3D for the Modern Web: Declarative 3D and glTF

Brian Coughlin ...@gmu.edu
GMU CS-752 Summer 2014

Abstract

The origins, goals and key aspects for both Declarative 3D and glTF are investigated in detail using selected examples, with particular focus in how both are designed to harmonize with the supporting ecosystem of WebGL enabled browsers, web standards, best practices and HTML5 frameworks. A case study "Swiss Army Knife - Tools Demo, glTF scene interaction and animation using CSS Transforms" is presented showing one example of what is possible today using a glTF scene with the HTML5 framework MontageJS.

Keywords: glTF, Declarative 3D, HTML5, Polyfill, DOM Integration, XML3D, X3DOM, WebGL

Introduction

Today 3D technology based on WebGL within Web browsers has passed key milestones that point to a bright and interesting future, with broader usage and adoption ahead. This progress is driving research, development, and experimentation across the Internet in academia and commercial endeavors.

To survey the projects and research of all such endeavours was not practical, so this paper examine two prominent technologies: Declarative 3D and glTF. These were both selected because they are well aligned with standards bodies, have a strong body of working source code, and have live examples openly deployed on the Web.

Origin and Goals of Declarative 3D

Officially Declarative 3D is a reference to the work of the W3C Community Group "Declarative 3D for the Web Architecture" [Declarative 3D CG 2011]. This group was formed in August 2011 by researchers and project leads from the German Research Center for Artificial Intelligence (DFKI), the Fraunhofer Institute for Computer Graphics Research (IGD) and the Web3D Consortium [Web3d 1999]. In practical terms, DFKI had been developing XML3D [XML3D 2014] with the Intel Visual Computing Institute (VCI), and likewise Franhofer IGD and the Web3D Consortium had been developing X3DOM [X3DOM 2014]. Rather than try to compete with each other, all parties banded together realizing that on the World Wide Web the "major media type ... still missing is 3D" [Jankowski, et. al. 2013].

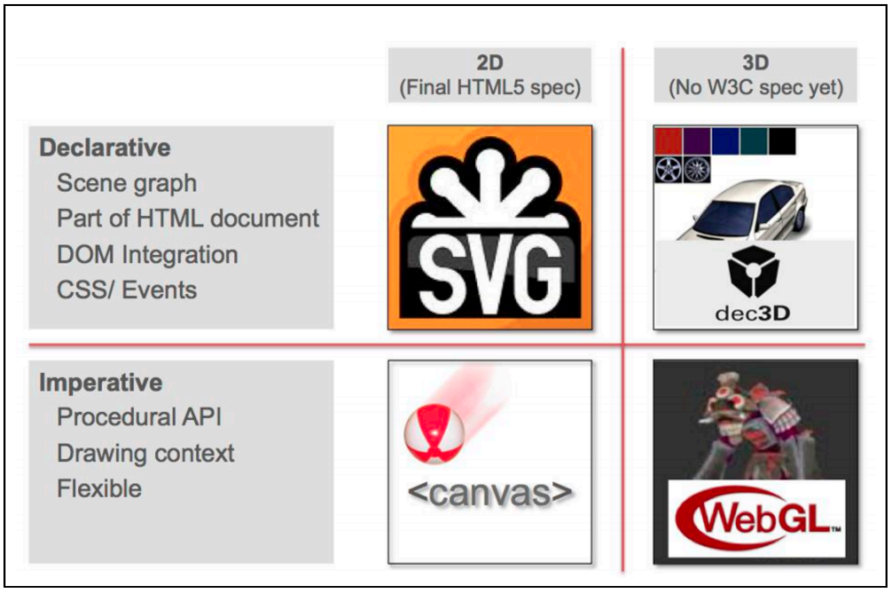

Figure 1: Position of Declarative 3D in the Web Graphics technology ecosystem [Jankowski, et. al. 2013].

Figure 1 above represents the high-level vision of the Declarative 3D Community Group. More specifically, Declarative 3D has stated its goal is to "evaluate the necessary requirements for a successful standardization of a declarative approach to interactive 3D graphics as part of HTML documents". [Declarative 3D Charter 2011]

Origin and Goals of glTF

Likewise Khronos has identified a related gap in the ecosystem - today there is no standard file type for 3D content in web browsers. So it has been promoting glTF as an optimal delivery format for 3D to WebGL/OpenGL devices. Khronos promotes it is analogous to jpeg for pictures and H.264 for video.

Audio	Video	Images	3D
MP3	H.264	JPEG	?
napster	YouTube	facebook	!

Figure 2: The lack of a standard 3D file format for the Web [Khronos 2013]

Khronos deserves some credit for identifying this gap. As suggested in Figure 2 above, with Youtube for video, or with Facebook/Instragram/et. al. for pictures, they each gained widespread adoption in part because their primary media formats were well established.

Khronos' glTF project compliments that of Declarative 3D quite well. Support for externally referenced 3D "generic data containers" (analogous to `<img src="http://facebook.com/jsmith/mycat.jpg">` in HTML) is a core design goal of Declarative 3D [Behr, et. al. 2012]. glTF can be considered as one implementation of just that.

The Roads Not Taken

The goals of Declarative 3D and glTF are quite appropriate given that other past and present technologies have significant limitations or gaps that have held back wider mainstream adoption of Web 3D graphics.

Web Browser Plugins: No Longer Relevant

The two most widely adopted browser plugins, Java and more recently Flash, are quickly being marginalized for multiple reasons [Kruger 2014]. Among those are many identified security risks, waning or no implementation support on iOS devices and Android devices, and a sandboxed bytecode application runtime that is out of step with modern open web standards. Google now is even warning users in search results for sites that run Flash.

Overall, Google is also dropping the cross-platform NPAPI plugin interface from Chrome and Mozilla is discouraging its use in Firefox. Microsoft stopped supporting NPAPI in Internet Explorer many versions ago, and only offers ActiveX as the basis for its IE plugin API which has never been a cross-platform solution.

WebGL: Vital but Not Enough

This may be confusing given that WebGL is prominently included in Figure 1 above. It is true that WebGL is the rendering layer underneath the Declarative 3D Architecture [Behr, et. al. 2012] and has been key to moving away from depending on Web Browser plugins for advanced rendering and graphics. But the low-level imperative programming model used by WebGL prevents it from gaining adoption by Internet content authors or most Internet developers. Most developers even with prior OpenGL experience (myself included) prefer to use popular WebGL javascript libraries like Three.js as an abstraction layer instead of writing direct low-level WebGL code.

Yet even then such higher-level Three.js WebGL code still uses an imperative paradigm unlike the declarative paradigm of proven web standards. For example, Figure 3 shows how it still takes roughly 15 lines of imperative code even using Three.js to place and orient a simple wireframe cube in a scene space.

