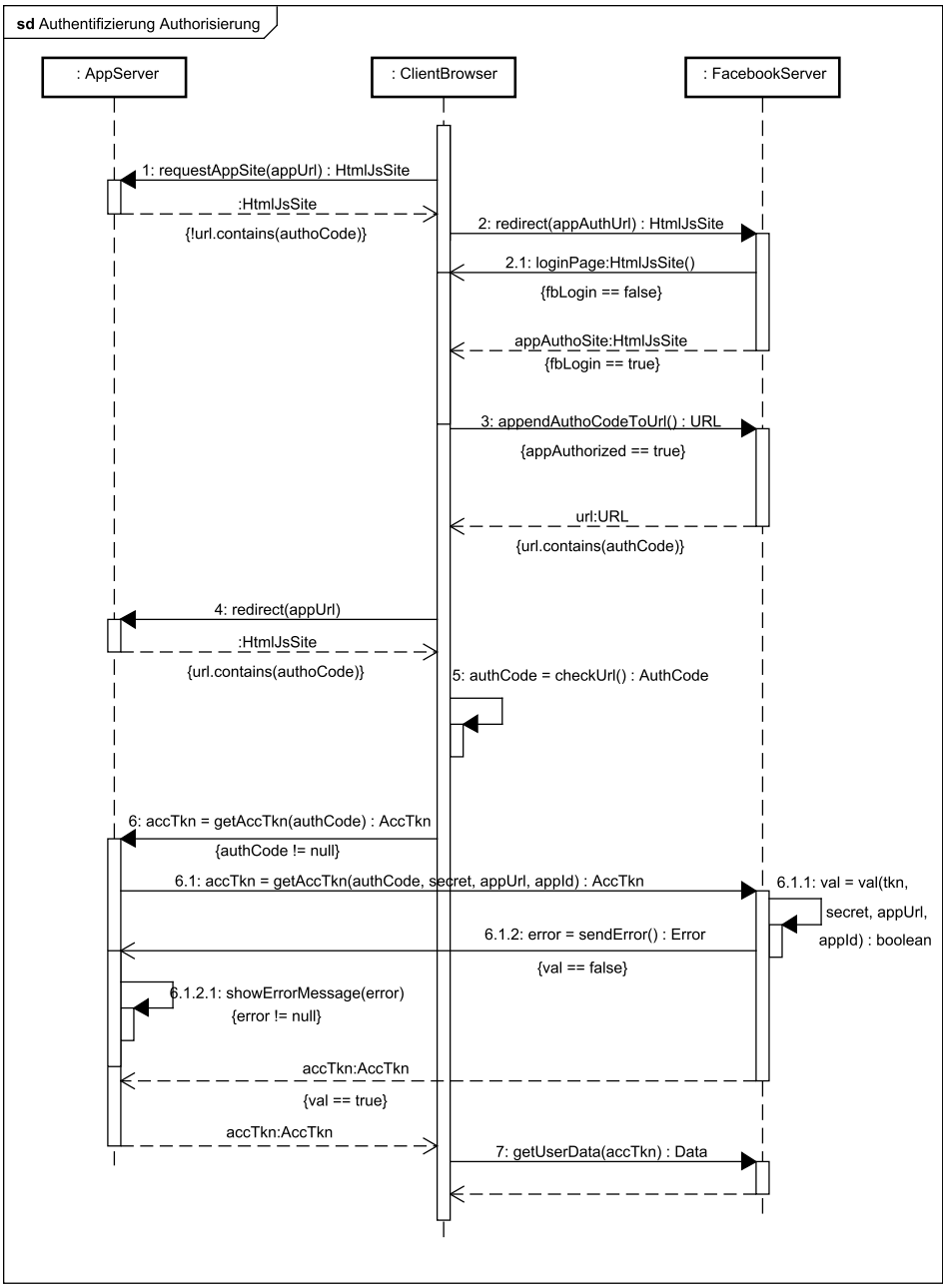


Abbildung 7: Schematischer Ablauf der Autorisierung und Authentifizierung als Sequenzdiagramm.

Die Antwort auf diese Anfrage ist ein String im JSON-Format (Listing 23 zeigt eine solche Antwort). Dieser String sollte praktischerweise in ein Java-Objekt konvertiert werden, um den Umgang zu erleichtern.

```
{
   "id": "123456789101112",
   "name": "Max Mustermann",
   "first_name": "Max",
   "last_name": "Mustermann",
   "link": "http://www.facebook.com/profile.php?id=123456789101112",
   "gender": "male",
   "email": "mustermann\u0040domain.de",
   "timezone": 2,
   "locale": "en_US",
   "verified": true,
   "updated_time": "2011-05-31T10:30:34+0000"
}
```

Listing 23: Antwort der in Listing 22 beschriebenen Anfrage

Das Parsen von JSON zu Java-Objekten wird von der Bibliothek Gson (`http://code.google.com/p/google-gson/`) übernommen. Dazu wird ein ein Java-Klasse angelegt, die die gleichen Variablenbezeichner für ihre Felder besitzt wie die JavaScript Object Notation. Wählt man andere Bezeichner, muss durch Annotationen der JSON-Variablenbezeichner angegeben werden (siehe hierzu Listing 24).

```
public class FacebookUserDto implements IsSerializable, Serializable{
  private static final long serialVersionUID = 1L;
  private String id;
  private String name;
  @SerializedName("first_name")
  private String firstName;
  @SerializedName("last_name")
  private String lastName;
  private String link;
  private String gender;
  private String email;
  private String timezone;
  private String locale;
  private boolean verified;
  @SerializedName("update_time")
  private String updateTime;

  // Default-Konstruktor, Getter und Setter sind aus Platzgründen entfernt
  }
```

Listing 24: Zielklasse zum Erstellen von JSON-Objekten.

Die Abfrage der Freunde des Benutzers läuft analog ab. Nur die URL, die auf die Graph API zugreift und die Zielklasse zum Parsens des JSON-Objekts ändert sich.

6 Beschreibung des Szenengraphs

In diesem Abschnitt wird erklärt, wie der Szenengraph entstanden ist und wie er funktioniert. Deshalb wird noch einmal kurz zusammengefasst, was ein Szenengraph ist, und welche Aufgaben er hat.
Ein Szenengraph ist ein System, das eine virtuelle Szene so strukturiert, dass sie durch die Hardware optimal abgearbeitet werden kann. Die Szene wird dabei in einer Baum aus Knoten räumlich organisiert. Gruppen-, Transformation- und Geometrieknoten sind dabei die Basis des Szenengraphs. Wird ein Knoten verändert, zum Beispiel durch eine Transformation, wirkt sich diese Änderung auf dessen Kinder aus.

6.1 Entwurf des Szenengraphs

Grundlage des Entwurfs bietet der minimale Szenengraph, bestehend aus Knoten für Gruppen, Transformationen und Geometrien [Mül06]. Die Geometrie wird dabei jeweils von einem Kern, genannt `Core` verwaltet, welche in ein `Leaf`, also einen Endknoten im Szenengraph, eingesetzt werden kann. Die Differenzierung zwischen `Core` und `Node`, beruht auf der Tatsache, dass sich die Eigenschaften dieser beiden Klassen und somit die Attribute, stark unterscheiden. So hat der `Core` die Aufgabe Geometrie zu rendern, wohingegen der `Node` oftmals nur eine Delegation der Traversierung durchführt. Szenengraphen werden deshalb im Vergleich zu mathematischen Graphen, als heterogene Graphen bezeichnet [Rei02].
Als weitere Rahmenbedingung muss angenommen werden, dass sich das vom Szenengraph genutzte GWT-WebGl-Binding, jederzeit ändern oder als zu fehlerhaft herausstellen kann. Deshalb ist eine Abstraktionsschicht, genannt „Rendering Abstraction Layer“, kurz RAL, über den Klassen, die die Methoden des WebGl-Bindings aufrufen, sinnvoll [SNR+11]. Ursprünglich ist dieser Entwurf dazu gedacht verschiedene 3D-Schnittstellen in ein und dem selben Szenengraph zu verwenden. So kann der RAL an nur einer Stelle des Programms entscheiden, welche 3D-Schnittstelle verwendet wird. Für den entwickelten Szenengraphen ermöglicht der RAL eine Implementierung verschiedener GWT-WebGL-Bindings, wodurch auf die genannten Probleme flexibibel reagiert werden kann. Umgesetzt wird der RAL durch das Abstract Factory Pattern nach der Gang of Four [GHJV95] (siehe Abb. 8).

In Abbildung 9 wird ein vereinfachter Entwurfsausschnitt des Szenengraphs für diese Umsetzung gezeigt [32]. Wie man sieht, werden Klassen, die keinen WebGl-Code enthalten, von der `AbstractSgFactory` erstellt. Im Diagramm sind dies beispielsweise die Klassen `Leaf` und `TranslationNode`. Dahingegen werden Klassen, die WebGl-Code und somit abhängig von einem WebGL-Binding sind, in einer konkreten Fabrik[33] instanziert, jedoch zuvor als abstrakte Methoden in der `AbstractSgFactory` deklariert.

6.2 Funktionsweise des Szenengraphs

Die Funktionsweise des entwickelten Szenengraphs unterscheidet sich im Vergleich zu einem normalen OpenGL-Szenengraph hauptsächlich davon, dass sich OpenGL und WebGL, wie in Abschnitt 2.1.5 schon beschrieben, unterscheiden. So fehlt in der WebGL-Spezifikation ein Matrix-Stack, welcher im Normalfall im Szenengraph die Aufgabe hat, per push und pop die Transformationsmatrix zu verwalten. Eine mögliche Lösung wäre es, den Matrix-Stack nachzuimplementieren. Intuitiver ist

[32] Das Diagramm ist aus Platzgründen nicht vollständig.

[33] Die konkreten Fabriken sind hierbei `G3dFactory` und `GwtGlFactory` (`GwtGlFactory` ist hierbei nicht implementiert und dient nur der Veranschaulichung).

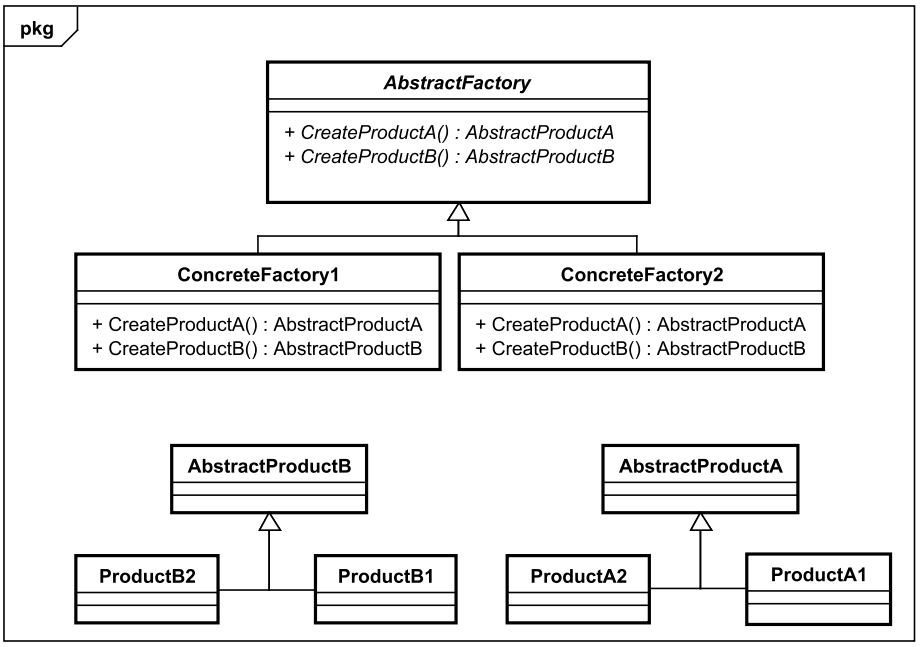


Abbildung 8: Das Abstract Factory Pattern. Dabei ist zu beachten, dass `ConcreteFactory1` nur Instanzen von `ProductA1` und `ProductB1` erstellt. Analog verhält sich `ConcreteFactory2`.

