(Article begins on next page)

3DHOP: 3D Heritage Online Presenter

Marco Potenziani[a], Marco Callieri[a], Matteo Dellepiane[a], Massimiliano Corsini[a], Federico Ponchio[a], Roberto Scopigno[a]

[a] *Visual Computing Lab, ISTI CNR, Pisa, Italy*

Abstract

3D Heritage Online Presenter (3DHOP) is a framework for the creation of advanced web-based visual presentations of high-resolution 3D content. 3DHOP has been designed to cope with the specific needs of the Cultural Heritage (CH) field. By using multiresolution encoding, it is able to efficiently stream high-resolution 3D models (such as the sampled models usually employed in CH applications); it provides a series of ready-to-use templates and examples tailored for the presentation of CH artifacts; it interconnects the 3D visualization with the rest of the webpage DOM, making it possible to create integrated presentations schemes (3D + multimedia). In its design and development, we paid particular attention to three factors: easiness of use, smooth learning curve and performances. Thanks to its modular nature and a declarative-like setup, it is easy to learn, configure, and customize at different levels, depending on the programming skills of the user. This allows people with different background to always obtain the required power and flexibility from the framework. 3DHOP is written in JavaScript and it is based on the SpiderGL library, which employs the WebGL subset of HTML5, implementing plugin-free 3D rendering on many web browsers. In this paper we present the capabilities and characteristics of the 3DHOP framework, using different examples based on concrete projects.

Keywords: online presentation, WebGL, 3D Web, web based 3D rendering, online 3D content deployment, Cultural Heritage

1. Introduction

It is becoming much easier to deal with 3D content on the web. Due to recent hardware and software advancements, the 3D web is moving away from the “swamp” of proprietary, heavy-weight plugins. Nevertheless, specific niches in the world of potential users of the 3D web media, which are somehow far from the mainstream use of 3D data, are still uncovered. One of these peculiar user groups is the one focusing on Cultural Heritage (CH) and using high resolution 3D models of real-world artifacts. Digital 3D models of CH artifacts are nowadays widespread and, beside their more “technical” uses (documentation, restoration support, study and measurement) they are becoming very valuable in dissemination, teaching and presentation to the public. Even if there are applications where lower-resolution hand-modeled 3D models may suffice, in many other cases high-resolution digitized geometries are essential to convey correct information.

This paper presents a software framework, 3DHOP (3D Heritage Online Presenter), designed to cope with the needs of this specific user group. The use of 3DHOP simplifies the creation of interactive visualization webpages, able to display high-resolution 3D models, with intuitive user interaction/manipulation; moreover, these resources can be deeply connected with the rest of the webpage elements (Figure 1).

Please note that CH is not the only application domain dealing with very high-resolution models and requiring a dense interconnection between those models and other data or media. In this sense, CH is a major domain of inspiration and assessment for our activity, but not the only application context for 3DHOP technology.

The most interesting characteristics of the 3DHOP framework are:

- The ability to work with extremely complex 3D meshes or point clouds (tens of million triangles/vertices), using a streaming-friendly multiresolution scheme.
- The ease of use for developers, especially those with background in web programming, thanks to the use of declarative-style scene creation and exposed JavaScript functions used to control the interaction.
- The availability of a number of basic building blocks for creating interactive visualizations, each one configurable, but at the same time providing sensible defaults and comprehensive documentation.

3DHOP is based on the WebGL subset of HTML5, and on SpiderGL [1], a JavaScript support library oriented to advanced Computer Graphics (CG) programming. Thanks to this, 3DHOP works without the need of plugins on most modern browsers (Google Chrome, Mozilla Firefox, Internet Explorer, Safari and Opera) on all platforms. On mobile devices the support is still ongoing in some cases, but this situation will improve in the near future. 3DHOP has been released as open source (GPL licence) in April 2014, and it is available to be tested and used. The downloadable package, with documentation, a series of tutorials (How-Tos) and a Gallery of examples is available at the website: **http://3dhop.net**.

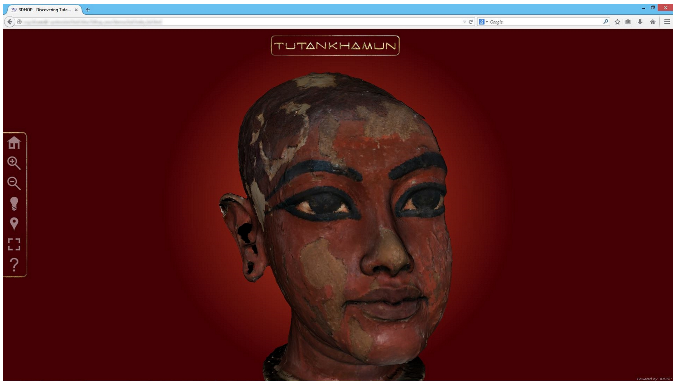

Figure 1: The *Tutankhamun* viewer: using 3DHOP to publish on the Web a high resolution 3D model explorable in a simple, intuitive and interactive way (the artifact is linked to additional multimedia information through hotspots). This example is available in the Gallery section of the 3DHOP website.

2. Related work

Here, we focus on three main aspects of the 3DHOP framework. First, we review the technologies to handle the 3D content on the Web, than we present some solutions about how to transmit the 3D content efficiently. For completeness we report also some works related to the offline visualization of huge models, by focusing mainly on papers related to our framework.

2.1. 3D content on the Web

As soon as three-dimensional content became a consolidated type of multimedia material, its visualization in the context of web pages became an issue, since 3D models were not considered as a "native" type of data. Initially, visualization of 3D components was devoted to embedded software components, such as Java applets or ActiveX controls [2].This led to a lack of standardization and to a quite limited use of 3D content on the web.

A first step to find at least a common format for 3D data were the efforts converging towards the Virtual Reality Modelling Language (VRML) [3], started in 1994, and the more recent X3D [4] (2004). However, 3D scene visualization was still delegated to external software components.

The advent of the WebGL standard [5], promoted by the Khronos Group [6], brought to a remarkable change. WebGL, which is a mapping of OpenGL|ES 2.0 [7] specifications in JavaScript, allows web browsers to directly access the graphics hardware. WebGL has been the starting point for a number of actions for having advanced 3D Graphics on the web. An interesting and up-to-date overview of the current status is provided by the survey from Evans et al. [8].

From a general point of view, the solutions proposed in literature can be divided in two groups:

- The first class of systems extended the effort of X3D by structuring the description of the 3D content in a *declarative* fashion [9], essentially based on the *scenegraph* concept. Two interesting examples of declarative programming solutions are X3DOM [10] and XML3D [11].
- Alternatively, the *imperative* approach considers the computation as "a series of statements which change a programme state". A number of high-level libraries have been developed to help non-expert users using WebGL. Most of them are based on the use of JavaScript as a basic language. They range from scene-graph-based interfaces, such as Scene.js [12], GLGE [13] and Three.js [14], to more programmer-friendly paradigms, such as SpiderGL [1] and WebGLU [15]. The most successful of these libraries is Three.js which has been used in several small and medium size projects.

The comparison between the declarative and imperative approaches is not trivial, since none of them is able to perform better in all the possible applications. The performance is mainly related to the complexity and goal of the 3D graphics application, as it will be also discussed in the next sections. Evans et al. [8] point out also that declarative approaches had a major impact in the research community, while imperative approaches were mainly used in the programming community.

From a more general point of view, the system presented in this paper deals also with the issue of integrating 3D models with other types of data, such as text or images. This has been recently taken into account by a few recent works that explored the integration of text and 3D models on the web [16, 17, 18]. The Smithsonian X3D explorer [19], developed as a "branch" application of the Autodesk Memento engine [20], is an alternative example where 3D models are associated/linked to additional content, but we miss detailed information on the structure and flexibility of the Smithsonian system.

2.2. 3D online streaming

The plugin-free solutions together with the availability of high-level libraries have pushed the development of rich 3D web applications, thus increasing the demand to transmit efficiently sophisticated (and often huge) 3D scenes.

As pointed out in many works [21, 22, 23], the transmission of 3D content should follow precise requirements in order to be efficient for web applications. First, the latency before visualization should be minimized. Second the model representation should permit different level of details (LoD) to account for the rendering capabilities of different devices. Having different LoD at disposal allows also to reduce the latency time before the first visualization. Compression is also another important aspect, to make it possible to provide large 3D datasets on connections with average bandwidth. For compressed streaming, decompression time becomes crucial in order to avoid bottlenecks.

Some recent works focused on a better organization of generic streamable formats [24, 25], but when the 3D structures become very big, it is necessary to think about ad-hoc solutions.

For the above reason, progressive compression methods are good candidates for streaming 3D content. Despite this, many methods based on progressive meshes (originally developed by Hoppe [26]) cannot be directly adapted for the Web because the research efforts in this direction have focused on obtaining high compression ratios and not, for example, to improve decompression time or to allow the progressive compression of attributes like color or texture.

Only in the last three years, some ad hoc compression methods for 3D streaming have been developed. Gobbetti et al. [27] proposed to transmit 3D models for which it is possible to compute a parametrization, so that they can be converted into a quad-based multi-resolution format. Behr et al. [22] used different quantization levels for the model vertices and transmit them using a set of nested GPU-friendly buffers (called *POP buffer*). This completely avoids the problem of decompression, making them suitable also for low-end devices, such as smartphones. Lavouè et al. [21] proposed an adaptation for the Web (reduced decompression time at the cost of a low compression ratio) of the progressive algorithm of Lee et al. [28] which is based on the valence-driven progressive connectivity encoding proposed by Alliez and Desbrun [29]. During the encoding the mesh is iteratively simplified (decimation+cleaning). At each simplification step the connectivity, the geometry and the color information of each removed vertex are encoded and written in the compressed stream. At the end, typically a triangle requires only 2.9 bytes to be represented (without color information).
Other research has been also conducted to handle other types of data, like point clouds [30], which may present different types of issues to contend with.

