X3DOM AS CARRIER OF THE VIRTUAL HERITAGE

Yvonne Jung, Johannes Behr, Holger Graf

Fraunhofer Institut für Graphische Datenverarbeitung, Darmstadt, Germany
{yjung, jbehr, hgraf}@igd.fhg.de

Figure 1. With MeshLab exported model of an old statue visualized via X3DOM in the same HTML page on 3 different platforms: iPhone App using WebKit extensions; Internet Explorer C++ based X3D plugin; WebGL-based implementation on Nokia N900.

KEY WORDS: 3D Internet, Declarative 3D in Web-Bowser, X3DOM, Virtual Heritage, Cultural Heritage, WebGL

ABSTRACT:

Virtual Museums (VM) are a new model of communication that aims at creating a personalized, immersive, and interactive way to enhance our understanding of the world around us. The term "VM" is a short-cut that comprehends various types of digital creations. One of the carriers for the communication of the virtual heritage at future internet level as de-facto standard is browser front-ends presenting the content and assets of museums. A major driving technology for the documentation and presentation of heritage driven media is real-time 3D content, thus imposing new strategies for a web inclusion. 3D content must become a first class web media that can be created, modified, and shared in the same way as text, images, audio and video are handled on the web right now. A new integration model based on a DOM integration into the web browsers' architecture opens up new possibilities for declarative 3D content on the web and paves the way for new application scenarios for the virtual heritage at future internet level. With special regards to the X3DOM project as enabling technology for declarative 3D in HTML, this paper describes application scenarios and analyses its technological requirements for an efficient presentation and manipulation of virtual heritage assets on the web.

1. INTRODUCTION

The trend in using more multimedia technologies in our everyday life has also an impact on digital heritage and its overall value chain from digitisation, processing, and presentation within VM platforms. Moreover, 3D interactive content being the information carrier of the future, still requires dedicated research efforts to enable a seamless process chain of integration, composition and deployment. In recent years 3D enhanced environments and 3D content are more and more seen as a provider for the understanding of complex causalities, advanced visual cognitive stimulus and easy interaction. Hence, complex causalities within the VH (Virtual Heritage) information space have to be adapted to the visitors' or users' cognitive capabilities allowing them personalised access to the heritage. The combination of multiple media and diverse ICT platforms have to actively support users in the understanding of 3D topics, providing new motivation in engaging the users within either individual or collaborative digital culture experiences. Thus, museums can make complex causalities immersively available and leverage visitors or VH consumers to a higher quality of experience of CH.

Coming along with 3D interactive content, we are facing a shift in interaction and presentation paradigms for the access to the Virtual Heritage. It still requires tremendous research activities and we are facing several great challenges within information pre-processing, concatenation and presentation being adaptively supported by (de-facto) standard ICT solutions within the CH (Cultural Heritage) domain. This encompasses hardware, e.g. displays and its scalability, but also adaptive software solutions to support a context change for interactive presentations with the ultimate vision to "bridge the gap between heritage-driven multi-media technologies and our natural environment".

On the other side, the internet can be seen as one carrier of future (learning) worlds in which socialising aspects combined

with motivating, easy to use, exhausting and understandable information can be accessed, retrieved and refined. Connecting modern rich media 3D technology with traditional web-based environments, interesting new possibilities for self-regulated and collaborative knowledge dissemination emerge (Jung, 2008). Here we need besides the acquisition and preparation of heritage driven 3D content new methodologies and tools which are able to comply with the requirements of highly dynamic knowledge and information processing within its presentation. This is required for several involved stakeholders, e.g. future digital curators or non-professional visitors of the museum at any age. New workflows for rich media content creation have to be elaborated for enabling e.g. digital curators to easily prepare and provide 3D heritage driven media on the web. Research activities should therefore focus on how to produce and elaborate sustainable and standardised solutions covering the overall content preparation pipeline for 3D content on the web.

Building on the lessons learned in web technology and its applications, we reflect on how to embed heritage driven multiple media content into browser front-ends. Major attention on the conceptual design has been devoted to:

- re-usable application environments allowing the integration of standardised media archiving formats,
- extensibility with respect to the web browser as major interoperable deployment platform,
- declarative heritage-driven 3D content for easy authoring and content concatenation.

Thus, in this paper, we first review suitable techniques for the web-based visualisation of heritage-driven objects before presenting our solution. Nowadays, most 3D rendering systems for web-based applications follow the traditional browser-plugin-based approach, which has two major drawbacks. On the one hand, plugins are not installed by default on most systems and the user has to deal with security and incompatibility issues. On the other hand, such systems define an application and event model inside the plugin that is decoupled from the HTML page's DOM content, thereby making the development of dynamic web-based 3D content difficult.