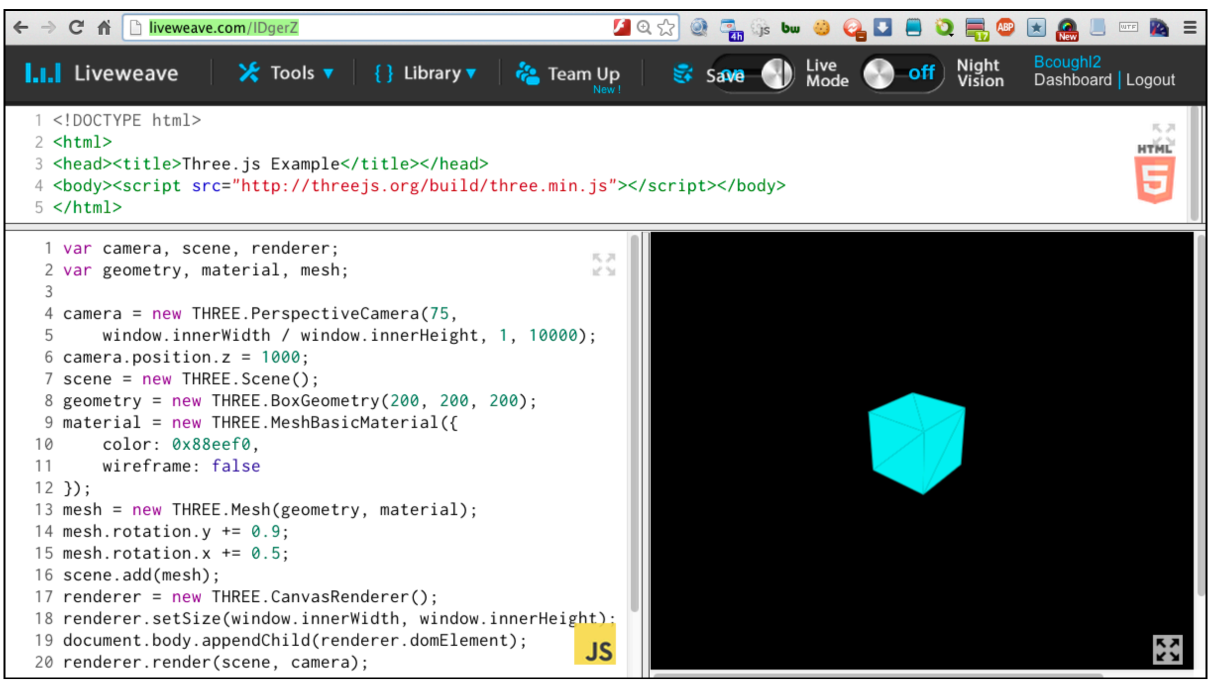

Figure 3: Three.js WebGL example [http://liveweave.com/IDgerZ]

Enabling Component Technologies for glTF and Declarative 3D

GPU Hardware and WebGL

As referenced in part by the blue sections for Figure 5 below, the underlying Hardware, Operating System and

Browser (also referred to as UA/"User Agent") layers have matured in the market ecosystem. All modern mainstream devices have some form of Graphic Processing Unit (GPU) hardware in silicon. These are not just desktop devices running Windows and OSX, but also mobile devices running iOS and Android [Sharma, 2014] which are already surpassing PCs in annual units sold. All of these device's latest operating systems offer WebGL support in at least one of their widely deployed User Agents, most notably with Windows supporting it in Internet Explorer 11 recently and Apple finally getting on board with Safari for iOS 8 targeted for general release in Fall 2014 [Jackson, et. al. 2014].

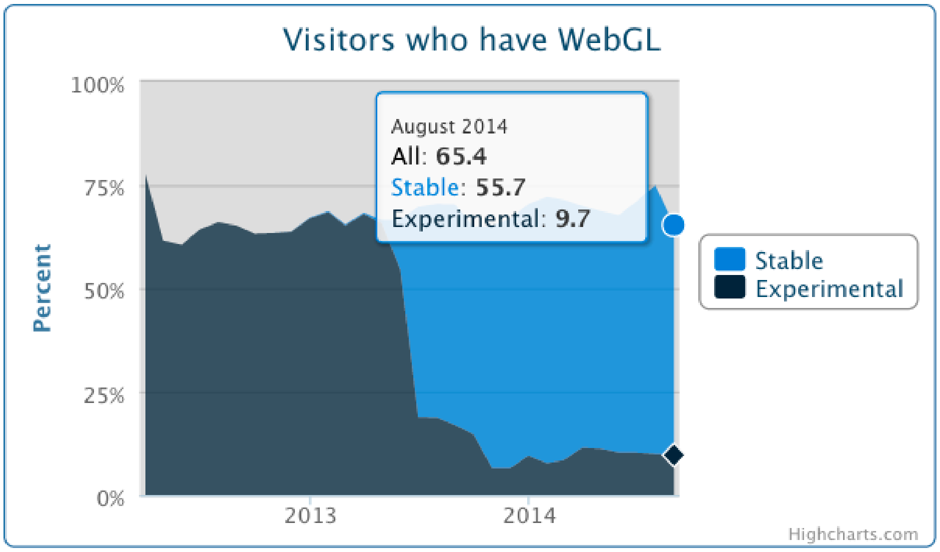

Figure 4: WebGLstats.com results to date of site visitors with User Agents that support WebGL

Figure 4 above shows a sample of webGLstats.com as of August 2014 with almost two thirds of its visitors' User Agents now having WebGL support. That result should only keep improving once iOS 8 is released later this year, and Internet Explorer 11 becomes more widely adopted among Windows users.

Other Key Web Platform Enablers

Equally important but less obvious are other key advances in the modern Web Platform listed below in Table 1. Individually they primarily benefit developers, but collectively they also open the door for more modern and sophisticated applications using WebGL and more.

Technology	Description	Developer Benefit
Polyfills [Sharp 2010]	Javascript code that dynamically adds one or both of the following: 1. support for otherwise missing functionality in older or less advanced User Agents 2. new functionality to the browser using Javascript and CSS (a.k.a. "prollyfill")	Liberate code from being spec'd to the least capable User Agent. (e.g. < IE v9) Rely on Polyfills to provide backwards compatibility with older browsers. Leverage Prollyfills to support new functionality and elements that embody your new standard(s).
"Evergreen" Browsers [Dale 2013]	All modern User Agents are now self-updating without requiring manual action(s) of the end user.	

Javascript Performance Improvements	All Modern browsers employ JIT compilation techniques [Wikipedia 2014]	Continually better performance of existing Javascript code as new browser versions evolve
TypedArrays [Khronos 2013]	Browser API for interoperability with native binary data	C-like arrays of raw data in Javascript can be passed directly to binary APIs like WebGL and others without requiring intermediate conversion steps.

Table 1: Other Key Web Platform Enablers for Declarative 3D

Polyfills

Of all the four technologies above in Table 1, Polyfills are particularly significant. Polyfills enable rapid innovation and experimentation with new functions and declarative elements on modern browsers *today*, without being held back by limitations of older browsers or waiting for browser vendors to implement new standards alongside the W3C specifications process.

Polyfills have spurred a recent groundswell of prominent developers to form the W3C Extensible Web Community Group to coordinate polyfill techniques with the W3C. In general, this represents an exiting new model for innovation in web browsers and more agile development of related standards. [Smus 2012] In the case of Declarative 3D, both XML3D and X3DOM use polyfill techniques to produce working implementations with their new namespaced elements, getting feedback from the developer community straight away.

Another example of web platform innovation not waiting for official standardization is Web Components [Webcomponents 2014], which is being used by many leading-edge frameworks including the Polymer Project, Mozilla X-tags and Bosonic.