The 3DHOP solution is based on a multi-resolution data structure which allows the client to efficiently perform view-dependent visualization. Together with the low granularity of the multi-resolution this approach allows interactive visualization of large 3D models with no high bandwidth requirements (a 8 Mbit/s is sufficient for good interaction with huge models). For further details see Section 4.1.

2.3. Offline visualization of huge 3D models

The visualization of complex geometries has been an issue in computer graphics well before the possibility to have web-based solutions.

Some of the issues related to 3D streaming had to be faced also in this context, and different approaches have been proposed, like LOD based [31, 32] methods, but one of the most interesting solutions was proposed by the seminal paper by Hoppe [26], which proposed a progressive refinement of the geometry during visualization. Following this work, a number of so-called multi-resolution and multi-triangulation solutions have been proposed. They mainly differ on the multiresolution representation [33, 34], on the support of color encoding [35], or on other aspects (a survey on these method was provided by Zhang [36]). Alternative research tracks are devoted to other types of data, like point clouds [37].

More recent work on this topic was devoted to the issue of data compression [28] or to overcome the fact that multi-resolution was mainly created for visualization and not for processing [38].

More in general, the data structures used for offline visualization may be adapted to web rendering, provided that they are compliant with its requirements (i.e. latency, decompression time). An alternative proposed solution was to still devote the rendering effort to a powerful server, and send to the user only a rendered image of the high resolution mesh [39].

3. Design choices of the 3DHOP framework

3DHOP has been designed with the aim of being easy to use, especially for people having a background in Web development, thus without requiring solid knowledge in CG programming.

Our core idea was to mimic the philosophy of those pre-made html/javascript components available online, for example for image slideshow, date or color picker, charts and graphs. These components can be simply plugged inside a webpage including some scripts and adding few lines of HTML, and used by just changing some variables; they interact with the rest of the webpage with a series of exposed javascript functions and events. Most web developers have experience with similar components, and they are indeed extremely useful, given their quick startup, different level of configurability (from a simple parameter change to advanced modding) and integration with the rest of the webpage. It is clear that directly using WebGL, or (better) relying on one of the higher level libraries, frameworks and paradigms like XML3D, X3DOM, Three.js, Scene.js, it could be possible to create interactive presentation like the ones made with 3DHOP (or the entire 3DHOP tool) from scratch, but this would still be an "ad-hoc" effort. 3DHOP may be somehow restricting, with respect to a project-specific custom viewer, but we believe the ready-made components and behaviours and their reusable nature make it a valuable tool.

Most of the design choices address specific needs of the CH domain, providing a series of features that are extremely relevant to this sector.

3.1. Background: situating 3DHOP w.r.t to the state-of-the-art

3DHOP is not a "silver bullet", able to support any possible application or visual communication project, but a framework designed to deal with specific needs.

It is an ideal tool to visualize high-resolution single objects (especially with dense models coming from 3D scanning, see Figure 2) or, more in general, a simple static scene composed of complex models. Conversely, 3DHOP is not suited to manage complex scenes made of low-poly objects (this is a common case when working with CAD, procedural or hand-modeled geometries).

3DHOP makes possible a fast deployment process when dealing with simple interaction mechanisms, making it a good choice for quickly creating interactive visualizations for a large collection of models. Additionally, 3DHOP integrates extremely well with the rest of the webpage, thanks to its exposed JavaScript functions. The ideal situation is having the logic of the visualization scheme in the page scripts, and using 3DHOP for the 3D visualization. Trying to build an interface directly in the 3D space using its components (i.e. clickable geometries used as buttons) is certainly possible, but the results do not scale well

Figure 2: The simplest 3DHOP incarnation, featuring a simple viewer for a single 3D model. This example is available in the *How-To* section of the 3DHOP website.

with the needed configuration work. In the following, three existing alternative solutions are analyzed, in order to better stress similarities and differences.

Unity [40] is one of the most common tools for displaying interactive 3D content on the web for CH applications, a de-facto standard in this specific field. It is natural, then, to compare 3DHOP with Unity. Unity is a full-fledged game engine, extremely powerful and complete, providing advanced rendering, sound, physics and a lot of pre-defined components and helpers. Unity supports the implementation of interactive visualizations holding the same level of graphics and interaction complexity as a modern videogame. It has a rapid development time when creating a simple visualization, but the complexity of use/development ramps up if it is necessary to employ the more complex interaction features. Moreover, Unity is not well suited to manage high-resolution sampled geometry (except for terrains), while it is really good with hand-modeled geometry. Its streaming capabilities requires to pay a fee and also requires server-side computations. Finally, even if there are different ways to interconnect the 3D visualization with the webpage, this is one of the more complex features to set up in Unity, conversely to the otherwise user-friendliness of the tool. All these features make Unity somehow complementary to 3DHOP: the web-integrated visualization of single, high-res artifacts finds in 3DHOP a better support, while the exploration of complex modelled scenes or even immersive environments are better managed in Unity.

Another popular solution for fast online deployment of 3D models is Sketchfab [41]. Widely used, even by the CH community, it is indeed extremely simple to use and offers data storage support. On the downside, Sketchfab has a limit on the geometrical complexity of the input models, making it difficult or impossible to upload 3D scanned models at full resolution. Moreover, the interaction with the 3D models is only partially configurable, making it difficult to tailor the interaction to the specific shape and characteristics of the model. Additionally, models are stored on a remote server, raising issues of data privacy and data property. Finally, being the result of a commercial initiative, the more advanced features (including the handling of more complex geometries) are available only in the Pro version.

X3DOM [10] is another development platform that gained a quite broad range of applications. As already introduced, the X3DOM structure derives from a declarative approach and the definition of the scene is obtained through a *scenegraph* concept and related commands. While X3DOM has several points in common with 3DHOP, it is misleading to compare them directly, since X3DOM is more akin to programming language (based on the declarative paradigm), while 3DHOP is a set of configurable components (built using a different paradigm). X3DOM does implement default field values (following the specifications of X3D), and it provides most of the basic components of 3DHOP. Nevertheless, even creating a simple visualization requires dealing with the complete setup of the rendering and interaction. No code for simple examples is directly available from the official website, making it difficult for those with limited programming skills to obtain a step-by-step understanding. Finally, X3DOM has a ready-to-use solution to handle high-resolution geometries [22], but its performances is worse than what can be obtained with 3DHOP (see the results of the comparison in Section 4.1.1).

3.2. Declarative-style setup

Two main development paradigms support the development of 3D web applications: the *declarative* approach for the management of 3D content, e.g. endorsed by X3DOM; and the *imperative* approach, supported by the introduction of WebGL in HTML5. The use of *declarative* 3D mimics the way the rest of the webpage is composed and managed: 3D entities (geometries, transformations, camera, animations...) are declared and controlled as they are part of the DOM structure (like, for example, a DIV or an image). This approach makes things much simpler for people coming from the web development side.

Conversely, the *imperative* approach works in a way that is more similar to the implementation of stand-alone visualization software, by tapping into the capabilities of the graphics card using a more low-level programming. In most cases, it is like having the browser running an extremely powerful, stand-alone software, disconnected from the rest of the information available on the website.

If we apply a strong simplification of the current status, we may argue that the declarative approach is much easier for web developers, not requiring specific knowledge on 3D programming, and provides seamless integration with the webpage, simplifying the development of interactive presentations of mixed data (3D/text/images/videos). On the other hand, the imperative approach enables the user to fully exploit the power of the graphic cards, at the cost of requiring much more effort in application implementation. Of course, things are never so simple, and lot of effort has been spent on both sides to reduce the separation of these two development paradigms. However, this dichotomy is still holding and, depending on the personal background, it is quite easy to approach 3D Web applications design only considering one of the two paradigms, ignoring or misjudging the possibility offered by the other.

Our goal was to bridge the gap between these two worlds, by providing a framework that aims to combine the ease of use of the declarative style (to define the elements of the visualization and their properties), with the rendering power provided by low-level programming. We will describe in Section 4.2 how the creation of the scene follows a declarative style in 3DHOP, enabling a quick and intuitive (yet, highly customisable) deployment. At the same time, the core of the rendering exploits the experience matured in the field of CG programming (see Section 4.1).

3.3. DOM interconnection

A quite common situation, especially when using imperative 3D systems, is the strong separation between the 3D visualization and the rest of the webpage. In most cases, the visualization tool is completely self-contained, not interacting with the elements of the page. This creates difficulties in creating multimedia presentations, where an action on the webpage elements does affect the 3D visualization and vice-versa.

The system presented by Callieri at al. [17] was aimed at establishing a strong connection between what happens in the 3D viewer and the DOM elements, thus creating an integrated presentation context for different media. While succeeding in effectively connecting the imperative 3D to the DOM, the system was still limited by its specialisation. It is possible, by changing some configuration files, to display a different dataset, but the new object should be quite similar in terms of structure and semantics (the tool was tailored to CH artifacts with scenes carved on their surface, like, for example the Trajan column).

Conversely, 3DHOP was designed to support the interconnection with the elements of the DOM in a more extended and configurable way. 3DHOP can work just as a blind viewer (if the user does not configure any DOM interaction), but it offers many ways to interconnect the visualization to the rest of the webpage. It is possible to change the visibility of the different models; select, read and animate the trackball position; activate hotspots and detect clicks on the 3D models/hotspots. Most of these features can be controlled just by invoking or by registering event-handling JavaScript functions provided in the framework. In this way, the web developer has the complete freedom to integrate 3DHOP with the specific website logic.

