Entwicklung einer Auszeichnungs- und Stylingsprache zur deklarativen Erstellung von WebGL-basierten 3D Inhalten

Julia Anita Kaupert

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im Juni 2016

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 27. Juni 2016

Julia Anita Kaupert

Inhaltsverzeichnis

Gendering

Aus Gründen der sprachlichen Vereinfachung sind alle Aussagen in dieser Arbeit als geschlechtsneutral zu verstehen. Die Wörter *Entwickler* und *Benutzer* schließen sowohl Männer als auch Frauen ein, wenn nicht explizit angegeben wird, dass es sich dabei nur um die männlichen Entwickler beziehungsweise Benutzer handelt.

Kurzfassung

Seit einigen Jahren stehen neben der WebGL API auch einige darauf basierende JavaScript-Bibliotheken für die Erstellung von 3D Objekten und Szenen im Web zur Verfügung. Bei näherer Betrachtung dieser Ansätze wird jedoch schnell klar, dass all diese hoch komplex und für unerfahrene Webentwickler im Bereich 3D schwierig anzuwenden und zu erlernen sind. Um die Erstellung von 3D Inhalten im Web zu vereinfachen, wird im Zuge dieser Arbeit eine Auszeichnungs- und Stylingsprache entwickelt. Diese beiden neu entwickelten Sprachen bauen auf die bekannte HTML- beziehungsweise CSS-Syntax auf, da sie leicht zu verstehen sind und bereits einen hohen Bekanntheitsgrad unter Webentwicklern haben. Mit diesem neuen Ansatz soll es ermöglicht werden, ohne Vorkenntnisse im Bereich Web 3D, einfach und schnell 3D Geometrien und Umgebungen im Browser darzustellen.

Abstract

For the creation of 3D objects and scenes in the browser, the WebGL API, as well as many JavaScript libraries based on it, have been around for quite some time. However, if you take a closer look, you will realize that these methods are highly complex and hard to use for web developers who are inexperienced in the field of web 3D. To make the creation of 3D content on the web easier, a markup and styling language will be developed within this thesis. These languages build upon the well known HTML- and CSS syntaxes which are easy to learn and widely known among web developers. With this new approach it should now be possible, to display 3D geometries and environments fast and easily within the browser, without any prior knowledge.

Kapitel 1

Einleitung

Webbrowser wurden in den letzten Jahren immer leistungsfähiger und können nun neben 2D Anwendungen auch komplexere Anwendungen und Grafiken darstellen. Während die Anzahl von 2D Anwendungen derzeit noch stark überwiegt, wurde die Entwicklung von 3D Szenen im Browser inzwischen ein immer wichtigeres Thema. Viele verschiedene Ansätze und Lösungswege wurden bereits entwickelt, wie zum Beispiel VRML, X3D, X3DOM und XML3D. Neuerdings stehen auch WebGL, sowie darauf basierende JavaScript-Lösungen zur Verfügung, die mit fast allen modernen Browsern kompatibel sind. Da alle diese Ansätze Nachteile mit sich bringen, wird in dieser Arbeit der neue Ansatz einer Auszeichnungs- und Stylingsprache, basierend auf WebGL beziehungsweise der JavaScript-Bibliothek Three.js, für die Entwicklung von 3D Inhalten im Web gezeigt und erklärt.

1.1 Motivation und Zielsetzung

Das Thema 3D im Web gewann in den letzten Jahren immer mehr an Bedeutung und auch die Anzahl an 3D Anwendungen im Browser wie Spiele und virtuelle Rundgänge steigt. Da der Trend immer stärker zu Cloudbasierten Anwendungen anstelle von installierbaren Programmen geht, überlegen bereits einige Unternehmen eine Browserversion ihres Programms zu entwickeln. Zukünftig könnten dann beispielsweise Modellierungsprogramme mit einer auf WebGL basierenden Lösung realisiert werden.

Die meisten der bereits bestehenden Ansätze, mit denen eine solche 3D Anwendung umgesetzt werden könnte, haben jedoch den Nachteil, dass sie sehr komplex und schwer erlernbar sind und auch Struktur, Aussehen und Funktionalität nicht voneinander getrennt werden. Dies führt speziell bei größeren Projekten zu unleserlichem Code, der schwer zu editieren und zu erweitern ist. Vor allem bei der Entwicklung mit WebGL oder JavaScriptbasierenden Lösungen wie beispielsweise Three.js sind viele Code-Zeilen nötig, nur um ein einfaches 3D Objekt zu erstellen oder dieses zu stylen und

zu animieren. Mit dem neuen Ansatz, der in dieser Arbeit vorgestellt wird, sollen durch eine einfache und weitverbreitete Art von Syntax die Komplexität verringert, sowie Code-Zeilen eingespart werden. Auch unerfahrenen Entwicklern im Bereich Web 3D soll es so ermöglicht werden, einfach und schnell 3D Szenen zu erstellen.

Ziel dieser Arbeit ist es also herauszufinden, ob die Entwicklungskomplexität sowie die Zeit, die für die Entwicklung einer 3D Szene gebraucht wird, mit dem neuen Ansatz durch Trennung von Struktur, Aussehen und Funktionalität verringert beziehungsweise verkürzt werden kann. Hierzu werden das Konzept sowie die Implementierung einer Auszeichnungs- und Stylingsprache beschrieben. Der neu entwickelte Ansatz wird dann anhand einer Test-Szene, die von mehreren Testpersonen implementiert wird, getestet und evaluiert. Die Zeit wird anschließend mit jener Zeit verglichen, welche die Testpersonen bei der Umsetzung mit der bekannten JavaScript Bibliothek Three.js benötigen. Zusätzlich werden die Testpersonen nach der Umsetzung befragt, wie sie die Entwicklung mit dem neuen Ansatz empfanden. So soll festgestellt werden, ob die Erstellung von 3D Szenen im Web mithilfe der Auszeichnungs- und Stylingsprache vereinfacht werden kann.

1.2 Struktur und Aufbau

Um einen besseren Überblick zu schaffen, wird diese Arbeit in sieben Kapitel unterteilt. Beginnend mit der Einleitung, welche die Motivation und die konkrete Zielsetzung dieser Arbeit enthält. Anschließend wird in Kapitel 2 der technische Hintergrund erklärt und ein kurzer Überblick über WebGL und Three.js gegeben. In Kapitel 3 werden bestehende Ansätze für die Entwicklung von 3D Szenen im Web, wie VRML, X3D, X3DOM und XML3D erläutert. Zusätzlich wird jeweils eine kurze Beispiel-Szene gezeigt und beschrieben. Im darauf folgenden Kapitel 4 wird das Konzept und das technische Design des neuen Ansatzes einer Auszeichnungs- und Stylingsprache gezeigt. Außerdem wird eine vollständige Beispiel-Szene erstellt und erklärt. Die Implementierung des JSCSS Parsers, sowie des HTML Transpilers und des CSS Transpilers wird in Kapitel 5 gezeigt. Kapitel 6 beschreibt zuerst die Methode, mit welcher der neue Ansatz getestet wird. Danach werden die Testpersonen kurz vorgestellt, die Testergebnisse tabellarisch gegenübergestellt und die Benutzerbeurteilungen, sowie deren Verbesserungs- und Erweiterungsvorschläge gezeigt. Zuletzt folgt in Kapitel 7 das Fazit der gesamten Arbeit, sowie einen Ausblick und zukünftige Verbesserungs- und Erweiterungsmöglichkeiten für die Auszeichnungs- und Stylingsprache.

Kapitel 2

Technischer Hintergrund

Dieses Kapitel gibt eine kurzen Überblick über die bei der Umsetzung verwendeten Bibliotheken. Diese sind, wie im Titel der Arbeit bereits erwähnt, WebGL und Three.js.

2.1 WebGL

WebGL[1] ist eine betriebssystem-übergreifende und lizenzfreie 3D Grafik API, die es ermöglicht, 3D Grafiken und Szenen im Web darzustellen. Die API basiert auf OpenGL[2] ES 2.0 und wird durch ein HTML5 `<canvas>` Element als Document Object Model (DOM) Schnittstelle bereitgestellt. Um möglichst viele Anwendungsfälle abzudecken, benutzt WebGL einfache und flexible Geometrien. Da es von OpenGL ES 2.0 abstammt, können Entwickler mit Vorkenntnissen in OpenGL einfach auf WebGL umsteigen [11]. Hier die offizielle WebGL Beschreibung des Entwicklers Khronos [12]:

> „WebGL™ is an immediate mode 3D rendering API designed for the web. It is derived from OpenGL® ES 2.0, and provides similar rendering functionality, but in an HTML context. WebGL is designed as a rendering context for the HTML Canvas element. [...] Given the many use cases of 3D graphics, WebGL chooses the approach of providing flexible primitives that can be applied to any use case. Libraries can provide an API on top of WebGL that is more tailored to specific areas, thus adding a convenience layer to WebGL that can accelerate and simplify development. However, because of its OpenGL ES 2.0 heritage, it should be straightforward for developers familiar with modern desktop OpenGL or OpenGL ES 2.0 development to transition to WebGL development“.

[1] https://www.khronos.org/webgl
[2] https://www.opengl.org

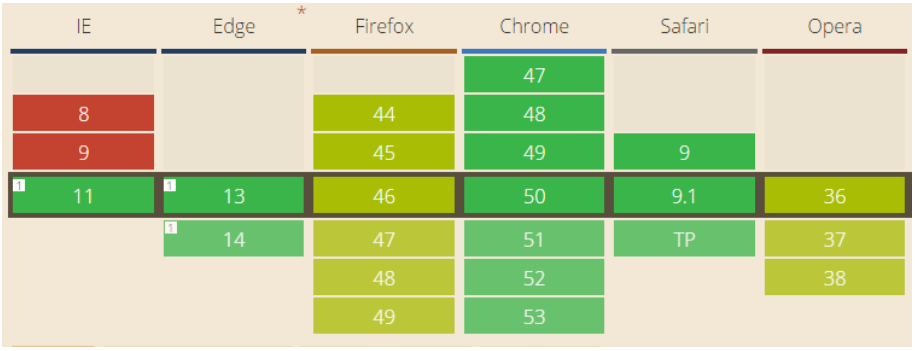


Abbildung 2.1: Browser-Unterstützung für WebGL [7].

WebGL ist Teil der HTML5 Familie und wird daher von fast allen Browsern unterstützt, die auch mit HTML5 kompatibel sind. Lediglich die Browser Firefox[3] und Opera[4] unterstützen WebGL noch nicht vollständig (siehe Abbildung 2.1). Zusätzlich funktioniert WebGL auch auf vielen mobilen Browsern. Die Kompatibilität des Browsers kann durch Aufruf der WebGL Report Seite[5] leicht getestet werden.

Um eine auf WebGL basierende Seite im Browser darzustellen, sind folgende Schritte notwendig [2]:

- Erstellen eines `<canvas>` Elements.
- Zugriff auf den Kontext des `<canvas>` Elements herstellen.
- Ansichtsfenster initialisieren.
- Hinzufügen von mindestens einem Buffer, der die zu rendernden Daten enthält (z.B. Eckpunkte).
- Definieren von mindestens einer Matrix, um die Transformationen des Buffers darzustellen.
- Erstellen eines Shaders, bzw. mehrerer Shader, je nach Bedarf.
- Initialisierung des Shaders bzw. der Shader mit Parametern.
- Zeichnen.

[3] https://www.mozilla.org/de/firefox/new
[4] http://www.opera.com/de
[5] http://www.doesmybrowsersupportwebgl.com

2.2 Three.js

Three.js[6] ist eine von vielen Open Source Bibliotheken und 3D Engines, wie zum Beispiel auch GLGE[7], Scene.js[8] und CubicVR[9], die für WebGL entwickelt wurden. Das Ziel von Three.js ist es, eine kleine und leicht zu benutzende 3D Bibliothek zu entwickeln [14]. Im Gegensatz zu WebGL, welches eine Vielzahl an Code-Zeilen erfordert, nur um zum Beispiel einen Würfel darzustellen, ist der Three.js Code nur ein Bruchteil davon [15]. Um etwas mit Three.js darzustellen, sind drei Schritte notwendig. Zuerst muss eine Szene erzeugt werden, woraufhin nur noch eine Kamera und ein Renderer hinzugefügt werden müssen. Three.js bietet viele Vorteile für Entwickler, wie zum Beispiel

- das Erstellen von einfachen, sowie komplexen 3D Geometrien,
- das Animieren und Bewegen von Objekten innerhalb einer 3D Szene,
- die Verwendung von Materialien mit Unterstützung von verschiedenen Texturen,
- das Laden von Objekten aus 3D Modellierungsprogrammen und
- das Erstellen von 2D Sprite-basierten Grafiken [1].

Außerdem ist die JavaScript-Bibliothek mit allen modernen Browsern, wie

- Mozilla Firefox seit Version 4.0,
- Google Chrome seit Version 9,
- Safari seit Version 5.1,
- Opera seit Version 12.00 und
- Internet Explorer seit Version 11

kompatibel [1]. Lediglich ältere Versionen des Internet Explorers unterstützen die Three.js Bibliothek nicht.