2. RELATED WORK

Besides the aforementioned browser plugins, Java3D (Sun, 2007) – a scene-graph system that incorporates the VRML/X3D (Web3D, 2008) design – was one of the first means for 3D in the browser. However, it never really was utilized for the web and today Java3D is no longer supported by Sun at all. The open ISO standard X3D in contrast provides a portable format and runtime for developing interactive 3D applications. X3D evolved from the old VRML standard, describes an abstract functional behaviour of time-based, interactive 3D multimedia information, and provides lightweight components for storage, retrieval and playback of real-time 3D graphics content that can be embedded into any application (Web3D, 2008). The geometric and graphical properties of a scene as well as its behaviour are described by a scene-graph (Akenine-Möller et al., 2008). Since X3D is based on a declarative document-based design, it allows defining scene description and runtime behaviour by simply editing XML without the need for dealing with low-level C/C++ graphics APIs, which not only is of great importance for efficient application development but also directly allows its integration into a standard web page. Further, using X3D means that all data are easily distributable and sharable to others. Despite proprietary rendering systems that all implement their own runtime behaviour, X3D allows developing portable 3D applications.

The X3D specification (Web3D, 2008) includes various internal and external APIs and has a web-browser integration model, which allows running plugins inside a browser. Hence, there exist several X3D players available as standalone software or as browser plugin. The web browser holds the X3D scene internally and the application developer can update and control the content using the Scene Access Interface (SAI), which is part of the standard and already defines an integration model for DOM nodes as part of SAI (Web3D, 2009), though there is currently no update or synchronization mechanism. To alleviate these issues, with the X3DOM framework (Behr et al., 2009) a DOM-based integration model for X3D and HTML5 was presented to allow for a seamless integration of interactive 3D content into HTML pages. The current implementation is mainly based on WebGL (Khronos, 2010), but the architecture also proposes a fallback model to allow for more powerful rendering backends, too (Behr et al., 2010), which will be explained in the next section. More information can be found online at http://www.x3dom.org/.

To overcome the old plugin-model, Khronos promotes WebGL as one solution for hardware accelerated 3D rendering in the web. The imperative WebGL API (WebGL, 2010) is a JavaScript (Crockford, 2008) binding for OpenGL ES 2.0 (Munshi et al., 2009) that runs inside a web browser, thereby allowing for native 3D in the web. The very first WebGL implementation was available in late September 2009 with a Mozilla Firefox 3.7 pre-alpha build. Since then, most other browsers like Apple WebKit, Google Chrome and Opera (except Microsoft's IE) followed with WebGL-enabled developer (and now beta) builds. By utilizing OpenGL ES 2.0 as basis, it was possible to define the WebGL specification in a platform independent manner, since on the one hand OpenGL 2.1 (the current standard for desktop machines) is a superset of ES 2.0. And on the other hand, most recent smartphones, like the iPhone or the Nokia N900, already have chips being conformant to that standard – even more, since the latest firmware update early June 2010, the built-in web browser of the Nokia N900 now also natively supports WebGL (and thereby X3DOM – compare Figure 1).

WebGL (WebGL, 2010) describes an additional 3D rendering context for the HTML5 <*canvas*> element (W3C, 2009a) by exposing the rendering API via new JavaScript objects and methods acting on the canvas object. The 3D rendering context is then acquired via *gl = canvas.getContext('webgl')*. If the returned *gl* object is defined and not null, the web browser supports WebGL – in this case the *gl* object provides all API calls. As mentioned, WebGL is based on the OpenGL ES 2.0 standard (Munshi et al., 2009), an OpenGL dialect that was developed for embedded and portable devices such as mobile phones with less powerful graphics chips. In contrast to standard desktop OpenGL (Shreiner et al., 2006) it has no support for the old fixed function pipeline (i.e., no matrix stack etc.) but is instead completely based on GLSL shaders (Rost, 2006). Thereby it is comparable to the OpenGL 3.x/4.x standard with the exception that more advanced features like transform feedback or geometry shaders that require rather recent GPUs are not supported. Another drawback is the fact that the web-developer has to deal with low-level graphics concepts (maths, GLSL-shaders, attribute binding, and so on). Moreover, JavaScript scene housekeeping can soon lead to performance issues, and there is still no uniform notion of metadata or semantics for the content possible.