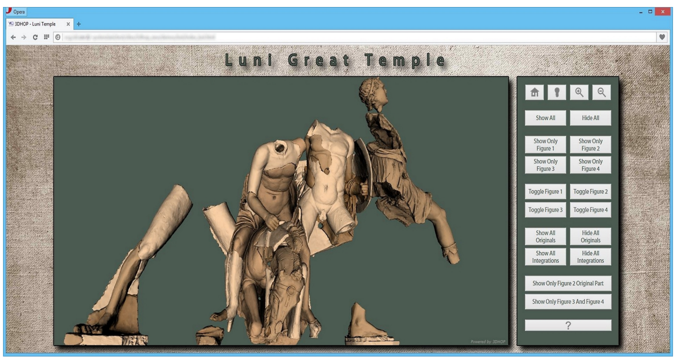

Figure 3: The *Luni Statues* viewer: in this example, four figures of the frieze of the Great Temple of Luni (Italy) are shown. Each statue has an original part and an integration (eight models for a total of 14 millions triangles); by using the visibility control, it is possible to control which subset of the pieces is shown. This example is available in the Gallery section of the 3DHOP website.

3.4. Exhaustive defaults and level of access

Another essential design choice of 3DHOP is to provide a default behavior, consistent with the common needs of our focus community. Each component of the viewer is configurable, but it is never mandatory to modify/update each parameter. The developer may just change a single needed parameter, and rely on defaults for the rest of them. In a wide sense, we follow the *batteries included* philosophy of Python, since we aim to simplify the life of the developer providing ready-to-use visualization components for online CH applications. In this way, our framework is much more accessible, and can be easily learned step by step (using the provided examples and *How-To* resources). This also provides a fast startup when deploying new content (in many cases it is only necessary to do minor changes to the provided examples) and it is ideal to automate the creation of "3D galleries" when a large number of objects have to be presented, since the basic visualization can be easily created by a script. A completely unskilled developer may readily start using 3DHOP to visualize his own dataset by simply downloading one of the examples and changing the name of the 3D model file. Then, it will be easy to modify the parameters of existing elements to achieve more advanced results. A web developer could approach the tool from another direction, by modifying the CSS/HTML to customize the graphic of the visualization. By using JavaScript, it will be then possible to connect the behavior of 3DHOP to the active elements of the webpage. A programmer with some skills in Javascript and computer graphics may modify the trackball or try to add a new trackball to obtain a different interaction, or to customise the rendering by changing the shaders or the rendering of the scene. More expert developers can add new elements in the scene, setup new event hooks and heavily modify the viewer.

3.5. Online and offline deployment

While the 3DHOP framework has been designed for online applications, we also made possible its use on a local machine. Given its minimal interface, compatible with touchscreens, and the ability to work without a dedicated server, 3DHOP is a good candidate for the creation of multimedia kiosks and interactive displays running on local machines inside a museum or an exposition. When deployed on the web, 3DHOP does not require a dedicated server or server-side computation; some space on a web-accessible server is enough to publish visualization webpages. This makes deployment easier also for institutions without complex IT infrastructure (like most museums); moreover, this self-publishing also avoids property and copyright issues (extremely important in the CH domain) related to the storage of restricted-access data to remote servers.

4. Inside the 3DHOP framework

3DHOP is based on the WebGL component of HTML5, and on the SpiderGL [1] library. This makes the framework extremely lightweight in terms of dependencies, and able to run on most modern browsers and platforms. 3DHOP does not need plugins or additional components installed in the client, nor specialized servers. The tool works on all major browsers: Firefox, Chrome, Internet Explorer, Safari, Opera on Windows, MacOS and Linux. Mobile support is still not complete, mainly due to the mobile browsers' support of WebGL not yet being as stable as in the PC market; on some Android platforms, the tool is working perfectly, but on other platforms and browsers the debugging is still ongoing. Touch- and multitouch-based input is supported.

4.1. Large models management

One of the key features of 3DHOP is the ability to manage very high resolution 3D meshes and point-clouds, by using a multiresolution approach. Displaying high resolution models on a web browser is not just a matter of optimizing the rendering speed, but it also involves considering the loading time and network traffic caused by transferring a considerable amount of data over the network. While WebGL gives direct access to the GPU resources, how data is transferred from a remote server to the local GPU is up to the programmer. Loading a high-resolution model in its entirety through the web requires transferring a single chunk of data on the order of tens of megabytes: this is definitely impractical, especially if the user has to wait for this file transmission to end before seeing any visual result.

The *multiresolution* approach ensures efficiency of both data transfer and rendering. Multiresolution schemes generally split the geometry into smaller chunks. For each chunk, multiple levels of detail are available. Transmission is *on demand*, requiring only to load and render the portions of the model strictly needed for the generation of the current view. While this approach is key to being able to render very large models at an interactive frame rate, it is also highly helpful with respect to the data transfer over a possibly slow network, since the data transferred will be divided into small chunks and only transferred when needed. The advantages of using this types of methods are the fast startup time and the reduced network load. The model is immediately available for the user to browse it, even though at a low resolution, and it is constantly improving its appearance as new data are progressively loaded. On the other hand, since refinement is driven by view-dependent criteria (observer position, orientation and distance from the 3D model), only the data really needed for the required navigation are transferred to the remote user.

We implemented one of those multiresolution schemes, called *Nexus* [34] (http://vcg.isti.cnr.it/nexus/), on top of the SpiderGL library [1] (http://vcg.isti.cnr.it/spidergl/), obtaining very good performance. Nexus is a multiresolution visualization library supporting interactive rendering of very large surface models. It belongs to the family of cluster based, view-dependent visualization algorithms. It employs a patch-based approach: the 3D model is split (according to a specific spatial strategy based on KD-trees) into patches; these initial patches represent the highest level of detail of the multiresolution representation. The number of triangles in each patch is halved, and adjacent patches are joined, in order to keep the number of triangles more or less uniform per patch. The different levels of detail are generated by iterating this process (bottom-up). The result is a tree structure containing each portion of the input object at multiple resolutions and, more importantly, the patches are organized and built to always match on common borders. This allows them to be assembled on-the-fly to build view-dependent representations at variable resolution.

At rendering time and based on the current view, the system decides which patches are better suited to represent the object given a target rendering speed and the maximum geometric error. Moreover, the batched structure allows for aggressive GPU optimization of the triangle patches, since the latter are encoded with triangle strips thus boosting GPU rendering performance.

At initial loading time, the "map" of the patch tree is downloaded, together with the lower-resolution patches. Then, depending on the view position, orientation and distance, the rendering algorithm decides which patches have to be fetched from the server to improve the current visualization, and queues a request. When each selected patch has been downloaded, the rendering is updated. The system continues this process of rendering-deciding-fetching-updating, trying to balance the amount of memory/data needed, the quality and speed of rendering and the network load.

All the data is contained in a single file. 3DHOP exploits the HTTP protocol capability to randomly access binary files to get specific data chunks inside each file, thus transferring only the needed portion of data. In this way, the viewer is able in a very short time to display a low-resolution version of the object, which is then progressively refined according to the user interaction, since the updates are view-dependent.

To give a practical demonstration of the capabilities of the multiresolution component, we provide some practical examples. The *Luni Statues* setup (Figure 3) provides visual inspection over eight 3D models, each one representing the original part and one or multiple integrations of each statue belonging to a Roman Temple in Luni (Italy), for a total of 14 million triangles. Another example is the *Helm* viewer (Figure 6) which shows a 3D model representing the actual state of an Etruscan helm and a second 3D model depicting the virtually restored version, each composed by 5 million triangles. Finally, the *Capsella Samagher* example (Figure 7) uses a 10 million triangles model and the *Pompei* viewer (Figure 8) is displaying a 20 million triangles mesh.

The conversion from a single-resolution 3D model to our multi resolution format is a one-time operation, done in a preprocessing phase. The 3DHOP user will convert its 3D assets using an executable (also open source, and included in the 3DHOP distribution). The obtained file is ready to be deployed on the Web server. It is important to note that our streamable multiresolution encoding does not require server-side computation and resident data-streaming daemons. It is the client that automatically fetches data from the inside of the file, jumping from one location to another in the data structure.

Finally, multiresolution allows also some degree of data protection. Most institutions do not want their 3D data to be downloaded without permission. When using a multiresolution encoding, the high-resolution 3D model is never transmitted to the remote user in a single file but in a set of pieces encoded

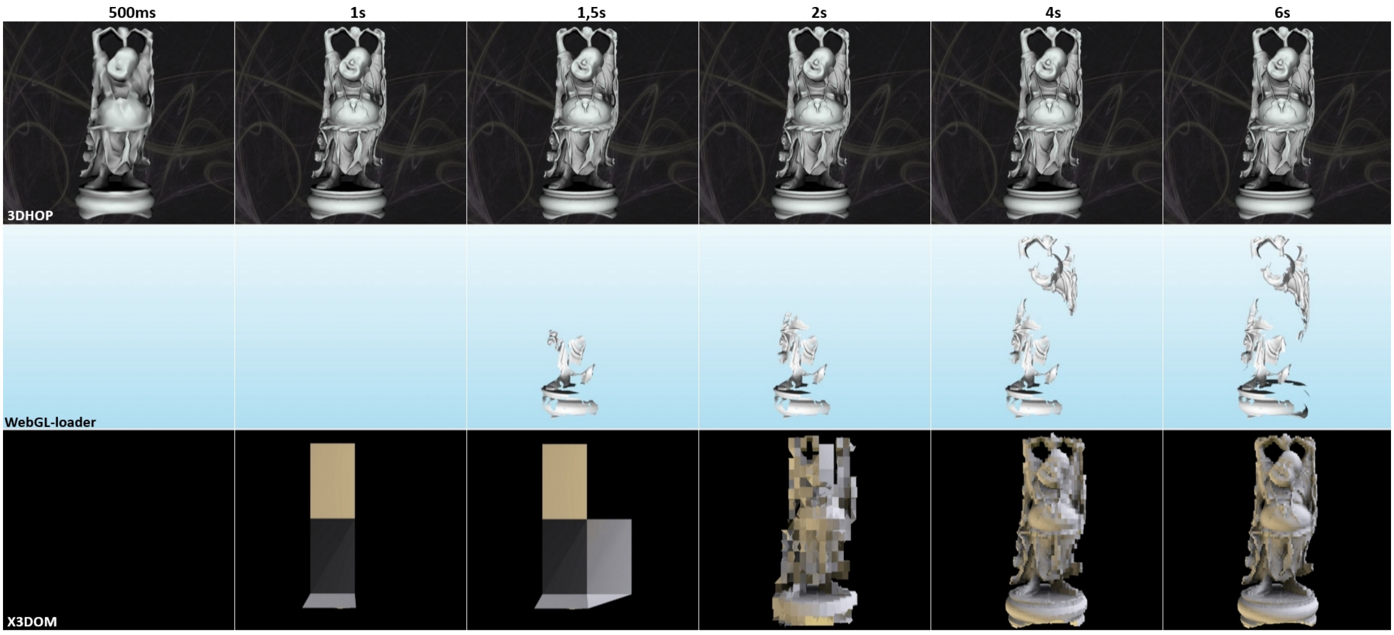

Figure 4: Comparative screenshots illustrating the web rendering of a 1M triangle mesh on a 5 Mbit/s Internet access, using the 3DHOP framework (first row), WebGL-loader (central row) and X3DOM binary POP Buffer Geometry (last row). All these test have been run on the same web server to ensure equal conditions. From the left screenshots are taken respectively at 500ms, 1s, 1.5s, 2s, 4s and 6s after loading the web page.