2.2.1 Aufbau einer einfachen Szene

In diesem Unterkapitel wird eine einfache Three.js Szene gezeigt und beschrieben. Der Code sieht dabei folgendermaßen aus:

```
1 <body>
2   <script src="js/three.min.js"></script>
3   <script>
4     var scene, camera, renderer;
5     init();
6     animate();
7
8     function init() {
```

[6] http://threejs.org
[7] http://www.glge.org
[8] http://www.scenejs.org
[9] http://www.cubicvr.org

```
 9       scene = new THREE.Scene();
10       var WIDTH = window.innerWidth;
11       var HEIGHT = window.innerHeight;
12
13       renderer = new THREE.WebGLRenderer({antialias:true});
14       renderer.setSize(WIDTH, HEIGHT);
15       document.body.appendChild(renderer.domElement);
16
17       camera = new THREE.PerspectiveCamera(45, WIDTH / HEIGHT, 0.1,
       20000);
18       camera.position.set(0,6,0);
19       scene.add(camera);
20
21       window.addEventListener('resize', function() {
22         var WIDTH = window.innerWidth;
23         var HEIGHT = window.innerHeight;
24         renderer.setSize(WIDTH, HEIGHT);
25         camera.aspect = WIDTH / HEIGHT;
26         camera.updateProjectionMatrix();
27       });
28
29       var light = new THREE.PointLight(0xffffff);
30       light.position.set(-100,200,100);
31       scene.add(light);
32
33       controls = new THREE.OrbitControls(camera, renderer.domElement);
34     }
35
36     function animate() {
37       requestAnimationFrame(animate);
38       renderer.render(scene, camera);
39       controls.update();
40     }
41   </script>
42 </body>
```

Zu Beginn wird in Zeile 2 die Three.js JavaScript-Datei eingebunden, um die Bibliothek nutzen zu können. Danach werden in Zeile 4 für die zuvor beschriebenen drei notwendigen Komponenten einer Szene globale Variablen angelegt. Die Erstellung der Szene erfolgt in Zeile 9. Das HTML-Element, auf dem der Renderer zeichnet, wird in den Zeilen 13 bis 15 erstellt und zum DOM hinzugefügt. Anschließend wird eine Kamera erstellt, positioniert, ausgerichtet und zur Szene hinzugefügt (siehe Zeile 17 bis 19) und ein *Eventlistener* ergänzt, der dem Renderer mitteilt, wenn sich die Größe seines Zeichenelements verändert (siehe Zeile 21 bis 27). In den Zeilen 29 bis 31 wird das Licht instanziiert, positioniert und zur Szene hinzugefügt und in Zeile 33 abschließend noch die Steuerung festgelegt, um mit der Maus innerhalb der Szene navigieren zu können. Die Funktion `animate` am Ende der Datei ist für das Rendern und Aktualisieren der Szene verantwortlich.

Kapitel 3

Aktueller Forschungsstand

In diesem Kapitel werden bereits existierende Technologien zur Erstellung von 3D Inhalten im Web vorgestellt und beschrieben. Während der neue Ansatz, der später genauer erklärt wird, aus einer Auszeichnungs- und Stylingsprache besteht, sind diese Projekte lediglich Auszeichnungssprachen und Dateiformate.

3.1 VRML

Virtual Reality Modeling Language (VRML)[1] ist ein ISO Standard Dateiformat zur Darstellung von drei-dimensionalen Vektorgrafiken im Web. Ursprünglich wurde VRML von SGI im Jahr 1994 entwickelt und war das erste Dateiformat, mit dem interaktive 3D Grafiken im Internet dargestellt wurden [2]. Es wurde 1997 zu einem internationalen Standard [8]. VRML ist eine einfache Textsprache zur Beschreibung von 3D Formen und interaktiven Umgebungen, deren Dateien mit .wrl enden. Für die Darstellung einer Szene ist ein externes Plugin, wie zum Beispiel Cortona3D[2], erforderlich.

3.1.1 Struktur und Code-Beispiel

Eine VRML Datei kann folgende (Groß-/Kleinschreibung unterscheidende) Komponenten enthalten [13]:

- VRML Header,
- Prototypen,
- Shapes, Interpolatoren, Sensoren und Skripte,
- Routers,
- Kommentare,
- Knoten,

[1] http://vrml2.de

[2] http://www.cortona3d.com/cortona3dviewer

- Felder und Feldwerte,
- definierte Knotennamen,
- benutzte Knotennamen.

In folgendem Code-Beispiel wird der Aufbau einer einfachen VRML Szene gezeigt [6]:

```
43 #VRML V2.0 utf8
44 # A Cylinder
45 Shape {
46   appearance Appearance {
47     material Material { }
48   }
49   geometry Cylinder {
50     height 3.0
51     radius 2.0
52     bottom TRUE
53     top TRUE
54     side TRUE
55   }
56 }
```

Am Beginn jeder VRML-Datei wird der Datei-Kopf erzeugt, in dem die Version und die Zeichencodierung angegeben werden. In diesem Fall beinhaltet die Datei VRML-Code, welcher der V2.0 Syntax entspricht und UTF-8 als Zeichencodierung nutzt (siehe Zeile 43). Kommentare beginnen, wie in Zeile 44, immer mit einer Raute (#) und gelten bis zum Ende der Zeile. In Zeile 45 wird eine *Shape Node* erstellt, die den Inhalt der Szene beschreibt. Jede *Shape Node* besteht aus einem Typ, einem Paar geschwungener Klammern, sowie optionalen Feldern innerhalb des Klammern-Paars. Zwischen Zeile 46 und 48 wird das Material des Objekts gesetzt. In der darauf folgenden Zeile wird die Geometrie (Box, Cone, Sphere, etc.), in diesem Fall ein Zylinder, erstellt. Die in den Zeilen 50 bis 54 angegebenen Felder und Werte spezifizieren die *Node* Attribute. Jedes Feld besteht aus einem Namen (height, radius, fontStyle, etc.), einem Datentyp (float, integer, etc.) und einem (Standard) Wert. Die Reihenfolge der Felder hat keine Auswirkung auf die Interpretation der Daten.

In Abbildung 3.1 ist die aus dem VRML-Code resultierende Szene dargestellt. Auf der Webseite von Web3d[3] sind weitere VRML Beispiele ersichtlich.

VRML konnte den Anforderungen der Unterstützer und Nutzer aus zwei Hauptgründen nicht gerecht werden [2]:

> „VRML unfortunately didn't gain the broad adoption its supporters had hoped for. From a purely technical standpoint, there were two contributing factors. First, the programmability was limited due to poor performance at the time of the available

[3] http://www.web3d.org/x3d/content/examples/Vrml2.0Sourcebook

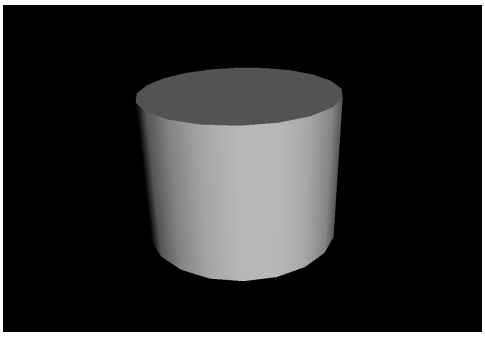


Abbildung 3.1: Die aus dem zuvor beschriebenem VRML-Code resultierende Szene.

> scripting languages' virtual machines. This meant that it wasn't possible to write generalpurpose code that affected the 3D scene, inherently limiting the domains to which VRML could be applied. Second, the rendering model was based on the original OpenGL API's fixed function graphics pipeline. This meant it was not possible to add new kinds of visual effects beyond those that had been designed into the system".

Der erste Grund dafür war die schlechte Performance. Der zweite, dass das Rendering Model auf der OpenGL API *fixed function graphics pipeline* basiert, weshalb es nicht möglich war, neue visuelle Effekte hinzuzufügen. VRML wurde 2004 durch den Standard X3D ersetzt [9].

3.2 X3D

Extensible 3D (X3D)[4] ist eine ISO Standard Szenengraph Sprache für interaktive 3D Grafiken im Web, sowie die dritte Generation von VRML, welche weiterhin durch Rückwärtskompatibilität unterstützt wird [4].

3.2.1 Struktur und Code-Beispiel

X3D benutzt die Szenengraph Datenstruktur, um virtuelle Umgebungen zu erstellen. Ein Szenengraph ist eine gerichtete, azyklische Baumstruktur, die alle Aspekte einer 3D Szene wie Geometrien, Shader, Positionen, Ausrichtungen, Animationen etc. beinhaltet. In diesem Kontext versteht man unter einem *directed acyclic graph*, kurz DAG, eine Baumstruktur mit einem Wurzelknoten und mehreren Unterknoten. Mit X3D aufgebaute Szenen weisen folgende Dateistruktur auf:

[4] http://www.web3d.org/x3d

```
57 <?xml version="1.0" encoding="UTF-8"?>
58 <!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.3//EN" "http://www.web3d.org
       /specifications/x3d-3.3.dtd">
59 <X3D version='3.3' width='400px' height='300px'>
60   <Scene>
61     <Transform translation='-3 0 0'>
62       <Shape>
63         <Box size='2 2 2'></Box>
64         <Appearance>
65           <Material diffuseColor='1 0 0'></Material>
66         </Appearance>
67       </Shape>
68     </Transform>
69   </Scene>
70   <Background skyColor="0.3 0.3 0.3"></Background>
71 </X3D>
```

Zu Beginn wird immer der Kopf der Datei erzeugt und die Version und die Zeichencodierung angegeben. In diesem Fall wird die XML Version 1.0 und die UTF-8 Codierung in der Datei verwendet (siehe Zeile 57) und in der darauf folgenden Zeile der Dokumenten-Typ für X3D gesetzt. Der `<X3D>` Tag in Zeile 59 beschreibt den Bereich, in dem die Szene dargestellt wird. Eine rechteckige Zeichenfläche mit einer Breite von 400px und einer Höhe von 300px wird angelegt und die verwendete X3D Version auf die aktuelle Standardversion 3.3 gesetzt [5]. Alle Elemente, wie beispielsweise `Light`, `Group` und `Object` müssen Kinder des `<Scene>` Tags in Zeile 60 sein. Um die folgende *Shape Node* drei Einheiten nach links zu verschieben, wird in Zeile 61 die *Transform Node* mit dem *Translation Field* und dem Wert -3 für die X-Achse hinzugefügt. Jede Geometrie muss in ein `<Shape>` Element wie in Zeile 62 verschachtelt werden. Im Inneren des Elements wird dann die Geometrie des Objekts über eine *Box Node* in Zeile 63 angelegt. Die Seitenlängen der Box Geometrie wird im *Size Field* auf zwei gesetzt. Über die *Appearance Node* in Zeile 64 im Inneren des `<Shape>` Tags wird die Standard-Farbe überschrieben und auf weiß gesetzt. In Zeile 65 wird innerhalb dieser *Node* der diffuse Anteil des Reflexionsverhaltens des Materials auf rot gesetzt. Zum Schluss wird in dieser Datei noch in Zeile 70 die Hintergrundfarbe der gesamten Szene auf grau gesetzt.

Um diese Szene nun im Browser darstellen zu können, ist ein Plugin wie zum Beispiel FreeWRL[5] nötig, das auf dem PC installiert sein muss. Die aus dem Code resultierende Szene sieht mit dem FreeWRL Plugin wie in Abbildung 3.2 aus. Die Web3d Webseite[6] stellt eine Liste mit Links zu weiteren Beispielen zur Verfügung.

[5] http://freewrl.sourceforge.net

[6] http://www.web3d.org/x3d/content/examples/X3dResources.html

Abbildung 3.2: Das Resultat des X3D-Codes, gerendert mit dem FreeWRL Plugin.

3.3 X3DOM

Bei X3DOM[7] handelt es sich um eine Open Source JavaScript Bibliothek für 3D Szenen im Web, die den X3D-Standard auf Basis von HTML5 und WebGL umsetzt. Deshalb ist nun kein zusätzliches Plugin mehr nötig, um eine X3D Szene im Browser darzustellen.

Der Name X3DOM setzt sich aus zwei Abkürzungen zusammen. Zum einen X3D, da X3DOM die X3D Syntax nutzt, und zum anderen DOM, da diese via X3D erzeugten Objekte über DOM Operationen manipuliert werden können. Dadurch ist es einfach, dynamische Änderungen, wie beispielsweise eine Farbänderung, an Objekten mit der `setAttribute` Funktion vorzunehmen. Dies ist die selbe Vorgehensweise wie zum Beispiel eine Änderung des Textes via JavaScript auf einer gewöhnlichen Webseite.

3.3.1 Struktur und Code-Beispiel

X3DOM Szenen sind sehr ähnlich aufgebaut, wie jene mit X3D. Der einzige Unterschied ist, dass der X3D-Code zusätzlich in eine normale HTML-Struktur inkludiert wird. Das bedeutet, dass es wie auf einer normalen Webseite einen `<html>` Tag gibt, der wiederum einen `<head>` Tag beinhaltet, in dem die JavaScript und CSS-Dateien von X3DOM eingebunden werden. Danach wird der `<body>` Tag hinzugefügt, in dem der Code der X3D Szene verschachtelt wird. Aufgrund des Standard HTML Aufbaus und den inkludierten X3DOM-Dateien wird kein Plugin zur Darstellung im Browser benötigt. Ein zusätzlicher Vorteil besteht darin, dass X3DOM dadurch kompatibel mit

[7] http://www.x3dom.org

Abbildung 3.3: Resultat des X3DOM-Code Beispiels im Browser.

den meisten modernen Browsern ist. Im folgenden Abschnitt wird der Code einer einfachen X3DOM Szene dargestellt, in der ein roter Würfel erzeugt wird [16]:

```
<html>
  <head>
    <script type='text/javascript' src='http://www.x3dom.org/download/
    x3dom.js'> </script>
    <link rel='stylesheet' type='text/css' href='http://www.x3dom.org/
    download/x3dom.css'></link>
  </head>
  <body>
    <x3d width='400px' height='300px'>
      <scene>
        <transform translation='-3 0 0'>
          <shape>
            <box size='2 2 2'></box>
            <appearance>
              <material diffuseColor='1 0 0'></material>
            </appearance>
          </shape>
        </transform>
      </scene>
      <background skyColor="0.3 0.3 0.3"></background>
    </x3d>
  </body>
</html>
```

In Abbildung 3.3 wird die aus dem oben beschriebenen X3DOM-Code resultierende Szene im Browser dargestellt.

