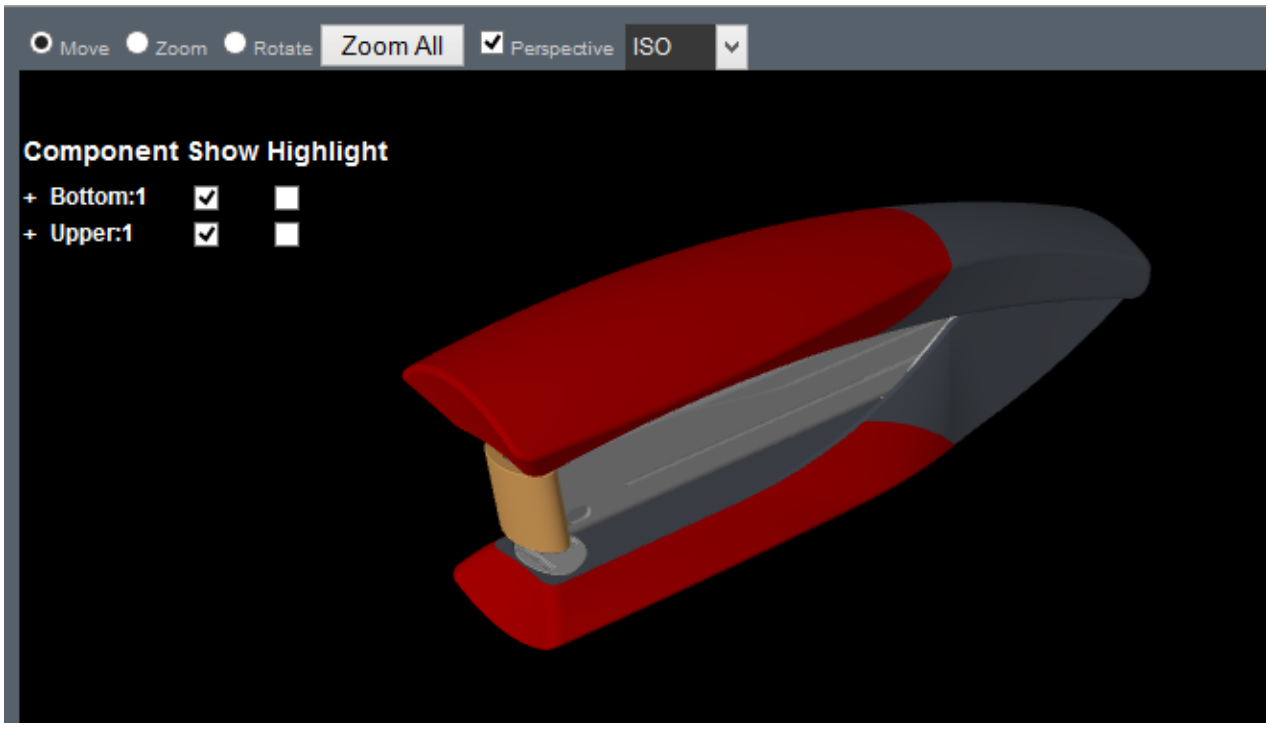
Projektbericht

WebGL-Publisher

von Lukas Mente und Stephan Schneider



Veranstaltung

Multimedia-Technologien

Betreuer

Prof. Martin Schober

Inhaltsverzeichnis

1 Einleitung 1

2 Die Technologie 2

2.1 WebGL 2

2.2 WebGL-Publisher 3

3 Funktionen des WebGL-Publisher 4

3.1 Import in WebGL-Publisher 4

3.2 Bearbeitungsmöglichkeiten im WebGL-Publisher 4

3.3 Export aus dem WebGL-Publisher 5

3.4 Export aus Autodesk Inventor 6

4 Exportierte Dateien 7

4.1 Modell (wpm-Datei) 7

4.2 HTML-Datei 10

4.2.1 Assembly Table 11

5 Praktische Anwendungsfälle 14

5.1 Animationen 14

5.1.1 Erstellte Beispielanimation 16

5.2 Ein- und Ausblenden von Bauteilgruppen 16

5.3 Hervorheben von Bauteilen 21

6 Abgabedateien 24

7 Fazit 25

7.1 Praktikabilität 25

7.2 Weitere Ansätze 26

1 Einleitung

Diese Projektarbeit beschäftigt dem Programm WebGL-Publisher.

Es werden drei grundsätzliche Fragestellungen behandelt:

1. Welche Möglichkeiten bietet das Programm zur Manipulation von CAD-Daten vor und nach dem HTML-Export?
2. Welche über den Standard hinausgehenden zusätzlichen Funktionen können implementiert werden, um den WebGL-Publisher effektiv als Werkzeug zur Erstellung von Multimedia-Dokumentationen und E-Learnings nutzen zu können?
3. Wie sind die Funktionen bezüglich der Praktikabilität in der Technischen Dokumentation zu bewerten und wo gibt es noch Verbesserungspotential?

Um diese Fragen zu beantworten, sind wir wie folgt vorgegangen:

Zunächst haben wir mit dem WebGL-Publisher an sich experimentiert, um zu erfahren, welche Funktionen die Benutzeroberfläche der Software bietet. Anschließend wurden die CAD-Daten exportiert.

Mit Hilfe der API-Beschreibung in der Online-Hilfe des WebGL-Publishers und den auf der WebGL-Publisher-Webseite verfügbaren Beispielen haben wir dann versucht zu verstehen, wie die ausgegebenen HTML-Dateien und 3D-Modellbeschreibungen aufgebaut sind und welche Möglichkeiten sich bieten, um nachträglich durch Programmierung einzugreifen.

Darauf aufbauend haben wir überlegt, welche Funktionen noch wünschenswert wären und wie sich diese Funktionen mit den vorhandenen Möglichkeiten realisieren lassen.

Zu guter Letzt haben wir ein Tutorial angefertigt, das die grundsätzliche Bedienung des WebGL-Publishers ebenso erklärt wie die Umsetzung der im Rahmen des Projekts entwickelten Anwendungsmöglichkeiten.

2 Die Technologie

2.1 WebGL

3D-Modelle nativ im Browser

Die Technologie WebGL ermöglicht es, 3D-Grafiken nativ im Browser darzustellen. WebGL kann mit Hilfe von HTML5-Canvas-Elementen direkt in HTML5-Seiten eingebunden werden. Als HTML5-kompatible Technologie wird WebGL bereits jetzt von den meisten modernen Browsern unterstützt. Auch die Ausgabe von WebGL-Grafiken auf mobilen Endgeräten ist in dem meisten Fällen bereits jetzt möglich.

Khronos Group

Entwickelt wurde WebGL von dem Industriekonsortium Khronos Group, dem unter anderem wichtige Firmen wie Apple, Intel, Samsung und Sony angehören. Wichtige Browser-Hersteller wie Apple, Google und Mozilla sind Mitglieder in der WebGL-Arbeitsgruppe. So können mit WebGL erzeugte Visualisierungen auch für hybride Apps und Web-Apps genutzt werden.

Anwendungsmöglichkeiten

Die Technologie hat zahlreiche Anwendungsmöglichkeiten, angefangen von der interaktiven Darstellung von Diagrammen bis hin zu ausgefeilten 3D-Spielen. Für Architekten, Marketingleute und Technische Redakteure ist WebGL aber vor allem wegen der interaktiven Darstellung von 3D-Modellen im Browser interessant. Der Nutzer kann das 3D-Modell selbst drehen und frei zoomen und so das technische Objekt aus allen Blickwinkeln in Augenschein nehmen. Darüber hinaus werden durch JavaScript-Programmierung viele weitere Funktionalitäten wie Tooltips, Animationen oder das Ein- und Ausblenden von einzelnen Bauteilen möglich. Dadurch kann die Interaktivität und damit auch der didaktische Nutzen der Modelle gesteigert werden.

Hardwarebeschleunigung

Um WebGL im Browser anzeigen zu können, ist es nicht nötig, Plugins oder Zusatzprogramme zu installieren. Ein weiterer Vorteil von WebGL ist, dass die Grafiken hardwarebeschleunigt sind. Das heißt, dass der Browser bei der Darstellung der Modelle direkt auf die Rechenleistung der Grafikkarte zugreifen kann. Die Modelle können dadurch in den meisten Fällen ruckelfrei dargestellt werden.