with a proprietary data structure. In this way, the malicious copy of the 3D data becomes quite complex and requires the design of ad-hoc procedures for downloading the whole geometric data and recombing them in the original model.

Smaller 3D models can also be managed using a single-resolution representation; currently, 3DHOP supports single-resolution models in PLY format [42] (but more importers will be added as future work). In this case, the model file is fetched from the server as a whole and parsed by 3DHOP. This solution is ideal for small geometries (less than 1MB), generally used to give a context to higher-resolution entities or small modelled 3D meshes. The management of geometries, may they be multi-resolution or single-resolution, is completely transparent to the user.

4.1.1. Web-based 3D rendering: comparison of existing solutions

We tested our rendering framework comparing it with the current state of art, in order to have tangible feedback about the effectiveness of our technical solution.

We chose to stream online the multiresolution version of a relatively simple mesh, the Happy Buddha model (1M triangles, vertex color, 22MBytes as binary .PLY file, previously used in similar comparison works [21]), with some of the approaches previously mentioned (see Section 2 and 3). In these test we used a limited bandwidth internet access and, of course, the same hardware and software equipment (desktop PC equipped with Intel Dual Core i3-3220 CPU at 3.30 GHz, 8 GB RAM, NVidia GeForce GT 620 1 GB RAM, OS Windows 8.1 and Google Chrome Browser ver. 43.0.2357.124m). Since our framework uses a view dependent algorithm, for the sake of accuracy, it must be said that all the test have been run at Full HD screen resolution (1920x1080 pixels, aspect ratio 16:9), however, when handling around 1M triangles per model (as in the Happy Buddha case) our rendering system is indifferent to this parameter.

We compared the 3DHOP framework results against the Google WebGL-loader [43], the X3DOM binary POP Buffer Geometry [22] approach, the Sketchfab [41] platform, and the Unity [40] graphics engine, in order to have a wide selection of competitors, ranging from complete system solutions (X3DOM, Sketchfab and Unity) to stand-alone streaming services (WebGL-loader), from progressive mesh techniques (POP Buffer Geometry) to hybrid systems (WebGL-loader) and to standard data streaming procedures (Sketchfab and Unity), from completely free projects (WebGL-loader and X3DOM) to mixed solutions (Sketchfab and Unity).

The results of this comparison can be easily understood by observing the screenshots in Figure 4, representing the time-lapse visualization of the aforementioned approaches, respectively caught after 500ms, 1s, 1,5s, 2s, 4s and 6s from launching the loading of the Web pages. Under these conditions, with limited bandwidth (5 Mbit/s, typical 3G+ connection speed) and meshes with millions of triangles, it can be easily seen that 3DHOP (first row in Figure 4) is performing better with respect to the WebGL-loader algorithm (central row in Figure 4) and to the X3DOM POP Buffers system (last row in Figure 4). Readily after the webpage loading (500 ms), a rough version of the geometry is already visible, and can be used for user interaction.

It should be noted that the Sketchfab and Unity results do not appear in Figure 4; this because both Sketchfab and Unity viewers do not use a progressive loading engine, and the model has to be fully downloaded before it is visible. In both cases,

	3DHOP	WebGL-loader	X3DOM
3,0 Mbit/s	**0,3 / 9,5**	2,0 / 19,4	0,6 / 44,5
5,0 Mbit/s	**0,2 / 4,8**	1,1 / 10,8	0,6 / 24,8
8,0 Mbit/s	**0,2 / 3,9**	0,7 / 6,8	0,6 / 15,2
20,0 Mbit/s	**0,2** / 3,7	0,3 / **2,7**	0,5 / 6,0
50,0 Mbit/s	**0,2** / 3,6	**0,2** / **1,1**	0,5 / 2,4

Table 1: Web rendering statistics for the Happy Buddha mesh (1M triangles) at different bandwidths (3, 5, 8, 20 and 50 Mbit/s), using 3DHOP framework, WebGL-loader and X3DOM binary POP Buffer Geometry. Each table cell shows two average time (values in seconds): the first one concerning the start of the rendering (time that the user will wait before seeing anything), the second one related to the end of the rendering (whole 3D model drawn time). All these test have been run on the same Web server to ensure equal conditions (bold values represent the best performance in each individual case).

the Happy Buddha model loaded after nearly 6 seconds from the web page launch. It is clear that this gap with respect to progressive multiresolution approaches is emphasized when the mesh size grows or the bandwidth decreases; on the other hand, it is also true that progressive multiresolution systems may continue updating and streaming data also after the other systems will have transferred the whole model.

This eventuality can also be found by observing the data in Table 1. In this case the same Happy Buddha test seen previously was performed at different bandwidths (ranging from 3 to 50 Mbit/s), this time taking into account the latency of the rendering (i.e. the time that the user will wait before seeing anything after running the web page) and the end of the data streaming process (i.e. the time taken to render the whole model). Under the aforementioned conditions the table clearly shows indeed that on fast networks (20 or 50 Mbit/s) progressive multiresolution approaches can employ a small amount of extra time to load the entire 3D model compared to the other approaches (an event that for our multiresolution algorithm does not occur with lower bandwidths, when 3DHOP performs better than any other). However it should be stressed once again that our framework is able to provide to the final user a draft (but illustrative and ready to use) version of the whole 3D model practically with no waiting times (300ms in the worst case, with 3 Mbit/s Internet access), consistently out-performing other competitors in any situation (regarding this feature).

It is worth remembering that, to ensure equal conditions, all the tests in this section have been run on the same Web server, and, with respect to the data in Table 1, they have been obtained by averaging five different measurements per cell data. Finally, it is right to clarify that, in order to obtain results less dependent on external network interferences, during these tests the server and client ran on the same network infrastructure, but that the acquired results are comparable with those obtained with the client and server placed on two different network subsystems.

Currently, no quantitative test was performed on mobile devices (since the mobile compatibility of 3DHOP is still not complete), but first results show that the performance of our framework will be good also on these systems (although the POP Buffer approach is extremely efficient on mobile devices due to the lack of decompression times).

Furthermore, the solutions introduced with the last software release (mesh compression, multi-thread JavaScript structure, frame-rate bounded streaming), suggest a further improvement of the performance. A more detailed description and evaluation of the current version of the view-dependent multiresolution engine can be found in [44].

4.2. Declarative-like scene setup

3DHOP has been designed to work with a few high-resolution geometries, and not with really complex scenes made of hundreds of entities. Anyway, it is necessary to define a *scene* to initialize the viewer. The definition of the scene has been implemented in a declarative fashion. All the scene elements are declared as JavaScript JSON structures, with properties and values, and assembled into a comprehensive scene structure. This structure is then parsed by 3DHOP at initialisation time to create the scene. We chose to use JSON because it is fairly easy to write and parse, it is human readable and easy to understand; XML would have been a good choice too, possibly a bit more verbose. With respect to a completely DOM-integrated approach, like XML3D, we are still somehow disconnected; the declarative approach is used to define the scene, which is an entity directly managed by the 3DHOP component, and all the interaction with the DOM passes through the 3DHOP viewer object, following the idea to create a self-contained component. We know this somehow offers a lower level of integration and less freedom, but also ensures a more immediate approach (just add the basic component to the webpage and it is ready-to-go) and a higher reusability (thanks to being self-contained).

The 3DHOP scene is composed of different elements: the *mesh* and the *instance* are the most basic. A *mesh* is simply a 3D model (single or multi-resolution). An *instance* is an occurrence of the mesh in the scene. This separation seems an unnecessary complication, given that the tool aims to be simple, but it is nevertheless the simplest way to have multiple objects sharing the same geometry.

Meshes and *instances* may have an attached transformation, specified either as a matrix (a 16-number vector) or by using the predefined SpiderGL functions. The most obvious use is to exploit the mesh transformation to bring the 3D model into a basic position/orientation (e.g. to put a 3D model originally not perfectly aligned to its axis into a “straight” position) and then to locate each *instance*, to set its specific position/orientation/scale.