During the last year, WebGL-based libraries such as WebGLU (DeLillo, 2009), which mimics the old OpenGL fixed-function pipeline by providing appropriate concepts, emerged as well as rendering frameworks building on top of WebGL by providing a JavaScript-based API. For instance GLGE (Brunt, 2010) is a scene-graph system that masks the low-level graphics API calls of WebGL by providing a procedural programming interface. Likewise, SpiderGL (Di Benedetto et al., 2010) provides algorithms for 3D graphics, but on a lower level of abstraction and without special structures like the scene-graph. These libraries are comparable to typical graphics engines as well as to other JavaScript libraries like jQuery (cp. http://jquery.com/), but none of them seamlessly integrates the 3D content into the web page in a declarative way nor do they connect the HTML DOM tree to the 3D content. In this regard, the aforementioned jQuery aims at simplifying HTML document traversing, event handling, and Ajax interactions, thereby easing the development of interactive web applications in general. However, using libraries like SpiderGL forces the web developer to learn new APIs as well as graphics concepts. But when considering that the Document Object Model (DOM) of a web page already is a declarative 2D scene-graph of the web page, it seems natural to directly utilize and extend the well-known DOM as scene-graph and API also for 3D content.

3. GETTING DECLARATIVE (X)3D INTO HTML5

Generally spoken, the open source X3DOM framework and runtime was built to support the ongoing discussion in both, the Web3D and W3C communities, of how an integration of HTML5 and declarative 3D content could look like, and allows including X3D (Web3D, 2008) elements directly as part of an HTML5 DOM tree (Behr et al., 2009; Behr et al., 2010). The proposed model thereby follows the original W3C suggestion to use X3D for declarative 3D content in HTML5 (W3C, 2009b): *"Embedding 3D imagery into XHTML documents is the domain of X3D, or technologies based on X3D that are namespace-aware"*. Figure 2 relates the concepts of X3DOM to SVG, Canvas and WebGL.

Figure 2. SVG, Canvas, WebGL and X3DOM relation.

3.1 DOM Integration

In contrast to other approaches, X3DOM integrates 3D content into the browser without the need to forge new concepts, but utilizes today's web standards and techniques, namely HTML, CSS, Ajax, JavaScript and DOM scripting. Figure 3 shows a simple example, where a 3D box is embedded into the 2D DOM tree using X3DOM. Though HTML allows declarative content description already for years, this is currently only possible for textual and 2D multimedia information.

Hence, the goal is to have a declarative, open, and human-readable 3D scene-graph embedded in the HTML DOM, which extends the well-known DOM interfaces only where necessary, and which thereby allows the application developer to access and manipulate the 3D content by only adding, removing or changing the DOM elements via standard DOM scripting – just as it is nowadays done with standard HTML elements like * *, **, *<img>* or *<canvas>* and their corresponding CSS styles. Thus, no specific plugins or plugin interfaces like the SAI (Web3D, 2009) are needed, since the well-known and excellently documented JavaScript and DOM infrastructure are utilized for declarative content design. Obviously, this seamless integration of 3D contents in the web browser integrates well with common web techniques such as DHTML and Ajax. Furthermore, semantics integration can be achieved with the help of the X3D metadata concept for creating mash-ups (i.e. a recombination of existing contents) and the like or for being able to index and search 3D content.

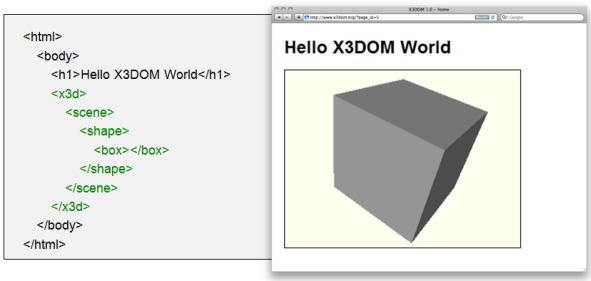

Figure 3. Simple example showing how the 3D content is declaratively embedded into an HTML page using X3DOM.

3.2 Interaction and Events

Most visible HTML tags can react to mouse events, if an event handler was registered. The latter is implemented either by adding a handler function via *element.addEventListener()* or by directly assigning it to the attribute that denotes the event type, e.g. *onclick*. Standard HTML mouse events like "onclick", "onmouseover", or "onmousemove" are also supported for 3D objects alike. Within the X3DOM system we also propose to create a new 3DPickEvent type, which extends the W3C MouseEvent IDL interface (W3C, 2000) to better support 3D interaction. The new interface is defined like follows:

```
interface 3DPickEvent : MouseEvent {
  readonly attribute float worldX;
  readonly attribute float worldY;
  readonly attribute float worldZ;
  readonly attribute float localX;
  readonly attribute float localY;
  readonly attribute float localZ;
  readonly attribute float normalX;
  readonly attribute float normalY;
  readonly attribute float normalZ;
  readonly attribute float colorRed;
  readonly attribute float colorGreen;
  readonly attribute float colorBlue;
  readonly attribute float colorAlpha;
  readonly attribute float texCoordS;
  readonly attribute float texCoordT;
  readonly attribute float texCoordR;
  object getMeshPickData (in DOMString vertexProp);
};
```

Figure 4. Three examples of on-site mobile Augmented Reality (AR) Cultural Heritage applications.