2.2 WebGL-Publisher

Der WebGL-Publisher ist ein preiswertes kleines Programm von CADMAI Software. Es bietet dem Anwender die Möglichkeit, eine Vielzahl von CAD-Formaten zu importieren. Danach kann der Anwender das 3D-Modell für den Export anpassen und z. B. Bauteile mit neuen Farben versehen oder Tooltipps einfügen.

WebGL auf Knopfdruck

Die wichtigste Funktion des WebGL-Publishers ist jedoch die Exportfunktion. Auf Knopfdruck wird eine HTML-Datei ausgegeben, die das fertige WebGL-3D-Modell enthält. Die HTML-Seite beinhaltet je nach Exportkonfiguration bereits alle wichtigen Schaltflächen, z. B. zur Änderung der Perspektive oder zum Umschalten der Mausfunktion von Zoomen auf Rotieren oder Verschieben.

Zusätzlich sind in der HTML-Datei bereits Funktionen und JavaScript-Bibliotheken enthalten, die es ermöglichen, Veränderungen an der Anzeige des Modells und dem Modell selbst vorzunehmen oder Animationen einzufügen.

WebGL-Publisher-Plugin

Zusätzlich zum eigenständigen WebGL-Publisher gibt es auch ein WebGL-Publisher-Plugin für die Programme Autodesk Revit (Architektur) und Autodesk Inventor (CAD).

Der Export mit diesem Plugin bietet einen großen Vorteil: Auf Wunsch wird zusätzlich eine Baumstruktur mit allen Bauteilen hierarchisch geordnet nach Baugruppen ausgegeben. Mit diesem so genannten Assembly Tree lassen sich Bauteile ganz einfach ein- und ausblenden oder durch Einfärbung hervorheben.

3 Funktionen des WebGL-Publisher

3.1 Import in WebGL-Publisher

CAD-Modelle werden über die Schaltfläche "Öffnen" in den WebGL-Publisher geladen. Dabei werden die folgenden Formate unterstützt:

- "WebGL-Publisher project"-Dateien (*.wpp)
- "CADMAI model"-Dateien (*.cmi, *.cmizip)
- Step-Dateien (*.stp, *.step)
- Iges-Dateien (*.igs)
- Wavefront-Dateien (*.obj)
- 3DS-Dateien (*.3ds)
- STL-Dateien (*.stl)
- DXF-Dateien (*.dxf)

Es ist also möglich, gängige Austauschformate für CAD-Daten zu importieren, was den Web-GL-Publisher zu einem sehr flexiblen Werkzeug macht.

3.2 Bearbeitungsmöglichkeiten im WebGL-Publisher

Über die Schaltfläche "Design" wird das Design-Menü aufgerufen, in dem sich alle Funktionen zum Bearbeiten eines CAD-Modells befinden.

Farbe

Unter der Registerkarte "Farbe" kann Bauteilen eine Farbe zugewiesen werden.

Texturen

Unter der Registerkarte "Texturen" können Bauteilen Texturen zugewiesen werden.

Transparenz

Unter der Registrierkarte "Transparenz" können Bauteile transparent oder unsichtbar gemacht werden.

Shader

Unter der Registrierkarte "Shader" können Bauteilen Texturen zugewiesen werden.

Generischer Shader

Unter der Registrierkarte "Generischer Shader" können bestimmte Rendering-Effekte eingestellt und den Bauteilen zugewiesen werden.

3D-Selektion

Unter der Registrierkarte "3D-Selektion" können Bauteilen Tooltips und Links zugewiesen werden. Diese erscheinen beim Überfahren des jeweiligen Bauteils mit der Maus.

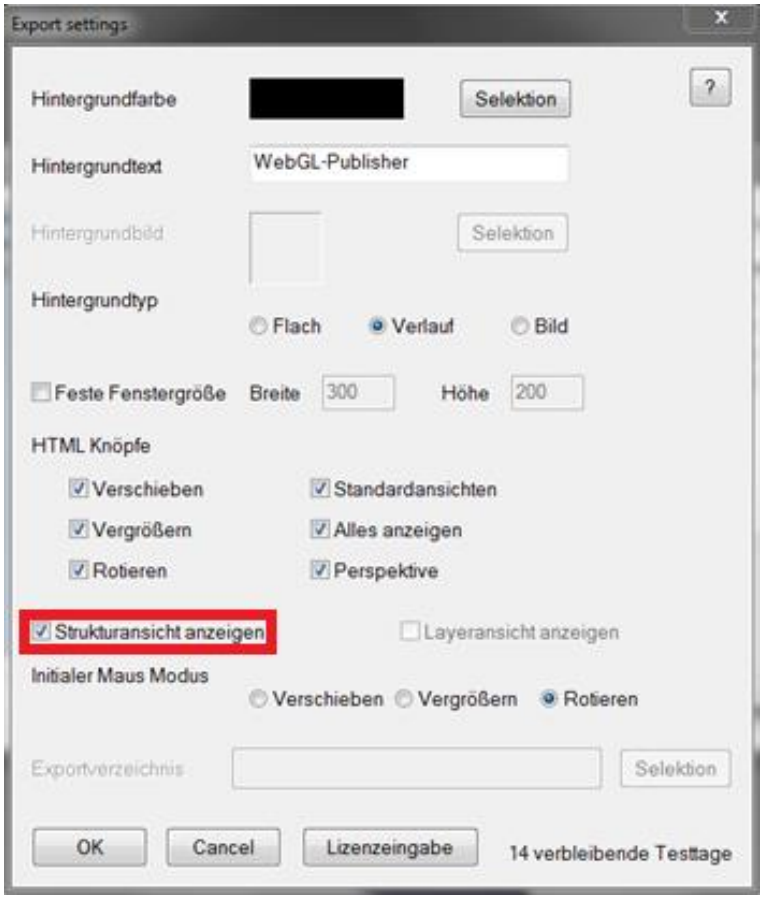
3.3 Export aus dem WebGL-Publisher

Die wichtigste Funktion des WebGL-Publishers ist der Export eines CAD-Modells. Mit der Schaltfläche "Export" wird das Modell umgewandelt und es werden folgende Dateien in das Zielverzeichnis exportiert:

- Das Modell im WebGL-Publisher-Modellformat (.wpm)
- Eine HTML Datei, die das exportierte Modell anzeigt
- Das Verzeichnis CmiBitmaps, das alle benötigten Texturbilder enthält
- Das Verzeichnis CmiScripts, das alle zur Darstellung notwendigen JavaScript-Bibliotheken enthält

3.4 Export aus Autodesk Inventor

Beim Export mit dem WebGL-Publisher-Plugin für den Autodesk Inventor[1] stehen die gleichen Möglichkeiten zur Verfügung wie beim Export aus dem WebGl-Publisher. Zusätzlich besteht allerdings noch die Möglichkeit, die Checkbox "Strukturansicht anzeigen" anzuhaken, die bewirkt, dass die Assembly Table mitexportiert wird[2].



[1] Wir haben uns eine Testversion des Autodesk Inventor auf http://www.autodesk.de/free-trials besorgt. Das WebGl-Publisher-Plugin gibt es als Testversion auf http://www.webgl-publisher.com/OrderDe.html.

[2] Die Checkbox steht auch beim Export aus dem WebGL-Publisher zur Verfügung, erzeugt aber mit den getesteten Modellen aber nur eine leere Assembly Table.

4 Exportierte Dateien

4.1 Modell (wpm-Datei)

Die wpm-Datei im JSON-Datenformat enthält alle Daten des umgewandelten CAD-Modells.

Dabei unterscheidet sich der Aufbau der wpm-Dateien, die direkt aus dem WebGL-Publisher exportiert werden, grundsätzlich von denen, die mit dem Inventor-Plugin erzeugt wurden.