An example of declaration of *meshes* and *instances* is the following:

```
meshes: {
  "Laurana": {
    url: "singleres/laurana.ply" },
  "Gargoyle": {
    url: "multires/gargo.nxs" },
  "Box": {
    url: "singleres/cube.ply",
    transform: {
      matrix:
        SglMat4.scaling([13.0, 0.5, 10.0])
    }
  }
},
```

```
modelInstances: {
  "Lady": {
    mesh: "Laurana",
    transform: {
      matrix: [1.0,    0.0,   0.0, 0.0,
               0.0,    1.0,   0.0, 0.0,
               0.0,    0.0,   1.0, 0.0,
               0.0, 235.0,-50.0, 1.0]
    }
  },
  "GargoRight": {
    mesh: "Gargoyle",
    transform: {
      matrix:
        SglMat4.mul(
          SglMat4.translation(
            [120.0, 0.0, 150.0]),
          SglMat4.rotationAngleAxis(
            sglDegToRad(-90.0),
            [0.0, 1.0, 0.0]))
    }
  },
  "GargoLeft": {
    mesh: "Gargoyle",
    transform: {
      matrix:
        SglMat4.translation(
          [-120.0, 0.0, 120.0])
    }
  },
  "Base": {
    mesh: "Box",
    transform: {
      matrix:
        SglMat4.translation(
          [0.0, -12.5, 0.0])
    }
  }
},
```

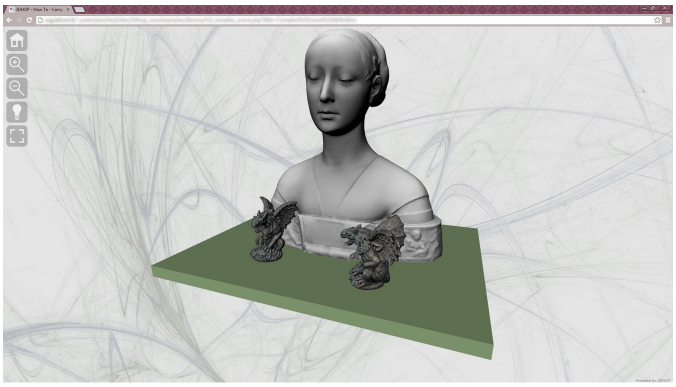

Figure 5: A simple scene in 3DHOP created by instancing geometries and applying transformations. This example is available in the *How-To* section of the 3DHOP website.

In this example a few simple elements are instantiated and arrayed in space, with the corresponding scene visible in Figure 5. A *mesh* element having the shape of a cube is scaled to become the base of the example in Figure 5, and positioned at the *instance* level. The other models are arranged (translated or rotated and translated) onto the base at *instance* level; the two gargoyles share the same *mesh* geometry.

A 3DHOP scene includes many other elements, which are presented in the following sections, e.g. the *trackball* (used to drive the interaction) or the *hotspot* elements used for picking. General scene parameters (e.g. the field of view and the custom scene centering) are also declared in the same way.

The declarative approach also has the advantage of more easily managing content retrieved from a database. The scene description is a JavaScript structure which can be easily filled with data retrieved by a query to a database; this would be less straightforward using an imperative-like setup.

4.3. Interaction components

A 3D viewer is not just a rendering engine, but also includes the components required to implement the user interaction. 3DHOP mostly uses the *object-in-hand* metaphor, where the camera is fixed and the object is manipulated by the user in front of it, generally using a trackball.

It is difficult, if not impossible, to create a single all-purpose trackball, able to cope with the specific geometric characteristics of every possible object. For this reason, we decided to implement a series of basic trackballs, letting the user to choose the more appropriate one. At the moment, the 3DHOP distribution includes three different trackballs (others will be added in the future):

- *Full-Sphere*: it is the trackball providing the more free interaction, enabling the user to rotate the object around all axes at the same time.
- *TurnTable*: this is the most flexible one, providing rotation around the vertical axis and tilting around the horizontal axis. With this trackball it is possible to reach almost all view positions around an object in a simple way, maintaining its verticality (e.g. preventing to rotate a statue head-down, feet-up).
- *Pan-Tilt*: this trackball is tailored to present bas-reliefs or objects whose detail is mostly located on a single plane.

Having a series of basic trackballs, implemented with simple, open and documented code, will allow developers to add new interaction modes coping with specific visualization needs. For this reason, each trackball in the distribution is a separate file, making it easier to use them as a codebase.

Trackballs can be configured with limits on their axes, to restrict the position reachable by the user. This is useful to avoid the user going, for example, below ground level in buildings, or behind objects with only a frontal part (like bas-reliefs). Trackballs can be also animated (we present an example in the next section).

In each 3DHOP viewer/installation there is only one trackball selected (*TurnTable* trackball is the default). To explicitly choose and configure a trackball, the developer has to specify the *trackball* element of the scene:

```
  trackball: {
    type: TurnTableTrackball,
    trackOptions: {
      startPhi      : 0.0,
      startTheta    : 0.0,
      startDistance : 2.5,
```

```
        minMaxPhi      : [-90, 120],
        minMaxTheta    : [-10.0, 75.0],
        minMaxDist     : [0.5, 3.0]
      }
    }
```

In the example above, the developer has chosen a *TurnTable*, starting exactly in front of the object (*phi* is rotation around vertical axis, *theta* the elevation angle) but a bit far from the object (distance 2.5 means that the camera distance is 2.5 times the size of the object bounding box). The trackball is limited both in the horizontal rotation (a bit to left, more to the right) and in the vertical one (not much below, a lot above); it is also impossible to go nearer than 0.5 and farther than 3.0 units from the object (again, expressed in multiples of the object size). Like in all configurations of 3DHOP components, it is not needed to specify *all* the parameters, since the unspecified ones will retain their default; it is sufficient to specify only the ones that need to be changed.

This approach, based on the trackball metaphor, is perfect to manipulate "objects", but it makes it much more difficult to navigate more complex scenes (such as buildings and terrains). We are currently working on interaction components more suited for exploring other types of geometries such as terrain models (with a Google earth-like approach), or the interior of a building (using a waypoint-based path).

4.4. *Interconnection with the DOM*

As introduced before, we wanted to create a framework offering basic viewers (if no other functions are configured), but also visualization components able to interact with the rest of the webpage. To this aim, we added a series of exposed functions and events, usable by a developer to allow 3DHOP components to interact with the rest of the web page logic. Our idea was to implement multiple, self-contained functions, with no high-level semantics attached, in order to provide the developer with a toolbox.

4.4.1. *Trackball automation*

The most basic interaction between a web page and the 3D visualization component is the control of the trackball. 3DHOP trackballs are able to give feedback on their current position: an exposed JavaScript function (*getTrackballPosition*) returns a structure containing the current state of the trackball. Another provided JavaScript function (*setTrackballPosition*) can be used to instantly move the trackball to a specific position by feeding it with a new state description. Additionally, it is possible to *animate* the trackballs to reach a certain position: instead of instantly changing its state, the camera follows a smooth animation path linking the current position with the specified one. These functions allow the developer to build, for example, a bookmarking mechanism for pre-selected views, a "share this view" button or an guided animated tour around the object. An example is shown in the *Helm* viewer (Figure 6), where the buttons on the right side of the window move the trackball to the views represented visually by the small icons.

4.4.2. *Visibility control*

Most visual presentation tools implement the control of the visibility of the different models. Model instances in 3DHOP can be configured in order to be visible or invisible at startup (visible is the default), and their visibility status can be changed at runtime using specific JavaScript functions exposed by the tool. An interesting trick is the tag-based selection of groups: in order to select the visibility status over groups of objects, the visibility functions do not work on a single instance, but on all instances thati have a specific tag. Model instances have a *tags* property, which is basically a series of strings. We can assign to each instance the tag of each "group" it belongs to or, if necessary, a unique tag. Using this simple mechanism, it is possible to address single entities as well as groups.

3DHOP exposes a function to set visibility and another one to toggle the visibility of a set of instances. For example, the *Luni Statues* viewer (Figure 3) presents four statues, each one composed of an original part and an integration; it is possible to make visible/invisible each statues either as a whole, or all the original parts or all the integrations of the entire set or, finally, the original/integration parts of a specific statue. In this example there are four statues, and for each statue there is one model for the original part and one for the integration. The original part of statue #1 has tags [*"figure1", "original"*]; the integration part of statue #1 has tags [*"figure1","integration"*], and so on for the other figures. Therefore, in order to make visible only the whole statue #1, the developer will use these calls:

```
setInstanceVisibility(HOP_ALL, false, false);
setInstanceVisibility("figure1", true, true);
```

Conversely, to show only original parts for statue #1 and #3:

```
setInstanceVisibility(HOP_ALL, false, false);
setInstanceVisibility("figure1", true, false);
setInstanceVisibility("figure3", true, false);
toggleInstanceVisibility("integration", true);
```

where HOP_ALL is a constant used to select all of the instances; the first parameter of *setInstanceVisibility* is the new visibility state; and the last parameter of both functions is used to force a redraw.

Visibility control is also used in the *Helm* viewer (Figure 6) to switch between the helm before and after restoration; there are two *instances* of different *meshes* in the same positions, and to switch between the two, one is hidden while the other is shown.

4.4.3. *Hotspots and picking*

Another widely available feature in web pages is the presence of clickable *hotspots*. This feature is often connected to something happening in the 3D visualization or elsewhere in the webpage. Depending on the visualization scheme, it may be interesting to have a picking component able to detect a pick on a hotspot, but also to detect a pick on an instance of a 3D model. 3DHOP does support both levels of interaction. In order to use this feature, the developer shall use two JavaScript functions to handle the picking (of hotspots and instances) and register these two functions to the handles exposed by 3DHOP.