3.4 XML3D

XML3D[8] ermöglicht die Integration von 3D Inhalten in eine Standard HTML Seite, sodass kein Plugin zur Darstellung im Browser benötigt wird und der Zugriff auf alle Objekte im DOM möglich ist. Die Bibliothek basiert auf WebGL und ist mit den meisten modernen Browsern kompatibel [17]. Außerdem wird Gebrauch von existierenden Web Technologien wie CSS, JavaScript, DOM und DOM Events gemacht [3]. Auch HTML5 Elemente werden wiederverwendet, wie zum Beispiel der `<img>` Tag, um eine Textur anzuwenden [17]. Über JavaScript und die DOM-API lassen sich Elemente und Attribute hinzufügen, verändern und entfernen.

3.4.1 Struktur und Code-Beispiel

Die auf .xhtml endende Datei besteht aus einer Standard HTML Struktur. Der XML3D-Code, der den Inhalt der Szene erstellt, kann direkt in der XHTML-Datei im `<body>` Tag eingebunden werden, wie in folgendem Code-Beispiel gezeigt wird:

```
93 <html>
94   <head>
95     <script type="text/javascript" src="http://www.xml3d.org/xml3d/
       script/xml3d.js"></script>
96   </head>
97   <body>
98     <xml3d id="MyXml3d" style="width: 300px; height: 300px; border: 1px
       solid gray" xmlns="http://www.xml3d.org/2009/xml3d">
99       <mesh type="triangles">
100        <int name="index">0 1 3  1 2 3  0 1 5  0 5 4  1 2 6  1 6 5  2 3
       7  2 7 6  3 0 4  3 4 7  4 5 7  5 6 7</int>
101        <float3 name="position">1 -1 -1  1 1 -1  -1 1 -1  -1 -1 -1  1 -1
        1  1 1 1  -1 1 1  -1 -1 1</float3>
102        <float3 name="normal">0 0 1  0 0 1  0 0 1  0 0 1  0 0 1  0 0 1
       0 0 1  0 0 1</float3>
103        <float3 name="diffuseColor">1 0 0</float3>
104      </mesh>
105    </xml3d>
106 </body>
107 </html>
```

Im Datei-Kopf in Zeile 95 wird zuerst die XML3D Bibliothek eingebunden, um diese verwenden zu können. Das `<xml3d>` Element in Zeile 98 definiert die Größe und das Erscheinungsbild des Render-Bereichs. In diesem Beispiel wird ein rechteckiger Bereich mit einer Breite von 300px und einer Höhe von 300px erzeugt, der von einem grauen Rahmen mit einer Stärke von 1px umrandet ist. Über das `<mesh>` Element wird der Szene ein einfaches Objekt hinzugefügt (siehe Zeile 99). Das `type` Attribut spezifiziert,

[8] http://xml3d.org

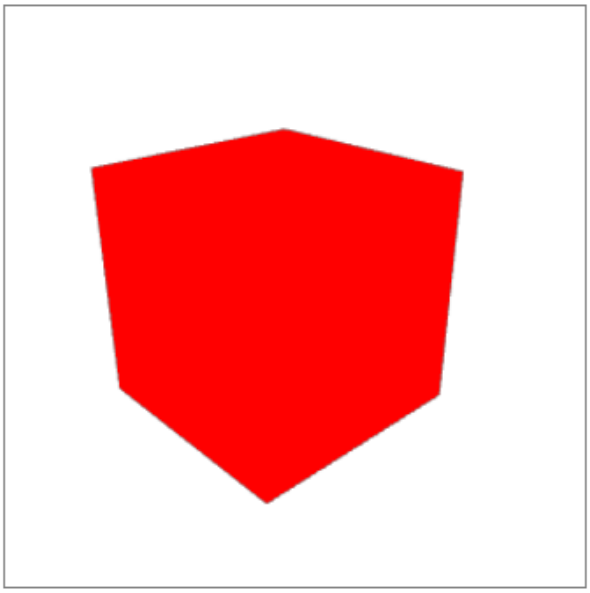


Abbildung 3.4: Das Resultat des XML3D-Codes im Browser.

welche Art von Mesh angezeigt werden soll. Dieses Attribut muss einen der folgende Werte annehmen: `tristrips`, `triangles`, `lines`, `linestrips`, `points` oder `derived`. In diesem Beispiel wird ein Dreieck angelegt. Dieses besteht aus Sets mit jeweils drei Eckpunkten. Das erste Set beschreibt das erste Dreieck, das zweite das zweite Dreieck usw. [17]. Innerhalb des `<mesh>` Elements in den Zeilen 100 bis 103 befinden sich sogenannte *Value* Elemente. Ein `<float3>` Element enthält eine Liste mit Gleitkommazahlen, die durch Leerzeichen getrennt sind. Diese werden als eine Reihe von 3D Vektoren interpretiert [17]. In diesem Beispiel werden über dieses Element die Positionen, sowie die Vertex Normalen gesetzt, um eine Box anstelle eines Dreiecks anzuzeigen, da kein anderer passender Geometrie-Typ zur Verfügung steht. Zusätzlich wird über das `<int>` Element noch der Index des Mesh beschrieben.

In Abbildung 3.4 ist das Ergebnis des oben beschriebenen Codes im Browser zu sehen. Auf der Webseite von XML3D[9] wird eine Liste von weiteren XML3D Beispielen zur Verfügung gestellt.

3.5 Vergleich

Abschließend werden nun in Tabelle 3.1 einige Eigenschaften nochmals zusammengefasst und verglichen. Während VRML eine einfache Textsprache

[9] http://xml3d.org/examples

Tabelle 3.1: Vergleich VRML, X3D, X3DOM, XML3D.

	VRML	**X3D**	**X3DOM**	**XML3D**
Syntax	Textsprache	XML	HTML/XML	HTML/XML
Plugin erforderlich	Ja	Ja	Nein	Nein
auf WebGL basierend	Nein	Nein	Ja	Ja
Geometrien	Box, Kegel, Zylinder, Kugel	Box, Kegel, Zylinder, Kugel, Text	Box, Kegel, Zylinder, Kugel, Text	Dreieck, weitere in Planung [17]
Objekt aus Datei laden	Nein	Nein	Nein	JSON
Animationen	Ja	Ja	Ja	Ja
Inline-Styles Unterstützung	keine	keine	keine	`display`, `transform`

ist, basieren X3D, X3DOM und XML3D auf der XML Syntax. Der Vorteil von X3DOM und XML3D gegenüber den Anderen besteht darin, dass kein Plugin installiert werden muss, um eine 3D Szene im Browser darzustellen. Alle Elemente können also direkt im HTML Grundgerüst im `<body>` Tag erstellt werden. Das Laden von Objekten aus Dateien, wie Three.js es ermöglicht, wird nur von XML3D aus einer JSON Datei unterstützt. XML3D unterstützt zusätzlich die zwei Standard CSS-Eigenschaften `display` und `transform`, die über *Inline-Styles* dem Objekt zugewiesen werden können. Jedoch werden derzeit nur Dreiecke von XML3D unterstützt, die aber beispielsweise zu einer Box umgeformt werden können. Keine der vier Herangehensweisen stellt eine Stylingsprache zur Verfügung. Die Struktur kann somit nicht vom Aussehen getrennt werden.

Kapitel 4

Eigener Ansatz und technisches Design

In diesem Kapitel wird ein neuer Ansatz einer Auszeichnungs- und Stylingsprache für die Erstellung von 3D Szenen im Web beschrieben, sowie der Aufbau des Projekts gezeigt.

4.1 Grundidee und Ziele

Ziel des Projekts ist es, die Entwicklung von 3D Szenen so einfach wie möglich zu machen, sowie Struktur, Aussehen und Funktionalität zu trennen. Außerdem soll es auch Entwicklern ohne Vorkenntnisse in 3D im Web ermöglicht werden, ohne viel Aufwand und Training 3D Szenen zu erstellen. Deshalb wird für die Umsetzung die bekannte HTML- und CSS-Schreibweise gewählt, da diese nicht nur einfach zu erlernen ist, sondern auch einen großen Bekanntheitsgrad unter Frontend-Entwicklern hat. Zusätzlich soll es durch diesen neuen Ansatz einfacher werden, den Code zu lesen, ihn zu verändern und zu erweitern, da sich alle Elemente der Szene in der HTML-Datei und alle Styles und Animationen im Stylesheet befinden.

Die Erstellung einer 3D Szene erfolgt nun in zwei Schritten. Zuerst kommt die neu entwickelte Auszeichnungssprache zum Einsatz. Über Custom HTML-Tags, die vom HTML Transpiler unterstützt werden, können 3D Objekte zur Szene hinzugefügt werden. Diese Objekte können eine ID und eine Klasse besitzen, über die das Element dann selektiert und gestaltet werden kann. Dies geschieht über die neue Stylingsprache, beziehungsweise mithilfe des CSS Transpilers.

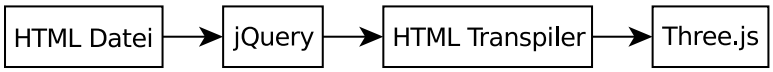


Abbildung 4.1: Ablauf der Übersetzung von HTML-Tags in Three.js Code.

4.2 Auszeichnungssprache

Um das Markup von Styling und Funktionalität zu trennen, werden neue Custom HTML-Tags (siehe Abschnitt 4.2.2) vom HTML Transpiler unterstützt. Mit deren Hilfe können einfache Geometrien, wie zum Beispiel eine Kugel oder ein Würfel, in eine Szene geladen werden. Außerdem besteht die Möglichkeit, Objekte aus Dateien zu laden. So können beispielsweise komplexere Objekte in 3D Programmen wie Blender[1] oder Maya[2] erstellt werden, als .obj Datei exportiert und über das neu unterstützte HTML-Element `<custom-object>` in die Szene geladen werden. Neben den Tags für die Erstellung von diversen Geometrien stehen auch noch drei weitere Tags zur Verfügung. Das `<grid>` Element wird verwendet, um ein Gitternetz für eine bessere Orientierung in der Szene hinzuzufügen und der `<view-point>` Tag, um die Ausrichtung der Kamera zu verändern. Für die Verschachtelung (siehe Abschnitt 4.2.3) von HTML Elementen wird der `<object3d>` Tag verwendet.

4.2.1 Aufbau und Ablauf

Zuerst werden alle Objekte, die zur Szene hinzugefügt werden sollen, über die neu entwickelten HTML-Tags innerhalb des `<body>` Elements der HTML-Datei eingefügt. Über jQuery[3] werden anschließend alle Tags innerhalb des `<body>` Tags selektiert und dann an den HTML Transpiler weitergegeben. Dieser ruft mithilfe von Namenskonventionen die zum Tag gehörige Three.js Funktion auf, über die das Objekt erzeugt und zur Szene hinzugefügt wird (siehe Abbildung 4.1).

4.2.2 Unterstützte Tags

In Tabelle 4.1 werden nun alle neuen HTML-Tags, die vom Transpiler unterstützt und in Three.js Code übersetzt werden können, aufgelistet und beschrieben. Wie die Objekte, die über diese Tags erstellt werden, im Browser dargestellt werden wird in Abbildung 4.2 gezeigt.

[1] https://www.blender.org
[2] http://www.autodesk.de/products/maya
[3] https://jquery.com

Tabelle 4.1: Unterstützte Custom HTML-Tags.

Tag Name	**Beschreibung**	**Attribute**	**Beispiel**
simple-sphere	Erstellt ein einfaches Kugel Objekt. Standardposition: x=0, y=0, z=0; Standardgröße: x=1, y=1, z=1; Standardfarbe: #ffffff;	id, class	`<simple-sphere id="sphere" class= "sphere_red"> </simple-sphere>`
simple-cube	Erstellt ein einfaches Würfel Objekt. Standardposition: x=0, y=0, z=0; Standardgröße: x=1, y=1, z=1; Standardfarbe: #ffffff;	id, class	`<simple-cube id="cube" class= "cube_red"> </simple-cube>`
simple-circle	Erstellt einen Kreis. Standardposition: x=0, y=0, z=0; Standardgröße: x=1, y=1, z=1; Standardfarbe: #ffffff;	id, class, radius, segments	`<simple-circle id="circle" class= "circle_red" radius="2" segments="30"> </simple-circle>`
simple-cylinder	Erstellt einen Zylinder. Standardposition: x=0, y=0, z=0; Standardgröße: x=1, y=1, z=1; Standardfarbe: #ffffff;	id, class, radiusTop, radiusBottom, height, radiusSegments	`<simple-cylinder id="cylinder" radiusTop="0.5" radiusBottom= "0.5" height="2" radiusSegments= "3"> </simple-cylinder>`

simple-torus	Erstellt einen Ring. Standardposition: x=0, y=0, z=0; Standardgröße: x=1, y=1, z=1; Standardfarbe: #ffffff;	id, class, radius, tube, radialSegments, tubularSegments, arc	`<simple-torus id="torus" class="torus_red" radius="3" tube="1" radialSegments= "8" tubularSegments= "50" arc="7"> </simple-torus>`
custom-object	Erstellt ein benutzerdefiniertes Objekt aus der angegebenen Datei. Standardposition: x=0, y=0, z=0;	id, class, src	`<custom-object id="tree" src="objects/tree .obj"> </custom-object>`
grid	Erstellt ein Gitternetz.	id, class, size, step, color	`<grid size="5" step="1" color="#ffffff"> </grid>`
view-point	Setzt die Position der Kamera.	coordX, coordY, coordZ	`<view-point coordX="0" coordY="3" coordZ="20"> </view-point>`
objects3D	Alle Objekte die verschachtelt werden sollen, müssen sich innerhalb dieses Elements befinden.	id	`<objects3D id= "objectsContainer"> </objects3D>`

4.2.3 Verschachtelung

Durch die Verschachtelung von Objekten soll die Vererbung von Eigenschaften gewährleistet werden. Wird also ein Tag in einen anderen verschachtelt und der Eltern-Tag beispielsweise entlang der X-Achse verschoben, so soll auch das Kind-Element mit dem Eltern-Element verschoben werden. Dasselbe gilt für Veränderungen der Positionen, Skalierungen und Rotationen. Lediglich Texturen und Materialien werden nicht weitervererbt. Alle Objekte, die ineinander verschachtelt werden sollen, müssen sich innerhalb des

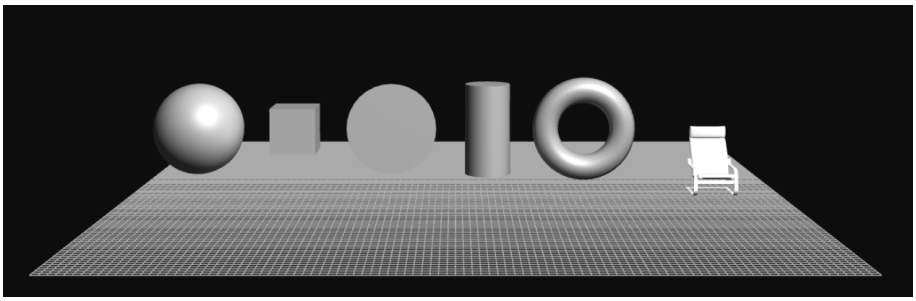


Abbildung 4.2: Aussehen aller über Custom HTML-Tags erstellten Objekte. Von links nach rechts: `<simple-sphere>`, `<simple-cube>`, `<simple-circle>`, `<simple-cylinder>`, `<simple-torus>`, `<custom-object>`.