Prolog

Das geht schon mit dem Prolog der Dateien los, auch wenn hier die Unterschiede noch nicht so groß sind. Auffällig ist lediglich, dass beim Inventor-Export die Positionsangaben um den Faktor 10 geringer sind und dass beim Inventor-Export das Feld "mNm" ausgefüllt ist.

```
{\
\"cR\":\"Copyright for model format: CADMAI Software GmbH\",\
\"clNm\":\"cmiModel\",\
\"ver\":\"3.0\",\
\"ct\":\"full\",\
\"mNm\":\"?\",\
\"llP\":[-82.0,-41.5,-16.0],\
\"urP\":[0.001,11.3,80.0],\
\"anims\":[],\
```

Abbildung 1: Prolog beim WebGL-Publisher-Export

```
{\
\"cR\":\"Copyright for model format: CADMAI Software GmbH\",\
\"clNm\":\"cmiModel\"\
,\"ver\":\"3.0\"\
,\"ct\":\"full\"\
,\"mNm\":\"Model1\"\
,\"sRc\":\"Inventor2015\"\
,\"llP\":[-8.2,-4.15,-1.6]\
,\"urP\":[0,1.13,8]\
```

Abbildung 2: Prolog beim Inventor-Export

Bauteilbeschreibungen

Bei der Beschreibung der einzelnen Teile treten dann schon größere Unterschiede auf. Während beim WebGL-Publisher-Export alles kompakt unter einem "clNm"-Element beschrieben wird, gibt es beim Inventor 2 "clNm"-Elemente. Beim Inventor scheinen im zweiten "clNm"-Element die Positionsdaten des Bauteils zu liegen, während der Namen und Farbe des Bauteils im ersten "clNm"-Element festgehalten sind.

Auch interessant sind die Bauteilnamen: Beim WebGL-Publisher haben die Bauteilnamen ("eNm") das Format "s + Zahl", beim Inventor wird der volle Namen des Bauteils (Befestigungsbuegel:1) angegeben, so wie er im CAD-Programm

festgelegt wurde. Außerdem gibt es beim Inventor im zweiten "clNm"-Element auch ein zweites "eNr"-Element, das bei den exportierten Modellen jedoch für alle Bauteile den Wert "body1" trägt.

In den Dateien lässt sich nachträglich auch noch die Farbe des Bauteils ändern, indem man in das Feld "col" die drei RGB-Werte (0-100 für jede Farbe, 100 entspricht 255 im RGB) und einen Transparenzwert einträgt (0.0 = unsichtbar, 1.0 = keine Transparenz).

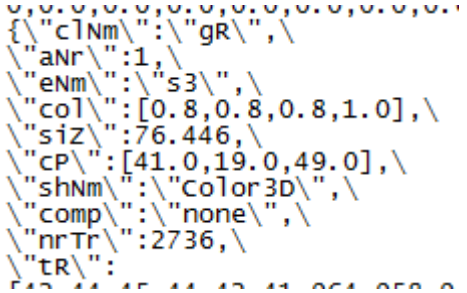


Abbildung 3: Bauteilbeschreibung beim WebGL-Publisher-Export

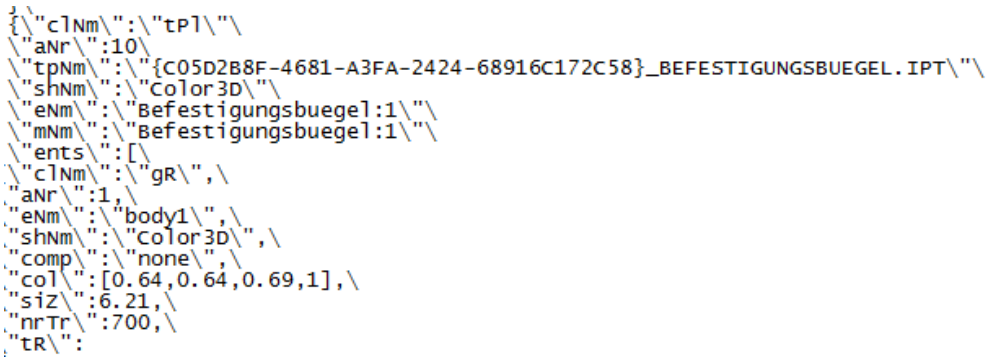


Abbildung 4: Bauteilbeschreibung beim Inventor-Export

Drittes "cLNm"-Element

Zusätzlich werden beim Inventor-Export am Ende der Datei nochmals alle Bauteile in einem dritten "clNm"-Element beschrieben, in dem teilweise die gleichen Dinge stehen wie in den anderen "clNm"-Elementen.

```
,{\"clNm\":\"cMp\"\
,\"aNr\":1\
,\"eNm\":\"Grundplatte:1\"\
,\"shNm\":\"Color3D\"\
,\"tpNm\":\"{1BE45C2F-4511-DCBF-ED44-B293E692E69C}_GRUNDPLATTE.IPT\"\
,\"mNm\":\"Grundplatte:1\"\
,\"urP\":[0,0.85,0]\
,\"llP\":[-8.2,-4.15,-1.6]\
,\"org\":[0,-4.15,0]\
,\"elA\":[-180,0,180]\
,\"col\":[0.75,0.75,0.75,1]\
}\
,{\"clNm\":\"cMp\"\
,\"aNr\":1\
,\"eNm\":\"Befestigungsbuegel:1\"\
,\"shNm\":\"Color3D\"\
,\"tpNm\":\"{C05D2B8F-4681-A3FA-2424-68916C172C58}_BEFESTIGUNGSBUEGEL.IPT\"\
,\"mNm\":\"Befestigungsbuegel:1\"\
,\"urP\":[-1.7,0.1,1.8]\
,\"llP\":[-6.5,-3.4,0]\
,\"org\":[-4.35,-1.19,1.6]\
,\"elA\":[0,0,180]\
,\"col\":[0.75,0.75,0.75,1]\
}\
```


Erkenntnisse

Insgesamt ist die Datenbeschreibung in den wpm-Dateien sehr mysteriös. Als wir am Anfang versucht haben, uns in die Materie einzuarbeiten, haben wir uns die wpm-Dateien angeschaut in der Hoffnung, Erkenntnisse daraus ziehen zu können, was nur teilweise gelungen ist.

Was sich sagen lässt ist, dass die Modelle beim Inventor-Export detaillierter beschrieben und anders aufgebaut sind als beim Export direkt aus dem WebGL-Publisher. Dies dürfte auch erklären, warum der Assembly Browser beim WebGL-Publisher-Export nicht funktioniert.

Selbst wenn beim WebGL-Publisher CAD-Modelle importiert werden, die in Bauteilgruppen hierarchisiert sind, scheint es nicht möglich sein, diese Daten zu erhalten. Und ohne Hierarchiedaten kann es schließlich auch keinen Assembly Browser geben.

Fehleranfälligkeit

Außerdem ist uns aufgefallen, dass es sehr gefährlich ist, mit den wpm-Modellen zu hantieren. Der kleinste Fehler kann dazu führen, dass das Modell vom Browser nicht mehr interpretiert werden kann und beim Laden des Modells eine Fehlermeldung erscheint:

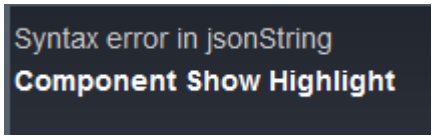


Abbildung 5: Fehlermeldung nach unvorsichtigen Manipulationen an der wpm-Datei

Deswegen haben wir es uns angewöhnt, vor Änderungen an wpm-Dateien immer eine Sicherungskopie zu machen.