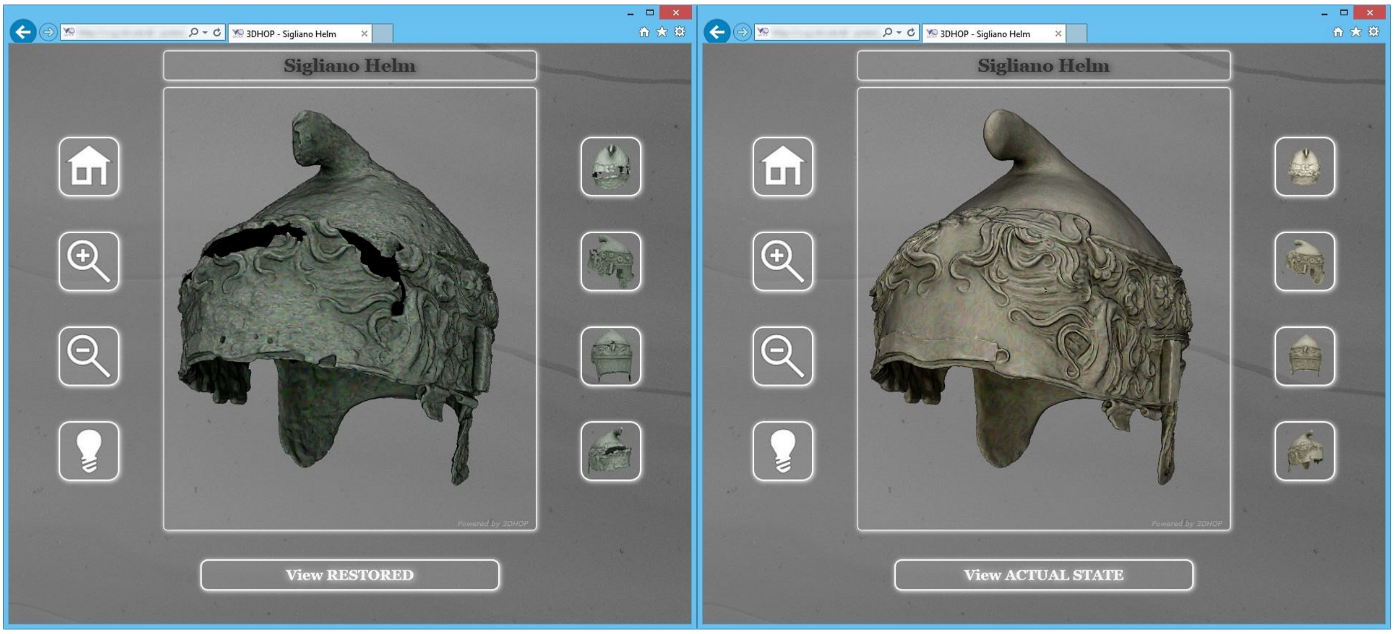

Figure 6: The *Helm* viewer allows to inspect an Etruscan helm either in its current state (image on the left) or in its virtual restoration version (image on the right), each represented by a 5 million triangle model. The user may switch between the two versions (using the ViewRestored/ViewActualState button), explore the model (it adopts the TurnTable trackball), and use the links on the right side of the window to go to interesting views of the model (these buttons will animate the trackball to reach the selected view position). This example is available in the Gallery section of the 3DHOP website.

The first function (hooked to *onPickedInstance*) is invoked every time a model instance has been clicked, and returns the name of the picked instance. The second one (hooked to *onPickedSpot*) is invoked every time a hotspot is clicked, again returning its name. A third function, which returns the exact XYZ coordinate of the clicked point under development and will be included in the next 3DHOP release.

In order to be more flexible, instead of just a single point, a hotspot may have an arbitrary shape and geometry. This is obtained by associating a *mesh* to the hotspot, similarly to the way a 3D model is specified when declaring an instance (a geometry is declared as a *mesh*, and then used in the declaration of the hotspot). In the simpler cases, a hotspot can be defined using a sphere or a cube model, moved to the correct position and appropriately scaled. In more complex situations, the user can provide a specific geometry, for example created using a 3D modeling tool. Picking is implemented using a basic CG method: when picking, the scene is rendered in an off-screen buffer, with each pickable object rendered as a solid unique color, which encodes its ID, while non-pickable objects are rendered solid black. The picked pixel is retrieved from this buffer: if black, nothing has been picked; if non-black, the color is transformed back into the ID of the picked object. This method does not require too many resources, and works pretty well also on complex scenes. The picking mechanism also works in realtime when the user moves the mouse, thus obtaining an "onOver" hook, and enables the hotspot geometry to light up. This feature may be deactivated when the scene is too complex, to speed up the rendering.

Hotspots may be made active or inactive using a tag-based mechanism similar to the one used in the visibility control, making it possible to define "hotspot groups" which can be independently activated/deactivated (e.g. to show different layers of information or linking). Each hotspot may have a specific color and an associated cursor.

An example of this kind of interaction is provided in the *Capsella Samagher* viewer (Figure 7): in this example, when a hotspot is picked some related presentation material (an image and a descriptive text) is shown in the left-most portion of the web page, and the view over the 3D model is moved to better frame the detail (using the trackball animation feature).

Figure 7: The *Capsella Samagher* viewer: in this example, the antique reliquary is presented with hotspots (light-blue regions). The hotspots, when picked, centers the view over the hotspot area and show the corresponding descriptive content (images and text) in the left-most part of the webpage. The Capsella model contains 10 million triangles. This example is available in the Gallery section of the 3DHOP website.

5. Using 3DHOP

The tradeoff between ease of use and flexibility is a major issue when creating a tool for non-expert developers. If the features are too simple or restricted, users with particular needs may not find proper support; on the other hand, an increase in flexibility could reduce simplicity of use. For this reason, the 3DHOP tool has been designed with different levels of entry, to be as straightforward as possible for the more simple cases but, at the same time, able to provide enough configurable features to support the huge variability of Cultural Heritage artworks and applications. Users with knowledge of JavaScript programming and web design will have no problem in using the framework, since its basic paradigm mimics the one normally employed in standard Web development.

5.1. 3DHOP for unskilled developers

Developers with limited programming skills may still use the framework using one of the following strategies:

- **Zero configuration**: since all the components have a set of safe defaults, it is possible to create a visualization page without configuring anything. This "minimal" visualization page is contained in a folder of the distribution, and can be readily used by the most inexperienced of users, since it is only necessary to change the 3D model file.
- ***How-Tos***: in addition to plain documentation, we opted to present the different features with *How-To* descriptions, detailing the parameter-based configuration of the visualization component. These pages contain reusable examples that can be modified following the content of the *How-To*. New *How-To* resources will be added as soon as new features and components are introduced in 3DHOP.
- **Templates**: in the *Gallery* page of the 3DHOP website, it is possible to find various examples (with different levels of complexity) which cover typical cases of usage in the CH field. The idea is to provide the developers with non-trivial usable templates, which can be used or customised with just minimal changes. After changing just the 3D model file (and the graphic elements, if needed), a completely unskilled developer may create their own visualization page without even modifying the HTML code. We are now working on better documentation for the templates, and on cleaning-up their HTML code for simpler use.

5.2. 3DHOP as a codebase

3DHOP has been designed to be configurable and flexible, and we are working on developing new components. Nevertheless, there are many projects where a specific solution is needed to fully exploit the data and to reach the communication goals. In these cases, 3DHOP may be seen as a "codebase". The modular structure of the tool facilitates the implementation of new, specialized components, or the tuning of existing ones. We believe that a skilled CG programmer and/or web developer may be able to heavily modify 3DHOP to cope with the particular needs of a project.

An example of this strategy is a modification of 3DHOP that we have designed for the web-based exploration of an entire *insula* (an area surrounded by four major streets) in the Pompei archaeological site. The basic version of 3DHOP was used as a starting point to create a customized viewer for the *Pompeii* model, presented in Figure 8.

The added value of this specific modification is the work done to extend the basic trackball to an interaction interface suited to the exploration of terrain-with-structures models. This system offers a double interaction method: a *bird-view* navigation and a *first-person-view* navigation. Both navigation methods are able to follow the height of the ground level, and collision detection with walls is available in first-person navigation. This new 3DHOP incarnation features also a new component, the *minimap* (an HTML5 canvas entity, see the small interactive map on the right-most portion of Figure 8). In each instant of the navigation, the current position of the viewer is shown on the map; clicking on any location in the minimap, the viewer is virtually moved to the desired location. Moreover, the system keeps track of the position of the viewer, not just showing the user location on the minimap, but also showing the name of the specific building/room the user is currently visiting (see the two textual fields on top-right, circled in red in Figure 8), retrieved from an existing web repository.

3DHOP is an open source tool, and the extension and modification of the framework is highly encouraged. We believe the 3DHOP framework has the potential to sprout an independent community of users, that could share examples, exchange experiences, and create connections. Following the first release of 3DHOP (April 2014), we have been contacted by several users willing to test and evaluate the framework. The first implementations by third parties are appearing (see Figure 9), and we are gathering suggestions and feedback.

6. Ongoing work, perspectives and conclusions

3DHOP is an ongoing effort, which already reached a level of consolidation that allowed us to disclose it and share with the community. We are regularly releasing new versions of the tool; one major update was made on October 2014, and the next one is scheduled for June 2015, as there are several features and extensions already on our roadmap. Since we conceive 3DHOP as a framework, there are many new components (or variations of the existing ones) that can be added to support the creation of more flexible and effective interactive visualizations. The main improvements would include:

- **New navigation and visualization features**: new trackball types and new scene manipulation functions are on the development list. Examples are the trackball used in the *Pompeii* explorer (Figure 8) that will be documented and added to the *Gallery*. Moreover, all geometries are