Declarative 3D in Detail

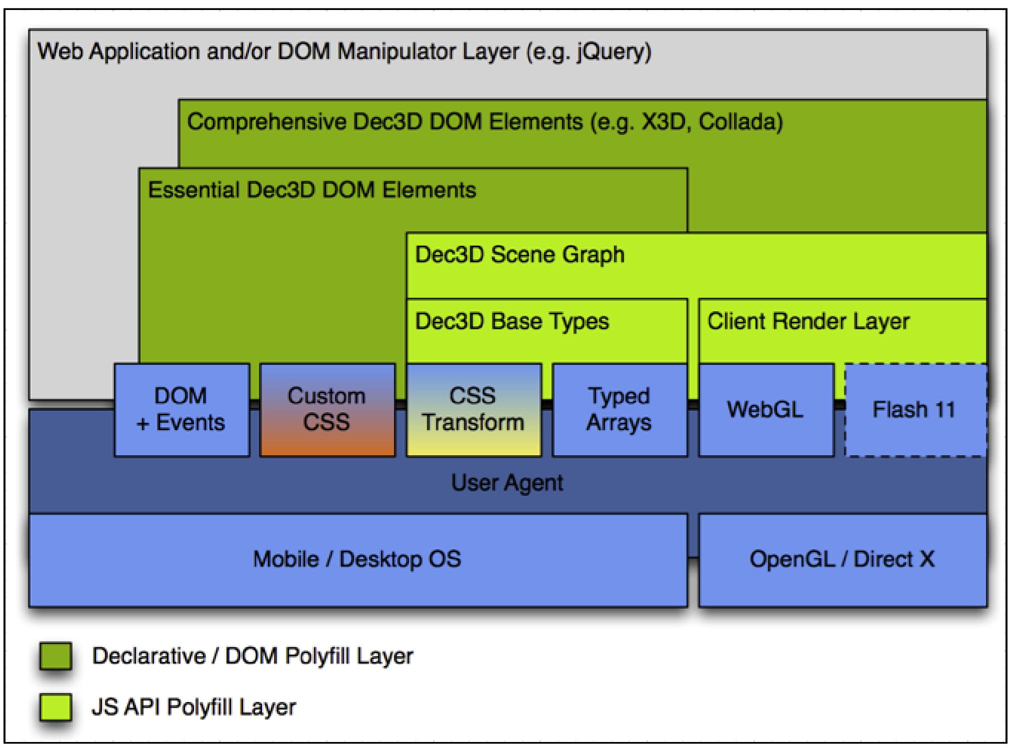

Figure 5: Declarative 3D "Polyfill Runtime Architecture" Software Stack [Behr, et. al. 2012]

Figure 5 shows the current Declarative 3D "Polyfill Runtime Architecture". Note the "DOM + Events", "Custom CSS" and "CSS Transform" layers - those reflect how Declarative 3D wants to bridge standard DOM techniques for page element interactions and CSS for page element presentation control with 3D scenes and more importantly node elements within those scenes. Typed Arrays are important just like with glTF to allow for direct bulk loads of binary object data directly into WebGL for rendering.

DOM bindings to Scene Graph Nodes: Essential vs Comprehensive DOM Elements

The Declarative 3D Scene Graph elements are layered higher in the stack because they can be mapped either to the "Essential Declarative 3D DOM Elements" or the "Comprehensive Dec3 DOM Elements". The "Comprehensive Dec3 DOM Elements" layer is designed to have the full original scene graph already captured in the Collada or X3D file. The idea then as used in X3DOM is to map *all* scene nodes directly onto the DOM as elements, which is logical in many cases but could cause problems rendering very complex X3D scenes with very elaborate scene graphs. Alternatively the "Essential Declarative 3D DOM Elements" layer appears to reflect the design of XML3D, which employs a more generic and intermediate set of DOM Elements which then can be more loosely and selectively coupled to scene or object mesh elements.

Both of these options, as well as general orientation to what Declarative 3D is about at in practice is best shown in the source code of the following two examples. One is using X3DOM and the other using XML3D.
X3DOM Example: Comprehensive Scene Element mapping to the DOM

```
1  <html>
2  <head>
3    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
4    <title>Sample X3DOM Page</title>
5    <script type='text/javascript' src='http://www.x3dom.org/download/x3dom.js'>
6    </script>
7    <link rel='stylesheet' type='text/css' href='http://www.x3dom.org/download/x3dom.css' />
8  </head>
9  <body>
10   <h2>X3DOM with External Scene Referance & <br>Triggered DOM Events </h2>
11   <script>
12     function redNose() {
13       if (document.getElementById('Deer__MA_Nose').getAttribute('diffuseColor') != '1 0 0')
14         document.getElementById('Deer__MA_Nose').setAttribute('diffuseColor', '1 0 0');
15       else
16         document.getElementById('Deer__MA_Nose').setAttribute('diffuseColor', '0 0 0');
17     }
18   </script>
19   <h3>
20     This example includes an external x3d scene.
21     <br>Reindeer's nose will turn off/on
22     via DOM events <br>triggered by clicking the reindeer.
23     <br><br>Example borrowed from the
24     <a href="http://doc.x3dom.org/tutorials/models/inline/example.html">
25       X3DOM tutorial.</a>
26   </h3>
27   <x3d width='350px' height='400px'>
28     <scene>
29       <viewpoint position="-1.94639 1.79771 -2.89271"
30       orientation="0.03886 0.99185 0.12133 3.75685"></viewpoint>
31       <Inline nameSpaceName="Deer" mapDEFToID="true" onclick="redNose();
32             " url="http://doc.x3dom.org/tutorials/models/inline/Deer.x3d" >
33     </scene>
34   </x3d>
35 </body>
```

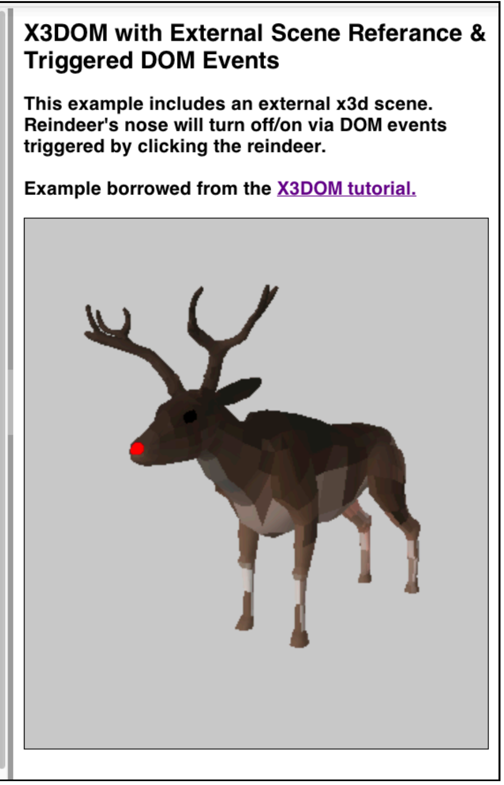

Figure 6: x3dom example source code and output
[http://plnkr.co/edit/ZbAbXNn6MdEgVXjN688H?p=preview]

Figure 6 shows the full page code for presentation of an X3D object. This example is quite straightforward given the maturity of X3D file format as a standard and widespread use in industrial, medical and commercial applications.

First note in lines 5 and 7 the x3dom javascript library and CSS class files are loaded, which bootstaps X3DOM to this page. That allows all the declarative code to be neatly enclosed within the `<x3d>` element tags from lines 27 to 34 which befits good declarative style perfectly.

Within those <x3d> tags, the X3D scene file is loaded by reference in line 32, and mapped into the DOM by the `nameSpaceName` and `mapDEFToID` attributes in line 31. Then the control for turning on (or off) the reindeer's nose is handled by the `onclick` DOM event also in line 31 associated with the `redNose()` javascript function in lines 11 - 17. `redNose()` uses the standard DOM query `document.getElementByID()` function to perform get and set methods on the diffuseColor of the `MA_Nose` Shape node within the `Deer.x3d` file, toggling the nose red or off per each mouse click by the user.