4.2 HTML-Datei

Darstellung eines CAD-Modells im Browser

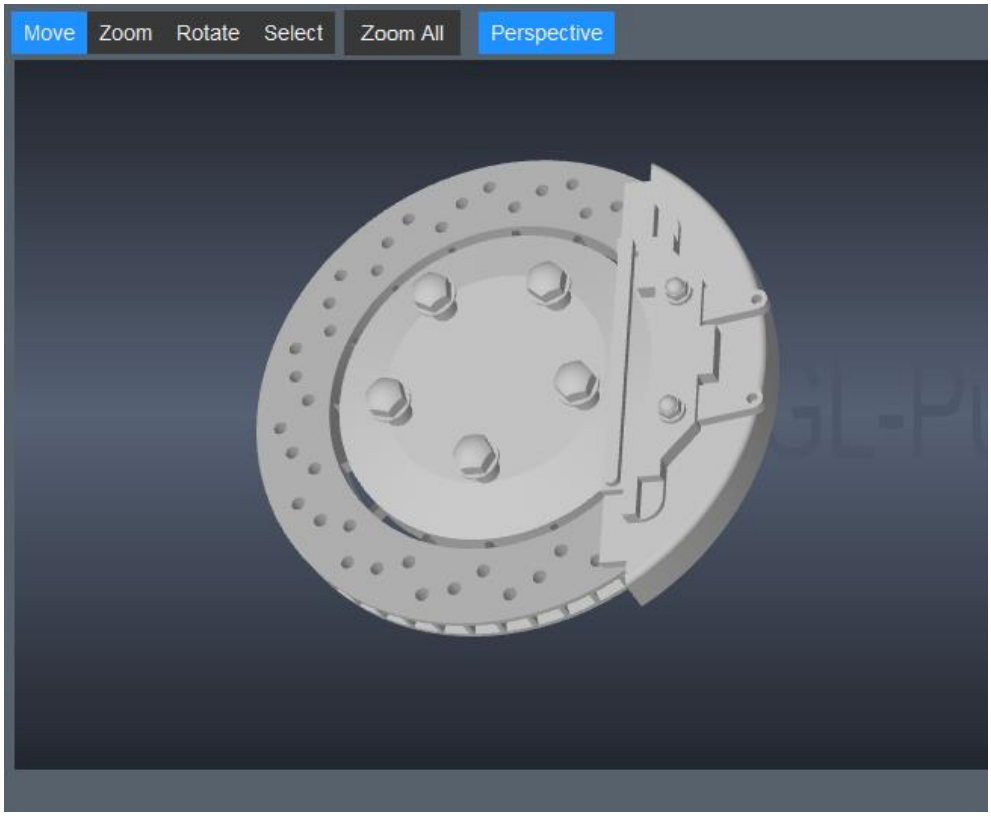


Abbildung 6: Infrastruktur in der ein Modell angezeigt wird

Je nach den Exporteinstellungen bringt die HTML-Datei bereits von Haus aus Schaltflächen mit wie z. B. eine Schaltfläche mit Funktionen zum Bewegen, Rotieren und Zoomen des Modells.

Außerdem gibt es in der ausgegebenen HTML-Datei bereits zahlreiche Funktionen, mit denen man Dinge verändern kann. Z. B. lässt sich über die ***ResizeWindow***-Funktion die Größe des Anzeigebereichs für das Modell festlegen oder über ***myCmiWindow.handleArrowKeys*** bestimmen, ob der Nutzer das Modell mit den Pfeiltasten auf der Tastatur drehen kann.

4.2.1 Assembly Table

Beim Export aus Autodesk Inventor liefert der WebGL-Publisher eine sogenannte Assembly Table[3] mit. Darin sind alle Bauteile enthalten und hierarchisch nach Baugruppen geordnet. Auf der HTML-Seite wird die Assembly Table wie folgt dargestellt:

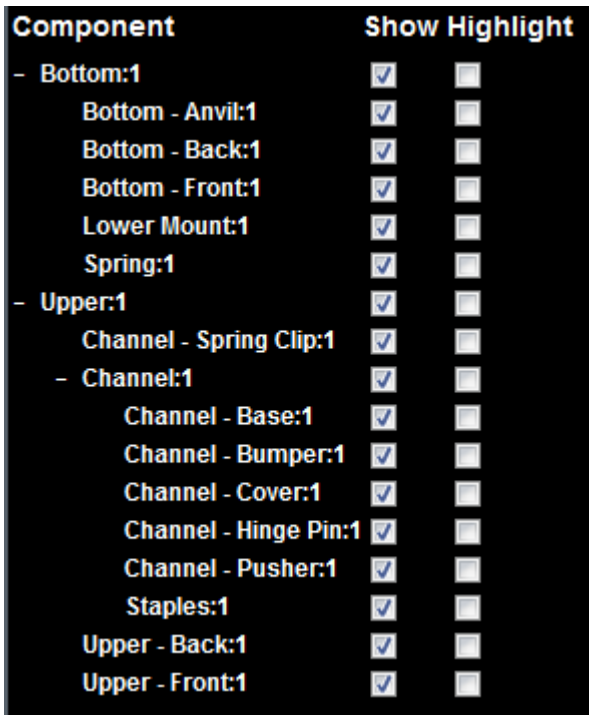


Abbildung 7 : Ausgeklappe Assembly Table

Interaktivität

Die Assembly Table ist interaktiv: Hinter den Bauteilen befinden sich zwei Checkboxen. In der Spalte Show können die Bauteile ein- und ausgeblendet werden. Wenn in der Spalte Highlight eine Checkbox aktiviert wird, dann wird das entsprechende Bauteil rot eingefärbt. Die Baugruppen und Bauteile auf erster Ebene sind nach dem ersten Laden bereits sichtbar. Bauteile, die sich in niedrigeren Hierarchiestufen befinden, müssen zunächst aufgeklappt werden.

[3] In anderen Kontexten heißt die Assembly Table auch Assembly Tree oder Assembly Browser

Tabellenkopf

```
<table id="AssemblyTable" style="position:absolute; top:60px; left:10px; font-weight:bold;
       color:#FFFFFF; border="0" cellpadding="0" cellspacing="0">
 <thead style="font-size:14px;">
   <tr>
     <th height="30" align="left">Bauteil </th>
     <th align="left">Anzeigen </th>
     <th align="left">Hervorheben </th>
   </tr>
 </thead>
```

Abbildung 8: Kopf der Assembly Table

Die Tabelle ist eine ganz normale HTML-Tabelle, der Tabellenkopf kann also nach Lust und Laune verändert werden. So lässt sich die Position der Tabelle ändern, man die Tabelle mit dem CSS-Befehl "***visibility: hidden;***" unsichtbar machen, wenn sie im konkreten Anwendungsfall nicht benötigt wird oder gar stört, und man kann die Beschriftung der Spalten in den <th>-Elementen anpassen bzw. auf Deutsch übersetzen wie im obigen Codebeispiel.

Tabellenrumpf

Änderungen, die in der HTML-Datei im tbody vorgenommen werden, haben ohne weitere Veränderungen im JavaScript-Code allerdings keine Auswirkungen, da die Tabelle jedes Mal neu konstruiert wird, wenn das WebGL-Modell geladen wird.

Im Tabellenrumpf können Baugruppen-Zeilen und Bauteil-Zeilen unterschieden werden.

Baugruppen-Zeilen

```
<tr>
    <td style="cursor: pointer;"><code></code>
        <a id="collaps-Cmp1"
        onClick="javascript:myAssemblyBrowser.switchCollectionState('collaps-Cmp1')">
        <code>- </code>Boden:1</a> 
    </td>
    <td>
        <input id="Sh-Cmp1" onClick="javascript:myAssemblyBrowser.switchVisibilityState('Sh-Cmp1')"
        checked="checked" type="checkbox">
    </td>
    <td>
        <input id="Hl-Cmp1" onClick="javascript:myAssemblyBrowser.switchHighlightState('Hl-Cmp1')"
        type="checkbox">
    </td>
 </tr>
```

Abbildung 9: Aufbau der Baugruppen-Zeilen

In den Baugruppen-Zeilen finden wir in der ersten Spalte die Funktion ***switchcollectionstate***, mit der sich die Baugruppe einklappen (collapse) oder ausklappen (expand) lässt. Wenn diese Funktion ausgeführt wird, wird der Tabellenrumpf komplett neu aufgebaut. Das hat zur Folge, dass die hierarchisch untergeordneten Zeilen einer Baugruppe physisch nicht mehr auf der HTML-Seite vorhanden sind, sobald man sie einklappt. Somit können die Checkboxen auch nicht mehr durch Funktionen angesprochen werden.

In den Spalten 2 und 3 werden die Checkboxen definiert, die zum Ein- und Ausblenden sowie zum Hervorheben dienen. Sie sind mit den entsprechenden Funktionen verknüpft.

Bauteil-Zeilen

Die Bauteil-Zeilen sind gleich aufgebaut wie die Baugruppen-Zeilen, nur fehlt das <a>-Element mit der Funktion zum Ein- und Ausklappen der Baugruppen.

Erzeugung der Assembly Table

Der Aufbau der Assembly Table erfolgt in drei Schritten:

1. Definition einer Variable:

   var myAssemblyBrowser;

2. Definition der Klasse für den Assembly Browser:

   myAssemblyBrowser = new CmiAssemblyBrowser ("myAssemblyBrowser","AssemblyTable");

   Parameter 1: (string browserName): Der Name der CmiAssemblyBrowser Variable als String

   Parameter 2: (string tableID): Die ID der HTML-Tabelle als String

3. Automatischer Aufbau der Tabelle, sobald das Modell geladen ist:

```
case "ModelOpened":
  SetStatusText("");
  myAssemblyBrowser.buildAssemblyTree(myCmiWindow.getCmiModel(),true);
  myAssemblyBrowser.rebuildTreeViewTable();
  break;
```

Abbildung 10: Aufbau der Assembly Table

Erkenntnisse

Daraus ergeben sich zwei Erkenntnisse:

1. Es ist möglich, auf einer Seite mit mehreren Modellen auch mehrere Assembly Tables anzuzeigen. Mit der Variable myAssmenblyBrowser2, der Table-ID AssemblyTable2 und dem Verweis auf das 2. myCmiWindow in der Build-Funktion funktioniert dies ausgezeichnet (siehe Beispieldatei "2 - Zwei Fenster mit Assembly Tables.html").