`<objects3D>` Containers mit der ID `objectsContainer` befinden. Außerdem muss jedes Eltern-Element eine ID besitzen, um die Kind-Elemente eindeutig zuordnen zu können. Ein Beispiel dafür wird in Abschnitt 4.4 gezeigt.

4.3 Stylingsprache

Die Grundidee ist es, in der gewohnten CSS Syntax über neu zur Verfügung gestellte 3D Eigenschaften, Styles auf Objekte anzuwenden. Dafür werden die beiden Selektoren ID (#) und Klasse (.) unterstützt. Über diese werden die Elemente aus dem DOM selektiert. Wie in gewöhnlichem HTML kann jedes Element über eine eindeutige ID verfügen. Zusätzlich steht das *Class*-Attribut zur Verfügung, welches im Gegensatz zu gewöhnlichem HTML zur Zeit des Prototyps aber nur eine einzelne Klasse pro Objekt erlaubt. IDs werden verwendet, um spezifische Eigenschaften anzuwenden und Klassen, um Styles wiederzuverwenden. Beispielsweise kann im Stylesheet eine Klasse angelegt werden, die eine Textur setzt. Diese Klasse kann dann auf mehrere Objekte angewandt werden, die dieses Aussehen annehmen sollen. Wird eine Eigenschaft durch eine ID und eine Klasse deklariert, so wird die in der ID deklarierte gesetzt. Enthalten mehrere gleich gewichtete Selektoren (mehrere IDs oder mehrere Klassen) die selbe Eigenschaft, so kommt die im Stylesheet zuletzt deklarierte zum Zug. Weitere Selektoren und eine erweiterte Kaskade[4] sind derzeit noch nicht implementiert.

4.3.1 Aufbau und Ablauf

Das Anwenden der neu entwickelten CSS-Eigenschaften auf Objekte erfolgt im Hintergrund über vier Schritte. Zuerst werden alle Styles aus dem Style-

[4] https://wiki.selfhtml.org/wiki/CSS/Kaskade

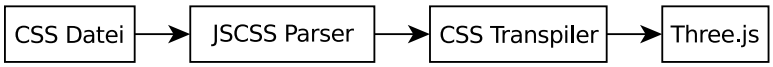


Abbildung 4.3: Ablauf der Übersetzung von 3D CSS in Three.js Code.

sheet ausgelesen. Diese werden dann vom JSCSS Parser geparst und an den CSS Transpiler weitergegeben. Dort wird der CSS-Code in JavaScript-Code übersetzt und mithilfe von Three.js das Styling auf die Objekte angewandt (siehe Abbildung 4.3).

4.3.2 Styling-Regeln

Wie bereits erwähnt, wird im Gegensatz zu gewöhnlichem CSS das Selektieren eines Objekts, auf das eine CSS-Eigenschaft angewandt werden soll, derzeit nur begrenzt vom CSS Transpiler unterstützt.

Gültige Regeln

Eigenschaften können auf mehrere Objekte angewandt werden, indem die Selektoren der einzelnen Objekte mit Beistrichen voneinander getrennt werden. In folgendem Code-Beispiel wird für das Objekt mit der ID `cube1`, das Objekt mit der ID `cube2` und alle Objekte mit der Klasse `cube` die Materialfarbe gesetzt.

```
108 #cube1, #cube2, .cube {
109   material-color: #ff0000;
110 }
```

Ungültige Regeln

Im Gegensatz zu Standard-CSS können Elemente derzeit nur über eine ID oder Klasse selektiert werden. Andere Selektoren wie beispielsweise Typselektoren, Attributselektoren oder Kindselektoren werden derzeit noch nicht unterstützt und auch das Selektieren eines Elements über mehrere IDs und Klassen ist zurzeit noch nicht möglich. In folgendem Beispiel wird versucht, einem Objekt mit der ID `cube1`, das auch die Klasse `cube` besitzt, eine Eigenschaft zuzuweisen.

```
111 #cube1.cube {
112   material-color: #ff0000;
113 }
```

Diese Eigenschaft kann jedoch derzeit nur auf einen der beiden Selektoren angewandt werden.

Das Selektieren verschachtelter Elemente, wie in folgendem Code-Beispiel, ist derzeit ebenfalls noch nicht möglich.

Tabelle 4.2: Unterstützte Styling Eigenschaften.

Eigenschaft	Beschreibung	Wert	Beispiel
size-x/y/z	Größe des Objekts.	Float	`size-x:2.5;`
scale-x/y/z	Skalierung des Objekts.	Float	`scale-x:2.5;`
position-x/y/z	Position des Objekts.	Float	`position-x:3.5;`
rotate-x/y/z	Rotation des Objekts in Grad.	Integer Grad	`rotate-x:90deg;`
material-color	Farbe für Basis-Shader des Mesh.	RGB hexadezimal	`material-color: #ff0000;`
texture	Textur zur Applikation auf das Mesh.	String mit Pfad zur Datei	`texture:"assets /textures /wood.jpg";`
shininess	Helligkeit des Glanzlichts. Je höher der Wert, desto stärker das Highlight des Objekts durch das Licht. Standardwert ist 30.	Integer	`shininess:50;`

```
114 #cube1 .cube {
115   material-color: #ff0000;
116 }
```

Hier wird versucht, dem Element mit der Klasse `cube`, das in einem Element mit der ID `cube1` verschachtelt ist, eine Materialfarbe zu setzen. Eine Lösung für dieses Problem wäre, dem inneren Element eine ID zuzuweisen, um dieses eindeutig selektieren zu können.

4.3.3 Unterstützte Eigenschaften

Wie bereits bekannt, können beispielsweise die Größe und die Hintergrundfarbe eines `<div>` Elements im 2D Kontext mithilfe von CSS geändert werden. Über die neu entwickelten CSS-Eigenschaften können nun auch 3D Objekte einfach angepasst und zum Beispiel die Größe oder Farbe eines Würfels verändert werden. In Tabelle 4.2 werden alle vom CSS Transpiler zum Zeitpunkt dieser Arbeit unterstützten Styling Eigenschaften aufgelistet und kurz beschrieben.

Tabelle 4.3: Unterstützte Animationen.

Eigenschaft	Beschreibung	Wert	Beispiel
fade-in/fade-out	Lässt ein Objekt innerhalb der angegebenen Zeitspanne ein- oder ausblenden.	Integer Millisekunden	`fade-in: 500ms;`
translate-x/y/z	Verschiebt ein Objekt um den angegebenen Wert entlang der lokalen X-, Y- oder Z-Achse.	Float Position, Integer Millisekunden	`translate-x: 2,3000ms;`
rotation-x/y/z	Lässt ein Objekt innerhalb der gegebenen Zeitspanne um den angegebenen Wert um die X-, Y- oder Z-Achse rotieren.	Integer Grad, Integer Millisekunden	`rotation-x: 180deg,3000ms;`
spin-x/y/z	Lässt ein Objekt dauerhaft um die X-, Y- oder Z-Achse rotieren.	String forward/backward oder left/right, je nach Achse	`spin-x: "forward";`

4.3.4 Animationen

Zusätzlich zum Styling können auch Animationen im 3D Raum, wie jene im 2D Raum über CSS3, auf Objekte angewandt werden. Mögliche Animationen sind das Ein- und Ausblenden, sowie das Verschieben und Rotieren von Objekten entlang der X-, Y- und Z-Achse. Über die in Tabelle 4.3 beschriebenen Eigenschaften können diese im Stylesheet den Objekten zugewiesen werden.

4.4 Beispiel-Szene

In diesem Abschnitt wird nun abschließend die Implementierung einer kompletten Beispiel-Szene mit der Auszeichnungs- und Stylingsprache gezeigt. Diese sieht im Browser wie in Abbildung 4.4 gezeigt aus.

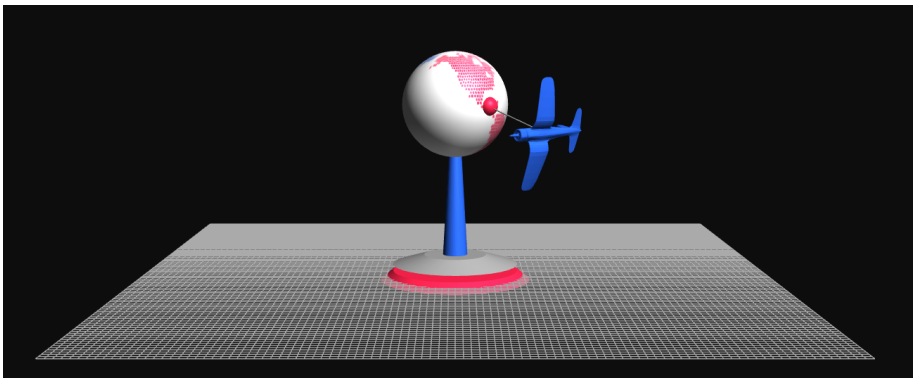


Abbildung 4.4: Aussehen der fertigen Beispiel-Szene.

Zuerst werden alle Elemente über HTML-Tags erzeugt (siehe Code-Zeile 117 bis 136). In diesem Fall besteht die Szene aus neun Objekten innerhalb des `<objects3d>` Elements, die teilweise verschachtelt werden, um das Styling beziehungsweise die Animationen später einfacher anwenden zu können.

```
117 ...
118 <body>
119   <div id="canvas">
120     <objects3d id="objectsContainer">
121       <grid size="5" step="0.1" color="#aaaaaa"></grid>
122       <view-point coordX="0" coordY="4" coordZ="22"> </view-point>
123       <simple-cylinder id="bottom-plate" radiusTop="1" radiusBottom="1.2
      " height="0.15" radiusSegments="32"></simple-cylinder>
124         <simple-cylinder id="bottom-ground" radiusTop="0.5" radiusBottom
      ="1" height="0.15" radiusSegments="32"></simple-sphere>
125         <simple-cylinder id="rod" radiusTop="0.06" radiusBottom="0.2"
      height="2.5" radiusSegments="32"></simple-cylinder>
126         <simple-sphere id="balloon">
127           <simple-sphere id="holder">
128             <simple-cylinder id="rope" radiusTop="0.01" radiusBottom="
      0.01" height="1.3" radiusSegments="32">
129               <custom-object class="plane" src="assets/plane.obj"></
      custom-object>
130             </simple-cylinder>
131           </simple-sphere>
132         </simple-sphere>
133     </objects3d>
134   </div>
135 </body>
136 ...
```

Danach werden diese Elemente in folgendem Code von Zeile 137 bis 176 über die neuen 3D CSS-Eigenschaften positioniert und angepasst.

```
137 #bottom-plate {
138   position-y: 0.1;
139   material-color: #fd234c;
```

```
140 }
141
142 #bottom-ground {
143   position-y: 0.25;
144 }
145
146 #rod {
147   position-y: 1.25;
148   material-color: #2964eb;
149 }
150
151 #balloon {
152   position-y: 2.5;
153   size-x: 1.7;
154   size-y: 1.7;
155   size-Z: 1.7;
156   material-color: #ffffff;
157   texture: "assets/textures/worldmap.jpg"
158 }
159
160 #rope {
161   position-y: -0.7;
162 }
163
164 #holder {
165   material-color: #fd234c;
166   size-x: 0.3;
167   size-z: 0.3;
168   size-y: 0.3;
169   position-z: 0.8;
170   rotate-x: -10deg;
171 }
172 .plane {
173   material-color: #2964eb;
174   position-y: -2.7;
175   position-x: 0.2;
176 }
```

Nachdem die statische Szene fertig implementiert wurde, werden nun zwischen Zeile 177 und 183 einige der Elemente animiert, sodass sich die Weltkugel dreht und vom Flugzeug umkreist wird.

```
177 #balloon {
178   spin-y: left;
179 }
180
181 #holder {
182   rotation-x: -75deg, 7000ms;
183 }
```

Das Element mit der ID `balloon` wird dauerhaft um die Y-Achse rotiert und das Objekt mit der ID `holder` wird innerhalb von sieben Sekunden -75 Grad um die X-Achse rotiert. Die Animation wird aufgrund der Verschachtelung in der HTML-Datei automatisch auf alle Kind-Elemente des Objekts

angewandt. Somit muss zum Beispiel die Rotations-Animation nur auf das Element mit der ID `holder` angewandt werden und alle verschachtelten Elemente, das Seil (`<simple-cylinder>` mit der ID `rope`) und das Flugzeug (`<custom-object>` mit der ID `plane`), rotieren mit.

Kapitel 5

Implementierung

In diesem Kapitel wird die Umsetzung des neuen Ansatzes erklärt und anhand von Code Beispielen verdeutlicht. Die Implementierung erfolgte in drei Schritten. Zuerst wurde mit der Implementierung des HTML Transpilers begonnen. Anschließend wurde nach einem CSS Parser gesucht und zum Schluss der CSS Transpiler entwickelt.