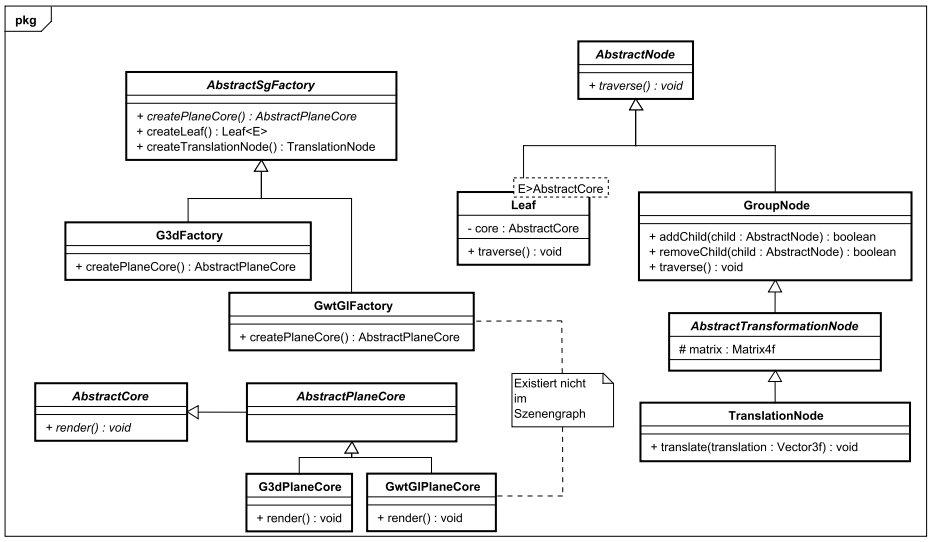


Abbildung 9: Ein vereinfachter Entwurfsausschnitt des Szenengraphs basierend auf dem Abstract Factory Pattern.

es jedoch, in jedem Knoten die aktuelle Transformationsmatrix[34] zu speichern. Transformationsknoten müssen dabei zusätzlich, die eigene Transformationsmatrix und die Transformationsmatrix des Vorgängerknotens speichern. Diese beiden zusätzlichen Matrizen werden dazu benötigt, um nach einer Aktualisierung der eigenen Transformationsmatrix, die aktuelle Transformationsmatrix neu zu berechnen und sie an ihre Kinder weiterzugeben. Somit ist der Matrix-Stack implizit in den Knoten des Szenengraphs gespeichert. Dabei ist eine Trennung zwischen Transformationsmatrix und Look-At-Matrix sinnvoll, da sich die Kamera unterschiedlich oft im Vergleich zu den Transformation der Knoten ändert. Würde man nur eine Modelviewmatrix, also das Produkt aus Look-At-Matrix und Transformationsmatrix in jedem Knoten verwalten, so müsste diese aktualisiert werden, sobald sich die Kamera oder in dem betreffenden Knoten die Transformation ändert. Außerdem kann mit der Trennung der beiden Matrizen ein vereinfachter Billboardeffekt umgesetzt werden. Dazu muss der für die Rotation verantwortlichen Teil in der Look-At-Matrix auf die Identitätsmatrix gesetzt werden. Dieser Teil befindet sich im oberen linken 3x3-Block der Look-At-Matrix [Fer11].

6.3 3D-Format des Avatars

Das 3D-Format für die Darstellung des Avatars unterliegt hauptsächlich technischen Anforderungen. Da mehrere Avatarmodelle dem Benutzer zur Verfügung gestellt werden sollen, macht es Sinn die Modelle auf dem Server, genauer gesagt im Blobstore zu speichern. Damit wird ermöglicht neue Modelle dynamisch hinzuzufügen bzw. abzurufen. Um eine problemlose Übertragung sicherzustellen, eignet sich daher ein textbasiertes Format, da wie in Abschnitt 5.1.2 erwähnt, letztendlich Strings übertragen werden. Des Weiteren sollte das Format von gängigen Modellierungstools unterstützt werden. Collada als XML-basierendes Format, das auch Animationen unterstützt, wäre ein geeignetes Format. Allerdings ist es mit einigen Problemen verbunden dieses populäre Format zu laden. Ein vernünftiges Auslesen einer Collada-Datei verlangt Bibliotheken, die das Schema von Collada in Klassen umwandeln und für eine Datei, die spezifischen Klassen mit ihren Werten instanziiert. Solche Bibliotheken[35] gibt es zwar für Java, jedoch besitzen diese viele Abhängigkeiten zu weiteren Bibliotheken. Clientseitiges Parsen ist aufgrund der Inkompatibilität dieser Bibliotheken mit dem GWT nicht möglich [36]. Serverseitig können diese Bibliotheken zwar eingesetzt werden, da Java auf der GAE nativ ausgeführt wird, jedoch ist, zumindest bei der Verwendung von JAXB, mit langen Ladezeiten und hoher CPU-Last zu rechnen. Dies hätte wiederum lange Antwortzeiten der App und hohe Serverlast zur Folge.
Ein 3D-Format zu dem auch neben diesen Anforderungen ein Loader leicht zu implementieren ist, ist das Md2-Format. Dieses Format unterstützt Animationen in Form von Keyframes. Keyframeanimationen beruhen darauf, dass von einer Animation nur eine Auswahl von Bildern, die sogenannten Keyframes abgespeichert werden. Zwischen den Keyframes kann, mittels Interpolation die Animation wiederhergestellt werden. Im Md2-Format werden für jedes Keyframe nur die Positionskoordinaten und die Normalen zu den Eckpunkten abgespeichert. Die Texturkoordinaten ändern sich während dem Verlauf der Animation nicht. Im Gegensatz zu Collada ist das Md2-Format ein binäres Format. Somit muss entweder eine Lösung gefunden werden binäre Daten effizient vom Server auf den Client zu übertragen, oder die entsprechenden Dateien müssen auf dem Server in ein textbasierendes Format gebracht werden. Zweitere Lösung scheitert daran, dass sich die Datengröße vervielfachen kann. Die Ursache hierfür liegt im Format, das die Float-Werte logischerweise binär abspeichert, so dass diese

[34] Mit der aktuellen Transformationsmatrix eines Knotens ist hierbei, die Multiplikation der Transformationsmatrizen der Elternknoten bis zu dem aktuellen Knoten, gemeint.

[35] JiBx, JAXB, CastorXML und XML Spy

[36] Wenn das Schema „misachtet“ wird und Normalen, Positions und Texturkoordinaten mit „Get-Node-By-Name-Befehlen“ ausgelesen werden, ist das Laden möglich. Entspricht aber nicht dem Sinn von Collada.

zu einem String umgewandelt werden müssen. Nimmt man z.B. einen Float-Wert mit 5 Stellen, so ist er als String 5 Zeichen lang und folglich auch 5 Byte groß. Ist der Float-Wert binär abgespeichert, belegt er nur 4 Byte. Zusätzlich besitzt das Md2-Format auch eine Kompression der Normalen, so dass es bei den Normalen zu einem noch größeren Speicherverbrauch kommt.
Es macht also Sinn, das Md2-Format direkt zu übertragen. Da ein Byte-Array, das übertragen wird, auch serialisert wird, kommt es zu einem ähnlichen Problem wie bei der Umwandlung der Float-Werte zu Strings. Jeder Byte des Byte-Arrays belegt natürlich nur ein Byte. Als String jedoch kann ein Byte bis zu drei Zeichen belegen, d.h. die Größe eines Byte-Arrays kann sich im schlimmsten Fall verdreifachen. Eine Lösung liegt in der Codierung in einen Base64-String. Die Base64-Codierung ist eine Umwandlung von Bytes in einen String mit einer vertretbaren Vergrößerung des Datenstroms. Das Laden des Modells kann dadurch auf Clientseite realisiert werden.

6.3.1 Laden des Md2-Formats

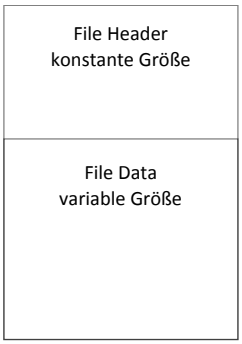


Abbildung 10: Aufbau einer Md2-Datei

Die Funktionen zum Laden des Md2-Formats basieren auf dem C++ Tutorial von [Hen11]. Durch die in JavaScript nicht vorhandenen Speicherzuweisungen mittels Pointern und den Unterschied zwischen OpenGL und WebGL, unterscheiden sich die Funktionen zum Laden und Rendern eines Md2-Models von denen in C++. Im Folgenden wird ein detaillierter Einblick über diese Probleme gegeben. Der Aufbau einer Md2-Datei ist in Abbildung 10 zu sehen. Sie besteht aus einem Header mit einer festen Größe und aus einem Teil mit variabler Größe. Der wichtigste Bestandteil des Headers sind die Offsets zu den variablen Daten in der Datei. Aus diesem Grund muss der Header als erstes geladen werden. Neben den Offsets beinhaltet der Header weitere Informationen, welche in Tabelle 1 im Header-Teil zu sehen sind. Der Header beginnt mit dem ersten Byte der Datei und ist 86 Bytes groß. Sinn ergeben diese Bytes nur, wenn man sie als vier Byte große Integers interpretiert.

An dieser Stelle kommt es zum ersten Problem. In C++ können die 86 Bytes des Headers durch die Benutzung von Pointern als Integer-Array repräsentiert und dadurch auch ausgelesen werden. In JavaScript gibt es jedoch keine Pointer. Eine Lösung ist der Gebrauch von Shiftingoperatoren und Operatoren für die bitweise Verundung, welche in JavaScript vorhanden sind. Diese Operatoren werden ebenso in Java und somit auch durch das GWT unterstützt. Mit den beiden Funktionen aus Listing 25 lassen sich einzelne Integer nacheinander aus der Datei auslesen. Das Array `data` ist dabei die komplette Md2-Datei als Bytearray und `dataPosition` gibt die momentane Position, also die Adresse in dem Array an. Wird `getNextInt` 17 mal aufgerufen, ist der komplette Header ausgelesen.

Nachdem der Header ausgelesen ist, kann der variable Teil der Datei folgen. Tabelle 1 zeigt ebenso die Inhalte des variablen Teils. Die Offsets zu den dort beschriebenen Daten variieren zwar, sind aber aus dem Header bekannt. Zum Rendern des Models werden für jedes Frame die Texturkoordinaten, die Normalen, die Positionen und der Zusammenhang[37] wie sie gerendert werden, benötigt. Die Struktur für jedes Keyframe (`KfmStruct`), beinhaltet lediglich Positionen und Normalen zu jedem Eckpunkt. Die Texturkoordinaten sind hingegen separat in den `TexStructs` abgespeichert, da diese sich nicht im Verlauf der Animation ändern. Der Zusammenhang wie sie gerendert steckt in den „OpenGL-

[37] Damit ist der Begin-Mode gemeint. Für Geometrie in WebGL also `TRIANGLES`, `TRIANGLE_STRIP` oder `TRIANGLE_FAN`.