2. Man kann die Zeile ***myAssemblyBrowser.rebuildTreeViewTable();*** auskommentieren oder löschen. Dann wird Tabelle so aufgebaut, wie in der HTML-Datei beschrieben ist, das heißt es wirken sich auch die vorgenommenen Änderungen im Tabellenrumpf aus. Dies ist allerdings nur so lange der Fall, bis man eine Baugruppe ein- oder ausklappt. Dann wird die Tabelle basierend auf den Spezifikationen in der Datei CmiAssemblyBrowser.js neu aufgebaut.

5 Praktische Anwendungsfälle

5.1 Animationen

Der WebGL-Publisher unterstützt in der derzeitigen Version (2.3.1) kein Erstellen von Animationen von der Benutzeroberfläche aus. Es gibt jedoch Beispielanimationen auf der Webseite vom Publisher[4]

Animationsbeispiel von der WebGL-Publisher Webseite

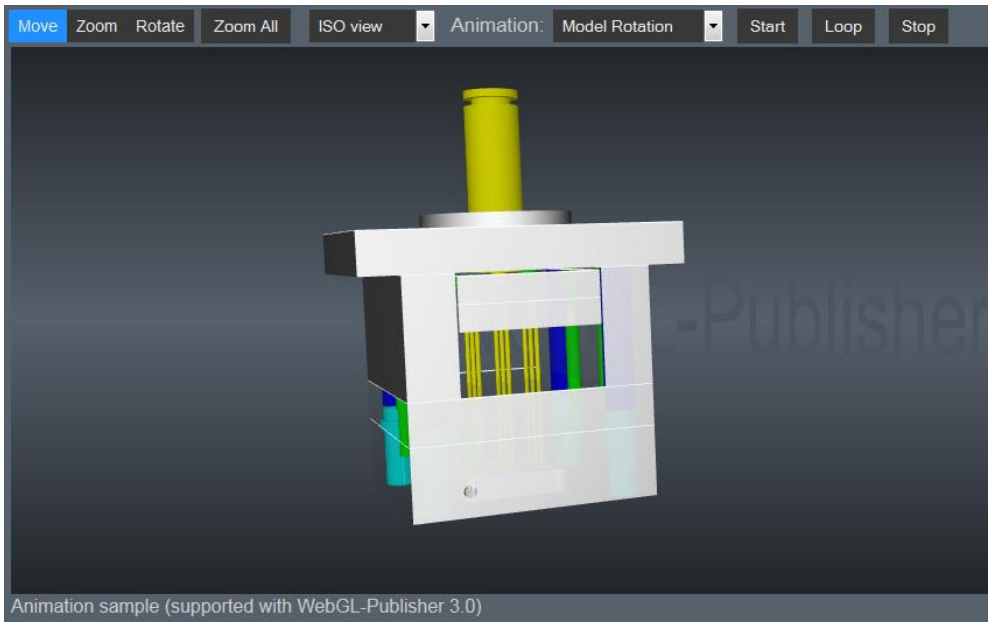


Abbildung 11: Animationsbeispiel auf der WebGl-Publisher Webseite

Im Vergleich zu der üblichen Umgebung, die uns der Publisher ausgibt, sind auf der Beispielseite einige Elemente dazugekommen. In der ersten Dropdown-Liste kann die Ansicht des Modells ausgewählt werden. In der zweiten Dropdown-Liste kann man zwischen einigen Beispielanimationen wählen. Dabei lässt "Start" die Animation einmal ablaufen, "Loop" lässt sie in einer Endlosschleife laufen und "Stop" hält eine laufende Animation an.

Aufgrund dieses Beispiels konnten wir darauf schließen, dass Funktionen zum Animieren in den JavaScript-Bibliotheken vorhanden und nur nicht im Programm integriert sind.

[4] http://www.webgl-publisher.com/AnimationSample.html

Wir haben die Beispielseite dann gespeichert, wobei neben der eigentlichen HTML-Datei ebenfalls alle benötigen JavaScript-Bibliotheken mitgespeichert wurden. Im Code der HTML-Datei haben wir die im Beispiel verwendeten Animationsfunktionen gefunden.

Beispiele möglicher Animationsfunktionen

```
case 0://none
  cmiAnimation.clearAnimation();
  break;
case 1://Rotation
  cmiAnimation.clearAnimation();
  cmiAnimation.addSceneRotation(100.0,80.0,0.0,0.0,90.0,0.0,20);
  cmiAnimation.addPause(0.5);
  cmiAnimation.addSceneRotation(100.0,80.0,0.0,0.0,-90.0,0.0,20);
  break;
case 2://Translation
  cmiAnimation.clearAnimation();
  cmiAnimation.addSceneTranslation(0.0,50.0,0.0,20);
  cmiAnimation.addPause(0.5);
  cmiAnimation.addSceneTranslation(0.0,-50.0,0.0,20);
  break;
case 3://Zoom
  cmiAnimation.clearAnimation();
  cmiAnimation.addSceneZoom(50.0,20);
  cmiAnimation.addPause(0.5);
  cmiAnimation.addSceneZoom(-50.0,20);
  break;
```

Abbildung 12: Ausschnitt von Animationsfunktionen aus dem Animationsbeispiel der Webseite

Mit diesen Grundlagen und der HTML-Datei als Grundgerüst konnten wir nun Animationen für eigene CAD-Modelle erstellen.

5.1.1 Erstellte Beispielanimation

Der folgende Quellcode stammt aus unserer in der Präsentation gezeigten Animation.

```
myCmiWindow.showOglStdView("Front");
cmiAnimation.clearAnimation();
cmiAnimation.addSceneZoom(45.0,20);
cmiAnimation.addSceneRotation(0.0,0.0,0.0,70.0,-10.0,200.0,20);
cmiAnimation.addPause (1);
cmiAnimation.addEntityBlinkingByNames(["s9","s10"],[1.0,0.0,0.0,1.0],2,0.3);
cmiAnimation.addEntityPositioningByName("s9",-145.0,50.0,200.0,20);
cmiAnimation.addEntityPositioningByName("s10",-145.0,-50.0,200.0,20);
cmiAnimation.addSceneZoom(-20.0,20);
cmiAnimation.addSceneRotation(0.0,0.0,0.0,0.0,-40.0,-33.0,20);
cmiAnimation.addPause (1);
cmiAnimation.addEntityBlinkingByNames(["s11"],[1.0,0.0,0.0,1.0],2,0.3);
cmiAnimation.addEntityPositioningByName("s11",-500.0,0.0,110.0,20);
cmiAnimation.addEntityBlinkingByNames(["s7"],[1.0,0.0,0.0,1.0],2,0.3);
cmiAnimation.addEntityPositioningByName("s7",-350.0,0.0,110.0,20);
```

Abbildung 13: Animationssequenz-Beispiel

Hier werden die angesprochenen Bauteile erst hervorgehoben und dann verschoben, um eine Demontage der jeweiligen Teile zu simulieren. Zusätzlich wird mit Zoom- und Rotationsfunktionen gearbeitet, damit der aktuelle Arbeitsschritt gut zu sehen ist.

Nähre Erläuterungen zu den einzelnen Animationsfunktionen finden Sie im Tutorial.

5.2 Ein- und Ausschalten von Bauteilgruppen

Mit Hilfe der Assembly Table ist es möglich, Bauteile ein- und auszuschalten, sofern man das Modell mit dem WebGL-Publisher-Plugin aus Autodesk Inventor exportiert hat.

Probleme

Die ursprüngliche Funktion bringt zwei Probleme mit sich, die in manchen Anwendungsfällen unerwünscht sein können:

1. Es lässt sich immer nur ein einzelnes Bauteil oder eine Bauteilgruppe mit einem Klick ausblenden. Dies ist z. B. unpraktisch, wenn man dem Nutzer einen Einblick in das Innenleben eines 3D-Modells gewähren will, dazu aber z. B. mehrere Bauteile aus verschiedenen Baugruppen ausgeblendet werden müssen.