For further explanation of this example, see the X3DOM documentation [X3DOM].

XML3D Example: Essential Scene Element mapping to the DOM

```
20 <xml3d id="MyXml3d" activeView="#defaultView" class="xml3d"
21        style="width: 600px; height: 400px;" >
22     <defs id="mainDef">
23         <transform id="t_Camera" rotation="0.772239 0.341341 0.535848 1.347139"
24                    scale="1.000000 1.000000 1.000000" translation="7.481132 -6.507640 5.343665"></transform>
25         <transform id="t_Lamp" rotation="0.205942 0.331517 0.920698 1.926274"
26                    scale="1.000000 1.000000 1.000000" translation="4.076245 1.005454 5.903862"></transform>
27         <transform id="t_Suzanne" rotation="0.364491 0.419457 0.831386 2.110558"
28                    scale="1.000000 1.000000 1.000000" translation="-0.470811 0.394874 0.000000"></transform>
29         <lightshader id="ls_Spot" script="urn:xml3d:lightshader:point">
30             <bool name="castShadow">true</bool>
31             <float3 name="attenuation">1.000000 0.033333 0.000000</float3>
32             <float3 name="intensity">1.000000 1.000000 1.000000</float3>
33         </lightshader>
34         <shader id="Material" script="urn:xml3d:shader:phong">
35             <float name="ambientIntensity">0.0</float>
36             <float3 name="diffuseColor">0.400000 0.120000 0.180000</float3>
37             <float3 name="specularColor">0.500000 0.500000 0.500000</float3>
38             <float name="shininess">0.2</float>
39         </shader>
40     </defs>
41     <view id="defaultView" orientation="0.772239 0.341341 0.535848 1.347139"
42           position="7.481132 -6.507640 5.343665"></view>
43     <group shader="#Material" transform="#t_Suzanne">
44         <mesh src="./suzanne.json" type="triangles"></mesh>
45     </group>
46     <group transform="#t_Lamp">
47         <light shader="#ls_Spot"></light>
48     </group>
49 </xml3d>
```

Figure 7: XML3D source code and output
[http://xml3d.github.io/xml3d-examples/examples/suzanne/suzanne.html]

If X3DOM can be described as mapping the full X3D scene graph down onto the DOM, XML3D takes a more generic and flexible approach. While not shown in Figure 7 above, within the `<head>` block in lines 5-12 of suzanne.html the base `xml3d.js` Javascript library is loaded along with the `xml3d-camera.js` library, along with jquery to bootstrap the page.

What is shown in Figure 7 above is several `<xml3d>` elements making up the scene graph, which is scaffolded declaratively and then applied to the suzanne object mesh. Compared to X3DOM, XML3D gives the page author more control over the final scene output, although he/she also has more work to do to construct a proper scene.

The "Suzanne" object like in the X3DOM example is loaded by reference in line 44, but the `suzanne.json` file is just a basic meshfile rather than a fully described scene graph. So most of the page's code declaratively adds all the scene elements in lines 21-48. Of particular note are the CSS transforms in lines 23-28 as well as the shader definition in lines 34-39 because they are then applied as style-like attributes to the mesh in line 43. Because they are declared separately from the mesh, they could be reused and applied to other meshes if this were a more complex scene. Finally note XML3D's `<view>` element on lines 41-42, which is remarkably similar to X3DOM's `<viewpoint>` element on lines 29-30 in Figure 6.

So while no changes are being made to the DOM in this example, items nested within the <xml3d> tags here are likewise part of the DOM and accessible using typical document.query*, document.get* etc methods. Finally, XML3D supports multiple mesh file formats including xml3d json (this example), xml3d xml, meshlab, and openCTM.

XML3D also supports a powerful mesh data transformation framework called XFlow, which is outside the scope of this paper.

XML3D and X3DOM together for Declarative 3D

These examples provide clear insights as to what Declarative 3D is about and how both sides of the "marriage" actually have much in common, while both bring different but complimentary strengths to the partnership. XML3D is a more green-field design and is more flexible and adaptable for the page author. However X3DOM clearly provides a straightforward way to present X3D assets within web pages using clean declarative style and best practices grounded in established standards.

Going forward it will be interesting to see how much traction Declarative 3D can gain with the W3C and browser vendors' future release roadmap. [W3C 2012] The general requirements being proposed by Declarative 3D are grouped into 15 "Essential" groups and are all worth serious consideration by the W3C, browser vendors and the Internet community at large. [Behr, et. al. 2012]

glTF in Detail

Figure 7: glTF pipeline progression of content authoring, conversion, delivery, rendering [Trevett 2013]

glTF, which is an abbreviation for "graphics library transmission format", is a file format optimized for delivery to WebGL-enabled web browsers. As shown in Figure 7, glTF is created from a Collada digital asset exchange (dae) files which became an ISO standard in 2013. Collada is widely supported as an export file type option across many types of 3D content creation software. Ironically while the collada DAE is a single file, the output of the collada2gltf converter is actually multiple files. Both glTF and Collada are supported and promoted by Khronos, which also manages the OpenGL and WebGL standards.

To understand what goes into glTF, it helps to first understand its "parent" format - the Collada dae file. Rather than being optimized for delivery, Collada is designed as an authoring tool file interchange format, which means it tries to be as detailed and explicit as possible to capture the full fidelity of the entire scene data so that it can be imported into any another 3D content authoring program. To optimize for delivery to WebGL browsers, collada2gltf culls through the dae file to select which scene elements are worth keeping vs those which can be discarded as overhead not needed for the end user. Furthermore, collada2gltf then optimizes the data elements it has decided to keep in multiple ways making them more readily consumable by webGL on the device.

glTF sourcecode is openly available on Github [Khronos 2014] although it is a work in progress currently at version 0.6. Table 2 below provides details on the discrete file components of glTF.

gltf file component elements (format)	Details
scene node hierarchy including materials, lights, cameras (json)	easily parsed by any web browser using a variety of free libraries. much lighter weight than XML-style markup see Figure 8 below for schema relationships
mesh vertices and indices (binary)	raw data meant for direct path into *GL using TypedArrays, optionally compressed using Open3DGC [Mammou 2013]
textures (png, jpeg, ...)	png, jpeg, etc. texture files passed on from Collada dae without modification
OpenGL shaders (glsl)	vertex and fragment shaders

Table 2: glTF components

Each glTF file component is designed to be as lightweight as possible for minimize the processing and rendering demands placed upon the Web Browser. Using the json format for the scene hierarchy is practical because it is much more easily parsed than XML and is also more compact so will take less time to download as well. The glsl shader files are very small and passed without modification into WebGL. Textures are simply png, jpeg or similar file types that likewise need no further modification. The mesh binary data, which can be the largest of all the component files, is raw binary data meant to be passed directly into buffers directly. Optional Open3DGC encoding is very efficient [Mammou 2013] and designed for fast decoding in javascript or C++ using arithmetic algorithms.