Header			
Offset	**Type**	**Anzahl der Elemente**	**Beschreibung**
0	`int`	1	Muss „IDP2“ entsprechen
4	`int`	1	Modellversion
8	`int`	1	Texturbreite
12	`int`	1	Texturhöhe
16	`int`	1	Größe eines Keyframes in Byte
20	`int`	1	Anzahl der Texturen
24	`int`	1	Anzahl der Eckpunkte des Modells
28	`int`	1	Anzahl der Texturkoordinaten
32	`int`	1	Anzahl der Dreiecke
36	`int`	1	Anzahl der OpenGL-Befehle
40	`int`	1	Anzahl der Keyframes
44 *1	`int`	1	Offset zu den Texturnamen
48 *2	`int`	1	Offset zu den Texturkoordinaten
52 *3	`int`	1	Offset zu den Dreiecks-Daten
56 *4	`int`	1	Offset zu den Keyframe-Daten
60 *5	`int`	1	Offset zu den OpenGL-Befehlen
64	`int`	1	Offset vom Dateiende
Teil mit variabler Größe			
siehe *1	`unsigned char`	65 * Anzahl der Texturen	Texturnamen
siehe *2	`TexStruct`	Anzahl der Texturkoordinaten	Texturkoordinaten
siehe *3	`TnglStruct`	Anzahl der Dreiecke	Dreiecksindizes
siehe *4	`KfrmStruct`	Anzahl der Keyframes	Daten der Keyframes
siehe *5	`int`	Anzahl der OpenGL-Befehle	OpenGL-Befehle

Tabelle 1: Inhalt der Md2-Datei.

```
1  private int getNextInt() {
2      byte[] tempBuffer = new byte [4];
3      for (int i = 0; i < tempBuffer.length; i++) {
4        tempBuffer[i] = data[dataPosition + i];
5      }
6      dataPosition += 4;
7      return convertByteArrayToInt(tempBuffer);
8  }
9
10 public int convertByteArrayToInt(byte[] array) throws IllegalArgumentException {
11     if (array.length != 4)
12       throw new IllegalArgumentException("Array length must be 4 bytes!");
13
14     int result = (0xFF & array[3]) << 24;
15     result |= (0xFF & array[2]) << 16;
16     result |= (0xFF & array[1]) << 8;
17     result |= (0xFF & array[0]);
18
19     return result;
20 }
```

Listing 25: Funktionen zum Auslesen und Konvertieren von vier Bytes zu einem Integer.

Befehlen“[38] (*5 aus Tabelle 1) der Md2-Datei.
Zu Beginn werden die Keyframes und damit die Positionen und Normalen ausgelesen. Wie man in Tabelle 2 sieht, untergliedert sich jedes Keyframe wieder in einen statischen Header und einen Teil mit variabler Größe.

Header		
Type	**Anzahl der Elemente**	**Beschreibung**
`float`	3	Skalierungsfaktor
`float`	3	Translationswert
`char`	16	Keyframename
Teil mit variabler Größe		
`VertexStruct`	Anzahl der Eckpunkte des Modells	komprimierte Position und Normale als 4 Byte große Struktur

Tabelle 2: Aufbau eines Keyframes.

Um die Position aus jeder der 4 Byte großen `VertexStructs` (siehe Tabelle 3) zu dekomprimieren, muss der 3D-Skalierungsfaktor aus dem Header (siehe Tabelle 2) komponentenweise mit den jeweils ein Byte großen Positionskoordinaten multipliziert werden. Danach wird der 3D-Translationswert ebenso komponentenweise an diese skalierte Positionskoordinate addiert. Hier ergeben sich abermals Konvertierungsprobleme. Zum einen von `byte` zu `float`, um den Skalierungsfaktor und den Translationswert auszulesen. Zum anderen von `byte` zu `unsigned char`, um den Normalenindex zu erhalten. Aufgrund von Java muss bei der zweiten Konvertierung sogar von `byte` zu `int` konvertiert werden, da Java den primitiven Datentyp `unsigned char` nicht unterstützt. Diese Konvertierung lässt sich jedoch, durch die Nutzung bitweiser Operatoren, mit der Funktion aus Listing 26 lösen.

Type	**Anzahl der Elemente**	**Beschreibung**
`unsigned char`	1	XKoordinate
`unsigned char`	1	YKoordindate
`unsigned char`	1	ZKoordinate
`unsigned char`	1	Normalenindex

Tabelle 3: Aufbau der `VertexStruct`.

```
1 public static int convertByteToUbyteAsInt(byte value){
2   int result = (0xFF & value);
3   return result;
4 }
```

Listing 26: Funktionen zum Konvertieren von einem `byte` zu einem `int`. Der Wert dieses `ints` entspricht danach dem Wert, den ein Byte als `unsigned char` in c hätte.

Das Problem vier Bytes zu einem `float` zu konvertieren erweist sich als schwerer, da die standard Java Funktion `Float.intBitsToFloat` vom GWT nicht unterstützt wird. Eine mögliche Lösung ist die Nutzung von `ArrayBufferViews`[39]. Sie beinhaltet eine Referenz auf einen `ArrayBuffer`, die je nach gewählten Typ der `ArrayBufferView`[40] dementsprechend interpretiert werden kann. Listing 27 zeigt die Funktion, die vier Bytes zu einem `float` konvertieren. Da es nicht

[38] Wörtliche Übersetzung, der Benennung aus [Hen11]

[39] Die `ArrayBufferViews` sind nur in JavaScript vorhanden. Ein GWT-Binding zu diesen ist in gwt-g3d (siehe Abschnitt 5.1.1) enthalten.

[40] Mögliche Typen sind: `Int8Array`, `Uint8Array`, `Int16Array`, `Uint16Array`, `Int32Array`, `Uint32Array`, `Float32Array`, `Float64Array`

möglich ist ein Bytearray als Buffer einer `Int8Array`- oder `Uint8Array`-Buffer-View zu setzen, sondern nur Integerarrays oder einen einzelnen `int`, muss das Array zuerst in einen `int` konvertiert werden (siehe Funktion aus Listing 25). Dieser `int` wird als Buffer einer `Int32Array`-View gesetzt, die sich den selben `ArrayBuffer` mit einer `Float32Array`-Buffer-View teilt. Die `Float32Array`-Buffer-View ist nun in der Lage den gesetzten `int` als `float` zu interpretieren.

```
1  public float convertByteArrayToFloat(byte[] array) throws IllegalArgumentException {
2    if (array.length != 4)
3      throw new IllegalArgumentException("Array length must be 4 bytes!");
4
5    int intValue = convertByteArrayToInt(array);
6    Int32Array int32Array = Int32Array.create(1);
7    int32Array.set(0, intValue);
8    Float32Array float32Array = Float32Array.create(int32Array.getBuffer(), 0, 1);
9    return float32Array.get(0);
10 }
```

Listing 27: Funktionen zum Konvertieren von vier Bytes zu einem `float`.

Mittels der beschriebenen Funktionen ist es möglich den Algorithmus 1 umzusetzen. Dieser zeigt im Ganzen, wie alle Normalen und Positionen für jedes Keyframe ausgelesen werden. Bisher fehlen noch die Texturkoodinaten, die zum korrekten Rendern gebraucht werden.

Algorithm 1 Auslesen der Normalen und Positionen aus den Keyframes

for $i = 0 \rightarrow numberOfKeyFrames$ **do**
  $keyFrame \leftarrow getKeyFrame(md2Data)$
  $scaleFactor3f \leftarrow getScaleFactor(keyFrame)$
  $translationValue3f \leftarrow getTranslationValue(keyFrame)$
  $keyFrameName \leftarrow getKeyFrameName(keyFrame)$

  for $j = 0 \rightarrow numberOfVerices$ **do**
    $position3f \leftarrow getPosition(keyFrame)$
    $position3f \leftarrow position3f * scaleFactor3f + translationValue3f$
    $normalIndex \leftarrow getNormalIndex(keyFrame)$
    $normal3f \leftarrow getNormal(normalIndex)$

    $positions[i][j] \leftarrow position3f$
    $normals[i][j] \leftarrow normal3f$
    $j \leftarrow j + 1$
  end for

  $saveName(keyFrameName)$
  $i \leftarrow i + 1$
end for

Um die Texturkoordinaten auszulesen, können entweder alle `TexStructs` (siehe Tabelle 1) auslesen werden oder die OpenGL-Befehle (ebenda) der Md2-Datei ausgelesen werden. Die Art, wie sie dort abgespeichert sind, hat jedoch Einfluss auf das Rendering. Werden die Texturkoordinaten aus der `TextStruct` ausgelesen kann das Modell nur mittels der Dreiecksindizes(`TnglStruct`) gerendert werden. Die Dreiecksindizes beinhalten sowohl einen Index zu den ausgelesenen Positionen,

der auch gleichzeitig für die Normalen gilt, als auch zu den Texturkoordinaten. Mit diesem Ansatz ist es nur möglich das Modell mit `TRIANGLES` zu rendern. Der zweite Ansatz, der auch in dieser Arbeit gewählt wurde, basiert auf dem Rendern mittels `TRIANGLE_STRIP` und `TRIANGLE_FAN`, was einen Performancevorteil verspricht. Dazu müssen als erstes die „OpenGL-Befehle“ aus der Datei geladen werden. Diese sind, wie der Header als Integer-Array zu interpretieren und werden analog geladen. Die Werte des gesamten Arrays sind semantisch als eine Liste von „Rendergruppen“ zu verstehen. Die erste Zahl jeder Gruppe gibt die Anzahl der Eckpunkte, die zur Gruppe gehören an. Das Vorzeichen der ersten Zahl entscheidet zusätzlich, ob die Gruppe mittels `TRIANGLE_STRIP` oder `TRIANGLE_FAN` gerendert wird. Die restlichen Werte der Gruppe, deren Größe jetzt bekannt ist, geben für jeden Eckpunkt nacheinander die Texturkoordinaten und den Index der ausgelesenen Normalen bzw. Position an. Sind alle Eckpunktdaten ausgelesen, beginnt eine neue Gruppe, die nach dem selben Schema ausgelesen wird. Algorithmus 2 veranschaulicht das Auslesen.

Algorithm 2 Auslesen der Texturkoordinaten

$i \leftarrow 0$
while $openGlCommands[i] \neq 0$ **do**
  $group \leftarrow getNewGroup()$
  $numberOfVerticesPerGroup \leftarrow openGlCommands[i++]$

  if $numberOfVerticesPerGroup < 0$ **then**
    $group.mode \leftarrow TRIANGLE\_FAN$
    $numberOfVerticesPerGroup \leftarrow -1 * numberOfVerticesPerGroup$
  else
    $group.mode \leftarrow TRIANGLE\_STRIP$
  end if
  $group.size \leftarrow numberOfVerticesPerGroup$

  for $j = 0 \rightarrow numberOfVerticesPerGroup$ **do**
    $group.texCoordS[j] \leftarrow toFloat(openGlCommands[i++])$
    $group.texCoordT[j] \leftarrow toFloat(openGlCommands[i++])$
    $group.index[j] \leftarrow openGlCommands[i++]$
  end for
  $saveGroup(group)$

end while

6.3.2 Rendern des Md2-Modells

Aus den geladenen Texturkoordinaten, dem Rendermode und dem Eckpunktindezes muss das Modell gerendert werden. In WebGL stehen dazu die Funktionen `drawElements` und `drawArrays` zur Verfügung. Aufgrund der ausgelesenen Indizes zu den Positionen und den Normalen eignet sich die `drawArrays` Funktion. Dabei gibt es das Problem, dass ein Eckpunkt mit der gleichen Position und Normale unterschiedliche Texturkoordinaten haben kann. Abbildung 11 zeigt dieses Problem. Das dargestellte Mesh kann nur durch Umwege mittels `drawArrays` gerendert werden.