5.1 HTML Transpiler

Dieses Kapitel beinhaltet die Implementierung des in Abschnitt 4.2.1 bereits grob beschriebenen Ablaufs der Übersetzung von HTML-Tags in Three.js Code. Alle für den HTML Transpiler notwendigen Funktionen werden im Namespace `threeDHTMLTranspiler` erstellt. In Abbildung 5.1 wird der Aufbau des HTML Transpilers gezeigt.

Zu Beginn wird die Funktion `assignEventhandler` (siehe Code-Zeilen 184 bis 201) ausgeführt.

```
184 assignEventhandler: function () {
185   $(document).on("documentIsReadyEvent", function () {
186     var htmlElements = document.getElementById('objectsContainer');
187     var elementsArray = [htmlElements];
188     threeDHTMLTranspiler.parseObjects(elementsArray, null);
189
190     for (var i = 0; i < threeDHTMLTranspiler.objectStack.length; ++i) {
191       var element = threeDHTMLTranspiler.objectStack[i].element;
192       var parentId = threeDHTMLTranspiler.objectStack[i].parentId;
193       threeDHTMLTranspiler.objectFunctions[element.localName](
194         parentId ? parentId : false,
195         element.id ? element.id : false,
196         element.className ? element.className : false,
197         element.attributes.src ? element.attributes.src : false,
198         element.attributes ? element.attributes : false
199       );
200     }
201 });
```

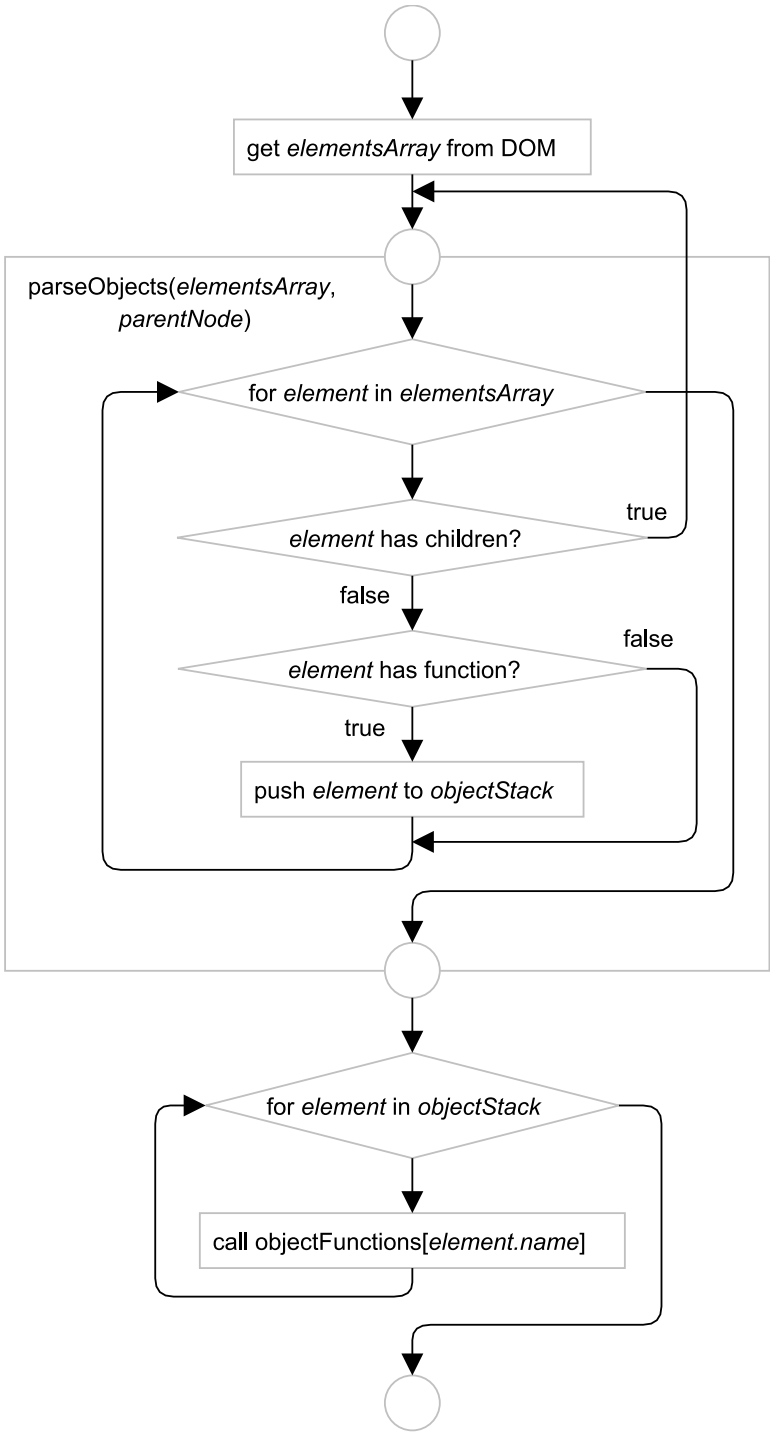


Abbildung 5.1: Aufbau des HTML Transpilers.

Wenn `documentIsReadyEvent` ausgelöst wird, werden in Zeile 186 alle Tags innerhalb des `<objects3d>` Elements mit der ID `objectsContainer`, mittels jQuery selektiert und in ein Array gespeichert. Dieses wird daraufhin in Zeile 188 der `parseObjects` Funktion übergeben. Es ist wichtig, hier ein Array zu übergeben, da sich `parseObjects` durch den möglichen rekursiven Aufruf für verschachtelte Elemente als ersten Parameter ein Array erwartet.

```
202 parseObjects: function (htmlElements, parentNode) {
203   for (var i = 0; i < htmlElements.length; ++i) {
204     var element = htmlElements[i];
205     if (element.children.length > 0) {
206       threeDHTMLTranspiler.parseObjects(element.children, element);
207     }
208     if (threeDHTMLTranspiler.objectFunctions.hasOwnProperty(element.
        localName)) {
209       threeDHTMLTranspiler.objectStack.unshift({
210         "element": element,
211         "parentId": element.parentElement.id
212       });
213     }
214   }
215 }
```

Um die Verschachtelung von mehreren HTML-Elementen zu unterstützen, werden der `parseObjects` zwei Parameter übergeben. Der erste beinhaltet alle HTML-Elemente und der zweite das jeweilige Eltern-Element. In Zeile 206 wird `parseObjects` rekursiv aufgerufen, solange bis von einem Element kein Kind-Element mehr gefunden wird. Ab Zeile 208 wird dann für jedes Element die ID und das dazugehörige Eltern-Element in das Array `objectStack` gespeichert. Hierfür wird die Funktion `unshift` verwendet, weil das Element an die erste Stelle gesetzt werden muss. Danach wird über dieses Array in den Zeilen 190 bis 200 im Eventhandler des `documentIsReadyEvent`s iteriert und via Namenskonvention die zum Tag gehörige `objectFunction` aufgerufen. Dieser wird als erster Parameter die ID des Elternelements übergeben, falls das Element in der HTML-Datei verschachtelt wurde. Weiters werden der Funktion noch, falls vorhanden, die ID des Elements, der Name der Klasse, der Pfad zur .obj Datei und alle Attribute, die im HTML-Tag gesetzt wurden, mitgegeben. In den Zeilen 216 bis 227 werden die `objectFunctions` für die `<simple-cube>`, `<simple-sphere>` und `<custom-object>` Tags gezeigt.

```
216 objectFunctions: {
217   "simple-cube": function (parentId, cssId, cssClass, src) {
218     threeDHTMLTranspiler.insertSimpleCube(parentId, 1, 1, 1, cssId,
      cssClass);
219   },
220   "simple-sphere": function (parentId, cssId, cssClass, src) {
221     threeDHTMLTranspiler.insertSimpleSphere(parentId, 1, 30, 30, cssId,
      cssClass);
222   },
```

```
223   "custom-object": function (parentId, cssId, cssClass, src) {
224     threeDHTMLTranspiler.insertCustomObject(parentId, cssId, cssClass,
        src);
225   },
226   ...
227 }
```

In dieser `objectFunction` wird dann die Funktion aufgerufen, die das Objekt erstellt und in die Szene einfügt. Für jeden Tag wurde somit eine eigene Funktion implementiert. Ein Beispiel für die zum `<simple-cube>` Tag gehörige Funktion `insertSimpleCube`, die einen Würfel erzeugt und hinzufügt, ist in folgendem Code zu sehen (siehe Zeile 228 bis 245).

```
228 insertSimpleCube: function (parentId, width, length, height, cssId,
        cssClass){
229   var material = new THREE.MeshPhongMaterial({color: 0xa0a0a0});
230     var mesh = new THREE.Mesh(new THREE.BoxGeometry(width, height,
        length), material);
231   var object = new THREE.Object3D();
232   object.cssId = cssId;
233   object.cssClass = cssClass;
234   object.add(mesh);
235
236   if (parentId != null && parentId != "objectsContainer") {
237     Designer.scene.traverse(function (sceneObject) {
238       if (sceneObject.cssId == parentId) {
239         sceneObject.add(object);
240       }
241     });
242   } else {
243     Designer.scene.add(object);
244   }
245 }
```

In Zeile 229 wird zunächst das Material gesetzt. Hierfür wurde das *MeshPhongMaterial*[1] verwendet, da dieses im Gegensatz zu *MeshBasicMaterial*[2] und *MeshLambertMaterial*[3] auch Glanzpunkte und interpolierte Normalen unterstützt. Danach wird in Zeile 230 eine `BoxGeometry` erstellt, die dem darunter erzeugten Objekt zusammen mit einer cssId und cssClass zugewiesen wird. In Zeile 236 wird geprüft, ob das soeben erstellte Objekt einem anderen Element zugewiesen werden soll. Anschließend wird die gesamte Szene nach einem Eltern-Element durchsucht und das neu erstellte Objekt darin verschachtelt. Somit übernimmt dieses die Manipulation des Eltern-Elements.

[1] http://threejs.org/docs/#Reference/Materials/MeshPhongMaterial
[2] http://threejs.org/docs/#Reference/Materials/MeshBasicMaterial
[3] http://threejs.org/docs/#Reference/Materials/MeshLambertMaterial

5.2 CSS Parser

Nach der Implementierung des HTML Transpilers wurde nach einem CSS Parser gesucht, um das 3D Stylesheet einlesen zu können und Zugriff auf die CSS Eigenschaften, sowie deren Werte zu bekommen. Nach kurzer Onlinerecherche wurde der JSCSS Parser gefunden, der einfach und schnell zu implementieren ist und alle Anforderungen erfüllt.

5.2.1 JSCSS Parser

Der JSCSSP ist ein mit JavaScript entwickelter CSS Parser. Dadurch ist dieser mit allen Browsern kompatibel. Er parst alle Styles einer Datei in einen String und speichert ein eigenes *CSS Object Model*, welches in Abbildung 5.2) gezeigt wird. Der Parser ist einfach zu verwenden und kann leicht verändert werden, da er nicht auf *Regular Expressions*[4] basiert [10].

In Abbildung 5.2 wird die, vom Parser nach dem Parsen der CSS-Datei zur Verfügung gestellte, Struktur gezeigt. Für die Umsetzung des in Abschnitt 5.3 beschriebenen CSS Transpilers wurden die drei Eigenschaften `property`, `mSelectorText` und `valueText` verwendet. In `mSelectorText` wird der CSS-Selektor gespeichert, auf dessen Objekt das Styling angewandt wird. In diesem Fall die ID mit dem Wert `#cube`. Die Eigenschaft `property` enthält die CSS-Eigenschaft `material-color`. Der Wert, in diesem Beispiel die Farbe grün (`#00ff00`), wird in der Eigenschaft `valueText` gespeichert. `getParsedStyles` ist eine der primären Funktionen des CSS Transpilers, welche den JSCSS Parser aufruft (siehe Zeile 246 bis 256).

```
246 getParsedStyles: function (callback) {
247   var source;
248   $.get("style/3Dstyles.css", function (data) {
249       source = data;
250       var parser = new CSSParser();
251       var sheet = parser.parse(source, false, true);
252       if (callback) {
253           callback(sheet);
254       }
255   });
256 }
```

Zuerst wird in Zeile 248 die CSS-Datei, die geparst werden soll, via AJAX eingelesen. Danach wird in Zeile 250 der CSSParser instanziiert, anschließend wird mithilfe der Funktion `parse` der Inhalt der zuvor angegebenen Datei geparst und in die Variable `sheet` gespeichert. In Zeile 251 werden der Funktion `parse` drei Parameter übergeben. Der erste enthält den Inhalt der zuvor eingelesenen Datei als String. Der zweite bestimmt, ob Leerzeichen enthalten bleiben. In diesem Fall werden diese nicht benötigt und somit

[4] *Regular Expressions* sind eine Art Filterkriterium, mit deren Hilfe Zeichenketten analysiert werden können.

Abbildung 5.2: Vom JSCSS Parser zur Verfügung gestellte Struktur.

`false` übergeben. Der letzte Parameter wird auf `true` gesetzt, um Kommentare zu unterstützen. Durch die Asynchronität eines AJAX-Aufrufs kann `getParsedStyles` keinen Wert zurückliefern, weshalb in den Zeilen 252 bis 254 ein `Callback` aufgerufen wird.

5.3 CSS Transpiler

Für die Umsetzung des CSS Transpilers wurden die in Abschnitt 5.2.1 bereits beschriebenen Eigenschaften aus dem *CSS Object Model* verwendet. In Abbildung 5.3 wird der Aufbau des CSS Transpilers gezeigt.