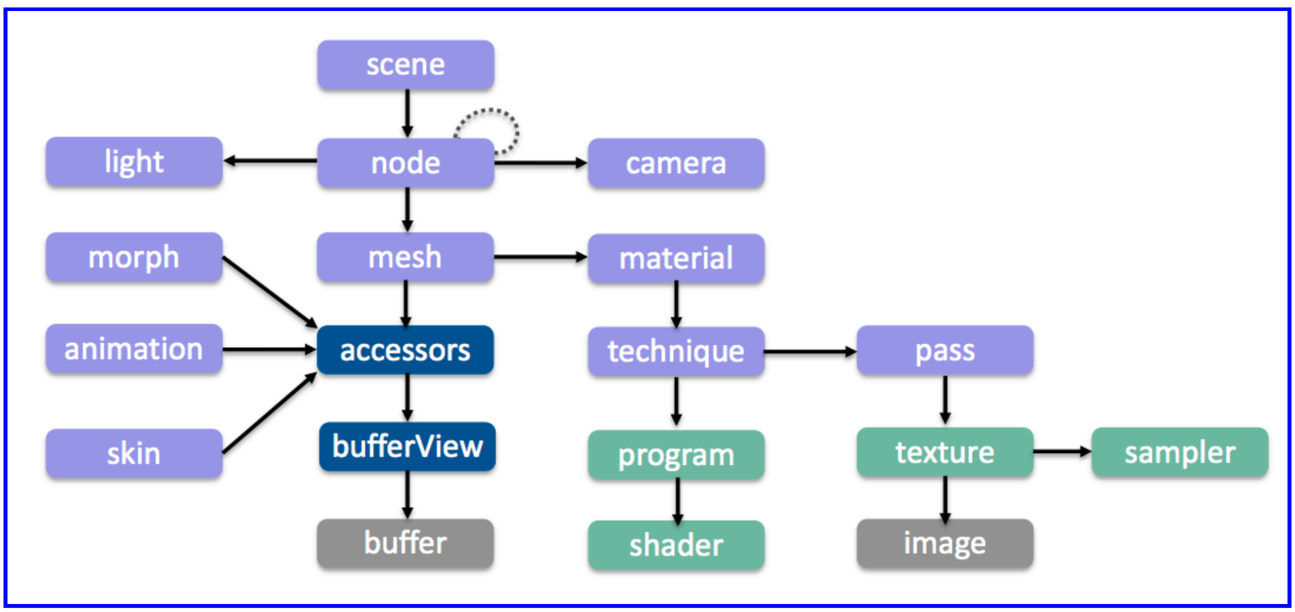

Figure 8: glTF Schema relationships

Figure 8 above shows the schema relationships of glTF elements. The case study that follows focuses mainly on the glTF node elements. Much more detailed analysis of collada2gltf algorithms and glTF schema components are available from several sources, such as Cozzi [Cozi 2013] et. al.

Case Study: "Swiss Army Knife - Tools Demo"
glTF scene interaction and animation using CSS Transforms

One of the key design goals of both Declarative 3D and glTF is to be interoperable with modern HTML5 standard techniques and tools. Here is one case study showing how that can be accomplished.

Case Study Scope

This case study presents a glTF scene within a WebGL canvas, where nodes with the glTF scene are animated using CSS3 Transform techniques triggered by DOM events when the user clicks radio button elements. This scope combines the DOM events aspect of the Declarative 3D X3DOM example with the CSS Transforms aspect of the Declarative 3D XML3D example.

Overview of the Process

At a high level, these were the steps followed:

1. Select and download a 3D scene or object from various open repositories on the Internet
2. Use Content Authoring tool to modify scene elements (optional) & export Scene to Collada file
3. Convert Collada scene file to glTF
4. Select javascript library or HTML5 framework(s) to present and interact with the 3D glTF scene file
5. Configure Your Development system
6. Modify Javascript, HTML and/or CSS file(s) as needed to compose desired presentation and interaction
7. Deploy to a web server

1. Selecting a Scene

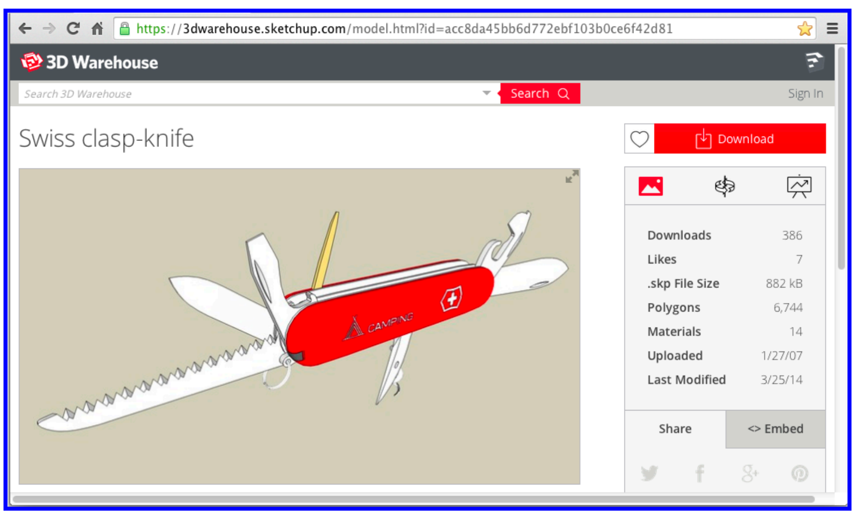

Figure 9: Original Swiss clasp-knife

This scene was downloaded as a Sketchup 6 model file (.skp).

2. Working with the Scene in Sketchup

The sketchup model was then edited to make its initial default state have all its tools closed as shown in Figure 9. The labels, corkscrew and the yellow toothpick assemblies and nodes were also deleted just to simplify the model somewhat. Finally, the knife's elements were named descriptively in the Sketchup Outliner panel to help match them to their node IDs in the final glTF json file later on.

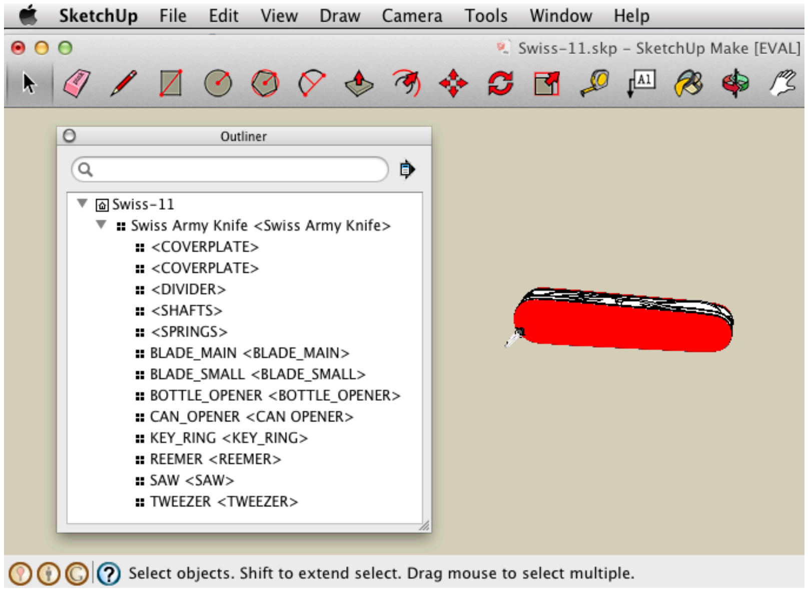

Figure 10: Final version of Modified Knife before exporting from Sketchup

3. Convert Scene to glTF

Once exported in Collada as `SwissArmyKnife.dae`, it is then converted to glTF using the collada2gltf command line tool as shown in Figure 11 below. Open3DGC binary compression was chosen as an option as well.