This allows the developer to use the 2D attribute (e.g. screenX) and/ or the 3D attributes (e.g. worldX or localX) if the vertex semantics are given appropriately (in this case the positions). The *getMeshPickData()* method additionally can be used to access generic vertex data. This way, the 2D/ 3D event now bubbles, as expected from standard HTML events, through the DOM tree and can be combined with e.g. a typical 2D event on the X3D element as is shown in the following code fragment:

```
<shape>
  <appearance>
    <material id="mat" diffuseColor="red"></material>
  </appearance>
  <box onclick="document.getElementById('mat').
       setAttribute('diffuseColor', 'green');">
  </box>
</shape>
```

3.3 Animations

There are several possibilities to animate virtual objects (e.g. for showing an ancient device in action etc.), ranging from updating attributes in a script every frame over standard X3D interpolator nodes up to using CSS-3D-Transforms und CSS-Animations, which are currently given as W3C working draft and only implemented in WebKit based web-browsers such as Apple Safari and Google Chrome. While X3D interpolators are supported by current Digital Content Creation (DCC) tools – an important point when processing the raw data and exporting to other formats – and are also able to animate vertex data (e.g. coordinates or colors), CSS animations are easily accessible using standard web techniques. The following code fragment shows an example on how to use CSS-3D-Transforms to update *Transform* nodes for animating their child nodes.

```
<style type="text/css">
  #trans {
    -webkit-animation: spin 8s infinite linear;
  }
  @-webkit-keyframes spin {
     from { -webkit-transform: rotateY(0); }
     to   { -webkit-transform: rotateY(360deg); }
  }
</style>
...
  <transform id="trans">
    <transform
      style="-webkit-transform: rotateY(45deg);">
    ...
    </transform>
  </transform>
...
```

3.4 HTML Profile and Render Backend

As mentioned, X3DOM is based upon the concepts of X3D, which defines several profiles, such as the interchange profile, that can be used as a 3D data format, and the immersive profile, which also defines means for runtime and behaviour control (Web3D, 2008). However, these profiles are not suitable for the integration into the HTML DOM due to several reasons, which are discussed in more detail in (Behr et al., 2009; Behr et al., 2010). Thus, we propose an additional "HTML" profile that basically reduces X3D to a 3D visualization component for HTML5 just like SVG for 2D (cf. Figure 2), while all interaction concepts are taken from standard DOM scripting. As also mentioned in (Behr et al., 2010), the general goal here is to utilize HTML, JavaScript, and CSS for scripting and interaction in order to reduce complexity and implementation effort.

The proposed "HTML" profile extends the X3D "Interchange" profile and consists of a full runtime with animations, navigation and asynchronous data fetching. On the one hand the latter is used for media data like <*img*> and <*video*>, which can directly be used to e.g. parameterize *Texture* nodes. On the other hand this is used for partitioning the scene data via an XMLHttpRequest (XHR) within *Inline* nodes, since 3D data can soon get very big, especially in the Virtual Heritage domain as shown in Figure 7 (bottom row). However, X3D *Script* nodes, Protos, and high-level pointing sensor nodes are not supported, whereas explicit (GLSL) shader materials as well as declarative materials – e.g. via the new *CommonSurfaceShader* node presented in (Schwenk et al., 2010) – are supported both.

While the concept targets at native browser support, the system design now supports different rendering and synchronization backends through a powerful fallback model that matches existing backends and content profiles (compare Figure 5). The flexible open-source implementation of X3DOM already provides various runtime/ rendering backends today. These intermediate solutions are implemented through a WebGL-layer (Behr et al., 2010), which supports WebGL, X3D/ SAI plugins, and native implementations, since using WebGL is slower due to JavaScript and not yet supported by all browsers. The current release of X3DOM supports a native implementation (that is closed source and only for the iOS platform right now), WebGL, and partially X3D/ SAI plugins (like the InstantPlayer ActiveX plugin that can be downloaded from http://www.instantreality.org/). A comparison of these backends is shown in Figure 1. Flash as an additional backend (see Figure 5) will be supported as soon as its 3D API layer (codename "Molehill") is available.