Zu Beginn wird in der `init` Funktion die zuvor beschriebene Funktion `getParsedStyles` (siehe Zeile 246 bis 256) aufgerufen.

```
257 init: function () {
258   ...
259   threeDCSSTranspiler.getParsedStyles(function (sheet) {
260     threeDCSSTranspiler.data.rules = sheet.cssRules;
261     threeDCSSTranspiler.data.rules = threeDCSSTranspiler.data.rules.sort
      (threeDCSSTranspiler.compareBySpecificity);
262
263     threeDCSSTranspiler.data.rules.forEach(function (rule) {
264       var objects = threeDCSSTranspiler.getObjectsBySelector(rule);
265
266       objects.forEach(function (object) {
267         rule.declarations.forEach(function (declaration) {
268           threeDCSSTranspiler.applyPropertyToObject(object,declaration);
269         });
```

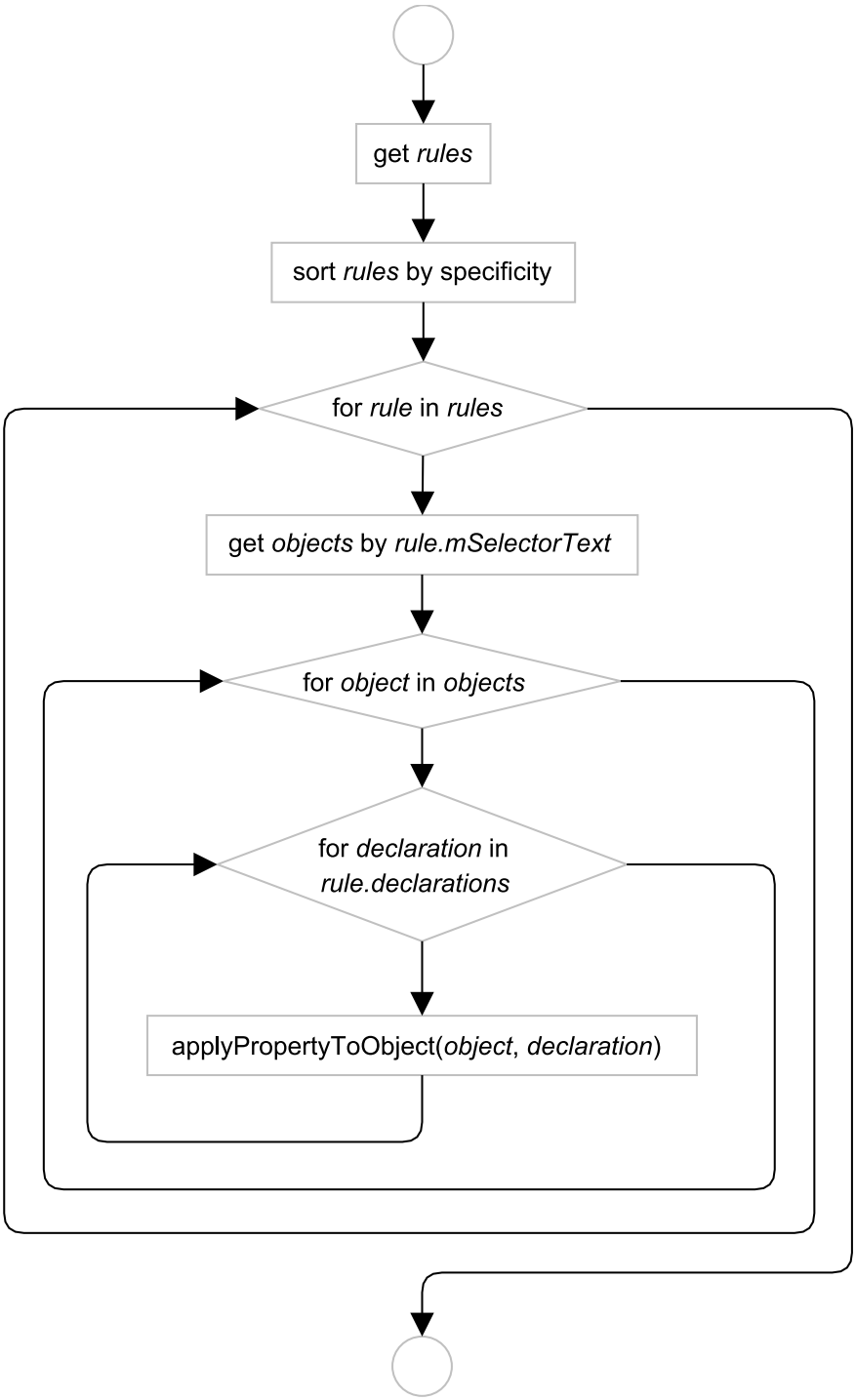


Abbildung 5.3: Aufbau des CSS Transpilers.

```
270       });
271     });
272   });
273 }
```

In Zeile 261 erfolgt anschließend innerhalb der `sort` Funktion der Aufruf der Funktion `compareBySpecificity`. In den Zeilen 274 bis 280 wird gezeigt, wie die zuvor eingelesenen Styles in dieser Funktion verglichen und entsprechend geordnet werden, um wie auch in gewöhnlichem CSS-Code die höhere Gewichtung einer ID im Vergleich zu einer Klasse zu unterstützen (siehe Abschnitt 4.3).

```
274 compareBySpecificity: function (a, b) {
275   if (a.mSelectorText[0] == '#' && b.mSelectorText[0] == '.')
276     return 1;
277   if (b.mSelectorText[0] == '#' && a.mSelectorText[0] == '.')
278     return -1;
279   return 0;
280 }
```

Im nächsten Schritt wird über alle geparsten `cssRules` aus der 3D CSS-Datei iteriert und die Funktion `getObjectsBySelector` aufgerufen. In der Funktion `getObjectsBySelector` von Zeile 281 bis 315 werden alle Objekte, die im Stylesheet über deren ID oder Klasse selektiert und angepasst werden, in das Array `objects` gespeichert.

```
281 getObjectsBySelector: function (rule) {
282   var regExp = /[#.][^,]{1,}/g;
283   var selectors = [];
284   var selector = null;
285   selector = regExp.exec(rule.mSelectorText);
286
287   do {
288     selectors.push(selector[0]);
289     selector = regExp.exec(rule.mSelectorText);
290   } while (selector != null);
291
292   var objects = [];
293   selectors.forEach(function (selector) {
294     switch (rule.mSelectorText[0]) {
295       case '#':
296         var idName = selector.substr(1);
297         Designer.scene.traverse(function (obj) {
298           if (obj.cssId == idName) {
299             objects.push(obj);
300           }
301           return;
302         });
303         break;
304       case '.':
305         var className = rule.mSelectorText.substr(1);
306         Designer.scene.traverse(function (obj) {
307           if (obj.cssClass == className) {
```

```
308             objects.push(obj);
309           }
310         });
311         break;
312     }
313   });
314   return objects;
315 }
```

In der `init` Funktion in Zeile 266 wird zuerst über alle `objects` und dann in Zeile 267 über alle `rule.declarations` iteriert. Innerhalb dieser beiden Schleifen erfolgt in Zeile 268 der `applyPropertyToObject` Aufruf. Dieser Funktion werden die Objekte und deren Styling als Parameter übergeben und in Zeile 321 die dazugehörige `propertyFunction` über Namenskonvention aufgerufen.

```
316  applyPropertyToObject: function (object, declaration) {
317   if (declaration.property == undefined) {
318     return;
319   }
320
321   threeDCSSTranspiler.propertyFunctions[declaration.property](object,
        declaration);
322 }
```

Hierbei ist zu erwähnen, dass einige Eigenschaften auf das Mesh angewandt werden und einige auf das Objekt selbst. Ein Beispiel für die Anwendung auf ein Mesh ist die Funktion `texture` (siehe Zeile 325 bis 330). Der Funktionsname `"texture"` muss hier exakt mit dem Wert, den die Eigenschaft `property` nach dem Parsen der Styles enthält, übereinstimmen. Deshalb wird dieser ebenfalls unter Anführungszeichen geschrieben.

```
323 propertyFunctions: {
324   ...
325   "texture": function (object, declaration) {
326     var textureLoader = new THREE.TextureLoader();
327     var url = declaration.valueText.replace(/\"/g, '');
328     object.children[0].material.map = textureLoader.load(url);
329     object.children[0].material.needsUpdate = true;
330   }
331   ...
332 }
```

In Zeile 328 wird die Textur mithilfe des Materials auf das Mesh des Objekts (`children[0]`) angewandt, da das verwendbare Material von der Art der Geometrie abhängig ist. Im Gegensatz dazu wird beispielsweise bei der Funktion `scale-x` der Wert, der im CSS gesetzt wird, dem Objekt selbst (`object`) zugewiesen (siehe Zeile 336).

```
333 propertyFunctions: {
334   ...
335   "scale-x": function (object, declaration) {
336     object.scale.x = declaration.valueText;
```

```
337   }
338   ...
339 }
```

Dies ist notwendig, um bestimmte CSS-Eigenschaften an alle verschachtelten Elemente zu vererben. Alle Kind-Elemente, also Polygon Mesh und verschachtelte Elemente, befinden sich im lokalen Koordinatensystem des Eltern-Elements und übernehmen Position, Skalierung und Rotation. Das Vererben von Texturen und Materialien wird dadurch nicht ermöglicht, weshalb diese auf alle Kind-Elemente erneut angewandt werden müssen.

5.3.1 Animationen

Für jede Animation wurde, wie auch für jede neue CSS-Eigenschaft, eine eigene Funktion geschrieben, die jedoch zusätzlich eine rekursive Komponente beinhaltet. In den Zeilen 340 bis 358 wird der Code für eine `fade-in` Animation gezeigt.

```
340 "fade-in": function (object, declaration) {
341   object.children[0].material.transparent = true;
342   object.children[0].material.opacity = 0;
343   var time = parseInt(declaration.valueText);
344   var lastUpdate = 0;
345   var fps = 0;
346
347   function step(timestamp) {
348     if (lastUpdate > 0) {
349       fps = 1000 / (timestamp - lastUpdate);
350       object.children[0].material.opacity += 1 / (time / (1000 / fps));
351     }
352     if (object.children[0].material.opacity < 1) {
353       window.requestAnimationFrame(step);
354     }
355     lastUpdate = timestamp;
356   }
357   step();
358 }
```

Zuerst wird die Transparenz des Materials aktiviert, wodurch das Objekt als letztes gerendert wird, um dahinterliegende Objekte durchscheinen lassen zu können. Mithilfe der `opacity` Eigenschaft kann der Wert dann von 0 bis 1 gesteuert werden. Die schrittweise Animation selbst wird durch die indirekt rekursive Funktion `step` ausgeführt. Da die `requestAnimationFrame` Funktion keine stabile Bildrate liefert, wird in den Zeilen 348 bis 351 versucht, durch eine Anpassung des Schrittbetrags, die Schwankungen der Framerate auszugleichen. Dabei werden zuerst die Frames pro Sekunde unter Berücksichtigung der Zeitdifferenz des letzten und vorletzten Frames berechnet. Anschließend wird in Zeile 350 herausgefunden, in wie viele Schritte die Animation unterteilt werden muss, um nach der im Stylesheet angegeben Zeit das Objekt vollständig eingeblendet zu haben. Dafür wird die angegebene

Zeit durch die Framedauer dividiert. Zum Schluss wird der zu erreichende Wert (in diesem Fall 1 für 100% Deckkraft) durch den Schrittbetrag dividiert, um den nächsten Gesamtwert der Animation zu berechnen.

Kapitel 6

Ergebnisse und Evaluierung

Nach Fertigstellung der Auszeichnungs- und Stylingsprache wurden diese mithilfe einiger Testpersonen getestet und evaluiert, um die in Kapitel 1 beschriebene Forschungsfrage beantworten zu können. Die Ergebnisse werden in diesem Kapitel erläutert.

6.1 Methode

Anhand einer Test-Szene sollte herausgefunden werden, ob die Entwicklung von 3D Szenen mithilfe einer Styling- und Auszeichnungssprache vereinfacht werden kann. Dafür wurden sechs Testpersonen gebeten, eine Test-Szene zuerst mit Three.js und danach die selbe Szene mithilfe der Auszeichnungs- und Stylingsprache zu implementieren. Beide Projekte wurden bereits im Vorhinein angelegt und die Basis-Szene (siehe Abbildung 6.1 und Code-Zeile 1 bis 42) mit Three.js erstellt. Auch die dafür notwendigen Objekt-Dateien wurden zur Verfügung gestellt.

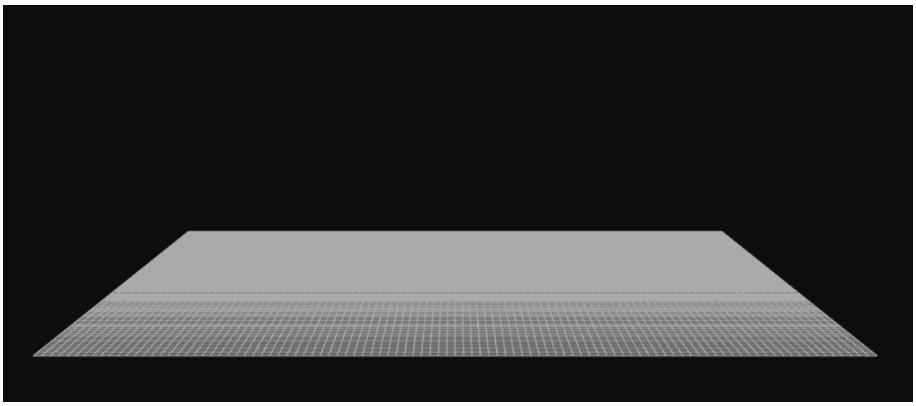


Abbildung 6.1: Ausgangssituation der Test-Szene.

Die Testpersonen mussten anschließend den fehlenden Three.js Code beziehungsweise beim neuen Ansatz den fehlenden HTML- und CSS-Code schreiben. Für die Umsetzungen hatten die Probanden jeweils 45 Minuten Zeit. Es wurde abwechselnd mit der Implementierung mit Three.js und dem neuen Ansatz begonnen. Die Zeit wurde beide Male gestoppt und nach der abgelaufenen Zeit der Fortschritt dokumentiert, falls eine Testperson die Szene innerhalb der vorgegebenen Zeit nicht fertigstellen konnte.