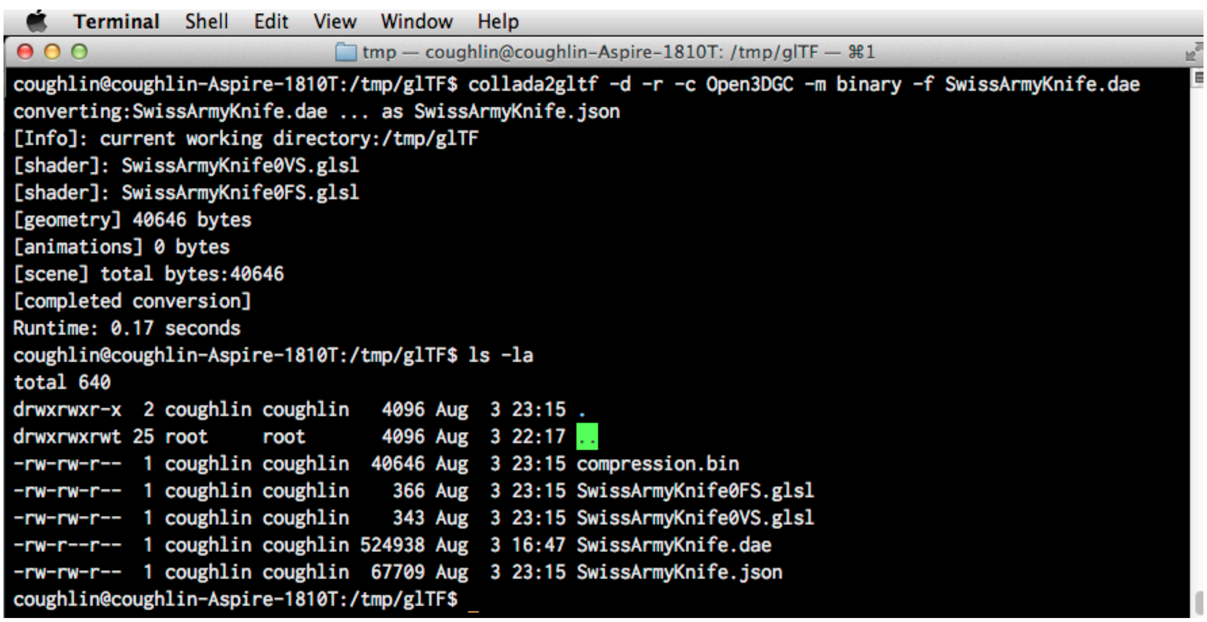

Figure 11: collada2gltf in action

4. Select Framework

MontageJS [MontageJS 2014] was selected as the framework for interacting and presenting this resulting glTF file. This decision was based on how MontageJS was already being used as the defacto framework for glTF Viewer pages [glTF Viewer 2014] and was the basis for other interesting working examples of other interactive 3D scenes, including some that were presented at the 2014 Apple WWDC event. [Jackson 2014] Also the MontageJS framework is first and foremost a general purpose HTML5 framework, and not just exclusively designed to support 3D content. This makes it exactly the kind of Web framework that Declarative 3D and glTF both seek to leverage via modern web techniques.

5. Configuring a development system

The montage framework currently offers free beta access to its online development tool Montage Studio [MontageStudio 2014] So a developer need only to point a Chrome web browser at this site to do all further development "in the cloud" using a free Github account to store the project files. However if one prefers to work within with their own local environment, instructions on how to configure those tools are also documented at the Montage Studio website. [MontageDocs 2014]

6. Modify JS/HTML/CSS files as desired

The desired user experience is to allow the user to interact with the Knife in a mock online shopping experience. This means the user can open any of the knife's tools by selecting HTML form elements outside of the scene, as shown in Figure 16. By default any Montage glTF scene also allows the user to orient the scene contents in 3D space with their mouse as well, which in this case nicely approximates the experience of holding and inspecting the actual knife.

a. Getting the glTF object onto the html page

The output files obtained from the collada2gltf tool were copied into `./assets/3d` within the project home directory. See also https://github.com/bcoughl2/btc-cs752/tree/gh-pages/assets/3d on github.

Then to reference those files, `/ui/main.reel/main.html` is updated as shown in Figure 12. Lines 22-35 are inside the `<script type="text/montage-serialization">` block which points the montage scene to `"/assets/3d/SwissArmyKnife.json"`. The scene gets presented in the `<body>` block in line 219.

/ui/main.reel/main.html

```
22                          "sceneView": {
23                                  "prototype": "mjs-volume/ui/scene-view.reel",
24                                  "properties": {
25                                          "element": { "#": "sceneView" },
26                                          "scene": { "@": "scene" }
27                                  }
28                          },
29
30                          "scene": {
31                                  "prototype": "mjs-volume/runtime/scene",
32                                  "properties": {
33                                          "path": "/assets/3d/SwissArmyKnife.json"
34                                  }
35                          },

217     <div data-montage-id="main" data-montage-skin="light">
218         <h1 align="center">Swiss Army Knife - Tools Demo</h1>
219                         <div align="center" data-montage-id="sceneView" class="scene"></div>
```

Figure 12: https://github.com/bcoughl2/btc-cs752/blob/gh-pages/ui/main.reel/main.html

b. Associate glTF Nodes with montage sceneView nodes

Once the knife visible in the page, reference nodes in the glTF json file must be defined and associated by node ID so that they can be accessed and animated. Figure 13 below shows one example of the Bottle Opener node using the node with ID23 to associate the correct node in the glTF `/assets/3d/SwissArmyKnife.json` scene to a node in the montage sceneView defined in `/ui/main.reel/main.html`. This process is also followed for 6 other glTF nodes corresponding to the Main Blade, Small Blade, Saw, Tweezers, Awl and Can Opener.