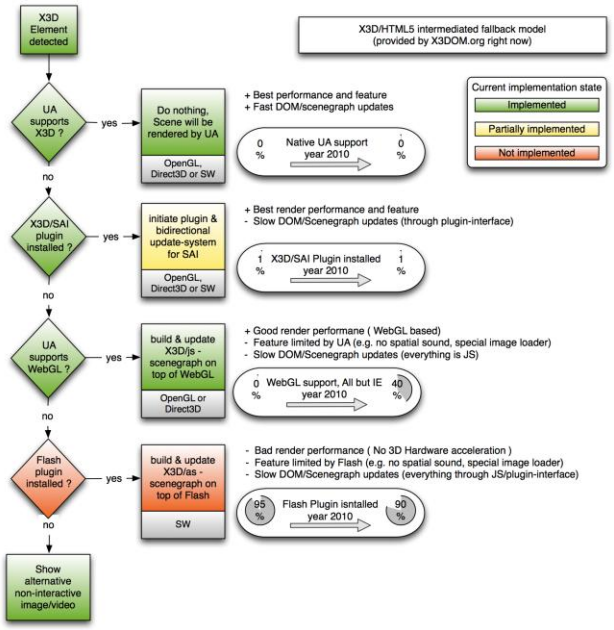

Figure 5. Fallback model: depending on the X3DOM profile and current browser environment, the system automatically chooses the appropriate backend rendering system.

4. WORKFLOW FOR THE HERITAGE ON THE WEB

Besides presentation, i.e. the rendering and user interface part, workflow issues must be considered, too, including tools and tool-chains as well as content and media authoring. While declarative representations help reducing the application development and maintenance efforts, the content first needs to be generated somehow. In general X3DOM is extremely helpful for application- or domain-specific production pipelines. First of all, the utilized format, namely X3D (Web3D, 2008), is an open ISO standard that is a superset of the older VRML ISO standard and which is supported by a large and growing number of Digital Content Creation (DCC) tools. Second, the X3DOM project itself provides a bundle of online- and offline-tools (e.g. plugins and re-coder, see Figure 6) to ease the production and processing of content items. Besides all these techniques the project provides also software components, tutorials, and examples on the web page, which explore and explain how to get the data from a specific DCC Tool, e.g. Maya or 3ds Max (Autodesk, 2011), into your 3D web application.

In virtual heritage, MeshLab (http://meshlab.sourceforge.net/) is an important tool to process and manipulate mesh datasets, which in addition can already export the 3D data into the X3D format, including textures, vertex colors, etc. However, when dealing with 3D scans the vast amount of data is an issue for several reasons. WebGL only supports 64k indices per mesh and therefore large models have to be split. X3DOM splits this automatically if necessary, but besides the memory footprint, loading the data, especially over the web, still takes time. Hence, data reduction should be considered as well. While progressive meshes and similar level-of-detail techniques are applicable here, the original set of normals and colors of the high-res mesh must be conserved for appropriate visual quality, wherefore normal and color maps can be used.

Another issue in the content pipeline one need to think of is annotations and metadata processing. A possible scenario here is 3D content that shall be annotated with metadata to allow for interlinking and concatenation with further information and additional content like HTML sites, multimedia, etc.

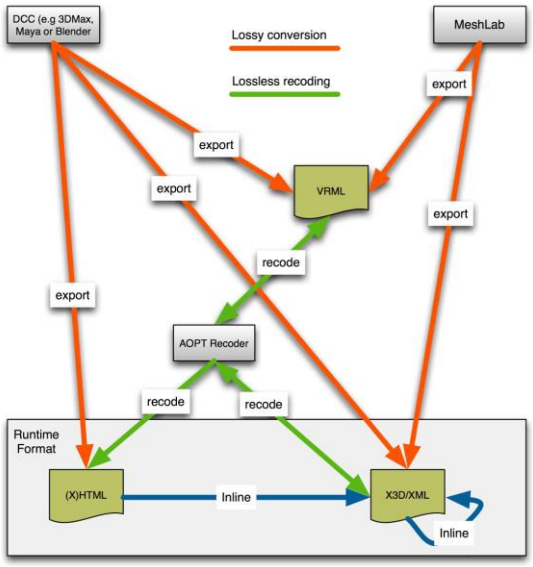

Figure 6. Interactive tools to export and recode data for X3DOM. MeshLab, as one major VH tool, can export X3D data directly, which can be used without further manual adoptions.

5. APPLICATION SCENARIOS AND RESULTS

There already exist several applications that demonstrate the capabilities of X3DOM. Some examples are discussed next in the context of typical scenarios and uses cases.

5.1 Primitive Exploration

One of the most basic use cases one can think of here is the examination of individual objects of the virtual heritage. In a typical scenario the 3D object is presented to the user such that he or she can examine it from all directions by simply moving and rotating it (or the virtual camera respectively) around with the mouse or a similar device. Concerning visualization this is a rather simple scenario in that the 3D scene itself keeps static. Here, Figure 7 shows some screenshots of the web-based visualization of Cultural Heritage objects provided by the V-MusT consortium. As can be seen, all geometric 3D objects are visualized in the web-browser by simply utilizing our open-source X3DOM framework for rendering the 3D content in real-time. This is especially notable in that this is still almost raw data stemming from 3D Laser scans, which is neither reduced nor somehow otherwise prepared for real-time rendering.