Eine mögliche Lösung wäre es die Normalen- und Positionskoordinaten von Eckpunkten, die mehrfache Texturkoordinaten besitzen, im Positions- und Normalenarray ebenfalls mehrfach abzu-

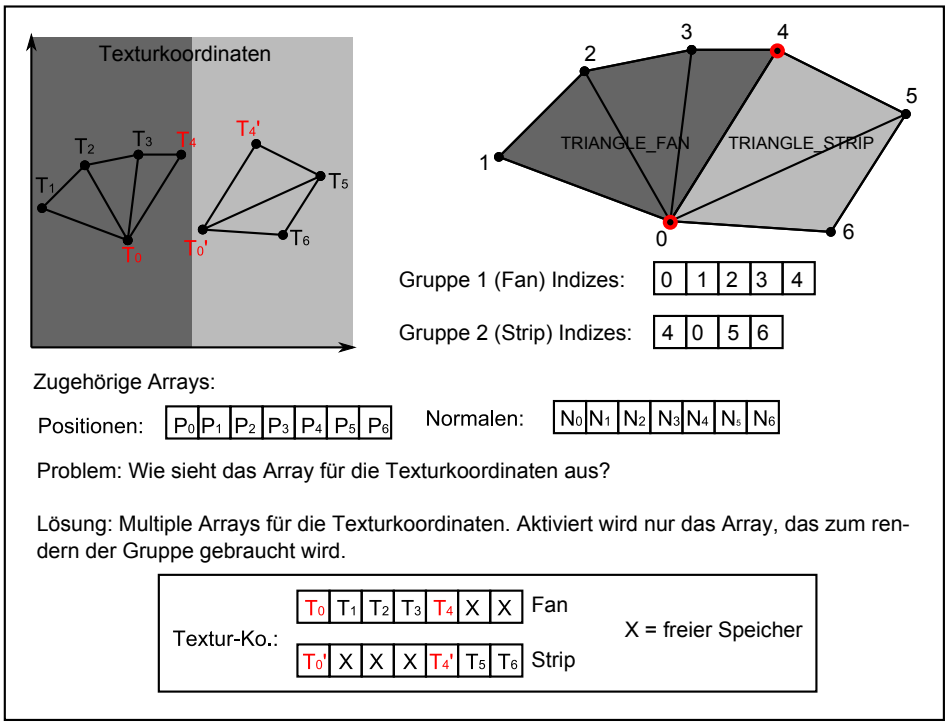


Abbildung 11: Ein Md2-Mesh mit doppelten Texturkoordinaten.

legen und gleichzeitig dem Indexarray neue Indizes hinzuzufügen. Dieser Ansatz müsste für jedes Keyframe wiederholt werden. Bedenkt man dabei, dass ein Md2-Modell bis zu 200 Keyframes besitzen kann, wäre der Aufwand und der dementsprechend Speicherverbrauch hoch. Besser ist es für jede Gruppe ein Texturkoordinatenarray anzulegen (wie auch in Abb. 11 zu sehen). Algorithmus 3 zeigt, wie die Texturkoordinaten aus den Gruppen in Arrays abgelegt werden müssen, damit die Indices für jeden Eckpunkt auf die richtige Texturkoordinate, Normale und Position zeigen.

Algorithm 3 Auslesen der Normalen und Positionskoordinaten aus den Keyframes

for $i = 0 \rightarrow groups.size$ **do**
  $texturCoordinates \leftarrow getNewTexCoord(numberOfModelVertices * 2)$
  $offset \leftarrow groups[i].index * 2$
  $texturCoordinates[offset] \leftarrow groups[i].texCoordS$
  $texturCoordinates[offset + 1] \leftarrow groups[i].texCoordT$
  $saveGroup(texturCoordinates)$
end for

Listing 28 zeigt, wie mit den gewonnenen Daten das Md2-Modell gerendert wird. Dabei wird auch die Animation des Md2-Formats einbezogen. Diese ist erst durch die Interpolation zwischen zwei Keyframes möglich ist. Dazu müssen die Positionen und Normalen von zwei Keyframes an den Shader übergeben werden. Zusätzlich wird ein Interpolationsfaktor übergeben, so dass im Shader anhand dieses Faktors linear, jeweils zwischen den Normalen und Positionen, eines Quell- und eines Ziel-Keyframes, interpoliert werden kann (siehe Shader in Listing 29).

Der Interpolationsfaktor liegt dabei im Intervall von Null bis Eins und wird mit jedem Bild, das gerendert wird erhöht. Entspricht der Wert des Faktors Null, wird das Quell-Keyframe gerendert, ent-

```
1
2  public void render() {
3    // Set interpolation Factor
4    animatedGeometryShader.bind();
5    animatedGeometryShader.setInterpolationFactor(animationManager.getInterpolationValue(), 0)
         ;
6
7    // Set texture
8    G3dGLContext.GL.activeTexture(TextureUnit.TEXTURE0);
9    G3dGLContext.GL.bindTexture(TextureTarget.TEXTURE_2D, texture);
10   animatedGeometryShader.setTexture(TextureUnit.TEXTURE0);
11
12   // Enable normal, vertex and texture attribute array
13   G3dGLContext.GL.enableVertexAttribArray(positionAttributeKeyFrame1);
14   G3dGLContext.GL.enableVertexAttribArray(normalAttributeKeyFrame1);
15   G3dGLContext.GL.enableVertexAttribArray(positionAttributeKeyFrame2);
16   G3dGLContext.GL.enableVertexAttribArray(normalAttributeKeyFrame2);
17   G3dGLContext.GL.enableVertexAttribArray(textureAttribute);
18
19   // Map normals and positions of source and destination key frames to the attribute arrays
20   mapBuffer(vertexBufferArray[animationManager.getSourceKeyFrameIndex()],
         positionAttributeKeyFrame1, 3);
21   mapBuffer(normalBufferArray[animationManager.getSourceKeyFrameIndex()],
         normalAttributeKeyFrame1, 3);
22   mapBuffer(vertexBufferArray[animationManager.getDestinationKeyFrameIndex()],
         positionAttributeKeyFrame2, 3);
23   mapBuffer(normalBufferArray[animationManager.getDestinationKeyFrameIndex()],
         normalAttributeKeyFrame2, 3);
24
25   // Loop over each rendering group
26   for (int i = 0; i < renderingGroupMode.length; i++) {
27     mapBuffer(texCoordBufferArray[i], textureAttribute, 2);
28
29     G3dGLContext.GL.bindBuffer(BufferTarget.ELEMENT_ARRAY_BUFFER, indexBufferArray[i]);
30     G3dGLContext.GL.drawElements(renderingGroupMode[i], renderingGroupSize[i],
           DrawElementsType.UNSIGNED_SHORT, 0);
31   }
32
33   // Restore state
34   G3dGLContext.GL.disableVertexAttribArray(positionAttributeKeyFrame1);
35   G3dGLContext.GL.disableVertexAttribArray(normalAttributeKeyFrame1);
36   G3dGLContext.GL.disableVertexAttribArray(positionAttributeKeyFrame2);
37   G3dGLContext.GL.disableVertexAttribArray(normalAttributeKeyFrame2);
38   G3dGLContext.GL.disableVertexAttribArray(textureAttribute);
39
40   // Calculate the animation values for the next frame
41   animationManager.calculateNextFrame();
42 }
43
44 private void mapBuffer(WebGLBuffer webGLBuffer, int attribute, int itemSize) {
45   G3dGLContext.GL.bindBuffer(BufferTarget.ARRAY_BUFFER, webGLBuffer);
46   G3dGLContext.GL.vertexAttribPointer(attribute, itemSize, DataType.FLOAT, false, 0, 0);
47 }
```

Listing 28: Rendern des Md2-Modells.

```
uniform mat4 uModelViewMatrix;
uniform mat4 uProjectionMatrix;
uniform mat3 uNormalMatrix;
uniform vec3 uLightPosition;
uniform vec2 uInterpolationValues;

attribute vec3 aPositionKeyFrame1;
attribute vec3 aPositionKeyFrame2;
attribute vec2 aTexCoord;
attribute vec3 aNormalKeyFrame1;
attribute vec3 aNormalKeyFrame2;

varying vec3 vNormal, vLightDir;
varying vec2 vTexCoord;

vec3 interpolate(vec3 vector1, vec3 vector2, float value){
 return vec3(vector1 + value * (vector2 - vector1));
}

void main() {
 // Interpolate the position
 vec3 interpolatedPosition = interpolate(aPositionKeyFrame1, aPositionKeyFrame2,
     uInterpolationValues.x);
 vec4 aPosition = vec4(interpolatedPosition,1);

 // Interpolate the normal position
 vec3 interpolatedNormal = interpolate(aNormalKeyFrame1, aNormalKeyFrame2,
     uInterpolationValues.x);
 vec3 aNormal = interpolatedNormal;

 vec4 worldPos = uModelViewMatrix * aPosition;
 gl_Position = uProjectionMatrix * worldPos;
 vNormal = uNormalMatrix * aNormal;
 vLightDir = uLightPosition - worldPos.xyz;

 vTexCoord = aTexCoord;
}
```

Listing 29: Vertex-Shader des Md2-Modells.

spricht er Eins, wird das Ziel-Keyframe gerendert. Die Werte dazwischen geben somit an, wie sehr das interpolierte Bild dem Quell- bzw. Ziel-keyframe ähneln wird. Sobald der Faktor Eins überschreitet, wird das Ziel-Keyframe zum Quell-Keyframe und ein neues Ziel-Keyframe muss gesetzt werden. Eine komplette Animation besteht daher aus mindestens zwei oder mehr Keyframes. Im Szenengraph überwacht ein Animationsmanager (zu sehen in Listing 28, Zeile 20-23) die Werte für die momentane Animation und kalkuliert daraus Quell- und Ziel-Keyframe, sowie den Interpolationsfaktor.
Damit die Animation über einen beliebigen Zeitraum abgespielt werden kann, wird sie wie in Abb. 12 zu sehen in einer Schleife durchlaufen.

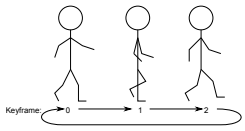


Abbildung 12: Schleife der Md2-Animation.

Im Szenengraph werden neben dem Core, der das beschriebene Rendering des Avatars übernimmt, weitere Knoten für die Translation, die Rotation, die Anzeige der Bewegungstrajektorie, die Sprechblase und den Namen des Avatars benötigt. Diese verschiedenen Knoten werden in einem Teilbaum zusammengefasst, dessen Einstiegspunkt ein Gruppenknoten bildet. Die Schnittstelle zu den verschiedenen Funktionen der einzelnen Knoten bildet wiederum ein extra Avatarknoten, welcher auch letztendlich an den Szenengraphenbaum angehangen werden kann (siehe Abbildung 13).

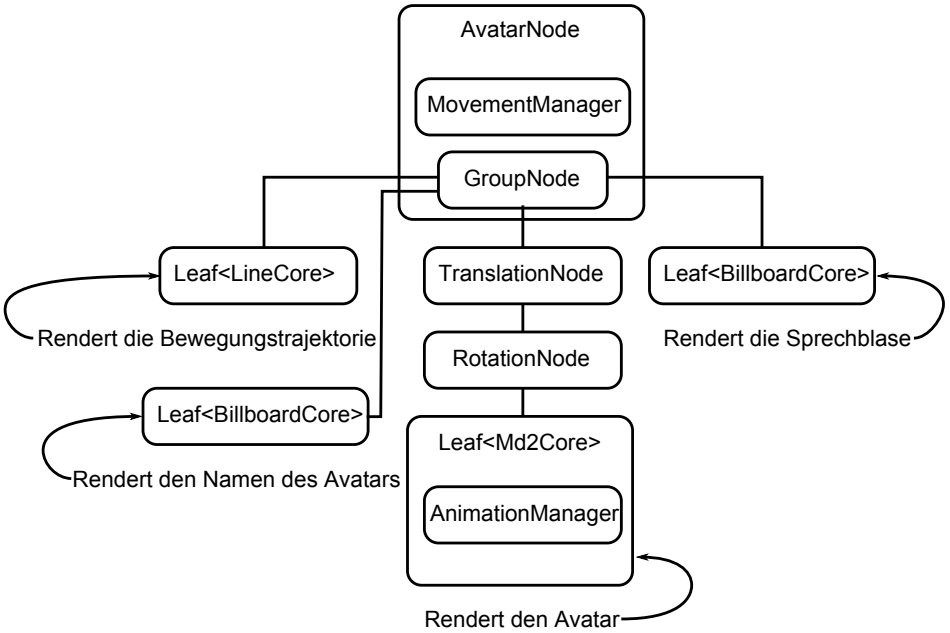


Abbildung 13: Aufbau des Avatarknotens.