2. Es ist nur möglich, dass der Nutzer Bauteile ein- und ausblendet. Eine automatische Ausblendung, um z. B. nicht mehr benötigte Teile in einer Animation verschwinden zu lassen, ist nicht möglich. Zudem weiß der Nutzer nicht unbedingt, welche Bauteile er ausblenden muss, um z. B. Beispiel einen Einblick in das Gerät zu erhalten. Damit er diese Bauteile nicht per Trial & Error suchen muss, würde sich z. B. eine Schaltfläche wie "Gerät öffnen" anbieten, mit der man die Bauteile auf einen Schlag ausblenden kann.

Um Bauteile auszublenden, wird beim Klick auf die Checkboxen im AssemblyTable folgende Funktion aufgerufen:

javascript:myAssemblyBrowser.switchVisibilityState('Sh-Cmp1')

Lösungsversuch 1

Als erstes haben wir natürlich versucht, diese Funktion von einer Schaltfläche aus aufzurufen, was bei einmaligem Betätigen der Schaltfläche auch funktioniert hat, sofern man der Schaltfläche die gleiche ID gegebenen hat wie der Checkbox. Beim zweiten Klicken, also beim Versuch des Wiedereinblendens ist jedoch nichts passiert.

Ein Blick in den Code von der Datei CmiAssemblyBrowser.js löst dieses Mysterium auf:

```
this.switchVisibilityState = function(ae) {
    if ((O5 == null) || (yk == null)) return -1;
    var wc = ae.split("-");
    var oI = null;
    var pL = document.getElementById(ae);
    oI = hn(O5, wc[1]);
    if (oI != null) {
        if (pL.checked) {
            this.sP(oI, true);
        } else {
            this.qW(oI, false);
            this.sP(oI, false);
        }
        if (GJ) GJ.lc().drawScene();
        return this.rebuildTreeViewTable();
```

Abbildung 14: Code von switchVisibilityState im CmiAssemblyBrowser[5]

[5] Wir haben inzwischen eine Möglichkeit gefunden, den JavaScript-Code aus den Bibliotheken schön darstellen zu können: http://codebeautify.org/jsviewer

Auch wenn sich einem Uneingeweihten die Funktion aufgrund der vielen kryptischen Variablen nicht ganz erschließt, scheint es wohl so zu sein, dass das Ergebnis des Funktionsaufrufs davon abhängt, ob die Checkbox auf der HTML-Seite angehakt ist oder nicht.

Lösungsversuch 2

Deshalb haben wir im nächsten Schritt versucht, eine Checkbox einzubinden, die die gleiche ID wie die Checkbox in der Tabelle hat. Dies hat dann auch einwandfrei funktioniert. Vor dem Hintergrund, dass es eigentlich unser Ziel war, mehrere Bauteile gleichzeitig von einer Schaltfläche aus ein- und ausblenden zu können, war dies aber keine zufriedenstellende Lösung.

Lösungsversuch 3

Also haben wir versucht, eine Funktion zu schreiben, die eine Checkbox anhakt, wenn Sie nicht angehakt ist und umgekehrt. Dies ist uns auch gelungen:

```
function unCheck ()
        {
        var isChecked = document.getElementById("Sh-Cmp2.1").checked;
        switch (document.getElementById("Sh-Cmp2.1").checked){
        case true:
        document.getElementById("Sh-Cmp2.1").checked = false;
        myAssemblyBrowser.switchVisibilityState('Sh-Cmp2.1');
        break;
        case false:
        document.getElementById("Sh-Cmp2.1").checked = true;
        myAssemblyBrowser.switchVisibilityState('Sh-Cmp2.1');
        break;
        };
```

Abbildung 15: Funktion zum Ein- und Ausschalten der Visibility-Checkbox der Komponente 2.1.

Diese Funktion bringt jedoch den Nachteil mit sich, dass sie sehr raumgreifend und kompliziert ist, da man sie für jedes Bauteil, das ausgeblendet werden soll, hinschreiben und die Komponentennummern händisch reinschreiben muss.

Lösung

Also haben wir uns überlegt, wie man die Funktion vereinfachen könnte und sind schließlich auf die Klickfunktion von jQuery aufmerksam geworden, mit der man Klicks auf Objekte ausführen kann. Wir haben also eine Funktion geschrieben, die bei jedem Klick auf eine Schaltfläche die gewünschten Haken setzt bzw. entfernt:

```
function unCheck1 ()

        {
        document.getElementById("Sh-Cmp2.1").click();
        document.getElementById("Sh-Cmp2.2").click();
        };
```

Abbildung 16: Verschlankte Funktion zum Ein- ud Ausblenden von Bauteilen

Diese Funktion funktioniert einwandfrei. Man muss nur für jedes Bauteil, das ein- und ausgeschaltet werden soll, eine Zeile einfügen, und die entsprechenden IDs der Checkboxen eintragen.

Tabelle ausklappen

Allerdings gab es ein zweites Problem: Diese Lösung funktioniert nur, wenn die Teile des Assembly Tables ausgeklappt, in denen sich die Checkboxen für die Bauteile befinden. Nicht ausgeklappte Teile des Assembly Tables befinden sich physisch nicht in der HTML-Datei, weil die Tabelle beim Auf und Zuklappen von Baugruppen immer neu aufgebaut wird. Man kann die entsprechenden Checkboxen also nicht mit der Funktion anklicken.

Außerdem wäre es dem Anwender nicht zuzumuten, dass er erst alle Teile der Assembly Table ausklappen muss, bevor er loslegen kann. Darüber hinaus gibt es in vielen Anwendungsfällen auch den Wunsch, die Tabelle unsichtbar zu machen, wenn sie nicht benötigt wird – und eine unsichtbare Tabelle kann schließlich schwer manuell ausgeklappt werden. Wir suchten also eine Lösung, die Tabelle automatisch beim Laden des Modells oder beim Aufruf der Ein-/Ausblendfunktion aufzuklappen.

Lösungsversuch 1

Naheliegend war es natürlich, es mit der jQuery-click-Funktion zu versuchen. Dann stellten wir jedoch fest, dass sich <a>-Elemente nicht mit dieser Funktion ansprechen lassen.

Lösung 1

Als nächstes versuchten wir, in die CmiAssemblyBrowser-Datei einzugreifen, um dafür zu sorgen, dass der Funktion zum Ein- und Ausklappen kein <a>-Element zugrunde liegt, sondern ein Kontrollkästchen, das sich über die Click-Funktion ein- und ausschalten lässt:

```
if (Yw[G].u7 != null) {
    if (Yw[G].UB) Dj = "<code>" + prefix + "</code>" + "<a id=\"collaps-"+Yw[G].id+"\"
    onclick=\"javascript:"+aJ+".switchCollectionState('collaps-"+Yw[G].id+"')\"><code>- </code>" +
    Yw[G].Dz + "</input> ";
    else Dj = "<code>" + prefix + "</code>" + "<a id=\"expand-"+Yw[G].id+"\" onclick=\"javascript:"
    +aJ+".switchCollectionState('expand-"+Yw[G].id+"')\"><code>+ </code>" + Yw[G].Dz + "</input> ";
    oL.style.cursor = "pointer";
} else {
    Dj = "<code>  " + prefix + "</code>" + Yw[G].Dz;
```

Abbildung 17: Ursprüngliche Funktion zum Aufbau der Assembly Table in CmiAssemblyBrowser

```
if (Yw[G].u7 != null) {
    if (Yw[G].UB) Dj = "<code>" + prefix + "</code>" + "<input id=\"collaps-"+Yw[G].id+"\" type="Checkbox"
    onclick=\"javascript:"+aJ+".switchCollectionState('collaps-"+Yw[G].id+"')\"><code>- </code>" +
    Yw[G].Dz + "</input> ";
    else Dj = "<code>" + prefix + "</code>" + "<input id=\"expand-"+Yw[G].id+"\" type="Checkbox" onclick=\"javascript:"
    +aJ+".switchCollectionState('expand-"+Yw[G].id+"')\"><code>+ </code>" + Yw[G].Dz + "</input> ";
    oL.style.cursor = "pointer";
} else {
    Dj = "<code>  " + prefix + "</code>" + Yw[G].Dz;
```