Additionally, by extending the web page with some standard JavaScript code for DOM scripting – where appropriate – the user can also interactively manipulate the data using standard 2D GUI elements (e.g. buttons and sliders) as for instance provided by the aforementioned JavaScript library jQuery. This can be useful to vertically or horizontally translate a clipping plane in order to cut away stratigraphic sequences and the like. Furthermore, it is also possible to allow the user to directly interact with an object by clicking on a certain point of interest etc., which then for instance triggers a popup HTML element containing some additional information. More concepts, though in the context of e-learning, are presented in (Jung, 2008).

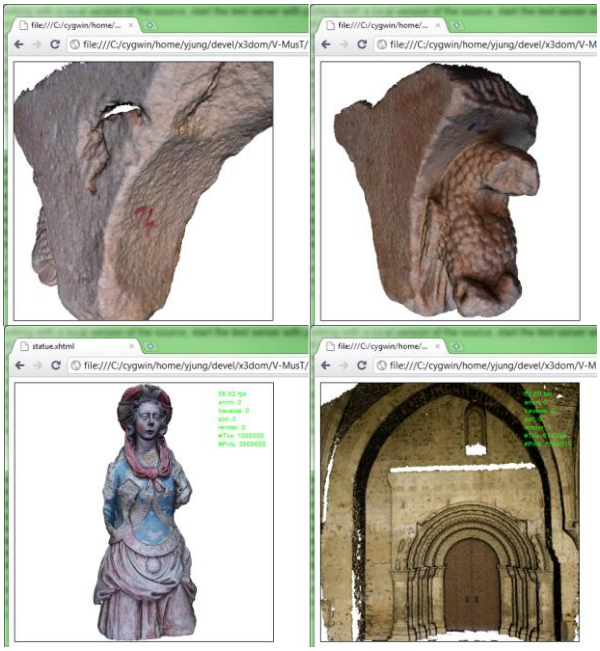

Figure 7. Virtual Heritage objects visualized with X3DOM. Top row: a reconstructed 3D capitel of an abbey that can be freely examined from all directions. Bottom row: the statue to the left is a 63 MB 3D scan and the front of the church shown to the right has 32 MB of vertex data.

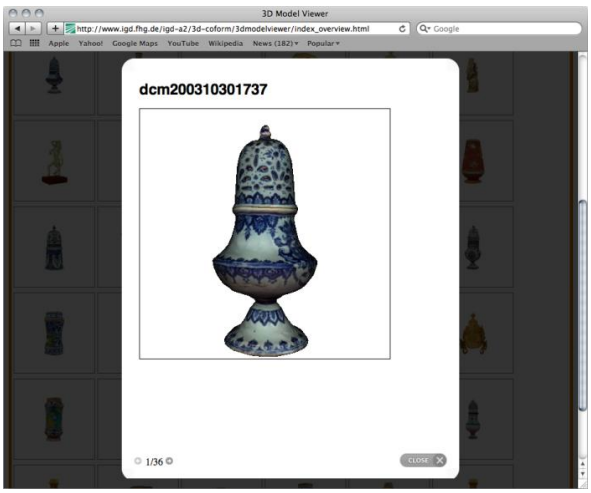

Figure 8. Coform3D – a line-up of multiple scanned 3D objects integrated with X3DOM and JavaScript into HTML.

Another, a bit more intricate application shows a line-up of 3D objects, as it is done with images or videos today. Here, 3D is used as just another medium alike. The 3D Gallery developed within the 3D-COFORM project (http://www.3dcoform.eu/) shows a line-up of over 30 virtual objects. Historian vases and statues were scanned with a 3D scanner. This allows not only a digital conservation of ancient artefacts but offers the possibility for convenient comparison, too. The results have been exported into the X3D file format. The application framework consists of a HTML page with a table grid with 36 cells, each filled with a thumbnail image of a virtual historical object. As soon as the user clicks on a thumbnail, a second layer pops up inside our HTML file showing the reconstructed object in 3D. The user can now closer examine it or he can close the layer to return to the grid again. Technically, we are opening a subpage with the declared X3D content which is rendered by X3DOM. The subpage is loaded inside an HTML iFrame within each layer inside the main page. Figure 8 shows a screenshot.