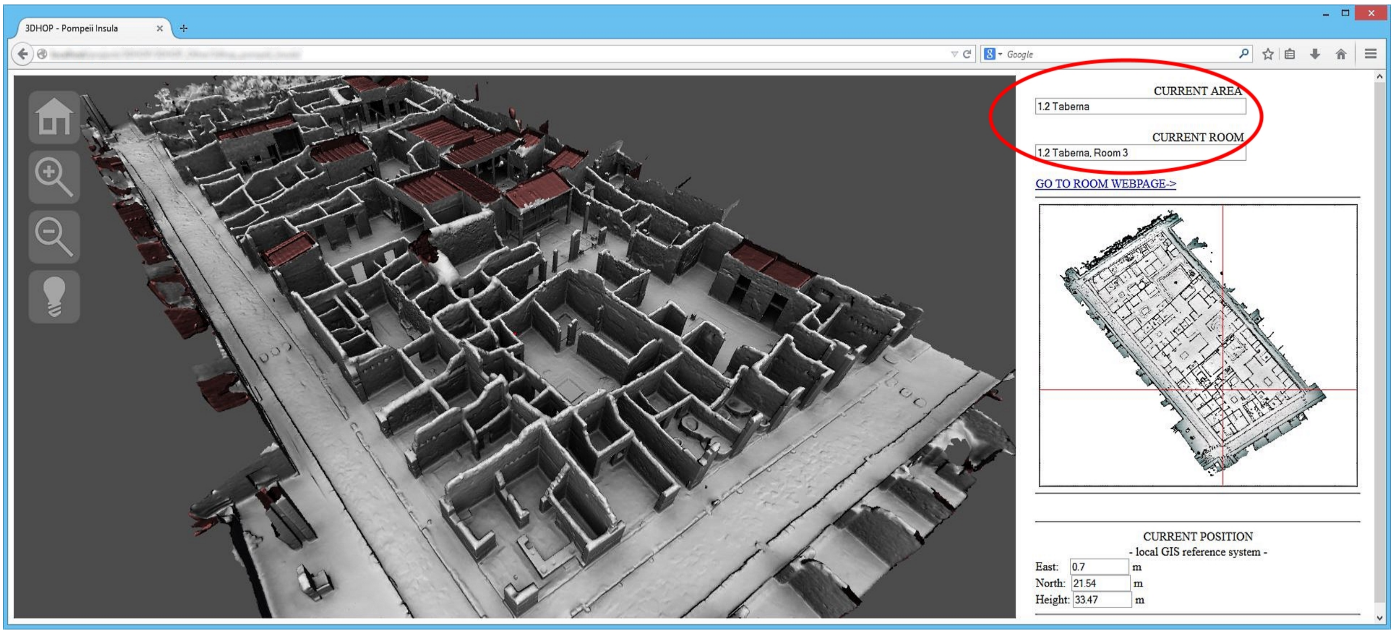

Figure 8: The *Pompeii* explorer: it allows to explore the entire Insula V 1 of Pompeii (using a 20 million triangle 3D model). Navigation is controlled by mouse inputs (using a custom terrain-enabled trackball) or by clicking on the minimap (see on the right of the window). The viewer keeps track of the current location of the user, showing the name of the room and of the house (text fields circled in red in the image). A test version is available at: http://3dhop.net/demos/insula/

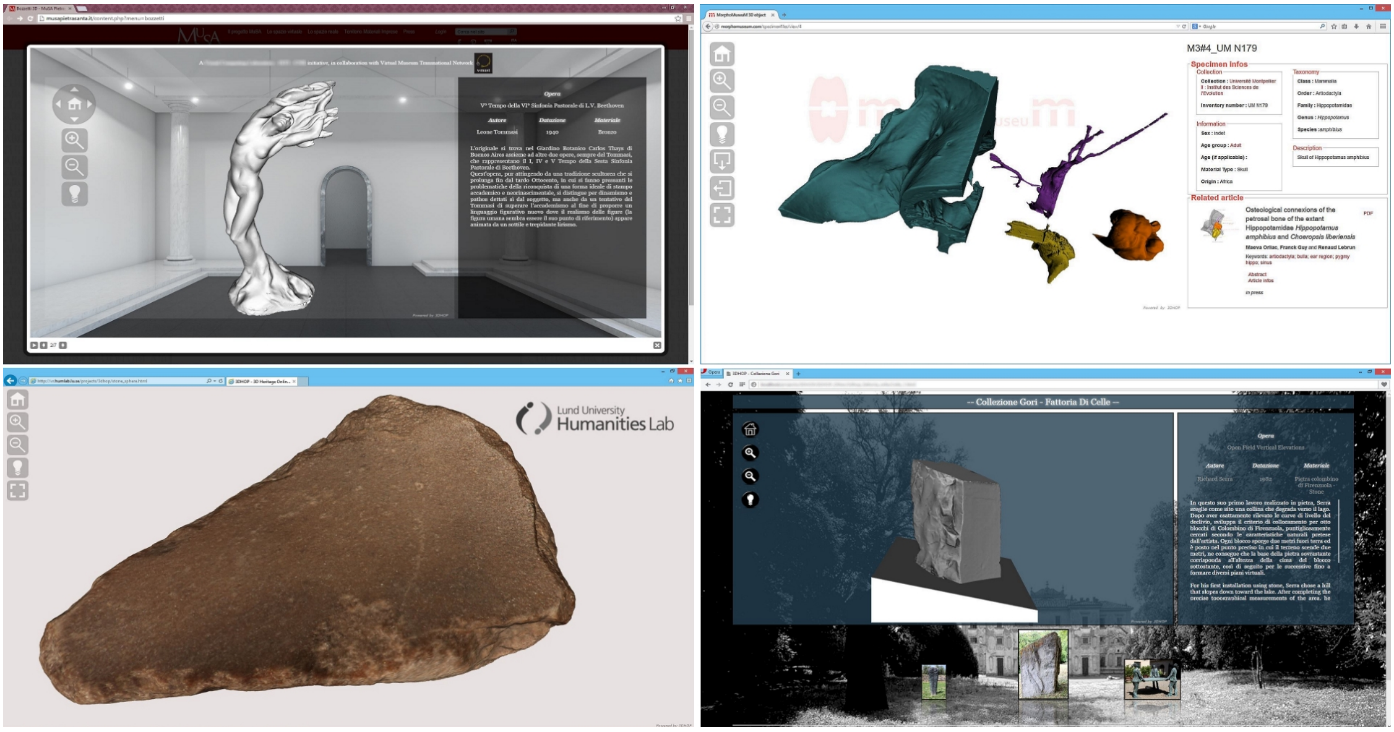

Figure 9: Four examples of independent projects developed by the community using 3DHOP (in clockwise order starting from the upper left): the *MuSA* viewer: presenting a collection of 3D artwork models, each one paired with a descriptive text (on the right of the page); the *Morpho Museum* project: publishing and sharing 3D models of vertebrates (the panel on the right contains specimen infos and links to the related article); the *Fattoria Celle* example: the Gori artworks collection opened to the public of the web (the 3D models are accessible by the slide show component in the bottom of the page); the *Humanitities Lab* experience: a simple viewer for high-resolution archaeological founding (by Lund University, Sweden).

currently rendered using the same basic shader. Our goal in the near future is to provide different, configurable shaders, which should be selectively attached to each instance.

- **Moving to dynamic definition of scenes**: at the moment, the scene definition is completely *static*. Once declared in the initialization, there is no way to modify the parameters of the different entities. We know that, in order to be fully compliant with the declarative paradigm, this feature will have to be added. Our development roadmap aims at reaching this functionality in a progressive way, starting from being able to modify the associated transformations, then to move to the other properties, and ending with the ability to dynamically add/remove entities.

- **Other types of media**: in the context of web visualization, other types of media could be effectively integrated into 3DHOP. One example is represented by *terrain* datasets. Terrains are defined in a 2D 1/2 space and can be managed more effectively than a 3D model using specialized strategies. A web-streamable multiresolution representation (based on quadtree) of a terrain will be soon integrated into 3DHOP, making it possible to add terrain geometry to a scene. This will be very useful to better cope with applications that involve landscapes of archeological interest. Moreover, we have available technology for the web-based streaming and visualization of relightable images, i.e. Reflection Transformation Images (RTI) [45, 46], currently under integration in the framework.

- **Authoring helpers and automatic services**: At the moment, there is not a visual editor or a wizard to set up a visualization scheme. This lack of guided tools may prevent some potential users from adopting 3DHOP despite its simplicity. For this reason, in the framework of the EC INFRA "ARIADNE" project we are implementing an automatic web service able to create presentation web pages, using a layout similar to the one shown in Figure 2. The web server accepts the upload of a 3D model plus some basic metadata provided with a simple web form and, after the unattended processing is completed, returns to the user the URL of the prepared visualization webpage (hosted on the same web server), plus a download link (to let the developer use the webpage and data on their own server, in case they want to).

To conclude, we have presented 3DHOP, a framework that aims at providing an easy way to create advanced 3D web content, offering the possibility to create and share advanced examples. Its modular structure has been designed to allow different utilisation levels of the framework but also to enable the creation of a community of users, so that examples and new components may be shared and re-used. We believe that this could be a helpful instrument to help the CH community to create and share advanced contents on the web, and use it not only for dissemination purposes, but also in the workflow of experts and practitioners.

Acknowledgements. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 313193 (EC INFRA "ARIADNE" project) and EC ERIC "DARIAH" project.

References

[1] Di Benedetto M, Ponchio F, Ganovelli F, Scopigno R. Spidergl: a javascript 3d graphics library for next-generation www. In: Proceedings of the 15th International Conference on Web 3D Technology. Web3D '10; New York, NY, USA: ACM. ISBN 978-1-4503-0209-8; 2010, p. 165–74. URL: `http://doi.acm.org/10.1145/1836049.1836075`. doi:`10.1145/1836049.1836075`.

[2] Microsoft. Microsoft ActiveX Controls. `http://msdn.microsoft.com/en-us/library/aa751968(VS.85).aspx`; 2013.

[3] Raggett D. Extending WWW to support platform independent virtual reality. Technical Report 1995;.

[4] Don Brutzmann LD. X3D: Extensible 3D Graphics for Web Authors. Morgan Kaufmann; 2007.

[5] Khronos Group . WebGL - OpenGL ES 2.0 for the Web. 2009.

[6] Khronos Group . Khronos: Open Standards for Media Authoring and Acceleration. 2009.

[7] Khronos Group . OpenGL ES - The Standard for Embedded Accelerated 3D Graphics. 2009.

[8] Evans A, Romeo M, Bahrehmand A, Agenjo J, Blat J. 3d graphics on the web: A survey. Computers & Graphics 2014;41(0):43 – 61. URL: `http://www.sciencedirect.com/science/article/pii/S0097849314000260`. doi:`http://dx.doi.org/10.1016/j.cag.2014.02.002`.