6.4 Statische Modelle

Das Einbinden statischer Modelle in den Szenengraph, das den Benutzern das Einrichten ihres Raums ermöglichen soll, wird durch das Obj-Format ermöglicht. Dieses Format ist unter Modellierungstools weit verbreitet, einfach zu laden und textbasierend. Eine Konvertierung von binären Daten ist daher nicht nötig.

6.4.1 Obj-Loader

Das Laden und Rendern eines Obj-Modells ist vergleichsweise einfach. Das Format beschreibt zeilenweise Positionen, Flächen und optional Texturkoordinaten sowie Normalen. Listing 30 zeigt eine quadratische Fläche, die im Obj-Format abgespeichert ist. Zum Laden des Modells muss die Datei nur zeilenweise eingelesen werden. Die erste Zeichenkette jeder Zeile beschreibt, ob die nachfolgenden Zahlen eine Position (`v`), eine Texturkoordinate (`vt`), eine Normale (`vn`) oder eine Fläche (`f`) beschreiben. Die Fläche wird durch Indizes auf die Positionen beschrieben. Optional können Inidizes auf Texturkoordinaten und Normalen angegeben werden. Werden die Inidizes auf die Normalen weggelassen, können sie aus dem Kreuzprodukt der Richtungsvektoren, der Fläche berechnet werden. In Listing 30 wird genau eine Fläche aus vier Eckpunkten beschrieben. Diese Fläche beinhaltet nur Indizes auf die Position und Texturkoordinate, welche jeweils durch ein „/“ getrennt werden. Die einzige Schwierigkeit besteht nur darin, die Fläche in Dreiecke zu konvertieren, da es in WebGL keine Rendermode gibt, der Vierecke rendern kann.

Listing 30: Quadratische Fläche im Obj-Format.

```
1
2  v 13.023772 -13.023773 -13.023772
3  v 13.023772 -13.023773 13.023772
4  v -13.023775 -13.023773 13.023771
5  v -13.023768 -13.023773 -13.023778
6
7  vt 0.726562 0.031250
8  vt 1.000000 0.000000
9  vt 0.992188 0.218750
10 vt 0.742188 0.246094
11
12 f 1/1 2/2 3/3 4/4
```

6.5 Picking

Picking, ein fester Bestandteil vieler Szenengraphen, wird gebraucht, um ein 3D-Objekt der Welt auszuwählen. Es kann auf unterschiedliche Weise implementiert werden. Es ist z.B möglich einen Strahl, ausgehend von der ausgewählten 2D-Koordinate, durch die Welt zu schicken und dessen Schnittpunkte mit den getroffenen Objekten zu analysieren.
Bei der Implementierung des Pickings auf Basis von WebGL sollte darauf geachtet werden, dass rechenintensive Aufgaben, wenn überhaupt vom Shader bearbeitet werden, da dieser auch in WebGL nativ ausgeführt wird und somit die nötige Performance mitbringt. Color-Picking ist deshalb eine gute Wahl.
In der Anwendung wird es gebraucht, um den Avatar in der Welt zu bewegen und Einrichtungsgegenstände zu verschieben oder zu rotieren. Aufgrund dieser Anforderungen unterteilt sich das implementierte Color-Picking in zwei Ebenen. Die erste Ebene dient dazu bei einem Mausklick das selektierte Objekt zu ermitteln. Die zweite bestimmt die genauen Weltkoordinaten des selektierten Punkts. Dabei wird die zweite Ebene nur für die Bewegung des Avatars in der Welt gebraucht.
Umgesetzt wird das Color-Picking durch einen „Color-Index-Manager“, der jedem Core, der Geometrie enthält und auswählbar sein soll, einen Color-Index zuweist. Der Color-Index hat ein Byte zur Verfügung und kann somit 256 verschiedene Werte speichern. Dieser Index wird im Alphakanal der Eckpunktfarbe von jedem Core gespeichert. Die restlichen drei Farbkanäle werden nur in den Cores der Bodengeometrie [41] gebraucht. Sie bilden jeweils die lokalen Koordinaten der Bodengeometrie auf

[41] diese besteht aus mehreren zusammengesetzten Rechtecken, welche in der X-Z-Ebene liegen.

Farben ab (siehe Abbildung 14). Dabei wird die maximale Ausdehnung jeder Achse auch auf den maximalen Wert des entsprechenden Farbkanals abgebildet.

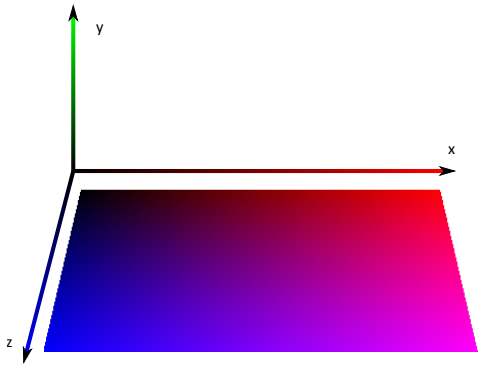


Abbildung 14: Abbildung der Weltkoordinaten der Bodengeometrie auf Farben.

Mit diesen Voraussetzungen ist man in der Lage nach einem Mausklick in einem Offscreen-Rendering-Pass die Szene mit den verschiedenen Color-Indizes der Cores zu rendern und den Pixel unter den 2D-Maus-Koordinaten zu ermitteln. Mit diesem Pixel und dessen Wert im Alphakanal, wird über die Hash-Map des Color-Index-Managers der selektierte Core ermittelt. Handelt es sich bei diesem Core um einen Core, der die Bodengeometrie verwaltet, kann über die Werte im RGB-Kanal, die angeklickte Weltposition zurückgerechnet werden. Dieses Verfahren hat auch seine Grenzen, wird eine sehr große Bodengeometrie verwendet, so dass ein Farbwert im Offscreen-Rendering auf mehrere Bildschirmpixel abgebildet werden, ist die zurückgerechnete Weltposition ungenau.

7 Beschreibung der App

In diesem Teil der Arbeit wird die letztendlich entwickelte Facebook-App mit ihren Funktionen vorgestellt[42]. Die App gliedert sich dabei in eine „Room View“, einen „Room Editor“, einen „Avatar Editor“ und eine Ansicht („Upload Content“), die es erlaubt neue Modelle für Avatare, die Inneneinrichtung und Texturen für den Hintergrund, hochzuladen. Wird die App gestartet gelangt man zuerst in die Room View und kann die anderen Ansichten über Tabs am oben Rand der App anwählen.

7.1 Room View

In der Room View kann man mit seinem Avatar durch seinen eigenen Raum und die Räume der anderen Benutzer navigieren (siehe Abb. 15). In der Textbox am unteren Rand der App können Nachrichten eingeben und abgeschickt werden, die über dem Avatar als Sprechblase erscheinen. Klickt man im Raum mit der Maus auf den Boden, wird über das Color-Picking die Koordinaten berechnet und der Avatar bewegt sich an die angeklickte Position. Am rechten Rand befindet sich der Dialog für die Raumwahl, über diesen kann der Raum gewählt werden, den man betreten will. Allerdings gibt es bisher nur die Möglichkeit seinen eigenen Raum und die Räume seiner Freunde zu besuchen, die ebenfalls die App benutzen.

Abbildung 15: Die Room View der App

[42]Die App ist unter der folgenden URL erreichbar: `http://apps.facebook.com/gwt_avatars_release/`

7.2 Room Editor

Im Tab des Room Editors, zu sehen in Abb. 16 kann der eigene Raum eingerichtet werden. Auf der oberen rechten Seite kann die Hintergrundtextur des Raums ausgewählt werden und darunter die Einrichtungsgegenstände. Mit der Toolbar in der unteren rechten Ecke können die platzierten Gegenstände verschoben, rotiert und gelöscht werden. Die Auswahl des gerade selektierten Gegenstandes wird durch einen weißen Kreis unter dem Gegenstand angezeigt und kann mit einem Mausklick gewechselt werden.

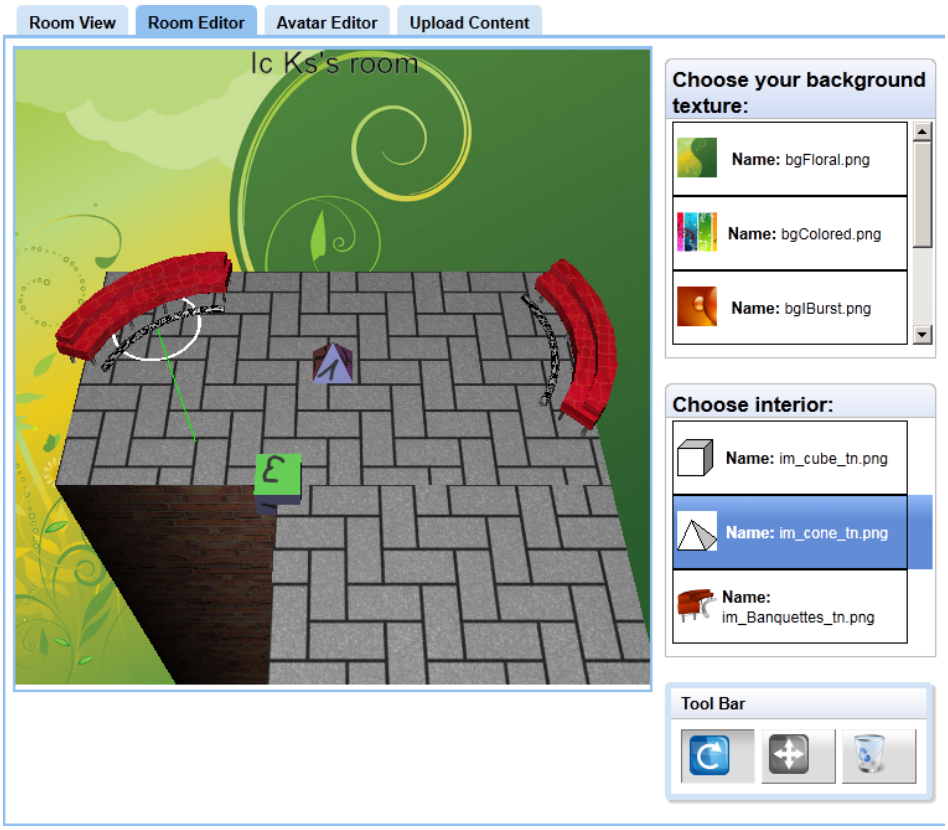


Abbildung 16: Der Room Editor der App

7.3 Avatar Editor

Der Avatar Editor in Abb. 17 ermöglicht dem Benutzer leider nur eine Auswahl zwischen mehreren Avataren. Die Zeit Editierfunktionen zu implementieren reichte nicht. Mit der Dialog Box unter dem Panel kann nur der Namen des Avatars geändert werden.