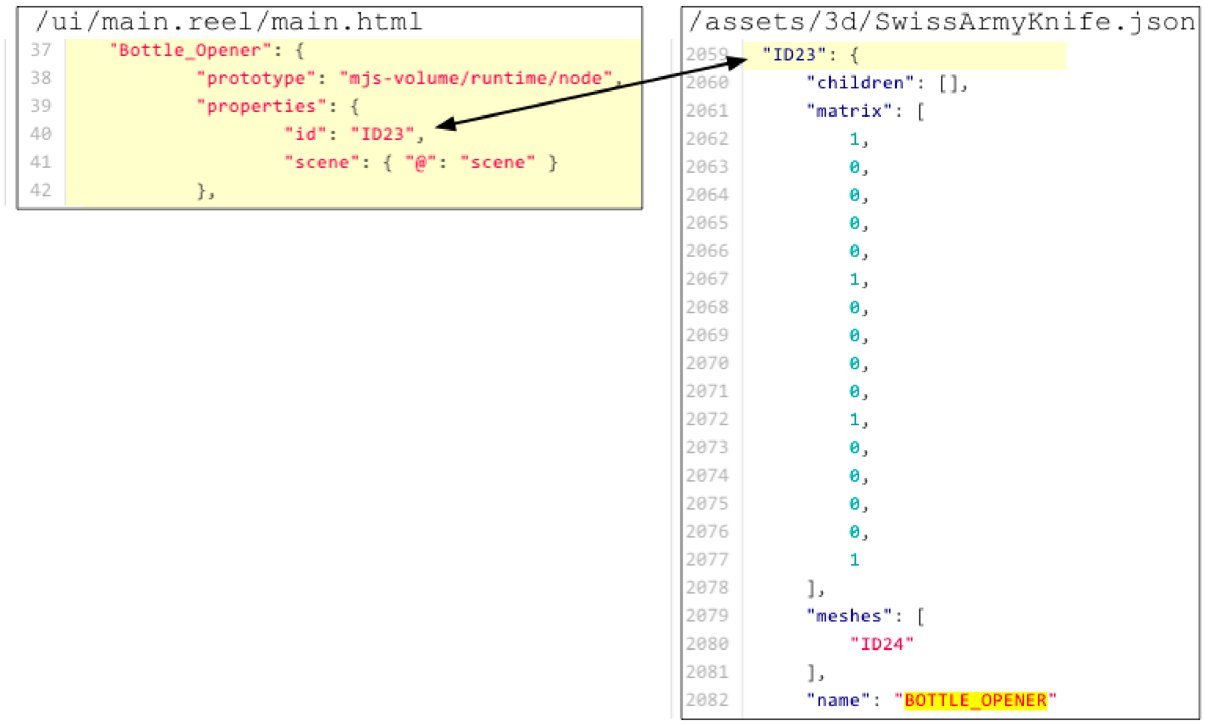

Figure 13: ID23 node linkage from `/ui/main.reel/main.html` sceneView to gltf scene file `/assets/3d/SwissArmyKnife.json`

c. To define the animation for each node, bindings between /ui/main/main.css CSS transformation classes and each node are created. Figure 14 is one example for the Bottle_Opener node.

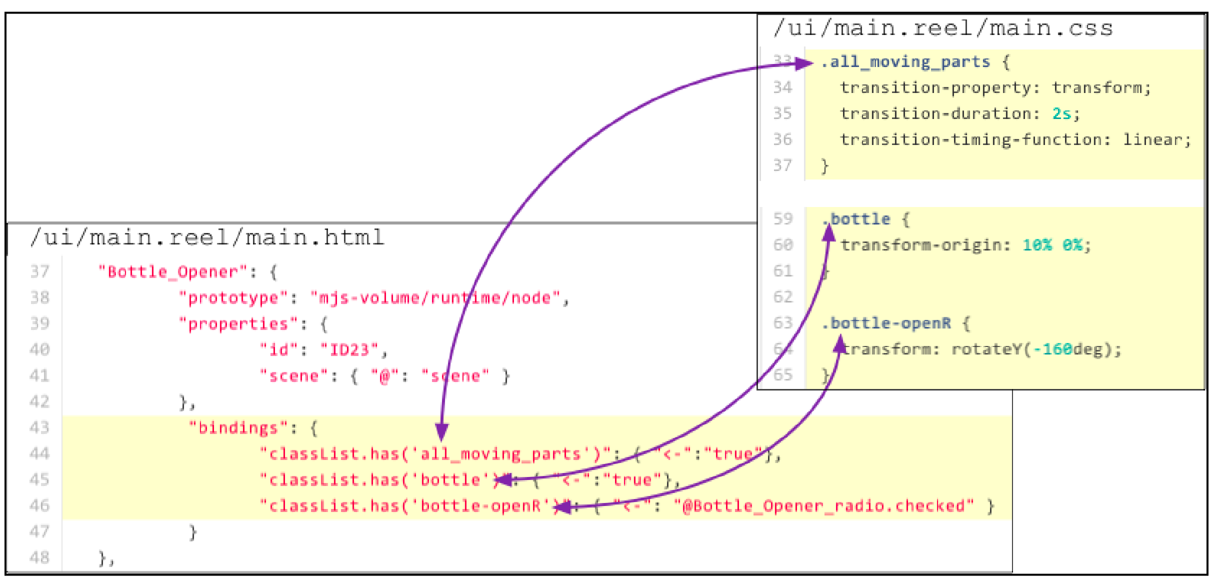

Figure 14: CSS transform class bindings with the Bottle_Opener node

d. Finally in Figure 15, radio buttons elements are created which both trigger a particular node's CSS animation (tool open) when checked, and which return that node to its initial state (tool closed) when unchecked. Note the arrow text "<–" on line 46 that maps the radio button state as an input to whether the "bottle-openR" CSS transform class gets triggered, which is initialized to false in line 144.

```
/ui/main.reel/main.html
                    "Bottle_Opener": {
...
                            "bindings": {
                                    "classList.has('all_moving_parts')": { "<-":"true"},
                                    "classList.has('bottle')": { "<-":"true"},
                                    "classList.has('bottle-openR')": { "<-": "@Bottle_Opener_radio.checked" }
                            }
                    },
...
                    "Bottle_Opener_radio": {
                            "prototype": "digit/ui/radio-button.reel",
                            "properties": {
                                    "element": {"#": "Bottle_Opener_radio"},
                    "checked": false,
                                    "value": "Bottle Opener",
                                    "radioButtonController": {"@": "RBcontroller"}
                            }
                    },
...
<body>
...
                                    <input type="radio" data-montage-id="Bottle_Opener_radio"> Bottle Opener
```

Figure 15: linking Bottle_Opener_radio button state to its CSS binding

7. Deploy to a web server

Throughout developing and iterating this code, it is easiest to run a simple local http server from the top level of the package directory. `npm minit` was used but any simple http server software should work all the same. When debugged and polished, to "go public" this project was deployed to github allowing easy hosting using Github's Pages feature.

The final working page for this case study is hosted at http://bcoughl2.github.io/btc-cs752/ based on the github repository files in the gh-pages branch at https://github.com/bcoughl2/btc-cs752/tree/gh-pages.

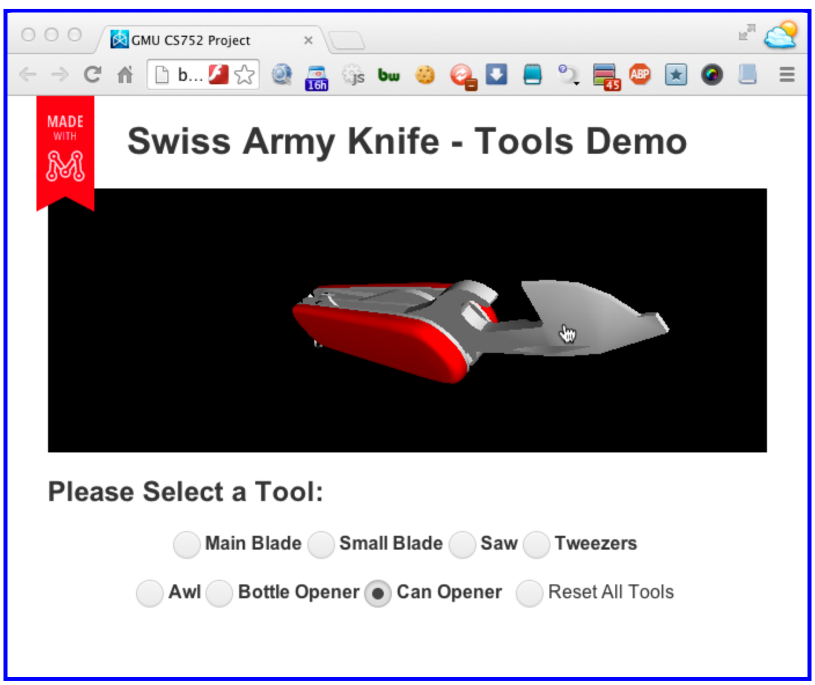

Figure 16: Final finished "Swiss Army Knife - Tools Demo" page!
http://bcoughl2.github.io/btc-cs752/

Lessons Learned

1. The glTF-related software used in the Case Study is still evolving and maturing, so there definitely were dead-ends with attempts to use certain combinations of software. For example, Collada files exported from Autodesk Maya and 3DS tools had problems either crashing the collada2gltf utility, or simply not rendering correctly in MontageJS. This could easily have occurred due to user errors in setting the OpenCollada export options as well which has a large number of options for the user to consider. Also the OpenCollada export is plugin code maintained by Khronos which is itself evolving with newer versions being released each year.