Abbildung 18: Geänderte Funktion zum Aufbau der Assembly Table in CmiAssemblyBrowser.js

Wenn man dann z. B. in die Ein- und Ausblenden-Funktion um eine Zeile erweitert, um die Assembly Table auszuklappen, dann funktioniert alles automatisch:

```
function unCheck1 ()
        {
        document.getElementById("expand-Cmp2").click();
        document.getElementById("Sh-Cmp2.1").click();
        document.getElementById("Sh-Cmp2.2").click();
        };
```

Abbildung 19: Um einen Ausklappbefehl erweiterte Funktion zum Ein- und Ausblenden

Lösung 2

Dann stellten wir fest, dass es im Gegensatz zur Ein- und Ausblendfunktion gar nicht nötig ist, den Umweg über den Simulationsklick zu gehen. Während beim Ein- und Ausblenden immer derselbe Parameter übergeben wird (***switchvisibilitystate ('Sh-Cmp1')***), wird beim für das Auf- und Zuklappen jeweils der Parameter Aufklappen (***switchCollectionState('expand-Cmp1')***) oder Zuklappen (***switchCollectionState('collapse-Cmp1')***) übergeben. Es ist also möglich, die Tabelle direkt mit einer Funktion aufzuklappen:

```
function unCheck1 ()
        {
        myAssemblyBrowser.switchCollectionState('expand-Cmp2');
        document.getElementById("Sh-Cmp2.1").click();
        document.getElementById("Sh-Cmp2.2").click();
        };
```

Abbildung 20: Um einen Ausklappbefehl erweiterte Funktion zum Ein- und Ausblenden, 2. Lösung

Lösung 3

Zu guter Letzt gibt es auch noch die Möglichkeit, die Tabelle so hinzuschreiben, wie man sie haben möchte. Das heißt, man exportiert das Modell aus dem Autodesk Inventor, ruft die HTML-Seite, klappt die gesamte Assembly Table auf und speichert die HTTML-Seite dann ab. In der gespeicherten HTML-Datei ist dann die gesamte ausgeklappte Tabelle in HTML-Code beschrieben.

Nun baut sich die Tabelle nach dem Aufruf der Seite jedoch immer wieder nach den Festlegungen in der Datei CmiAssemblyBrowser.js neu auf und dann klappen alle Elemente wieder zu. Dies lässt sich jedoch verhindern, wenn man im JavaScript-Teil der HTML-Datei folgende Zeile auskommentiert oder löscht:

```
myAssemblyBrowser = new CmiAssemblyBrowser("myAssemblyBrowser","AssemblyTable");
myAssemblyBrowser.buildAssemblyTree(myCmiWindow.getCmiModel(),true);
myAssemblyBrowser.rebuildTreeViewTable();
```

Abbildung 21: Programmzeile zum Neuaufbau der Assembly Table nach dem Laden der Seite

5.3 Hervorheben von Bauteilen

Problemstellung

Die Tooltips, mit denen z. B. die Bauteilnamen eingeblendet werden, wenn man mit der Maus über ein Bauteil fährt, ist eine schöne Funktion. Jedoch gibt es in den aus dem WebGL-Publisher exportierten Daten keine Möglichkeit, den umgekehrten Weg zu gehen und z. B. ein Bauteil hervorzuheben, wenn man mit der Maus über eine Schaltfläche mit dem Namen des Bauteils fährt. Diese Funktion könnte aber sehr nützlich sein, wenn man das 3D-Modell als eine Art beigelte 3D-Grafik nutzen möchte. Wenn der Nutzer ein bestimmtes Bauteil sucht, dann ist es unpraktisch, wenn er mit der Maus so lange über jedes Bauteil im Modell fahren muss, bis der gesuchte Tooltip eingeblendet wird, zumal es ja auch kleinere oder versteckte Bauteile geben kann oder Bauteile, die von der Perspektive des Anwenders aus gesehen verdeckt sind.

Lösung

Zur Lösung des Problems kann man die Highlight-Funktionen der Assembly Table nutzen. Der Mechanismus ist dabei genau derselbe wie beim Ein- und Ausblenden: Man kann in den JavaScript-Teil der HTML-Datei jQuery-Click-Funktionen einfügen, die Klicks auf die Checkboxen in der Spalte Highlight der (unsichtbaren) Assembly Table simulieren. So lassen sich die entsprechenden Teile z. B. beim Bewegen der Maus über eine Schaltfläche mit ***Mouseover*** und ***Mouseout*** rot einfärben und wieder auf die ursprüngliche Farbe zurücksetzen.

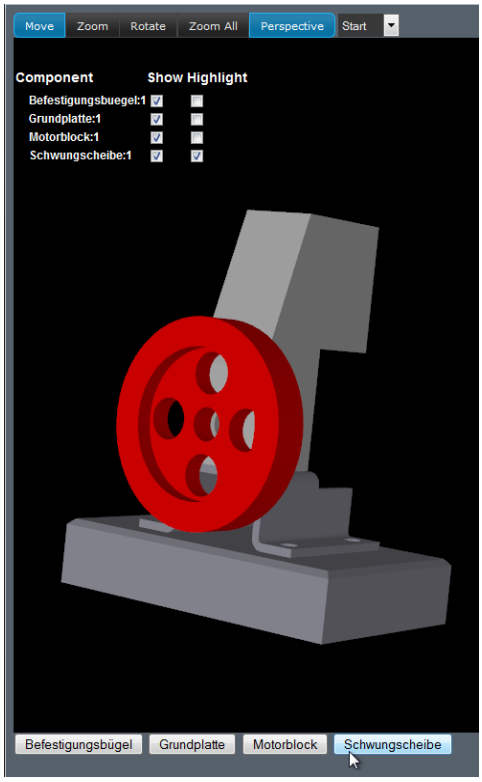


Abbildung 22: Hervorhebung der Schwungscheibe beim Mouseover über die Schaltfläche

Im gezeigten Beispiel wurden vier Schaltflächen auf der HTML-Seite eingefügt, die folgende Funktionen aufrufen:

```
function befestigungsbuegel ()
    {
    document.getElementById("Hl-Cmp2").click();
            };

function grundplatte ()
    {
    document.getElementById("Hl-Cmp1").click();
            };

function motorblock ()
    {
    document.getElementById("Hl-Cmp4").click();
            };

function schwungscheibe ()
    {
    document.getElementById("Hl-Cmp3").click();
            };
```

Abbildung 23: Funtionen zum Hervorheben von Bauteilen

Noch eleganter geht dies natürlich, wenn man den Komponentennamen als Parameter übergibt:

```
<div>
    <input id="teil1" type="Button" value="Befestigungsbügel" onMouseover="highlight('Hl-Cmp2');" onMouseout="highlight('Hl-Cmp2');"/>
    <input id="teil2" type="Button" value="Grundplatte" onMouseover="highlight('Hl-Cmp1');" onMouseout="highlight('Hl-Cmp1');"/>
    <input id="teil3" type="Button" value="Motorblock" onMouseover="highlight('Hl-Cmp4');" onMouseout="highlight('Hl-Cmp4');"/>
    <input id="teil4" type="Button" value="Schwungscheibe" onMouseover="highlight('Hl-Cmp3');" onMouseout="highlight('Hl-Cmp3');"/>
</div>
```

Abbildung 24: Übergabe der Komponentennamen als Parameter

Dann ist nur noch folgende kleine Funktion nötig, um den Effekt zu erzielen:

```
function highlight (cmp)
    {
    document.getElementById(cmp).click();
            };
```

Abbildung 25: Highlight-Funktion mit Parameter

Auf diese Art und Weise lassen sich auch mehrere Teile mit einem Mouseover bzw. Mouseout gleichzeitig hervorheben bzw. zurücksetzen:

```
<input id="teil1" type="Button" value="Befestigungsbügel"
onMouseover="highlight('Hl-Cmp2');highlight('Hl-Cmp1');highlight('Hl-Cmp3');highlight('Hl-Cmp4');"
onMouseout="highlight('Hl-Cmp2');highlight('Hl-Cmp1');highlight('Hl-Cmp3');highlight('Hl-Cmp4');"/>
```

Abbildung 26: Hervorhebung von 4 Teilen mit einem Mouseover

Die hier aufgeführte Lösung lässt sich auch für das Ein- und Ausschalten von Bauteilen verwenden und wurde in der entsprechenden Beispieldatei auch so umgesetzt.