5.2 Dynamic (Walkthrough) Scenarios

Other possible scenarios in CH embrace walkthrough worlds and the inspection of larger models like ancient city models and similar territories in virtual archaeology. With the Cathedral of Siena (cp. Figure 9) a classical guided walkthrough scenario is described in (Behr et al., 2001). Generally, floor plans (Figure 9, right) are a commonly used metaphor for navigation. This is for two reasons: for one thing the plan allows the user to build a mental model of the environment, and for another it prevents him from getting lost in 3D space. In this regard, camera paths with predefined animations are another frequently used means for guided navigation. In X3DOM camera animations can be easily accomplished by using one of the aforementioned animation methods, like for instance X3D interpolators.

Figure 9.The famous Digital Cathedral of Siena (cf. Behr et al., 2001): the left image shows the rendered 3D view of the cathedral's interior and a virtual guide, and the right image shows the 2D user interface.

Alternatively, the scene author can only define some interesting views and let the system interpolate between them. The resulting smooth camera animations are implemented following (Alexa, 2002). These animations are automatically generated if one binds the camera, e.g. when switching between different Viewpoint nodes (or cameras), which are part of the content. The same method is also used to calculate the animation-path if the current view is being resetted or if the current camera-view shall be moved to the 'show all' position.

As explained previously, it is furthermore possible to freely navigate within the 3D scene in order to closely examine all geometric objects. This is done using the "examine" navigation mode. Besides this, the user can also walk or fly through e.g. a reconstructed city model or an old building as shown in Figure 9. Like every X3D runtime, also the current WebGL-/ JS-based implementation of X3DOM provides some generic interaction and navigation methods. As already outlined, interactive objects are handled by HTML-like events, while navigation can either be user-defined or controlled using specific predefined modes. Therefore, we added all standard X3D navigation modes, i.e. "examine", "walk", "fly" and "lookAt". The content creator is free to activate them, for instance directly in the X3D(OM) code with *<navigationInfo type='walk'>*, or to alternatively write his own application-specific navigation code. In the WebGL-based implementation the modes use the fast picking code (required for checking front and floor collisions) based on rendering the required information into a helper buffer as described in (Behr et al., 2010), which performs well even for larger worlds.

5.3 (Mobile) On-Site AR Scenarios

Figure 4 shows some examples of on-site mobile Augmented Reality CH applications. AR as a rapidly emerging technology combined with the ubiquitous computing power of modern mobile devices means having the desired information in ones pocket. With the help of the video-see-through effect the information – such as 2D images from former times as shown in Figure 4 (right) or the 3D reconstruction of an old temple as shown in Figure 10, which shows some results from Archeoguide (cf. e.g. Vlahakis et al., 2002) – can be superimposed onto the video image, or the real world respectively, by using computer-vision-based tracking techniques. Archeoguide is an example of an outdoor AR system, that utilizes X3D for content description and runtime behaviour, whereas the whole application logic is written in JavaScript. The X3D scene consists of three different layers: the video in the background, the 3D reconstruction of a temple that does not exist anymore, and the user interface. For being able to realize both, the tracking as well as the rendering part, the aforementioned Mixed Reality framework Instant Reality is used as basis.

Figure 10. Archeoguide – example of an outdoor AR system (note the virtual temples and additional information that is rendered on top of the real scene, where only ruines are left).

In this context the term Mixed Reality means to be able to bring together (web-) content and location-based information directly on site. Especially when producing content for (mobile) MR applications, the unification of 2D and 3D media development is an essential aspect. Other important factors for authoring and rapid application development are declarative content description, flexible content in general (not only for the cultural heritage domain, but also for the industry etc.), and interoperability – i.e., write once, run anywhere (web/ desktop/ mobile). In X3DOM this is achieved by utilizing the well-known JavaScript and DOM infrastructure also for 3D in order to bring together both, open architectures and declarative content design known from web design as well as "old-school" imperative approaches known from game engine development. The app-independent visualization furthermore enables context sensitive and on-demand information retrieval, which is even more of interest for distributed content development using available web standards. But when limiting oneself to the pure WebGL-based JS layer of X3DOM, at the moment special apps for handling the tracking part are still needed (e.g. by using Flash or the InstantPlayer plugin), because access to the camera image data is required but not yet supported in HTML5. However, with the recently proposed <*device*> tag even this might change in the near future.

6. CONCLUSIONS

We have presented a scalable framework for the HTML/X3D integration, which on the one hand provides a single declarative developer interface, that is based on current web standards, and which on the other supports various backends through a powerful fallback model for runtime and rendering modules. This includes native browser implementations and plugins for X3D as well as a purely WebGL-based scene-graph – hence easing the deployment of 3D content and bringing it back to the user's desktop or mobile device.