Given the limited time available to complete this project, investigating root cause(s) for these issues became impractical. Using professional-grade 3D content creation tools like Maya and 3DS properly entails a significant learning curve if using them for the first time. Autodesk however deserves credit for making full-featured student versions of these tools freely available for academic use, unlike others which typically only offer trial versions that stop working after a short period.

2. Per good HTML5 practice, presentation aspects of the "Swiss Army Knife - Tools Demo" page is governed by CSS (in the `/ui/main.reel/main.css` file in this project's case). Ironically the most difficult to debug part of the software code written for this project was this CSS. Early versions of the project had different browsers presenting the scene differently which required adding or adjusting certain CSS selectors and their properties.

3. MontageJS documentation normally prescribes using an optimization tool called "mop" (https://www.npmjs.org/package/mop) to combine and minify javascript and CSS files to streamline http delivery of those elements from server to client web browser. However MontageJS developers have identified that mop has some compatibility issues with the MontageJS 3D Components used in this particular project. [MontageForum 2014]

4. Animations using CSS Transforms as supported in web browsers today are fairly simple so they won't apply to all use cases. First, it can only treat node components of scenes as rigid bodies, and CSS transforms are not designed for complex animated movements either but instead simple translations or rotations over some specified amount of time. Also the Xflow technology used of XML3D is quite powerful but also more complex to learn, and so far is only proven to work with XML3D but not glTF. Finally glTF does include its own schema elements for storing animations defined upstream in the content authoring tool if they are included in the Collada file, although how to trigger or control them in a webGL context is not clear.

Areas of Further Study and Research

With the maturity of WebGL well past its tipping point, here are some suggested areas for further research:

- Rendering glTF scenes in Declarative 3D (XML3D probably) !! Feasibility, Options, etc.
- 3D scene file formats: specification and optimization across multiple domains including transmission, client processing, progressive rendering, et. al.
- Optimal Strategies for mapping 3D scene graphs to Web Browser DOM or shadow DOM tree structures
- Declarative techniques for animating WebGL graphics: compare XFlow, W3C CSS & SVG standards, etc.

Conclusion

This paper and case study show how Declarative 3D and glTF are viable and compelling options for bringing 3D content more into mainstream use across the modern Web. Both already are usable today not only by themselves but also with HTML5 frameworks such as MontageJS, which was successfully used in the Case Study and which validated several of the key principles and design goals of both glTF and Declarative 3D.

References

- Declarative 3D W3C Community Group. 2011. http://www.w3.org/community/declarative3d/
- Web3d Consortium. 1999. http://www.web3d.org/
- XML3D Project. 2014. http://xml3d.org/
- X3DOM. 2014. http://www.x3dom.org/
- Jacek Jankowski, Sandy Ressler, Kristian Sons, Yvonne Jung, Johannes Behr, and Philipp Slusallek. 2013. Declarative integration of interactive 3D graphics into the world-wide web: principles, current approaches, and research agenda. In Proceedings of the 18th International Conference on 3D Web Technology (Web3D '13). ACM, New York, NY, USA, 39-45. DOI=10.1145/2466533.2466547 http://doi.acm.org/10.1145/2466533.2466547
- Declarative 3D Group Charter: Scope. 2011. Retrieved August 2014 from Declarative 3D wiki hosted by the W3C. http://www.w3.org/community/declarative3d/wiki/Declarative_3D_Group_Charter#Scope
- Sharma, Bharat. 2014. "Top 10 Best GPU In Mobile And Tablet Devices 2014" http://www.trickolla.com/2014/01/top-10-best-gpu-in-mobile-and-tablet.html
- Jackson, Dean and Eidson, Brady. 2014. "WebGL - Creating Interactive Content with WebGL" http://devstreaming.apple.com/videos/wwdc/2014/509xxwli42i4gs6/509/509_creating_3d_interactive_content_with_webgl.pdf
- Sharp. Remy. 2010. "What is a Polyfill?" http://remysharp.com/2010/10/08/what-is-a-polyfill/
- Kruger, Marcus (Goo Technologies). 2014. Retrieved 2014 from Wired.com. "Flash Is Dead ... Long Live WebGL". http://innovationinsights.wired.com/insights/2014/05/flash-dead-long-live-webgl/
- Khronos. 2013. "COLLADA/glTF BOF" https://www.khronos.org/assets/uploads/developers/library/2013-siggraph-collada-bof/COLLADA-BOF_SIGGRAPH-2013.pdf
- Wikipedia July 16, 2014. http://en.wikipedia.org/wiki/List_of_ECMAScript_engines
- Mammou, Khaled. (Khronos) December 2013. "Open 3D Graphics Compression" https://github.com/KhronosGroup/glTF/wiki/Open-3D-Graphics-Compression
- Cozi, Patrick. December 2013. "glTF and rest3d" http://cis565-fall-2013.github.io/lectures/12-04-glTF-and-rest3d.pptx
- Behr, Johannes and Sons, Kristians. 2012. "Declarative 3D as a Polyfill: TPAC 2012". "Declarative 3D Essentials" https://docs.google.com/presentation/d/1m50QV4MBJjn0iH0yI7Xb3NoM05IDsa8pbJm9-9PRA8U/present#slide=id.i0
- Dale, Tom. May 2013. "Evergreen Browsers" http://tomdale.net/2013/05/evergreen-browsers/
- Khronos. 2013. Typed Array Specification. http://www.khronos.org/registry/typedarray/specs/latest/
- Smus, Boris. 2012. "How the Web should work". http://smus.com/how-the-web-should-work/
- WebComponents. 2014. http://webcomponents.org/
- W3C. 2012. TPAC 2012. Session Ideas: Declarative 3D as Polyfill http://www.w3.org/wiki/TPAC2012/SessionIdeas#Declarative_3D_as_Polyfill
- Khronos. July 2014. "glTF - the runtime asset format for WebGL, OpenGL ES, and OpenGL." repository hosted on github.com. https://github.com/KhronosGroup/glTF
- Trevett, Neil (Khronos). June 2013. 2013 Web3d Conference Presentation. "3D Transmission Format". https://www.khronos.org/assets/uploads/developers/library/2013-web3d-conference/glTF-and-3D-Transmission-Web3D.pdf
- glTF Viewer. 2014. http://playsign.tklapp.com:8000/glTF-webgl-viewer/
- x3dom documentation. http://doc.x3dom.org/tutorials/models/inline/index.html
- MontageStudio. 2014. https://work.montagestudio.com/
- MontageStudio Documentation. 2014. http://docs.montagestudio.com/montagejs/tutorial-3d-applications-with-montagejs.html and http://docs.montagestudio.com/montagejs/montagejs-setup.html
- Montage Forum. 2014. http://forum.montagestudio.com/t/3d-scene-inconsistency-across-browsers-osx-safari-help-please/64
- MontageJS Source Code. 2014. Hosted on Github. https://github.com/montagejs/montage