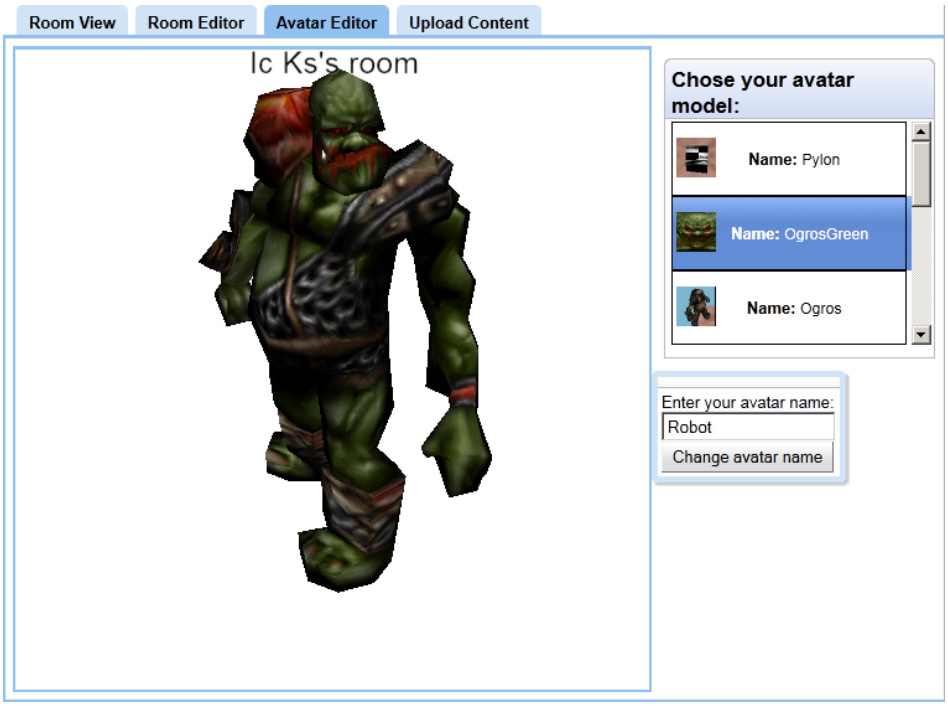


Abbildung 17: Der Avatar Editor der App

7.4 Upload Content

In dieser letzten Ansicht (Abb. 18) können neue Avatare, Modelle für die Inneneinrichtung und Hintergrundtexturen für den Raum in einem Zip-Archiv hochgeladen werden. Die Modelle und Texturen sind nur wenig später nach der Verarbeitung auf dem Server verfügbar. Da der Server die Inhalte des Archivs nur anhand des Dateinamens und der Dateiendung erkennt, bietet diese Funktion ein Einfallstor für Angreifer. Der Server könnte beispielsweise durch manipulierte Dateien ausgelastet werden. Besser wäre es die Dateien anhand ihres spezifischen Headers, der Dateigröße und auf Fehler zu überprüfen. Dieser Ansatz wäre jedoch mit einem vielfach größeren Implementierungsaufwand verbunden gewesen.

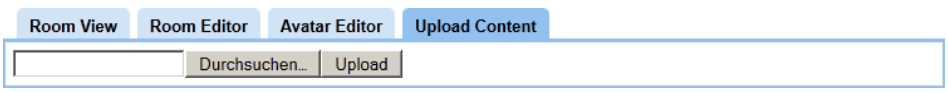


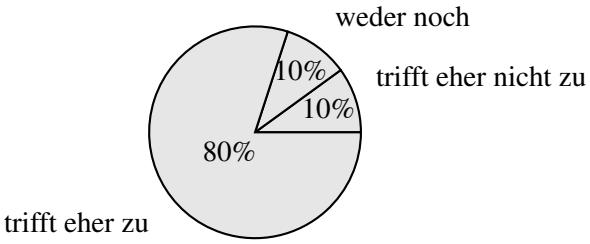
Abbildung 18: Die Ansicht zum Hochladen neuer Modelle und Texturen

8 Evaluierung der Anwendung

Nach der Beschreibung der App, werden in diesem Abschnitt die Ergebnisse der Evaluation präsentiert. Evaluiert wurde die App mithilfe einer Onlineumfrage, die auf der App-Seite verlinkt wurde. Von 234 Personen, die die App zumindest einmal aufgerufen haben (Stand Oktober 2011) haben, nahmen 10 an der Umfrage Teil.
Die Umfrage zielt dabei hauptsächlich auf die Erfüllung der Pattern aus Abschnitt 3.2, den Mehrwert und die Performance der App ab. Dazu wurden 18 Aussagen formuliert, welche die Teilnehmer, jeweils unter Benutzung der Likert-Skala, bewerten mussten.
Der nun folgende Teil zeigt zu den relevanten Aussagen jeweils das Ergebnis als auch eine Interpretation dazu.

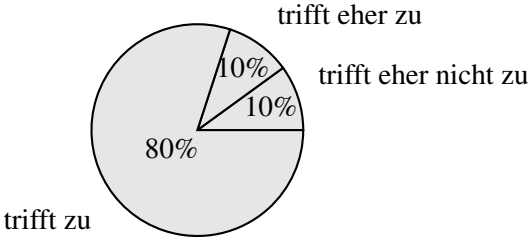
8.1 Mehrwert

- **Item 1:** Die App bringt dem sozialen Netzwerk einen Mehrwert.



Deutlich ist zu erkennen, dass Großteil einen Mehrwert durch die App wahrnimmt. Diese Aussage ist deshalb von Bedeutung, da in der virtuellen Welt lediglich grundlegende Funktionen zur Verfügung stehen. Es war daher nicht zweifelsfrei zu erwarten, dass in der subjektiven Wahrnehmung der einzelnen Benutzer etwas Neues und Einzigartiges wahrgenommen wird. Insbesondere unter der Annahme, dass die Mehrheit der Teilnehmer bereits Kontakt zu dreidimensionalen Inhalten z.B aus Spielen und Filmen hatte.

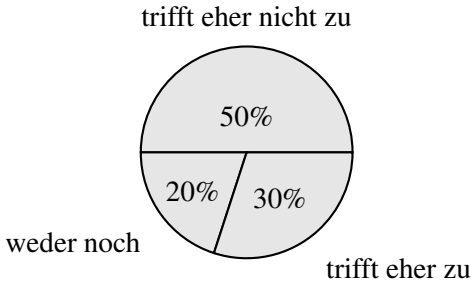
- **Item 2:** Hätte die App mehr Funktionen, würde sie dem sozialen Netzwerk einen Mehrwert bringen.



Diese Aussage muss im Zusammenhang mit der vorherigen Aussage (siehe **Item 1** 8.1) betrachtet werden. Wäre die erste Aussage überwiegend negativ, d.h. nicht zustimmend beantwortet worden, wäre ein Mangel der implementierten Funktionen ein möglicher Grund. Die Einschätzung dieser Aussage hätte diesen Mangel entlarvt. Da die vorherige Aussage, jedoch überwiegend Zustimmung erhielt, sind auch die Einschätzungen dieser Aussage zu einem Großteil zustimmend ausgefallen.

8.2 Spaßfaktor

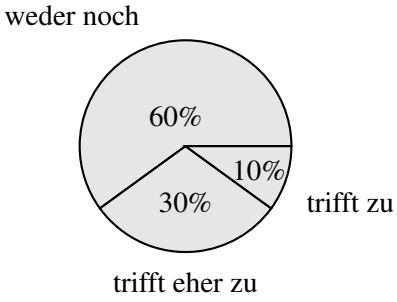
- **Item 3:** Der Spaßfaktor der App ist niedrig.



Das Ergebnis dieser Aussage ist trotz, des eher gering vermuteten Spielspaßes, mangels der gebotenen Möglichkeiten, gegenteilig ausgefallen. Allerdings ist zu erwähnen, dass die meisten Teilnehmer diese Umfrage bereits nach einer kurzen „Antestphase“ beantwortet haben. Somit könnten die wenigen Möglichkeiten der App, in einem kleinen Zeitrraum, dennoch ein positives Feedback hinterlassen haben.

8.3 Performance

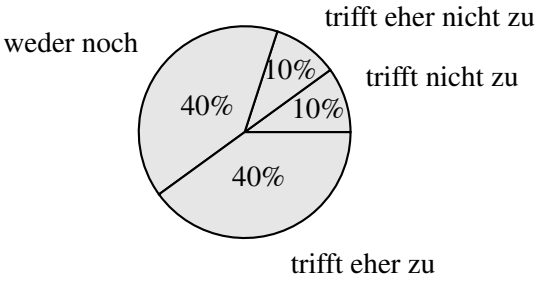
- **Item 4:** Die Performance der App ist gut.



Im Anbetracht der Tatsache, dass während der erstmaligen Benutzung der App, jede Textur und jedes Modell heruntergeladen werden muss, ist das Ergebnis ausreichend gut ausgefallen. Darüber hinaus muss bedacht werden, dass die App in Richtung Performance noch nicht optimiert ist. Auf Windows-Rechnern kommt erschwerend hinzu, dass WebGL-Code und der GLSL-Shadercode, sofern keine Änderungen an Chrome oder Firefox vorgenommen wird, in DirectX 9 kompatiblen Code konvertiert wird. Dieser Umstand hat zur Folge, dass sich die Ladezeit insbesondere bei dem Gebrauch vieler Shader, wie es in der Implementierung dieser Arbeit der Fall ist, oder komplexer Shader, spürbar verlängert. Eine Optimierung an diesen Stellen, könnte daher die Ladezeiten wesentlich verkürzen und zu einem besseren Ergebnis führen.

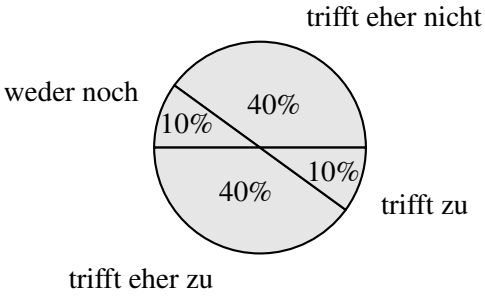
8.4 Pattern 1

- **Item 5:** Die Grafik der App ist ist gut.



Wie aus dem Diagramm zu sehen ist, empfindet die Mehrheit der Teilnehmer die Grafik als annehmbar. Überraschend, wenn man beachtet, dass es sich bei dem Md2-Format der Avatare, um ein bereits 14 Jahre altes 3D-Format handelt und der Boden eines jeden Raumes lediglich aus zwei Rechtecken besteht. Das Ziel aus dem ersten Pattern, nämlich den Benutzer in der virtuelle Welt „versinken“ zu lassen, kann aufgrund der folgenden Ergebnisse nur ansatzweise als erfüllt angesehen werden.

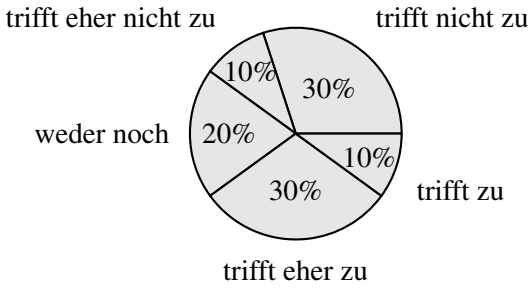
- **Item 6:** Die dargestellten Objekte und Grafiken haben einen Bezug zur realen Welt (der Würfel und der Pylon sind davon ausgenommen).