The benefit of our proposed model is the tight integration of declarative (X)3D content directly into the HTML DOM tree without the need to forge new concepts, but by using today's (web) standards. Similar to images or videos today, 3D objects become just another medium alike. As a thin layer between HTML and X3D we deliver a connecting architecture that employs well-known standards on both sides, such as the CSS integration, thereby easing the users' access. Even more, by building upon appropriate standards, we also give a perspective towards more sustainable 3D contents.

7. ACKNOWLEDGEMENTS

Thanks to Daniel Pletinckx and VisualDimension for providing some of the 3D assets and models.

8. REFERENCES

Akenine-Möller, T., Haines, E., Hoffmann, N., 2008. Real-Time Rendering. AK Peters, Wellesley, MA, 3rd edition.

Alexa, M., 2002. Linear combination of transformations. In Proc. SIGGRAPH '02, ACM, New York, USA, pp. 380-387.

Autodesk, 2011. Autodesk 3ds Max 2011. http://area.autodesk.com/3dsmax2011/features.

Behr, J., Fröhlich, T., Knöpfle, C., Kresse, W., Lutz, B., Reiners, D., Schöffel, F., 2001. The Digital Cathedral of Siena - Innovative Concepts for Interactive and Immersive Presentation of Cultural Heritage Sites. In Bearman, David (Ed.): Intl. CH Informatics Meeting. Proceedings: CH and Technologies in the 3rd Millennium. Mailand, pp. 57-71.

Behr, J., Eschler, P., Jung, Y., Zöllner, M, 2009. X3DOM - a DOM-based HTML5/ X3D integration model. In Stephen Spencer, editor, Proceedings Web3D 2009: 14th Intl. Conf. on 3D Web Technology, New York, USA, ACM, pp. 127–135.

Behr, J., Jung, Y., Keil, J., Drevensek, T., Eschler, P., Zöllner, M., Fellner, D., 2010. A scalable architecture for the HTML5/ X3D integration model X3DOM. In Stephen Spencer, editor, Proceedings Web3D 2010: 15th Intl. Conference on 3D Web Technology, New York, USA, ACM Press, pp. 185-194.

Di Benedetto, M., Ponchio, F., Ganovelli, F., Scopigno, R., 2010. SpiderGL: a JavaScript 3D graphics library for next-generation WWW. Web3D 2010, New York, USA, ACM Press, pp. 165-174.

Jung, Y., 2008. Building Blocks for Virtual Learning Environments. In Cunningham, S. (Ed.) et al.; Eurographics: WSCG 2008, Communications Papers. Plzen, University of West Bohemia, pp. 137-143.

Brunt, P., 2010. GLGE. http://www.glge.org/.

Crockford, D., 2008. JavaScript: The Good Parts. O'Reilly, Sebastopol, CA.

DeLillo, B., 2009. WebGLU JavaScript library. http://github.com/OneGeek/WebGLU.

Khronos, 2011. WebGL specification, working draft. https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/doc/spec/WebGL-spec.html.

Munshi, A., Ginsburg, D., Shreiner, D., 2009. OpenGL ES 2.0 Programming Guide. Addison-Wesley, Boston.

Rost, R., 2006. OpenGL Shading Language. Addison-Wesley, Boston, 2nd edition.

Schwenk, K., Jung, Y., Behr, J., Fellner, D., 2010. A Modern Declarative Surface Shader for X3D. In ACM SIGGRAPH: Proceedings Web3D 2010: 15th Intl. Conference on 3D Web Technology. New York, ACM Press, pp. 7-15.

Shreiner, D., Woo, M., Neider, J., Davis, T., 2006. OpenGL Programming Guide. Addison-Wesley, Boston, 5th edition.

Sun, 2007. Java3d. https://java3d.dev.java.net/.

Vlahakis, V., Ioannidis, N., Karigiannis, J., Tsotros, M., Gounaris, M., Stricker, D., Gleue, T., Dähne, P., Almeida, L., 2002. Archeoguide: An Augmented Reality Guide for Archaeological Sites. In IEEE Computer Graphics and Applications 22 (5), pp. 52-60.

W3C, 2009a. Html 5 specification, canvas section. http://dev.w3.org/html5/spec/Overview.html#the-canvas-element.

W3C, 2009b. Html 5 specification draft, declarative 3D scenes section. http://www.w3.org/TR/2009/WD-html5-20090212/no.html#declarative-3d-scenes.

W3C, 2000. Document Object Model Events. http://www.w3.org/TR/DOM-Level-2-Events/events.html#Events-MouseEvent.

Web3D, 2008. X3D. http://www.web3d.org/x3d/specifications/.

Web3D, 2009. Scene access interface (SAI). http://www.web3d.org/x3d/specifications/ISOIEC-FDIS-19775-2.2-X3D-SceneAccessInterface/.