Nachdem die Testpersonen die Szene vollständig implementiert hatten oder die Zeit abgelaufen war wurden den Testpersonen noch einige Fragen gestellt, die zusätzlich zu den persönlichen Angaben wie Alter, Ausbildung, Beruf und genauer Tätigkeit im Beruf folgendermaßen lauteten:

- Haben Sie bereits Erfahrung mit 3D im Web und wenn ja, welche?
- Haben Sie bereits Erfahrung mit Three.js und wenn ja, welche?
- Mit welcher Technik fiel Ihnen die Umsetzung der Szene leichter?
- Was hat Ihnen an der Umsetzung mit der als leichter empfundenen Technik besser gefallen als mit der anderen?
- Wie hat Ihnen die Umsetzung mit der Auszeichungs- und Stylingsprache insgesamt gefallen?
- Gibt es Verbesserungs- oder Erweiterungsvorschläge für die Auszeichnungs- und Stylingsprache?

6.1.1 Testpersonen

Für die Evaluierung des Projekts wurden Webentwickler, aber auch Personen mit wenig Erfahrung im Web-Bereich ausgewählt. Wie aus Tabelle 6.1 entnommen werden kann, arbeiten die meisten Probanden als Webentwickler und verfügen über einen Bachelor- oder Master-Abschluss im IT-Bereich. Während die ersten drei Testpersonen hauptsächlich JavaScript Kenntnisse vorweisen konnten, lagen die Stärken bei den Testpersonen 4 bis 6 eher im Bereich HTML und CSS. Lediglich zwei Probanden hatten bereits Vorkenntnisse in 3D im Web beziehungsweise auch mit der Bibliothek Three.js. Eine weitere Testperson, die als UI/UX Designer arbeitet, hatte abgesehen von HTML- und CSS-Grundkenntnissen keinerlei Erfahrung mit Webentwicklung.

6.1.2 Aufgabenstellung

Den zuvor vorgestellten Testpersonen wurde folgende Aufgabe gestellt:

Erstellen Sie eine Szene, die einen Wagen und einen Baum beinhaltet (siehe Abbildung 6.2). Der Wagen soll aus den beiden Dateien *rad.obj* und *wagen.obj* zusammengebaut und auf die Hälfte der Größe skaliert werden. Die Textur *metall.jpg* soll auf das Wagen Objekt angewandt werden. Der Baum besteht aus zwei einfachen Geometrien – einem transformierten Wür-

Tabelle 6.1: Testpersonen im Überblick.

Testperson	**Ausbildung**	**berufliche Tätigkeit**	**Vorkenntnisse Web 3D**
Testperson 1	MSc, Interactive Media, FH Hagenberg	Fullstack Webentwickler, JavaScript, Meteor, PHP, Contao	WebGL, Three.js, blend4web
Testperson 2	MSc. Digitale Medien, FH Hagenberg	Frontend Webentwickler, JavaScript, Ember.js, .NET	WebGL, Three.js
Testperson 3	BSc, Software Engineering, FH Hagenberg	Frontend Webentwickler, JavaScript, Ember.js, .NET	keine
Testperson 4	MSc. Digitale Medien, FH Hagenberg	Frontend Webentwickler, HTML, CSS, Ember.js	keine
Testperson 5	Kolleg für Mediengestaltung und Werbung, BFI Linz	UI/UX Designer	keine
Testperson 6	BSc, Software Engineering, TU Wien	Frontend Webentwickler, CSS, Ember.js	keine

fel und einer Kugel – und soll positioniert werden (5,0,0). Die Kugel soll grün (#009436) und das Rechteck braun (#944F00) eingefärbt werden. Positionieren Sie den gesamten Wagen in der Mitte des Gitternetzes (0,0,0) und verschieben Sie ihn innerhalb von fünf Sekunden entlang der X-Achse, sodass dieser den Baum berührt. Auch die Reifen sollen sich innerhalb dieser fünf Sekunden mitdrehen. Für die Szene wurden folgende Elemente zur Verfügung gestellt:

- Datei für das Grundgerüst des Wagens (*assets/wagen.obj*).
- Datei mit einem Reifen und einer halben Achse (*assets/rad.obj*).
- Datei mit Textur (*assets/textures/metall.jpg*).

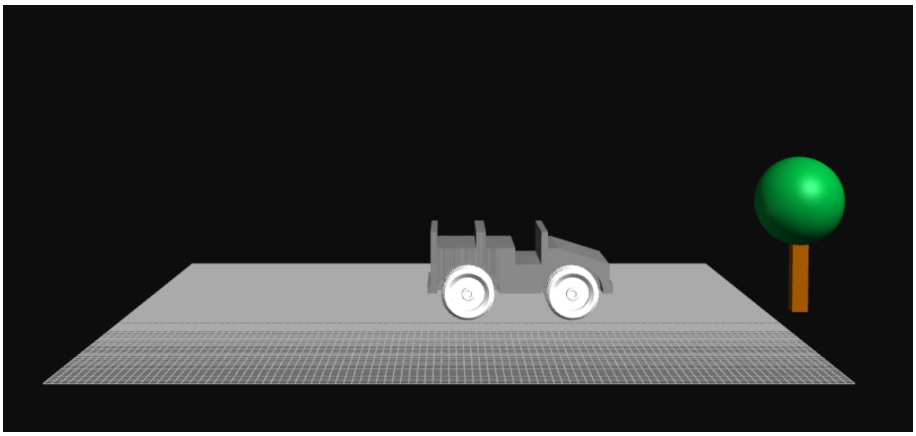


Abbildung 6.2: Die fertige Test-Szene.

6.2 Ergebnisse

In diesem Abschnitt werden nun alle gemessenen Zeiten für die Umsetzungen der Szene angeführt und verglichen. In Tabelle 6.2 werden die Gesamtzeiten beider Umsetzungen gegenübergestellt.

Es stellte sich heraus, dass jede der sechs Probanden bei der Umsetzung mit der Auszeichnungs- und Stylingsprache schneller war, als mit Three.js. Testperson 1 benötigte sogar nur knapp ein Drittel, Testperson 2 nur die Hälfte der Zeit mit dem neuen Ansatz im Vergleich zur anderen Implementierung. Bei den anderen Testpersonen können nur Vermutungen angestellt werden, da diese die Umsetzung mit Three.js nicht abschließen konnten, mit dem neuen Ansatz jedoch schon. Es wird vermutet, dass alle Probanden mindestens doppelt so viel Zeit für die vollständige Umsetzung mit reinem Three.js Code benötigt hätten, als bei der Umsetzung mit der Auszeichnungs- und Stylingsprache.

Um dies zu verdeutlichen, wird in den Tabellen 6.3 und 6.4 veranschaulicht, wie viel Zeit für die jeweiligen Schritte der Implementierung benötigt wurde. Zuerst die Zeit, die gebraucht wurde, um die Objekte zu erstellen und in die Szene zu laden, gefolgt von der Zeit, die für das Styling benötigt wurde und in der letzten Spalte die Dauer zum Animieren der Objekte.

Wie in Tabelle 6.3 ersichtlich, gelang es vier Testpersonen nicht, die Szene mit Three.js fertig zu implementieren. Testperson 3 gelang es innerhalb der 45 Minuten nur die Objekte zu erstellen und die Räder zu animieren. Der Rest des Stylings und die Animationen wurden nicht umgesetzt. Testperson 4 konnte alle Objekte erstellen und das Styling der einzelnen Objekte fertigstellen, jedoch in der vorgegebenen Zeit keine Animationen implementieren. Testperson 5 erstellte alle Objekte und stylte den Baum fertig, jedoch fehlten am Ende der Zeit das Styling und die Animation des Wagens. Test-

Tabelle 6.2: Gesamtzeiten der beiden Umsetzungen im Vergleich in Minuten.

Testperson	**Gesamtzeit Three.js**	**Status Three.js**	**Gesamtzeit neuer Ansatz**	**Status neuer Ansatz**
Testperson 1	36:20	abgeschlossen	12:45	abgeschlossen
Testperson 2	42:10	nicht abgeschlossen	18:30	abgeschlossen
Testperson 3	45:00 (Umsetzung wurde nach 45 Minuten abgebrochen)	nicht abgeschlossen	42:10	abgeschlossen
Testperson 4	45:00 (Umsetzung wurde nach 45 Minuten abgebrochen)	nicht abgeschlossen	23:40	abgeschlossen
Testperson 5	45:00 (Umsetzung wurde nach 45 Minuten abgebrochen)	nicht abgeschlossen	35:25	abgeschlossen
Testperson 6	45:00 (Umsetzung wurde nach 45 Minuten abgebrochen)	nicht abgeschlossen	22:40	abgeschlossen

person 6 gelang es, alle Objekte zu erstellen und zu stylen, jedoch reichte die vorgegebene Zeit nicht aus, um den Wagen und die Räder zu animieren.

Mit dem neuen Ansatz der Auszeichnungs- und Stylingsprache konnten im Gegensatz zur Three.js Umsetzung alle Testpersonen innerhalb der vorgegebenen Zeit von 45 Minuten die Szene fertigstellen. Die schnellste Zeit liegt dabei bei 12 Minuten und 45 Sekunden, die langsamste bei 42 Minuten und 10 Sekunden (siehe Tabelle 6.4). Große Zeitunterschiede zeigten sich vor allem bei der Erstellung der Objekte. Alle Testpersonen benötigten maximal die Hälfte der Zeit mit der neuen Auszeichnungssprache im Gegensatz

Tabelle 6.3: Zeiten der Umsetzung mit Three.js in Minuten (*nicht vollständig implementiert).

Testperson	**Gesamtzeit**	**Objekte erstellen**	**Styling**	**Animation**
Testperson 1	36:20	11:25	18:45	06:10
Testperson 2	42:10	09:35	23:15	09:10
Testperson 3	45:00*	13:40	31:20*	-
Testperson 4	45:00*	38:20	06:40*	-
Testperson 5	45:00*	23:25	21:35*	-
Testperson 6	45:00*	19:00	24:10	01:50*

Tabelle 6.4: Zeiten der Umsetzung mit dem neuen Ansatz in Minuten.

Testperson	**Gesamtzeit**	**Objekte erstellen**	**Styling**	**Animation**
Testperson 1	12:45	01:56	08:19	02:30
Testperson 2	18:30	04:00	09:10	05:20
Testperson 3	42:10	06:00	27:10	09:00
Testperson 4	23:40	03:20	17:20	03:00
Testperson 5	35:25	07:30	22:40	05:15
Testperson 6	22:40	15:00	04:20	02:40

zur Umsetzung mit reinem Three.js Code. Besonders groß war die Zeitdifferenz bei Testperson 4, die für die Erstellung der Objekte mit Three.js 38 Minuten und 20 Sekunden benötigte und mit dem neuen Ansatz 3 Minuten 20 Sekunden. Dies ist vermutlich auf die berufliche Tätigkeit der Testperson zurückzuführen, da diese im Beruf hauptsächlich mit HTML und CSS arbeitet und wenig Erfahrung mit JavaScript hat.

Ein deutlicher Unterschied ist auch bei der Zeit für das Styling der Objekte zu erkennen. Drei der Testpersonen konnten das Styling mit Three.js nicht abschließen, mit dem neuen Ansatz jedoch schon. Auch hier konnte jede der Testpersonen mindestens die Hälfte der Zeit bei der Entwicklung mit der Stylingsprache einsparen. Lediglich zwei Personen gelang es, die Animationen mit Three.js fertigzustellen, da beide bereits Erfahrung mit Three.js hatten. Dabei brauchte Testperson 1 mit Three.js 6 Minuten und 10 Sekunden und nur 2 Minuten 30 Sekunden mit der Stylingsprache, also ein Drittel der Zeit.

Dies zeigt, dass selbst Personen mit Erfahrung im Bereich Web 3D beziehungsweise auch mit der JavaScript Bibliothek Three.js bei der Implementierung mit dem neuen Ansatz deutlich schneller waren. Sowohl bei der Erstellung der Objekte, als auch bei der Implementierung des Stylings oder

der Animationen konnten somit mit den beiden neu entwickelten Sprachen deutlich kürzere Zeiten erreicht werden, als bei der Implementierung mit reinem Three.js Code.

6.3 Benutzerbeurteilung

Nach Fertigstellung der Szene mit beiden Ansätzen wurden den Testpersonen noch einige Fragen bezüglich der beiden Umsetzungen gestellt.

Mit welcher Technik fiel Ihnen die Umsetzung der Szene leichter? Alle Testpersonen, egal ob mit Three.js Erfahrung oder ohne, gaben an, dass ihnen die Umsetzung mit der Auszeichnungs- und Stylingsprache viel leichter fiel als mit reinem Three.js Code.

Was hat Ihnen an der Umsetzung mit der als leichter empfundenen Technik besser gefallen? Als Gründe dafür wurden die Einfachheit des Codes, die vertraute Art der CSS und HTML Syntax, sowie die Trennung von Struktur, Aussehen und Funktionalität genannt. Außerdem meinten vier der Probanden, dass viel Zeit bei der Entwicklung eingespart werden kann, da weniger Code-Zeilen nötig sind als mit einer JavaScript-Lösung, um das gewünschte Ergebnis zu erzielen.

Wie hat Ihnen die Umsetzung mit der Auszeichungs- und Stylingsprache insgesamt gefallen? Drei der Testpersonen empfanden die Umsetzung als ausgezeichnet, die anderen drei als sehr gut[1].

6.4 Verbesserungs- und Erweiterungsmöglichkeiten

Zuletzt wurden die Testpersonen noch befragt, ob es Verbesserungs- oder Erweiterungsvorschläge für die Auszeichnungs- und Stylingsprache gibt. Jeder Proband nannte mindestens zwei Vorschläge. Alle Ideen und Anregungen der Testpersonen werden in diesem Abschnitt näher beschrieben.