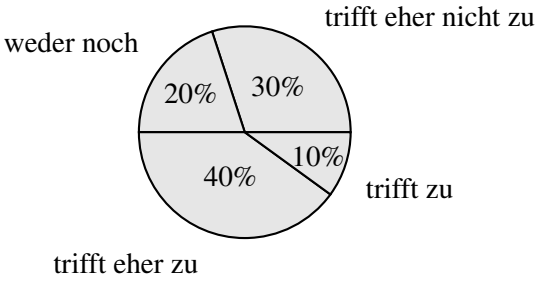
An dieser Stelle ist leider festzustellen, dass die zu bewertende Aussage, nicht präzise genug gestellt wurde. Aus der Verteilung der Bewertungen schließt sich weder eine klare Zustimmung, noch eine Ablehnung. Nachteilig hat sich, mit Sicherheit die Benutzung fertiger Md2-Modelle, die hauptsächlich im Ego-Shooter Quake II Verwendung finden, ausgewirkt. Die Erstellung eigener Modelle, stellte sich als sehr zeitintensiv heraus. Somit war die Verwendung schon fertiger Modelle die einzige Möglichkeit mehrere Modelle zur Auswahl anzubieten. Der Wiedererkennungswert und die Identifikation mit diesen oft sehr phantasievollen und martialischen Modellen, könnte daher die Bewertung dementsprechend beeinflusst haben.
Der im ersten Pattern geforderte hohe Wiedererkennungswert, kann daher der virtuellen Welt nicht zugesprochen werden.

8.5 Pattern 2

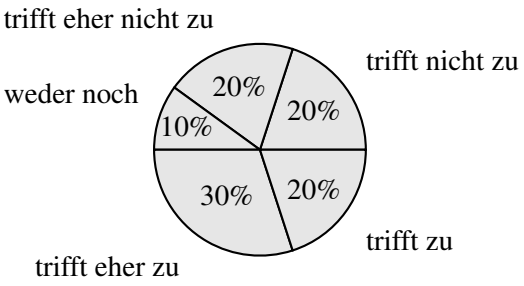
- **Item 7:** Die virtuelle Welt reagiert auf Aktionen meines Avatars.



- **Item 8:** Man fühlt sich in die virtuelle Welt einbezogen.



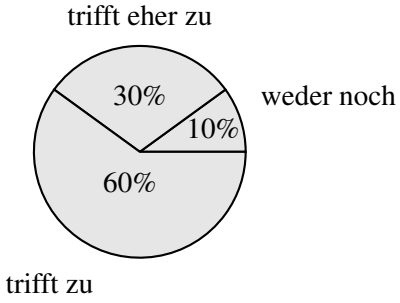
- **Item 9:** Das fehlen von Physik (z.B. Kollisionen und Schwerkraft) irritiert.



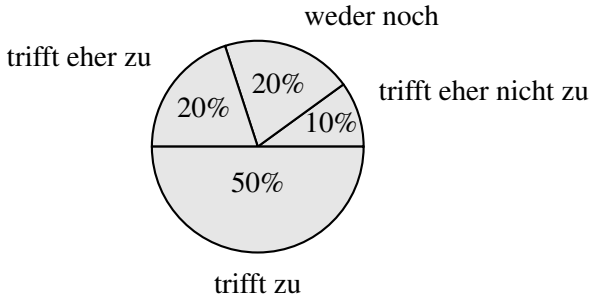
Die Ziele des zweiten Patterns, welches dem Benutzer den Eindruck vermitteln soll, dass er von der virtuellen Welt wahrgenommen und einbezogen wird, sind der allen drei Ergebnissen nach, nicht in ausreichendem Maß erfüllt worden. Dies liegt wahrscheinlich nicht nur an der fehlenden Physik. Es liegt auch die Vermutung nahe, dass die Anzeige des Raumnamens und der Bewegungstrajektorie, nicht explizit als Reaktion der virtuellen Welt wahrgenommen wurde. Außerdem sind zwei einfache Features für ein spürbares Feedback der virtuellen Welt zu wenig.

8.6 Pattern 3

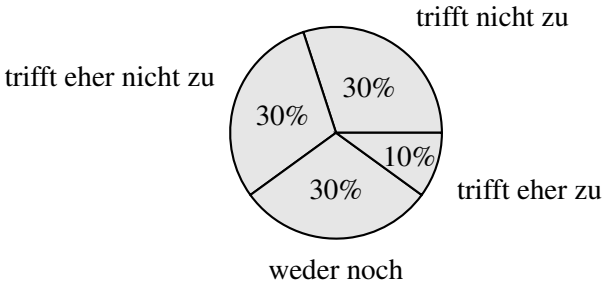
- **Item 10:** Es ist leicht die Orientierung in der virtuellen Welt zu behalten.



- **Item 11:** Man ist sich bewusst, in welchem Raum man sich befindet.

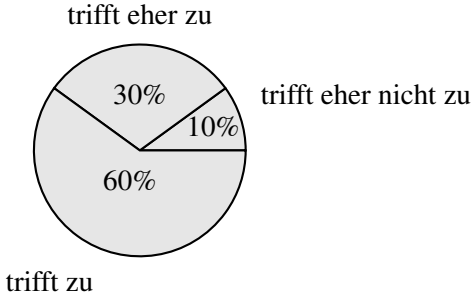


- **Item 12:** Das Wechseln von einem Raum zum anderen ist irritierend.

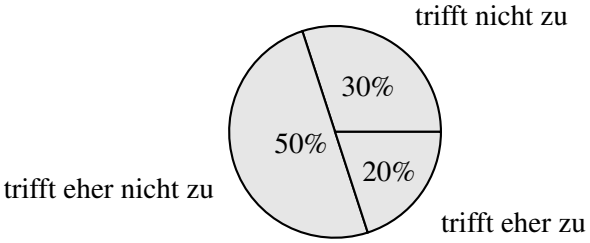


Die Ergebnisse dieser drei Aussagen müssen im Zusammenhang mit dem dritten Pattern gesehen werden. Der Inhalt dieses Pattern besagt, dass „Superkräfte“ ausschließlich dort genutzt werden sollen, wo sie gebraucht werden. Es sollte dabei herausgefunden werden, ob die Teleportation zwischen den Räumen sinnvoll ist. Dazu wurde als erstes allgemein die Orientierung in der virtuellen Welt bewertet. Betrachtet man die Ergebnisse von **Item 10**, kann diese als gut bezeichnet werden. Allerdings sollten Verbesserungen bei der Kenntlichmachung der Räume erfolgen, da sich anscheinend einige Teilnehmer nicht bewusst waren, in welchem Raum sie sich befinden (siehe **Item 11**). Außerdem wurde ein Raumwechsel zum Teil als irritierend empfunden (siehe **Item 12**). Dies lässt sich womöglich, darauf zurückführen, dass sich die verschiedenen Räume nur anhand ihrer Größe und dem eingeblendeten Raumname, unterscheiden.

- **Item 13:** Das Navigieren durch die virtuelle Welt fällt aufgrund der Perspektive leicht.



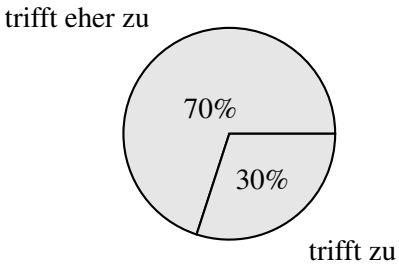
- **Item 14:** Die Perspektive auf die Welt ist störend.



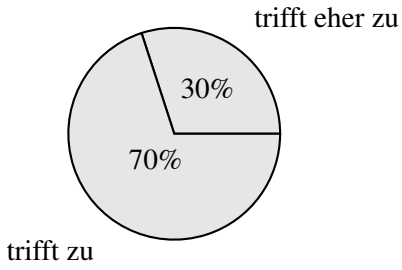
Als zweite „Superkraft“ sollte die Third-Person-Ansicht bewertet werden. Deutlich ist zu erkennen, dass es den Benutzern mit dieser Perspektive leicht fällt, zu navigieren. Zusammengefasst mit der Teleportation, kann sich sich das gewählte Konzept, als durchaus brauchbar bezeichnen lassen.

8.7 Pattern 4

- **Item 15:** Die App ist mit diesem Umfang an Möglichkeiten auf lange Sicht langweilig.



- **Item 16:** Aufgaben oder Spiele fehlen.



Leider sind auch die Ziele des vierten Patterns, Aktivitäten in der virtuellen Welt anzubieten, die über eine längere Zeitspanne das Interesse der Benutzer aufrecht erhalten, nicht ausreichend erfüllt. Dieses Ergebnis lässt sich vielleicht darauf zurückführen, dass von den zehn Beteiligten nur vier ihren Raum eingerichtet haben. Somit blieb für diese Benutzer als einzige Aktivität das Chatten über die Sprechblasen mit anderen Benutzern. Aber auch mit dem Einrichten des Raums ist davon auszugehen, dass der Ansporn nicht ausreicht ein dauerhaftes Interesse zu wecken.

8.8 Zusammenfassung der Ergebnisse

Fässt man die Ergebnisse der Umfrage zusammen, so besteht noch ein großer Bedarf an Verbesserungen, um von einer spannenden und fesselnden virtuellen Welt zu sprechen. Die dafür nötigen Aktivitäten, Aufgaben und Spiele sind jedoch aus programmiertechnischer Sicht eine große Herausforderung. Darüber hinaus, kann jedoch auch festgehalten werden, dass der momentane Zustand der Anwendung wahrscheinlich eine geeignete Basis darstellt.
Neben der Bewertung der Aussagen, konnten die Teilnehmer optional Kommentare und Anmerkung abgeben. Darin wurde vor allem eine intensivere Interaktion gefordert, z.B durch Minispiele, Avatargesten und einen Avatareditor mit mehr Möglichkeiten. Zudem hat sich mehrfach herauskristallisiert, dass es Zugang zu öffentlichen Räumen, d.h. zu Räumen, die jeder betreten kann, ohne mit dem Raumeigentümer befreundet zu sein, geben sollte.

9 Fazit und Ausblick

Als Schlussfolgerung lässt sich klar sagen, dass es mit der aktuellen Web-Technologie möglich ist virtuelle Welten ohne Plugin in den Browser zu bringen. Diese können sich allerdings nicht in allen Belangen mit einer heutigen virtuellen Welt messen, die nativ ausgeführt wird. Solche Welten werden zum Teil von mehreren Tausend Benutzern gleichzeitig besucht und bieten eine hoch realistische Grafik. JavaScript und die damit einhergehenden Web-Technologien sind diesem Leistungshunger noch nicht gewachsen. Jedoch wird dieser Umstand mit der Zeit in den Hintergrund treten. So wird der voranschreitende Wettbewerb zwischen den Browserherstellern bei dem u.a. die Ausführungsgeschwindigkeit von JavaScript, als Indikator für eine Verbesserung der Browser gilt, die Performanceprobleme beseitigen. Rechenintensive Web-Anwendungen, wie eine virtuelle Welt, profitieren davon sehr.
Der Spielraum für eine Optimierung der Grafik auch im Bezug auf diese Arbeit, ist hierbei noch nicht ausgeschöpft. Betrachtet man beispielsweise, die in Echtzeit gerenderte dreidimensionale Büste aus Abb. 19, wird ersichtlich, dass selbst High-End-Grafik in WebGL gerendert werden kann. Der Flaschenhals ist hauptsächlich beim Transport und der Verwaltung der 3D- und Texturdaten zu suchen; d.h. wie muss der Weg der Daten, vom Server auf den Client, der sie wiederum verändert und abschließend an die Grafikkarte weiterreicht, effizient gestaltet werden.
Auf der Grafikkarte angelangt, stellen selbst Effekt wie Subsurface-Scattering, das genutzt wird um die Haut der Büste zu rendern, kein Problem mehr dar.