[9] Jankowski J, Ressler S, Sons K, Jung Y, Behr J, Slusallek P. 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; New York, NY, USA: ACM. ISBN 978-1-4503-2133-4; 2013, p. 39–45. URL: `http://doi.acm.org/10.1145/2466533.2466547`. doi:`10.1145/2466533.2466547`.

[10] Behr J, Eschler P, Jung Y, Zöllner M. X3dom: a dom-based html5/x3d integration model. In: Proceedings of the 14th International Conference on 3D Web Technology. Web3D '09; New York, NY, USA: ACM. ISBN 978-1-60558-432-4; 2009, p. 127–35. URL: `http://doi.acm.org/10.1145/1559764.1559784`. doi:`10.1145/1559764.1559784`.

[11] Sons K, Klein F, Rubinstein D, Byelozyorov S, Slusallek P. Xml3d: Interactive 3d graphics for the web. In: Proceedings of the 15th International Conference on Web 3D Technology. Web3D '10; New York, NY, USA: ACM. ISBN 978-1-4503-0209-8; 2010, p. 175–84. URL: `http://doi.acm.org/10.1145/1836049.1836076`. doi:`10.1145/1836049.1836076`.

[12] Kay L. SceneJS. `http://www.scenejs.com`; 2009.

[13] Brunt P. GLGE: WebGL for the lazy (web site). `http://www.glge.org/`; 2010.

[14] Dirksen J, editor. Learning Three.js: The JavaScript 3D Library for WebGL. Packt Publishing; 2013.

[15] DeLillo B. WebGLU: A utility library for working with WebGL . `http://webglu.sourceforge.org/`; 2009.

[16] Jankowski J, Decker S. A dual-mode user interface for accessing 3d content on the world wide web. In: Proceedings of the 21st international conference on World Wide Web. WWW '12; New York, NY, USA: ACM. ISBN 978-1-4503-1229-5; 2012, p. 1047–56. URL: `http://doi.acm.org/10.1145/2187836.2187977`. doi:`10.1145/2187836.2187977`.

[17] Callieri M, Leoni C, Dellepiane M, Scopigno R. Artworks narrating a story: a modular framework for the integrated presentation of three-dimensional and textual contents. In: Web3D, 18th International Conference on 3D Web Technology. 2013, p. 167–75.

[18] Russell BC, Martin-Brualla R, Butler DJ, Seitz SM, Zettlemoyer L. 3D Wikipedia: Using online text to automatically label and navigate recon-

structed geometry. ACM Transactions on Graphics (SIGGRAPH Asia 2013) 2013;32(6).

[19] Smithsonian I. Smithsonian X 3D. http://3d.si.edu/; 2011.

[20] Autodesk . Project Memento. http://memento.autodesk.com; 2011.

[21] Lavoué G, Chevalier L, Dupont F. Streaming compressed 3d data on the web using javascript and webgl. In: Proceedings of the 18th International Conference on 3D Web Technology. Web3D '13; New York, NY, USA: ACM. ISBN 978-1-4503-2133-4; 2013, p. 19–27.

[22] Limper M, Jung Y, Behr J, Alexa M. The pop buffer: Rapid progressive clustering by geometry quantization. Computer Graphics Forum 2013;32(7):197–206.

[23] Limper M, Wagner S, Stein C, Jung Y, Stork A. Fast delivery of 3d web content: A case study. In: Proceedings of the 18th International Conference on 3D Web Technology. Web3D '13; New York, NY, USA: ACM. ISBN 978-1-4503-2133-4; 2013, p. 11–7.

[24] Limper M, Thöner M, Behr J, Fellner DW. SRC - a streamable format for generalized web-based 3d data transmission. In: Polys NF, Chesnais A, Gobbetti E, Döllner J, editors. The 19th International Conference on Web3D Technology, Web3D '14, Vancouver, BC, Canada, August 8-10, 2014. ACM. ISBN 978-1-4503-3015-2; 2014, p. 35–43. URL: http://doi.acm.org/10.1145/2628588.2628589. doi:10.1145/2628588.2628589.

[25] Sutter J, Sons K, Slusallek P. Blast: A binary large structured transmission format for the web. In: Proceedings of the Nineteenth International ACM Conference on 3D Web Technologies. Web3D '14; New York, NY, USA: ACM. ISBN 978-1-4503-3015-2; 2014, p. 45–52. URL: http://doi.acm.org/10.1145/2628588.2628599. doi:10.1145/2628588.2628599.

[26] Hoppe H. Progressive meshes. In: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '96; New York, NY, USA: ACM. ISBN 0-89791-746-4; 1996, p. 99–108.

[27] Gobbetti E, Marton F, Rodriguez MB, Ganovelli F, Di Benedetto M. Adaptive quad patches: An adaptive regular structure for web distribution and adaptive rendering of 3d models. In: Proceedings of the 17th International Conference on 3D Web Technology. Web3D '12; New York, NY, USA: ACM. ISBN 978-1-4503-1432-9; 2012, p. 9–16.

[28] Lee H, Lavou G, Dupont F. Rate-distortion optimization for progressive compression of 3d mesh with color attributes. The Visual Computer 2012;28(2):137–53.

[29] Alliez P, Desbrun M. Progressive compression for lossless transmission of triangle meshes. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '01; New York, NY, USA: ACM. ISBN 1-58113-374-X; 2001, p. 195–202.

[30] Evans A, Agenjo J, Blat J. Web-based visualisation of on-set point cloud data. In: Proceedings of the 11th European Conference on Visual Media Production. CVMP '14; New York, NY, USA: ACM. ISBN 978-1-4503-3185-2; 2014,URL: http://doi.acm.org/10.1145/2668904.2668937. doi:10.1145/2668904.2668937.

[31] Rossignac J, Borrel P. Multi-resolution 3d approximations for rendering complex scenes. In: Falcidieno B, Kunii T, editors. Modeling in Computer Graphics. IFIP Series on Computer Graphics; Springer Berlin Heidelberg. ISBN 978-3-642-78116-2; 1993, p. 455–65. URL: http://dx.doi.org/10.1007/978-3-642-78114-8_29. doi:10.1007/978-3-642-78114-8_29.

[32] Funkhouser TA, Séquin CH. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In: Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques. SIGGRAPH '93; New York, NY, USA: ACM. ISBN 0-89791-601-8; 1993, p. 247–54. URL: http://doi.acm.org/10.1145/166117.166149. doi:10.1145/166117.166149.

[33] Shaffer E, Garland M. A multiresolution representation for massive meshes. IEEE Transactions on Visualization and Computer Graphics 2005;11(2):139–48. URL: http://dx.doi.org/10.1109/TVCG.2005.18. doi:10.1109/TVCG.2005.18.

[34] Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchio F, Scopigno R. Batched multi triangulation. In: Proceedings IEEE Visualization. Conference held in Minneapolis, MI, USA: IEEE Computer Society Press; 2005, p. 207–14. URL: http://vcg.isti.cnr.it/Publications/2005/CGGMPS05.

[35] Borgeat L, Godin G, Blais F, Massicotte P, Lahanier C. Gold: Interactive display of huge colored and textured models. ACM Trans Graph 2005;24(3):869–77. URL: http://doi.acm.org/10.1145/1073204.1073276. doi:10.1145/1073204.1073276.

[36] Zhang Y, Xiong H, Jiang X, Shi J. A Survey of Simplification and Multiresolution Techniques for Massive Meshes: A Survey of Simplification and Multiresolution Techniques for Massive Meshes. Journal of Computer-aided Design & Computer Graphics 2010;22:559–68. doi:10.3724/SP.J.1089.2010.10617.

[37] Wimmer M, Scheiblauer C. Instant points: Fast rendering of unprocessed point clouds. In: Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics. SPBG'06; Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. ISBN 3-905673-32-0; 2006, p. 129–37. URL: http://dx.doi.org/10.2312/SPBG/SPBG06/129-136. doi:10.2312/SPBG/SPBG06/129-136.

[38] Ganovelli F, Scopigno R. Ocme: out-of-core mesh editing made practical. IEEE Computer Graphics and Applications 2012;32(3):46–58. URL: http://vcg.isti.cnr.it/Publications/2012/GS12.

[39] Koller D, Turitzin M, Levoy M, Tarini M, Croccia G, Cignoni P, et al. Protected interactive 3d graphics via remote rendering. ACM Trans on Graphics 2004;23(3):695–703. URL: http://vcg.isti.cnr.it/Publications/2004/KTLTCCS04.

[40] UnityTechnologies . Unity: Create the games you love with unity. More info on: http://unity3d.com/; 2014.

[41] Sketchfab . Publish and find the best 3d content. More info on: https://sketchfab.com/; 2014.

[42] GeorgiaTech . The ply file format. More info on: http://www.cc.gatech.edu/projects/large_models/ply.html; 2014.

[43] Google . Google Code WebGL Loader. https://code.google.com/p/webgl-loader/; 2011.

[44] Ponchio F, Dellepiane M. Fast decompression for web-based view-dependent 3d rendering. In: Proceedings of the 20th International Conference on 3D Web Technology. Web3D '15; New York, NY, USA: ACM. ISBN 978-1-4503-3647-5; 2015, p. 199–207.

[45] Malzbender T, Gelb D, Wolters H. Polynomial texture maps. In: Proceedings of the 28th annual conference on Computer graphics and interactive techniques - ACM Siggraph'01. ACM; 2001, p. 519–28.

[46] Mudge M, Malzbender T, Chalmers A, Scopigno R, Davis J, Wang O, et al. Image-Based Empirical Information Acquisition, Scientific Reliability, and Long-Term Digital Preservation for the Natural Sciences and Cultural Heritage . In: Roussou M, Leigh J, editors. EG 2008 Tutorials Proceedings. Crete, Greece: Eurographics Association; 2008,.