6 Abgabedateien

Ordner Beispieldateien:

Umsetzungsbeispiele für die im Projektbericht beschriebenen Funktionen

- 1 - Animation: Beispielanimation Ausbau Bremsscheibe
- 2 - Zwei Fenster mit Assembly Tables: Zwei WebGL-Modelle auf einer Seite mit jeweils einem funktionsfähigen Assembly Table
- 3 - Ein- und Ausblenden: Gleichzeitiges Ein- und Ausblenden von zwei Bauteilen über eine Schaltfläche, um das Innenleben des Kopfteils eines Tackers sichtbar zu machen
- 4 - Hervorheben: Hervorheben von 4 verschiedenen Bauteilen über Schaltflächen mit Mouseover und Mouseout.
- Die zugehörigen Modelle und JavaScript- und jQuery-Bibliotheken

Ordner Tutorial:

- Tutorial zu Grundfunktionen des WebGL-Publishers und zur Umsetzung der in diesem Projektbericht beschriebenen Erweiterungen
- Reduzierte Beispieldateien, aus denen der Code kopiert werden kann

Ordner Projekt:

- Projektbericht
- Projektplan
- Zeiterfassung
- Powerpoint-Abschlusspräsentation

7 Fazit

7.1 Praktikabilität

Der WebGL-Publisher ist ein feines kleines Programm, das sehr preiswert ist und mit dem sich für diesen Preis erstaunlich viel machen lässt.

Wer 3D-Modelle nativ im Browser darstellen möchte, ist damit gut bedient. Es stehen viele Funktionen zur Verfügung, die beim Export mitgeliefert werden können, wie z. B. Perspektivwechsel auf Standardansichten wie Isometrie, oben, links, etc. Auch die Möglichkeit, JavaScript-Code in die Tooltips der Modelle einbinden zu können, ist eine interessante und leicht umsetzbare Option.

Probleme mit großen Modellen

Uns ist aufgefallen, dass der WebGL-Publisher Probleme bekommt, wenn sehr große Modelle importiert werden. Bei übergroßen Modellen kann es sogar zum Absturz des Programms kommen.

Außerdem sind bei großen Modellen die exportieren wpm-Dateien so groß, dass ein sinnvolles Arbeiten mit den Dateien im Editor kaum möglich ist. Sehr lange Lade- und Speicherzeiten sind in diesen Fällen an der Tagesordnung.

Animationen

In punkto Animationen ergibt sich ein geteiltes Bild: Die Animationen lassen sich nicht direkt in der WebGL-Publisher Benutzeroberfläche erstellen, sondern nur programmieren. Für komplexe Animationen, in denen sich Bauteile drehen und verschiedene Bewegungen zusammenspielen, scheint dies eine sehr unpraktische Lösung zu sein,

Animationen, die zeigen sollen, wie etwas auseinandergebaut wird wie in unserem Beispiel lassen sich jedoch gut umsetzen.

Doch auch hier wäre eine bessere Umsetzung direkt im WebGL-Publisher schön: Wenn man die Teile einfach packen könnte und diese dann entlang von gewissen Achsen auf die richtige Position verschieben könnte, wäre das eine große Arbeitserleichterung. Die Programmierung beruht bezüglich des Findens der richtigen Koordinaten und der richtigen Ansicht zu sehr auf dem Trial-and-Error-Prinzip.

Außerdem ist es schade, dass die mitgelieferten Aminationsfunktionen z.B. nur die Rotation relativ zum Blickwinkel erlauben und dass man keine absolute Position angeben kann, wie das Modell nach der Animation ausgerichtet werden soll. Dies wäre nötig, wenn man bei Animationen Bauteile auch dann in den Vordergrund drehen will, wenn der Nutzer das Modell schon gedreht hat. Zwar kann man am Anfang der Animation immer einen Perspektivwechsel einfügen, aber das sieht nicht sonderlich schön aus, wenn das Modell zuerst ruckartig in die gewünschte Startposition gebracht wird und sich dann erst langsam in die richtige Position dreht. Eine fließende Animation vom Startblickwinkel zum gewünschten Endblickwinkel wäre an dieser Stelle deutlich schöner, ist jedoch nicht realisierbar.

Ein- und Ausblenden / Hervorheben

Das Ein- und Ausblenden und das Hervorheben von Bauteilen sind beides schöne Funktionen, die den didaktischen Nutzen der 3D-Darstellung erhöhen können.

Das Problem ist jedoch, dass für diese Funktionen der Assembly Browser benötigt wird, der nur beim Export mit dem WebGL-Publisher-Plugin aus dem Autodesk Inventor verfügbar ist.

Wenn der Autodesk Inventor dazugekauft werden muss, ist die ganze Sache nicht mehr so billig. Eine Lizenz kostet 1325 Euro. Außerdem fallen noch 178,50 Euro für WebGL-Publisher-Plugin an, also insgesamt 1503,50 Euro. Sicherlich ist dieser Betrag für ein Unternehmen noch bezahlbar, aber vielleicht gibt es zu diesem Preis schon andere Programme, die noch mehr können.

Wenn die Assembly Table jedoch vorhanden ist, lassen sich das Ein- und Ausblenden und das Hervorheben mit wenigen Zeilen Code sehr gut und schnell integrieren.

7.2 Weitere Ansätze

Im Laufe des Projekts sind wir noch auf einige weitere Verbesserungsmöglichkeiten gestoßen, die sich jedoch entweder im Zeitrahmen des Projekts nicht mehr umsetzen ließen oder aufgrund mangelnder programmiertechnischer Kenntnis nicht umgesetzt werden konnten.

Kombination Animation / Assembly Table

Offensichtlich ergeben sich Kombinationsmöglichkeiten aus den Animationen und den Animationen und den Funktionen, die die Assembly Table bietet. Die Highlight-Funktion steht dabei nicht so im Vordergrund, weil sich die Aufmerksamkeitslenkung auf die relevanten Bauteile auch mit einer Blink-Animation bewerkstelligen lässt. Was jedoch nützlich ist, ist die Möglichkeit, einzelne Bauteile auszublenden. So können bei einer Ausbauanimation die ausgebauten Teile einfach ausgeschaltet werden und sind nicht mehr im Weg, so wie es jetzt in unserer Beispielanimation mit den Schrauben der Fall ist. Das Ausblenden der Schrauben wäre leicht umzusetzen gewesen, doch leider ist unsere Autodesk-Testversion abgelaufen, sodass wir kein Assembly-Table-fähiges Modell der Bremsscheibe mehr exportieren können.

Koordinaten für Blickwinkel ausgeben lassen

Ein weiteres Problem ist, dass man zur Programmierung von Animationen entweder ein gutes räumliches Darstellungsvermögen braucht oder per Trial & Error die richtigen Perspektiven finden muss.

Schön wäre die Option, sich ein Testmodell im Browser so hinzudrehen, wie man es braucht, auf einen Knopf zu drücken und dann die aktuellen Blickwinkelkoordinaten angezeigt zu bekommen. Dann könnte man die Blickwinkelkoordinaten einfach in die HTML-Datei eintragen, mit der das Modell angezeigt werden soll.

Da die Blickwinkelkoordinaten mit ***CmiWindow.rotateSceneAbsolut*** übergeben werden können, können diese Werte mit ausreichender Programmierkenntnis wahrscheinlich auch ausgegeben werden. Diese würde eine große Arbeitserleichterung bedeuten und wäre künftig besonders dann wichtig, wenn es auch möglich wäre, absolute Zielblickwinkel-Koordinaten für Animationen zu setzen.

Kombination Animation / weitere Programmierung

Natürlich lassen sich Animationen noch mit weiteren Programmierungen erweitern. Denkbar wäre z. B., dass zuzüglich einer Ausbauanimation auch noch Handlungsschritte angezeigt werden könnten. Der jeweilige gerade aktive Handlungsschritt könnte eingefärbt werden. Die in den Handlungsschritten verwendeten Bauteilnamen könnten mit einer Mouseover-Highlight-Funktion so gestaltet werden, dass das entsprechende Bauteil aufleuchtet, wenn man mit der Maus über den Bauteilnamen fährt. Zudem könnte man Start, Stopp, Vor- und Zurück oder Pause-Schaltflächen einführen, mit der sich die Animation steuern lässt. So wäre es z. B. denkbar, dass der Nutzer einen Animationsschritt wiederholen kann, wenn er auf eine Zurück-Schaltfläche klickt.