Abbildung 19: Subsurface-Scattering mit WebGL

Außer den Performance- und Grafikproblemen, bereiten auch komplexe, echtzeitfähige Interaktionen zwischen mehreren Benutzern Schwierigkeiten. Hier ist es unbedingt notwendig, dass alternativen zum HTT-Protokoll spezifiziert und entwickelt werden. WebSockets mit ihrer bidirektionalen Verbindung sind daher der ideale Ansatz. Nur durch eine solche Verbindung wird die Voraussetzung geschaffen, beispielsweise Kollisionen zu simulieren. Die Channel API bzw. das zugrunde liegende Polling-Verfahren, das in dieser Arbeit verwendet wurde, bietet wegen der hohen Latenz nicht die Möglichkeit, mehrfach in der Sekunde Daten zu verschicken bzw. zu empfangen.
Außer diesen Erschwernissen, muss deutlich gesagt werden, dass es während der Verwendung, dieser noch sehr neuen Technologien häufig zu vielen kleineren Problemen gekommen ist. So lassen sich Strategien und Verfahren, die sich bereits in der Web-Entwicklung mit JavaScript bewährt haben, nicht immer ohne weiteres mit dem GWT und der GAE umsetzen. Zudem sind Dokumentation und Support oft nicht in dem Umfang vorhanden, wie es für die vorherrschenden Web-Technologien der Fall ist.

Beobachtet man all diese Probleme, so wird man feststellen, dass die meisten von ihnen dank der rasanten Web-Entwicklung in naher Zukunft nicht mehr existieren werden.
Positiv wirkt sich dabei auch die Verschiebung von Desktop-Applikation ins Web aus. Dieser Umbruch ist deutlich an der Entwicklung von Betriebssystemen zu sehen, die gänzlich auf die Cloud setzen [Ihl11]. Die Applikationen und Daten solcher Betriebssysteme lagern vorwiegend im Web. Ob sich bei diesem Umbruch WebGl als vorrangige 3D-Technologie durchsetzen wird, ist wie in Abschnitt 2.1.6 erwähnt, schwer einzuschätzen. Flash wird zumindest davon profitieren, dass es bereits eine große Gruppe an Flash-Spiele-Entwicklern gibt. Daher liegt die Annahme, dass diese eher dazu geneigt sind auf Basis von Flash, 3D Spiele zu entwickeln, nicht fern.

Abbildungsnachweis

- Abb. 6 Authorization. In [UNB11b]
- Abb. 5 Authentication. In: [UNB11b]
- Abb. 8 Semantischer Nachbau des Abstract Factory Patterns. In: [GHJV95]
- Abb. 10 Md2 file architecture. In: [Hen11]
- Abb. 12 Animation figure. In: [Hen11]
- Tab. 1 Quake II's Md2 model file format. In: [Hen11]
- Abb. 2.] Datastore and Blobstore. In: [Tho11]
- Abb. 19 WebGL Skin. In: [UNB11v]
- Abb. 1 Von: [CP11]

Literatur

[BEJZ09] Johannes Behr, Peter Eschler, Yvonne Jung, and Michael Zöllner. X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[BGP09] Nicoletta Blas, Franca Garzotto, and Caterina Poggi. Web engineering at the frontier of the web 2.0: Design patterns for online 3d shared spaces. *World Wide Web*, 12:345–379, December 2009.

[BPO08] Marion Boberg, Petri Piippo, and Elina Ollila. Designing avatars. In *Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts*, DIMEA '08, pages 232–239, New York, NY, USA, 2008. ACM.

[BRDA11] Rozenn Bouville Berthelot, Jérôme Royan, Thierry Duval, and Bruno Arnaldi. Scene graph adapter: an efficient architecture to improve interoperability between 3d formats and 3d applications engines. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 21–29, New York, NY, USA, 2011. ACM.

[CP11] Ewen Cheslack-Postava. Kataspace application built on sirikata, September 2011. `http://www.sirikata.com/blog/2010/kataspace-application-built-on-sirikata/`.

[Dew09] Anirudh Dewani. Gwt: The technical advantage. 2009.

[DWYW09] Nicolas Ducheneaut, Ming-Hui Wen, Nicholas Yee, and Greg Wadley. Body and mind: a study of avatar personalization in three virtual worlds. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 1151–1160, New York, NY, USA, 2009. ACM.

[ere11] By erez. *Creating a Facebook app with Google App Engine and Google Web Toolkit*, August 2011. http://techo-ecco.com/blog/creating-a-facebook-app-with-google-app-engine-and-google-web-toolkit/.

[Fer11] António Ramires Fernandes. *Billboarding Tutorial*, August 2011. http://www.lighthouse3d.com/opengl/billboarding/index.php?billCheat.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[Grü11] Dominik Grüntjens. Opengl es 2.0. In *Vorlesung „Echtzeit Rendering“*, 2011.

[GU10] Daniel Guermeur and Amy Unruh. *Google App Engine Java and GWT Application Development*. Packt Publishing, November 2010.

[H.11] Jason H. *gwt-gae-channel*, September 2011. http://code.google.com/p/gwt-gae-channel/.

[Hen11] David Henry. *The Quake II's MD2 file format*, September 2011. http://tfc.duke.free.fr/old/models/md2.htm.

[HL11a] Eran Hammer-Lahav. *Beginner's Guide to OAuth*, August 2011. http://www.khronos.org/registry/webgl/specs/latest/.

[HL11b] Eran Hammer-Lahav. *Beginner's Guide to OAuth*, August 2011. http://hueniverse.com/2007/10/beginners-guide-to-oauth-part-i-overview/.

[Ihl11] Jens Ihlenfeld. Googles frontalangriff auf microsoft, September 2011. http://www.golem.de/1105/83420.html.

[Int11] Alexa Internet. The top 500 sites on the web, September 2011. http://www.alexa.com/topsites.

[LD11] Seb Lee-Delisle. Webgl and molehill : an overview of in-browser gpu 3d, September 2011. http://sebleedelisle.com/2011/06/webgl-and-molehill-an-overview-of-in-browser-gpu-3d/.

[Mül06] Stefan Müller. Hierarchien. In *Vorlesung „Computergraphik 2009“*, pages 23–24, 2006.

[Pad11] PaderDesign. *Second Life Economic Statistics - Diagrammed*, September 2011. http://www.paderdesign.de/secondlife/stats/.

[Pau10] Peter Paulis. 3d webpages. Master's thesis, May 2010.

[Rei02] Dirk Reiners. Scene graph rendering. *Structure*, pages 1–18, 2002.

[Rud11] Jesse Ruderman. *Same origin policy for JavaScript*, August 2011. https://developer.mozilla.org/en/Same_origin_policy_for_JavaScript.

[Sen11] Christian Senk. *gwt-ws*, September 2011. http://code.google.com/p/gwt-ws/.

[Sey11] Brett Seyler. *Google, Android, and the Future of Games on the Web*, September 2011. http://blogs.unity3d.com/2010/05/19/google-android-and-the-future-of-games-on-the-web/.

[SNR+11] Markus Sareika, Giang Phuong Nguyen, Gerhard Reitmayr, Michael Idziorek, Antti Juustila, Thorsten Fröhlich, Ann Morrison, Zsolt Szalavari, and Peter Peltonen. *Integrated Project on Interaction and Presence in Urban Environments*, August 2011. http://www.ipcity-ist.eu/wp-content/uploads/2010/03/D5.4%20Final%20report%20on%20infrastructure.pdf.

[Ste11] Andy Stevko. *GWT Dev Plugin fails due to XSS policy violation when hosted in external domain's iframe*, August 2011. http://code.google.com/p/google-web-toolkit/issues/detail?id=4468.

[Tho11] Lilli Thompson. *Building Game Development Tools with App Engine, GWT, and WebGL*, September 2011. http://www.google.com/events/io/2011/sessions/building-game-development-tools-with-app-engine-gwt-and-webgl.html.

[UNB11a] UNBEKANNT. *About The Khronos Group*, September 2011. http://www.khronos.org/about.

[UNB11b] UNBEKANNT. *Authentication*, August 2011. http://developers.facebook.com/docs/authentication/.

[UNB11c] UNBEKANNT. *The Channel API (Java)*, August 2011. http://code.google.com/appengine/docs/java/channel/.

[UNB11d] UNBEKANNT. *Coding Basics - JavaScript Native Interface (JSNI)*, September 2011. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideCodingBasicsJSNI.html.

[UNB11e] UNBEKANNT. *Concepts*, September 2011. http://code.google.com/p/objectify-appengine/wiki/Concepts?tm=6.

[UNB11f] UNBEKANNT. *Deferred Binding*, September 2011. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/FAQ_Client.html.

[UNB11g] UNBEKANNT. *Developer's Guide - Client-side Storage (Web Storage)*, August 2011. http://code.google.com/webtoolkit/doc/latest/DevGuideHtml5Storage.html.

[UNB11h] UNBEKANNT. Flash platform runtime penetration, September 2011. http://www.adobe.com/products/player_census/flashplayer/mobile_penetration.html.

[UNB11i] UNBEKANNT. Flash player penetration, September 2011. http://www.adobe.com/products/player_census/flashplayer/.

[UNB11j] UNBEKANNT. *Graph API*, August 2011. http://developers.facebook.com/docs/reference/api/.

[UNB11k] UNBEKANNT. *JRE Emulation Library*, September 2011. http://www.gwtapps.com/doc/html/jre.html.

[UNB11l] UNBEKANNT. *An open protocol to allow secure API authorization in a simple and standard method from desktop and web applications.*, August 2011. http://oauth.net/.

[UNB11m] UNBEKANNT. *Permissions*, August 2011. http://developers.facebook.com/docs/reference/api/permissions/.

[UNB11n] UNBEKANNT. *Publishing*, September 2011. http://unity3d.com/unity/publishing/.

[UNB11o] UNBEKANNT. *Publishing Web*, September 2011. http://unity3d.com/unity/publishing/web.

[UNB11p] UNBEKANNT. *The simplest convenient interface to the Google App Engine datastore*, September 2011. http://code.google.com/p/objectify-appengine/.

[UNB11q] UNBEKANNT. Sirikata a bsd licensed open source platform for games and virtual worlds, September 2011. http://www.sirikata.com/blog/.

[UNB11r] UNBEKANNT. Stage3d apis for adobe flash player and adobe air, September 2011. http://labs.adobe.com/technologies/flashplatformruntimes/features/stage3d.html.

[UNB11s] UNBEKANNT. Unity, flash & 3d on the web, September 2011. http://blogs.unity3d.com/2011/02/27/unity-flash-3d-on-the-web/.

[UNB11t] UNBEKANNT. *URL Fetch Java API Overview*, August 2011. http://code.google.com/intl/de-DE/appengine/docs/java/urlfetch/overview.html.

[UNB11u] UNBEKANNT. *WebGL and OpenGL Differences*, August 2011. http://www.khronos.org/webgl/wiki/WebGL_and_OpenGL_Differences.

[UNB11v] UNBEKANNT. Webgl skin, September 2011. http://www.chromeexperiments.com/detail/webgl-skin/.

[UNB11w] UNBEKANNT. *What Is Google App Engine?*, August 2011. http://code.google.com/intl/de-DE/appengine/docs/whatisgoogleappengine.html.

[UNB11x] UNBEKANNT. *Writing Files to the Blobstore (Experimental)*, September 2011. http://code.google.com/appengine/docs/java/blobstore/overview.html#Writing_Files_to_the_Blobstore.

[VJP07] Asimina Vasalou, Adam N. Joinson, and Jeremy Pitt. Constructing my online self: avatars that increase self-focused attention. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '07, pages 445–448, New York, NY, USA, 2007. ACM.

[WLH07] Richard S. Wright, Benjamin Lipchak, and Nicholas Haemel. *OpenGL(R) SuperBible: Comprehensive Tutorial and Reference (4th Edition) (OpenGL)*. Addison-Wesley Professional, Upper Saddle River, NJ, USA, 2007.