6.4.1 Auszeichnungssprache

Um die Auszeichnungssprache noch besser nutzen zu können, wären weitere Geometrien wie 3D Text oder Octahedron, die auch von Three.js zur Verfügung gestellt werden, hilfreich. Von zwei Testpersonen wurde der Wunsch

[1] Bewertungsschlüssel: ausgezeichnet, sehr gut, gut, befriedigend, mangelhaft, schlecht.

geäußert, die neu entwickelten Tags als sogenannte *Standalone Tags* zu implementieren, um Code und Zeit zu sparen. Dabei handelt es sich um Elemente, die keinen Inhalt haben und deshalb nur aus einem Tag bestehen anstatt aus Anfangs- und End-Tag. Außerdem wurde die Kurzschreibweise, wie beispielsweise HAML[2], von Testperson 4 erwähnt, mit deren Hilfe Code eingespart werden kann. Der HTML-Code von Zeile 117 bis 136 würde dann folgendermaßen aussehen:

```
359 ...
360 %body
361   #canvas
362     %objects3d#objectsContainer
363       %grid{:color=>"#aaaaaa", :size=>"5", :step=>"0.1"}
364       %view-point{:coordX=>"0", :coordY =>"4", :coordZ=>"22"}
365
366       %simple-cylinder#bottom-plate{:height=>"0.15", :radiusBottom=>"1.2
       ", :radiusSegments=>"32", :radiusTop=>"1"}
367       %simple-cylinder#bottom-ground{:height=>"0.15", :radiusBottom=>"1"
       , :radiusSegments=>"32", :radiusTop => "0.5"}
368         %simple-cylinder#rod{:height=>"2.5", :radiusBottom=>"0.2", :
       radiusSegments=>"32", :radiusTop=>"0.06"}
369         %simple-sphere#balloon
370           %simple-sphere#holder
371             %simple-cylinder#rope{:height=>"1.3", :radiusBottom=>"0.01",
        :radiusSegments=>"32", :radiusTop=>"0.01"}
372               %custom-object.plane{:src=>"assets/plane.obj"}
373 ...
```

Dies würde den Code zusätzlich kürzer und leichter lesbar machen. Allen Tags sollen außerdem mehrere IDs und Klassen gegeben werden können. Das `<objects3d>` Element, in das zurzeit alle Objekte verschachtelt werden müssen, um mit Verschachtelungen arbeiten zu können, soll außerdem nicht mehr benötigt werden. Da einige Testpersonen Probleme mit dem Verständnis des, teils durch das CSS gedrehten, Koordinatensystems äußerten, soll das Koordinatensystem des Objekts angezeigt werden.

6.4.2 Stylingsprache

Wie in Abschnitt 4.3 erwähnt, ist es derzeit nur möglich ein Objekt im Stylesheet über eine ID oder Klasse anzusprechen. Weitere Selektoren, die im Standard CSS möglich sind, wie zum Beispiel Typselektoren, Attributselektoren, Kindselektoren oder auch Pseudoelemente werden derzeit noch nicht unterstützt. Dies fanden zwei der Testpersonen störend, weshalb diese zukünftig ebenfalls unterstützt werden sollen. Außerdem sollen die beiden in Abschnitt 4.3.2 beschriebenen, ungültigen Regeln unterstützt werden, um Objekte spezifischer selektieren, stylen und animieren zu können. Auch die Kaskade muss dann aufgrund der Unterstützung mehrerer unterschiedlicher

[2] http://haml.info

Selektoren erweitert werden. Zusätzlich wurde der Wunsch geäußert, einige Farben in Form von Namen im CSS-Code zu unterstützen. Dadurch soll zum Beispiel Folgendes, wie auch in Standard CSS für die Eigenschaften `color` und `background-color`, unterstützt werden, ohne den dazugehörigen Hexadezimal oder RGB Farbcode zu kennen:

```
374 #object {
375     material-color: blue;
376 }
```

Auch die Verwendung einer CSS Kurzschreibweise, wie zum Beispiel Sass[3], wäre von Vorteil, wenn die zur Zeit noch ungültigen Regeln aus Abschnitt 4.3.2 unterstützt werden, meinten zwei der Probanden. Die Testpersonen äußerten außerdem den Wunsch, Animationen verzögern zu können, wie es auch in Standard CSS über die `animation-delay` Eigenschaft möglich ist. Somit könnten mehrere Animationen nacheinander beziehungsweise verzögert ausgeführt werden. Auch die aus Standard CSS bekannte Eigenschaft `animation-iteration-count` wäre hilfreich, um Animationen mehrmals hintereinander ausführen zu können, meinte eine der Testpersonen.

[3] http://sass-lang.com

Kapitel 7

Fazit

In dieser Arbeit wurden zu Beginn in Kapitel 2 die technischen Grundlagen, die im Laufe der Arbeit und des Projekts benötigt wurden, beschrieben. Diese sind WebGL und Three.js. Außerdem gibt es noch weitere Ansätze für die Erstellung von 3D Szenen im Browser, die nicht auf JavaScript basieren, wie zum Beispiel VRML, X3D, X3DOM und XML3D. Diese sind reine Auszeichnungssprachen und trennen, wie auch bei Three.js, daher die Struktur, das Aussehen und die Funktionalität nicht voneinander. Außerdem wird für VRML und X3D ein Plugin benötigt, um eine 3D Szene im Web darstellen zu können, da diese im Gegensatz zu den anderen nicht auf WebGL basieren.Deshalb wurde im Zuge dieser Arbeit eine Auszeichnungs- und Stylingsprache entwickelt, die auf Three.js basiert, jedoch Struktur, Aussehen und Funktionalität trennt. Die 3D Objekte werden nun in einer HTML-Datei erstellt und deren Aussehen in einem Stylesheet über neu entwickelte CSS-Eigenschaften bearbeitet. Um dies zu unterstützen, wurde der JSCSS Parser sowie ein HTML Transpiler und ein CSS Transpiler entwickelt. Die Implementierung wurde in Kapitel 5 beschrieben und erklärt.

Die Evaluierung wurde anschließend mithilfe von sechs Testpersonen durchgeführt. Diese wurden gebeten, eine Test-Szene mit Three.js und anschließend mit der Auszeichnungs- und Stylingsprache jeweils innerhalb von 45 Minuten umzusetzen. Die Zeit wurde beide Male gestoppt. Mit Three.js konnten lediglich zwei der sechs Testpersonen die Test-Szene fertigstellen. Mit dem neuen Ansatz gelang allen Personen die vollständige Umsetzung der Szene. Außerdem benötigten alle Probanden maximal die Hälfte der Zeit mit der neuen Auszeichnungssprache, im Gegensatz zur Umsetzung mit reinem Three.js Code. Bei der Befragung gaben alle Testpersonen an, dass ihnen die Umsetzung mit dem neuen Ansatz viel leichter fiel, da ihnen die Art der Syntax vertraut war und weniger Code-Zeilen nötig sind, als mit den anderen zuvor beschriebenen Ansätzen. Drei der Testpersonen bewerteten den neuen Ansatz mit ausgezeichnet, die anderen drei mit sehr gut. Somit stellte sich heraus, dass die Erstellung von 3D Szenen mithilfe der Auszeichungs-

und Stylingsprache vereinfacht werden kann und weniger Zeit in Anspruch nimmt als beispielsweise mit reinem Three.js.

7.1 Ausblick

Das Thema 3D im Web sorgt bereits seit einigen Jahren für Gesprächsstoff. Zwar wird 3D das Internet in den nächsten Jahren nicht beherrschen, jedoch wird die Anzahl von 3D Anwendungen im Web vermutlich weiter steigen.

Zukünftige Anwendungsgebiete für Web 3D

Immer aufwendigere Browser-Spiele und diverse 3D Visualisierungen von Produkten oder diversen Abläufen werden zukünftig im Web zu finden sein. Da auch der Trend immer mehr zu Cloud-basierten Anwendungen anstelle von installierbaren Programmen geht, sind auch Online Versionen von Maya, Blender oder anderen Modellierungsprogrammen denkbar und durchaus mit WebGL realisierbar. Ein bereits bestehendes Beispiel für eine Cloud-basierte 2D und 3D Anwendung mit WebGL ist Autodesk A360[1].

Erweiterungen für die Auszeichnungs- und Stylingsprache

Trotz der sehr positiven Rückmeldungen der Testpersonen äußerten diese, wie in Kapitel 6.4 ausführlich beschrieben, einige Verbesserungs- und Erweiterungsvorschläge. Deshalb soll dieses Projekt weiterentwickelt werden. Die Auszeichnungssprache soll um einige neue Geometrien erweitert werden, aus *Standalone Tags* bestehen, sowie die Verschachtelung verbessert werden. Auch die Stylingsprache soll weiter entwickelt werden und die in Kapitel 4.3.2 beschriebenen ungültigen Regeln und auch einige weitere Selektoren, wie beispielsweise Typselektoren und Kindselektoren, unterstützen. Die Animationen sollen zukünftig verzögert und mehrmals hintereinander ausgeführt werden können. Mit diesen und weiteren Verbesserungen und Erweiterungen soll die Entwicklung von 3D Szenen im Web zukünftig noch einfacher werden.

[1] https://a360.autodesk.com

Anhang A

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

A.1 PDF-Dateien

Pfad: /

Kaupert_Julia_2016.pdf Masterarbeit (Gesamtdokument).

A.1.1 Literatur

Pfad: /Literatur

*.pdf . . . . . Literatur als PDF Dokumente.

A.1.2 Online-Quellen

Pfad: /Online-Quellen

*.pdf . . . . . Online-Quellen als PDF Dokumente.

A.2 Abbildungen

Pfad: /Abbildungen

*.pdf . . . . . Vektorgrafiken.

*.png . . . . . Original Rasterbilder.

Quellenverzeichnis

Literatur

[1] Jos Dirksen. *Learning Three.js: The JavaScript 3D Library for WebGL*. Packt Publishing, 2013 (siehe S. 5).

[2] Tony Parisi. *WebGL: Up and Running*. O'Reilly Media, Inc., 2012 (siehe S. 4, 7, 8).

[3] Kristian Sons. *XML3D – Interactive 3D Graphics for the Web*. 2010 (siehe S. 13).

Online-Quellen

[4] Don Brutzman. *X3D: Extensible 3D Graphics for Web Authors*. 2008. URL: https://diglib.eg.org/bitstream/handle/10.2312/cgems04-11-1355/CGEMS_X3dGraphicsForWebAuthorsModule_Brutzman2008February1.pdf?sequence=1 (besucht am 13. 02. 2016) (siehe S. 9).

[5] Web3D Consortium. *Recommended Standards*. 1999-2016. URL: http://web3d.org/standards/version/ (besucht am 05. 03. 2016) (siehe S. 10).

[6] Michael M. Heck David R. Nadeau John L. Moreland. *Introduction to VRML 97*. 1998. URL: http://www.sdsc.edu/~moreland/courses/Siggraph98/vrml97/slides/mt0016.htm (besucht am 12. 02. 2016) (siehe S. 8).

[7] Alexis Deveria. *Can I use*. 2016. URL: http://caniuse.com/#search=webgl (besucht am 02. 05. 2016) (siehe S. 4).

[8] Daly Realism Don Brutzman. *X3D for Web Authors*. 2013. URL: http://x3dgraphics.com/slidesets/X3dForWebAuthors/TutorialX3dSceneGraph.pdf (besucht am 21. 02. 2016) (siehe S. 7).

[9] e-teaching.org. *VRML/Virtual Reality Modeling Language*. 2015. URL: https://www.e-teaching.org/materialien/glossar/vrml (besucht am 17. 03. 2016) (siehe S. 9).

[10] Daniel Glazman. *JSCSSP a CSS parser in JavaScript*. 2010-2011. URL: http://www.glazman.org/JSCSSP/ (besucht am 09. 03. 2016) (siehe S. 31).

[11] Khronos Group. *OpenGL ES 2.0 for the Web*. 2016. URL: https://www.khronos.org/webgl/ (besucht am 11. 02. 2016) (siehe S. 3).

[12] Khronos Group. *WebGL Specification*. 2014. URL: https://www.khronos.org/registry/webgl/specs/1.0/#1 (besucht am 11. 02. 2016) (siehe S. 3).

[13] Bozana Bokan Manuel Schiewe. *VRML / X3D und 3D-Präsentationen*. URL: http://vrml2.de/ (besucht am 17. 03. 2016) (siehe S. 7).

[14] mrdoob. *Github*. 2016. URL: https://github.com/mrdoob/three.js/ (besucht am 11. 02. 2016) (siehe S. 5).

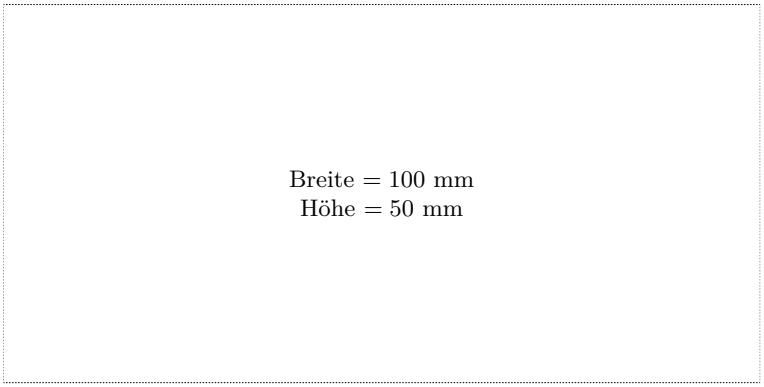
[15] *Three.js - Creating a scene*. 2016. URL: http://threejs.org/docs/#Manual/Introduction/Creating_a_scene (besucht am 11. 02. 2016) (siehe S. 5).

[16] x3dom.org. *X3DOM Documentation: Overview*. 2013. URL: http://doc.x3dom.org/ (besucht am 13. 02. 2016) (siehe S. 12).

[17] XML3D.ORG. *XML3D*. 2016. URL: http://xml3d.org (besucht am 14. 02. 2016) (siehe S. 13–15).

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —