

This is a pre print version of the following article:

3DHOP: 3D heritage online presenter / Potenziani, Marco; Callieri, Marco; Dellepiane, Matteo; Corsini, M; Ponchio, Federico; Scopigno, Roberto. - In: COMPUTERS & GRAPHICS. - ISSN 0097-8493. - 52:(2015), pp. 129-141. [10.1016/j.cag.2015.07.001]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

03/05/2026 05:07

(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

^aVisual 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>.

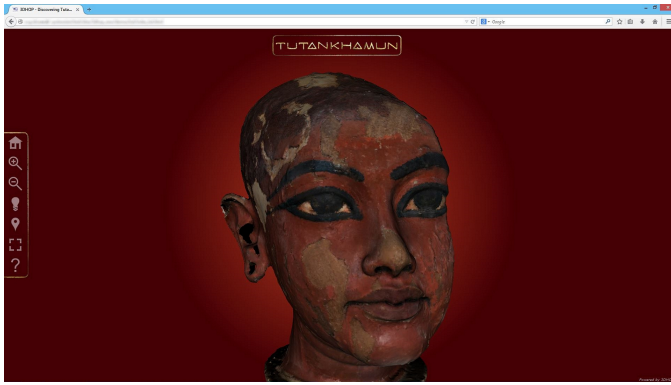


Figure 1: The *Tutankhamun* viewer: using 3DHOP to publish on the Web a high resolution 3D model explorable in a simple, intuitive and interactive way (the artifact is linked to additional multimedia information through hotspots). This example is available in the Gallery section of the 3DHOP website.

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.

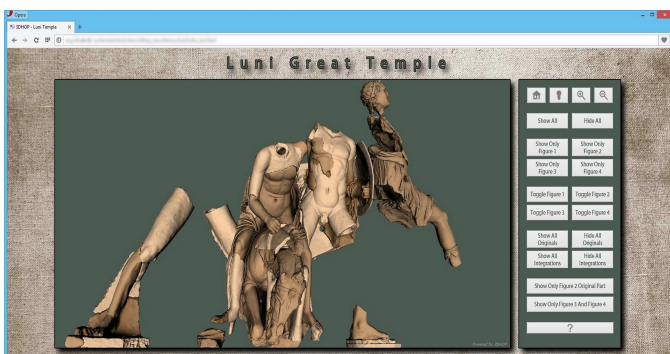


Figure 3: The *Luni Statues* viewer: in this example, four figures of the frieze of the Great Temple of Luni (Italy) are shown. Each statue has an original part and an integration (eight models for a total of 14 millions triangles); by using the visibility control, it is possible to control which subset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

440 plugins or additional components installed in the client, nor spe-
441 cialized servers. The tool works on all major browsers: Firefox,
442 Chrome, Internet Explorer, Safari, Opera on Windows, MacOS
443 and Linux. Mobile support is still not complete, mainly due to
444 the mobile browsers' support of WebGL not yet being as stable
445 as in the PC market; on some Android platforms, the tool
446 is working perfectly, but on other platforms and browsers the
447 debugging is still ongoing. Touch- and multitouch-based input
448 is supported.

449 4.1. Large models management

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

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

482 We implemented one of those multiresolution schemes, called
483 *Nexus* [34] (<http://vcg.isti.cnr.it/nexus/>), on top of the SpiderGL
484 library [1] (<http://vcg.isti.cnr.it/spidergl/>), obtaining very good
485 performance. Nexus is a multiresolution visualization library
486 supporting interactive rendering of very large surface models.
487 It belongs to the family of cluster based, view-dependent visu-
488 alization algorithms. It employs a patch-based approach: the
489 3D model is split (according to a specific spatial strategy based
490 on KD-trees) into patches; these initial patches represent the
491 highest level of detail of the multiresolution representation. The
492 number of triangles in each patch is halved, and adjacent patches
493 are joined, in order to keep the number of triangles more or less
494 uniform per patch. The different levels of detail are generated

495 by iterating this process (bottom-up). The result is a tree struc-
496 ture containing each portion of the input object at multiple res-
497 olutions and, more importantly, the patches are organized and
498 built to always match on common borders. This allows them
499 to be assembled on-the-fly to build view-dependent representa-
500 tions at variable resolution.

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

507 At initial loading time, the “map” of the patch tree is down-
508 loaded, together with the lower-resolution patches. Then, de-
509 pending on the view position, orientation and distance, the ren-
510 dering algorithm decides which patches have to be fetched from
511 the server to improve the current visualization, and queues a
512 request. When each selected patch has been downloaded, the
513 rendering is updated. The system continues this process of
514 rendering-deciding-fetching-updating, trying to balance the amount
515 of memory/data needed, the quality and speed of rendering and
516 the network load.

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

524 To give a practical demonstration of the capabilities of the
525 multiresolution component, we provide some practical exam-
526 ples. The *Luni Statues* setup (Figure 3) provides visual inspec-
527 tion over eight 3D models, each one representing the original
528 part and one or multiple integrations of each statue belonging
529 to a Roman Temple in Luni (Italy), for a total of 14 million tri-
530 angles. Another example is the *Helm* viewer (Figure 6) which
531 shows a 3D model representing the actual state of an Etruscan
532 helm and a second 3D model depicting the virtually restored
533 version, each composed by 5 million triangles. Finally, the
534 *Capsella Samagher* example (Figure 7) uses a 10 million tri-
535 angles model and the *Pompei* viewer (Figure 8) is displaying a
536 20 million triangles mesh.

537 The conversion from a single-resolution 3D model to our
538 multi resolution format is a one-time operation, done in a pre-
539 processing phase. The 3DHOP user will convert its 3D as-
540 sets using an executable (also open source, and included in the
541 3DHOP distribution). The obtained file is ready to be deployed
542 on the Web server. It is important to note that our streamable
543 multiresolution encoding does not require server-side computa-
544 tion and resident data-streaming daemons. It is the client that
545 automatically fetches data from the inside of the file, jumping
546 from one location to another in the data structure.

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

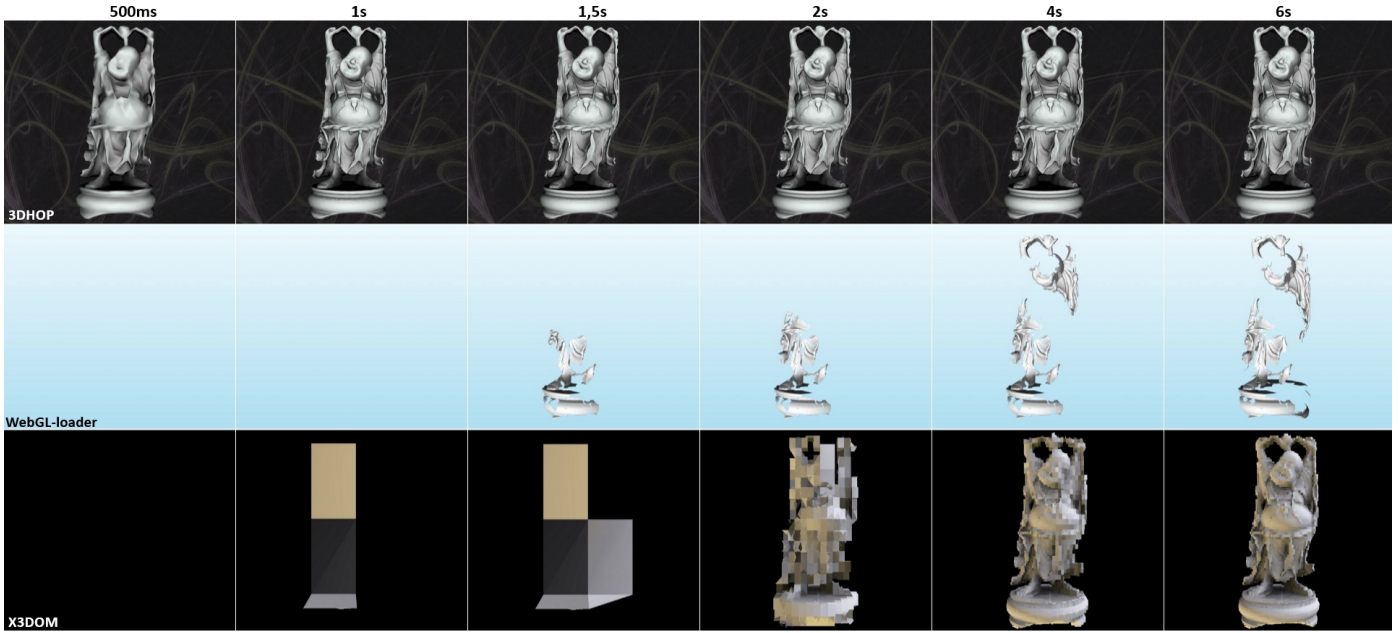


Figure 4: Comparative screenshots illustrating the web rendering of a 1M triangle mesh on a 5 Mbit/s Internet access, using the 3DHOP framework (first row), WebGL-loader (central row) and X3DOM binary POP Buffer Geometry (last row). All these test have been run on the same web server to ensure equal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

552 with a proprietary data structure. In this way, the malicious
 553 copy of the 3D data becomes quite complex and requires the
 554 design of ad-hoc procedures for downloading the whole geo-
 555 metric data and recombining them in the original model.

556 Smaller 3D models can also be managed using a single-
 557 resolution representation; currently, 3DHOP supports single-
 558 resolution models in PLY format [42] (but more importers will
 559 be added as future work). In this case, the model file is fetched
 560 from the server as a whole and parsed by 3DHOP. This solution
 561 is ideal for small geometries (less than 1MB), generally used
 562 to give a context to higher-resolution entities or small modelled
 563 3D meshes. The management of geometries, may they be multi-
 564 resolution or single-resolution, is completely transparent to the
 565 user.

566 4.1.1. Web-based 3D rendering: comparison of existing solu- 567 tions

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

571 We chose to stream online the multiresolution version of
 572 a relatively simple mesh, the Happy Buddha model (1M tri-
 573 angles, vertex color, 22MBytes as binary .PLY file, previously
 574 used in similar comparison works [21]), with some of the ap-
 575 proaches previously mentioned (see Section 2 and 3). In these
 576 test we used a limited bandwidth internet access and, of course,
 577 the same hardware and software equipment (desktop PC equip-
 578 ped with Intel Dual Core i3-3220 CPU at 3.30 GHz, 8 GB
 579 RAM, NVidia GeForce GT 620 1 GB RAM, OS Windows 8.1
 580 and Google Chrome Browser ver. 43.0.2357.124m). Since our
 581 framework uses a view dependent algorithm, for the sake of ac-
 582 curacy, it must be said that all the test have been run at Full HD

583 screen resolution (1920x1080 pixels, aspect ratio 16:9), how-
 584 ever, when handling around 1M triangles per model (as in the
 585 Happy Buddha case) our rendering system is indifferent to this
 586 parameter.

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

598 The results of this comparison can be easily understood by
 599 observing the screenshots in Figure 4, representing the time-
 600 lapse visualization of the aforementioned approaches, respec-
 601 tively caught after 500ms, 1s, 1.5s, 2s, 4s and 6s from launch-
 602 ing the loading of the Web pages. Under these conditions, with
 603 limited bandwidth (5 Mbit/s, typical 3G+ connection speed)
 604 and meshes with millions of triangles, it can be easily seen that
 605 3DHOP (first row in Figure 4) is performing better with respect
 606 to the WebGL-loader algorithm (central row in Figure 4) and to
 607 the X3DOM POP Buffers system (last row in Figure 4). Read-
 608 ily after the webpage loading (500 ms), a rough version of the
 609 geometry is already visible, and can be used for user interac-
 610 tion.

611 It should be noted that the Sketchfab and Unity results do
 612 not appear in Figure 4; this because both Sketchfab and Unity
 613 viewers do not use a progressive loading engine, and the model
 614 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:

```

701 meshes: {
702   "Laurana": {
703     url: "singleres/laurana.ply" },
704   "Gargoyle": {
705     url: "multires/gargo.nxs" },
706   "Box": {
707     url: "singleres/cube.ply",
708     transform: {
709       matrix:
710         SglMat4.scaling([13.0, 0.5, 10.0])
711     }
712   }
713 },

```

```

714 modelInstances: {
715   "Lady": {
716     mesh: "Laurana",
717     transform: {
718       matrix: [1.0, 0.0, 0.0, 0.0,
719               0.0, 1.0, 0.0, 0.0,
720               0.0, 0.0, 1.0, 0.0,
721               0.0, 235.0, -50.0, 1.0]
722     }
723   },
724   "GargoRight": {
725     mesh: "Gargoyle",
726     transform: {
727       matrix:
728         SglMat4.mul(
729           SglMat4.translation(
730             [120.0, 0.0, 150.0]),
731           SglMat4.rotationAngleAxis(
732             sglDegToRad(-90.0),
733             [0.0, 1.0, 0.0]))
734     }
735   },
736   "GargoLeft": {
737     mesh: "Gargoyle",
738     transform: {
739       matrix:
740         SglMat4.translation(
741           [-120.0, 0.0, 120.0])
742     }
743   },
744   "Base": {
745     mesh: "Box",
746     transform: {
747       matrix:
748         SglMat4.translation(
749           [0.0, -12.5, 0.0])
750     }
751   }
752 },

```

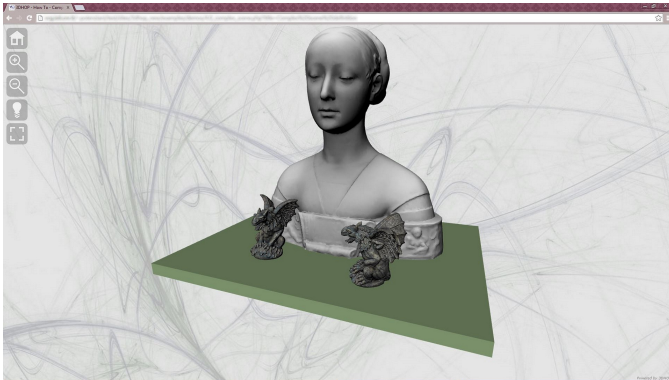


Figure 5: A simple scene in 3DHOP created by instantiating geometries and applying transformations. This example is available in the *How-To* section of the 3DHOP website.

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

760 A 3DHOP scene includes many other elements, which are

761 presented in the following sections, e.g. the *trackball* (used to
762 drive the interaction) or the *hotspot* elements used for picking.
763 General scene parameters (e.g. the field of view and the custom
764 scene centering) are also declared in the same way.

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

770 4.3. Interaction components

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

776 It is difficult, if not impossible, to create a single all-purpose
777 trackball, able to cope with the specific geometric characteris-
778 tics of every possible object. For this reason, we decided to im-
779 plement a series of basic trackballs, letting the user to choose
780 the more appropriate one. At the moment, the 3DHOP distribu-
781 tion includes three different trackballs (others will be added in
782 the future):

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

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

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

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

```

809 trackball: {
810   type: TurnTableTrackball,
811   trackOptions: {
812     startPhi      : 0.0,
813     startTheta   : 0.0,
814     startDistance : 2.5,

```

```

815     minMaxPhi      : [-90, 120],
816     minMaxTheta   : [-10.0, 75.0],
817     minMaxDist    : [0.5, 3.0]
818   }
819 }

```

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

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

840 4.4. Interconnection with the DOM

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

850 4.4.1. Trackball automation

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

868 4.4.2. Visibility control

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

882 3DHOP exposes a function to set visibility and another one
883 to toggle the visibility of a set of instances. For example, the
884 *Luni Statues* viewer (Figure 3) presents four statues, each one
885 composed of an original part and an integration; it is possible to
886 make visible/invisible each statues either as a whole, or all the
887 original parts or all the integrations of the entire set or, finally,
888 the original/integration parts of a specific statue. In this exam-
889 ple there are four statues, and for each statue there is one model
890 for the original part and one for the integration. The original
891 part of statue #1 has tags [*“figure1”, “original”*]; the integra-
892 tion part of statue #1 has tags [*“figure1”, “integration”*], and
893 so on for the other figures. Therefore, in order to make visible
894 only the whole statue #1, the developer will use these calls:

```

895 setInstanceVisibility(HOP_ALL, false, false);
896 setInstanceVisibility("figure1", true, true);

```

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

```

898 setInstanceVisibility(HOP_ALL, false, false);
899 setInstanceVisibility("figure1", true, false);
900 setInstanceVisibility("figure3", true, false);
901 toggleInstanceVisibility("integration", true);

```

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

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

911 4.4.3. Hotspots and picking

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

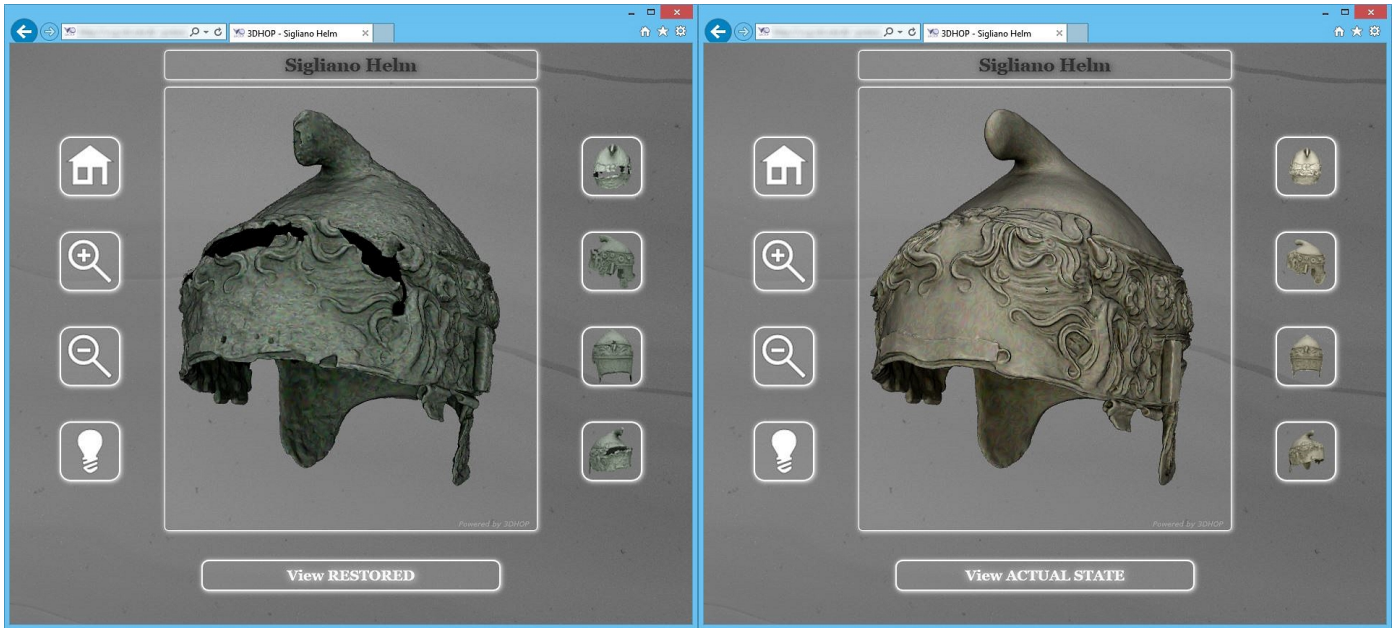


Figure 6: The *Helm* viewer allows to inspect an Etruscan helm either in its current state (image on the left) or in its virtual restoration version (image on the right), each represented by a 5 million triangle model. The user may switch between the two versions (using the ViewRestored/ViewActualState 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.

922 The first function (hooked to *onPickedInstance*) is invoked every
 923 time a model instance has been clicked, and returns the name of the
 924 picked instance. The second one (hooked to *onPickedSpot*) is
 925 invoked every time a hotspot is clicked, again returning its name.
 926 A third function, which returns the exact XYZ coordinate of the
 927 clicked point under development and will be included in the next
 928 3DHOP release.

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

951 Hotspots may be made active or inactive using a tag-based
 952 mechanism similar to the one used in the visibility control,

953 making it possible to define “hotspot groups” which can be
 954 independently activated/deactivated (e.g. to show different layers
 955 of information or linking). Each hotspot may have a specific
 956 color and an associated cursor.

957 An example of this kind of interaction is provided in the
 958 *Capsella Samagher* viewer (Figure 7): in this example, when a
 959 hotspot is picked some related presentation material (an image
 960 and a descriptive text) is shown in the left-most portion of the
 961 web page, and the view over the 3D model is moved to better
 962 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.

963 5. Using 3DHOP

964 The tradeoff between ease of use and flexibility is a major
965 issue when creating a tool for non-expert developers. If the fea-
966 tures are too simple or restricted, users with particular needs
967 may not find proper support; on the other hand, an increase in
968 flexibility could reduce simplicity of use. For this reason, the
969 3DHOP tool has been designed with different levels of entry,
970 to be as straightforward as possible for the more simple cases
971 but, at the same time, able to provide enough configurable fea-
972 tures to support the huge variability of Cultural Heritage art-
973 works and applications. Users with knowledge of JavaScript
974 programming and web design will have no problem in using the
975 framework, since its basic paradigm mimics the one normally
976 employed in standard Web development.

977 5.1. 3DHOP for unskilled developers

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

- 981 • **Zero configuration:** since all the components have a set
982 of safe defaults, it is possible to create a visualization
983 page without configuring anything. This "minimal" vi-
984 sualization page is contained in a folder of the distribu-
985 tion, and can be readily used by the most inexperienced of
986 users, since it is only necessary to change the 3D model
987 file.
- 988 • **How-Tos:** in addition to plain documentation, we opted
989 to present the different features with *How-To* descriptions,
990 detailing the parameter-based configuration of the visu-
991 alization component. These pages contain reusable ex-
992 amples that can be modified following the content of the
993 *How-To*. New *How-To* resources will be added as soon as
994 new features and components are introduced in 3DHOP.
- 995 • **Templates:** in the *Gallery* page of the 3DHOP website,
996 it is possible to find various examples (with different lev-
997 els of complexity) which cover typical cases of usage in
998 the CH field. The idea is to provide the developers with
999 non-trivial usable templates, which can be used or cus-
1000 tomised with just minimal changes. After changing just
1001 the 3D model file (and the graphic elements, if needed),
1002 a completely unskilled developer may create their own
1003 visualization page without even modifying the HTML
1004 code. We are now working on better documentation for
1005 the templates, and on cleaning-up their HTML code for
1006 simpler use.

1007 5.2. 3DHOP as a codebase

1008 3DHOP has been designed to be configurable and flexible,
1009 and we are working on developing new components. Neverthe-
1010 less, there are many projects where a specific solution is needed
1011 to fully exploit the data and to reach the communication goals.
1012 In these cases, 3DHOP may be seen as a "codebase". The mod-
1013 ular structure of the tool facilitates the implementation of new,

1014 specialized components, or the tuning of existing ones. We be-
1015 lieve that a skilled CG programmer and/or web developer may
1016 be able to heavily modify 3DHOP to cope with the particular
1017 needs of a project.

1018 An example of this strategy is a modification of 3DHOP that
1019 we have designed for the web-based exploration of an entire
1020 *insula* (an area surrounded by four major streets) in the Pompei
1021 archaeological site. The basic version of 3DHOP was used as
1022 a starting point to create a customized viewer for the *Pompeii*
1023 model, presented in Figure 8.

1024 The added value of this specific modification is the work
1025 done to extend the basic trackball to an interaction interface
1026 suited to the exploration of terrain-with-structures models. This
1027 system offers a double interaction method: a *bird-view* naviga-
1028 tion and a *first-person-view* navigation. Both navigation meth-
1029 ods are able to follow the height of the ground level, and colli-
1030 sion detection with walls is available in first-person navigation.
1031 This new 3DHOP incarnation features also a new component,
1032 the *minimap* (an HTML5 canvas entity, see the small interactive
1033 map on the right-most portion of Figure 8). In each instant of
1034 the navigation, the current position of the viewer is shown on
1035 the map; clicking on any location in the minimap, the viewer is
1036 virtually moved to the desired location. Moreover, the system
1037 keeps track of the position of the viewer, not just showing the
1038 user location on the minimap, but also showing the name of the
1039 specific building/room the user is currently visiting (see the two
1040 textual fields on top-right, circled in red in Figure 8), retrieved
1041 from an existing web repository.

1042 3DHOP is an open source tool, and the extension and mod-
1043 ification of the framework is highly encouraged. We believe the
1044 3DHOP framework has the potential to sprout an independent
1045 community of users, that could share examples, exchange ex-
1046 periences, and create connections. Following the first release of
1047 3DHOP (April 2014), we have been contacted by several users
1048 willing to test and evaluate the framework. The first implemen-
1049 tations by third parties are appearing (see Figure 9), and we are
1050 gathering suggestions and feedback.

1052 6. Ongoing work, perspectives and conclusions

1053 3DHOP is an ongoing effort, which already reached a level
1054 of consolidation that allowed us to disclose it and share with
1055 the community. We are regularly releasing new versions of the
1056 tool; one major update was made on October 2014, and the next
1057 one is scheduled for June 2015, as there are several features and
1058 extensions already on our roadmap. Since we conceive 3DHOP
1059 as a framework, there are many new components (or variations
1060 of the existing ones) that can be added to support the creation of
1061 more flexible and effective interactive visualizations. The main
1062 improvements would include:

- 1063 • **New navigation and visualization features:** new track-
1064 ball types and new scene manipulation functions are on
1065 the development list. Examples are the trackball used in
1066 the *Pompeii* explorer (Figure 8) that will be documented
1067 and added to the *Gallery*. Moreover, all geometries are

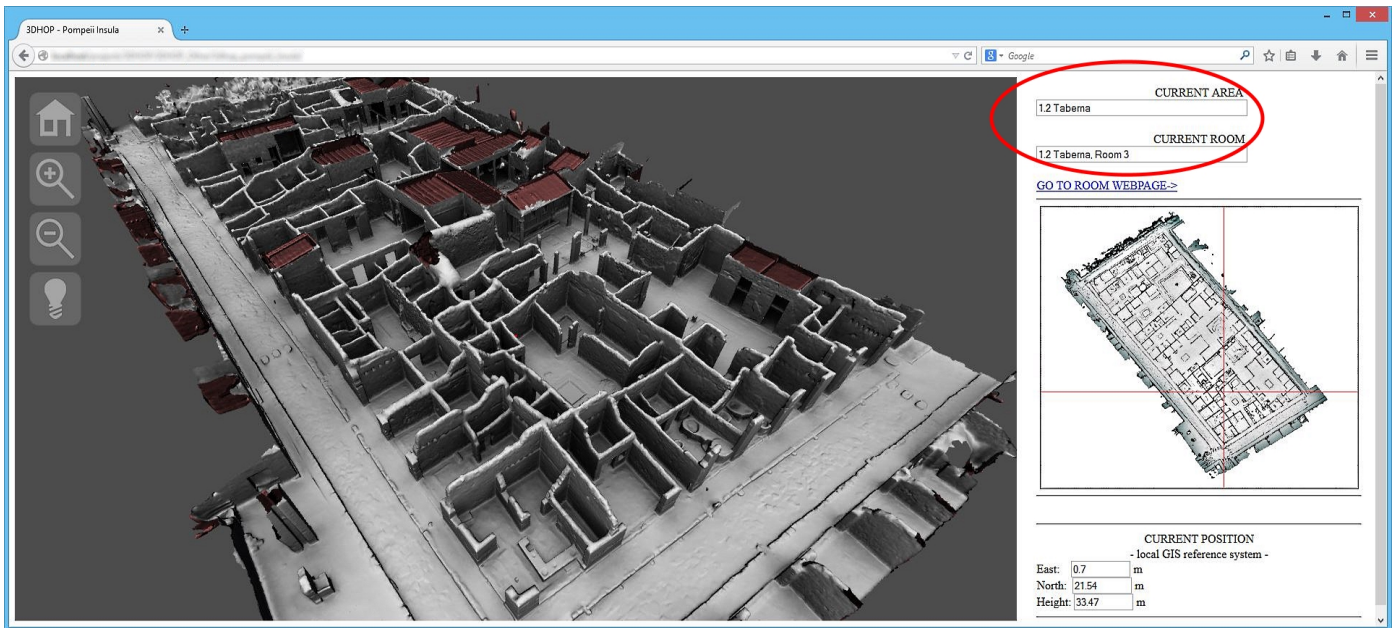


Figure 8: The *Pompeii* explorer: it allows to explore the entire Insula V 1 of Pompeii (using a 20 million triangle 3D model). Navigation is controlled by mouse inputs (using a custom terrain-enabled trackball) or by clicking on the minimap (see on the right of the window). The viewer keeps track of 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/>

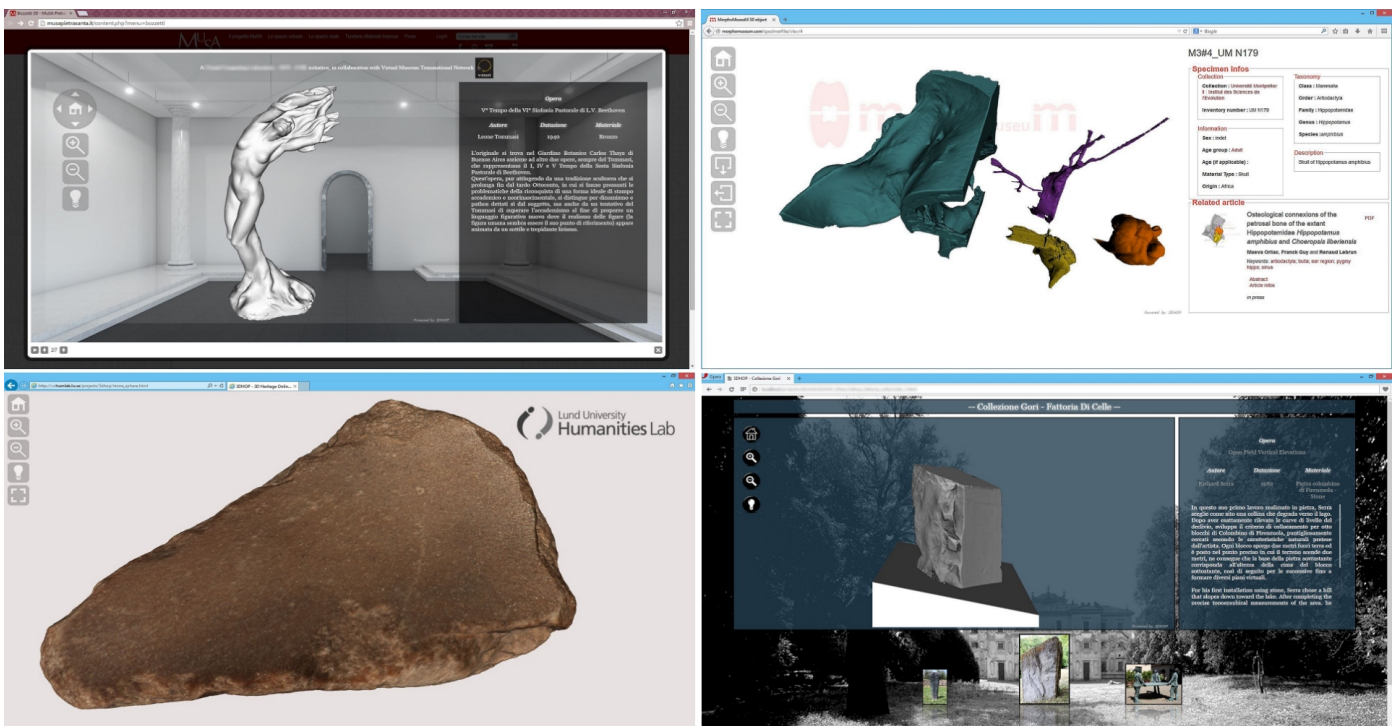


Figure 9: Four examples of independent projects developed by the community using 3DHOP (in clockwise order starting from the upper left): the *MuSA* viewer: presenting a collection of 3D artwork models, each one paired with a descriptive text (on the right of the page); the *Morpho Museum* project: publishing 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 *Humanities Lab* experience: a simple viewer for high-resolution archaeological founding (by Lund University, Sweden).

1068 currently rendered using the same basic shader. Our goal
1069 in the near future is to provide different, configurable
1070 shaders, which should be selectively attached to each in-
1071 stance.

1072 • **Moving to dynamic definition of scenes:** at the moment,
1073 the scene definition is completely *static*. Once declared
1074 in the initialization, there is no way to modify the pa-
1075 rameters of the different entities. We know that, in order
1076 to be fully compliant with the declarative paradigm, this
1077 feature will have to be added. Our development roadmap
1078 aims at reaching this functionality in a progressive way,
1079 starting from being able to modify the associated trans-
1080 formations, then to move to the other properties, and end-
1081 ing with the ability to dynamically add/remove entities.

1082 • **Other types of media:** in the context of web visual-
1083 ization, other types of media could be effectively inte-
1084 grated into 3DHOP. One example is represented by *ter-*
1085 *rain* datasets. Terrains are defined in a 2D 1/2 space and
1086 can be managed more effectively than a 3D model using
1087 specialized strategies. A web-streamable multiresolution
1088 representation (based on quadtree) of a terrain will be
1089 soon integrated into 3DHOP, making it possible to add
1090 terrain geometry to a scene. This will be very useful
1091 to better cope with applications that involve landscapes
1092 of archeological interest. Moreover, we have available
1093 technology for the web-based streaming and visualiza-
1094 tion of relightable images, i.e. Reflection Transformation
1095 Images (RTI) [45, 46], currently under integration in the
1096 framework.

1097 • **Authoring helpers and automatic services:** At the mo-
1098 ment, there is not a visual editor or a wizard to set up a
1099 visualization scheme. This lack of guided tools may pre-
1100 vent some potential users from adopting 3DHOP despite
1101 its simplicity. For this reason, in the framework of the
1102 EC INFRA "ARIADNE" project we are implementing
1103 an automatic web service able to create presentation web
1104 pages, using a layout similar to the one shown in Fig-
1105 ure 2. The web server accepts the upload of a 3D model
1106 plus some basic metadata provided with a simple web
1107 form and, after the unattended processing is completed,
1108 returns to the user the URL of the prepared visualization
1109 webpage (hosted on the same web server), plus a down-
1110 load link (to let the developer use the webpage and data
1111 on their own server, in case they want to).

1112 To conclude, we have presented 3DHOP, a framework that
1113 aims at providing an easy way to create advanced 3D web con-
1114 tent, offering the possibility to create and share advanced exam-
1115 ples. Its modular structure has been designed to allow different
1116 utilisation levels of the framework but also to enable the cre-
1117 ation of a community of users, so that examples and new com-
1118 ponents may be shared and re-used. We believe that this could
1119 be a helpful instrument to help the CH community to create and
1120 share advanced contents on the web, and use it not only for dis-
1121 semination purposes, but also in the workflow of experts and

1122 practitioners.

1123
1124 **Acknowledgements.** The research leading to these results has
1125 received funding from the European Union Seventh Framework
1126 Programme (FP7/2007-2013) under grant agreement n. 313193
1127 (EC INFRA "ARIADNE" project) and EC ERIC "DARIAH"
1128 project.

1129 References

- 1130 [1] Di Benedetto M, Ponchio F, Ganovelli F, Scopigno R. Spidergl: a java-
1131 script 3d graphics library for next-generation www. In: Proceedings
1132 of the 15th International Conference on Web 3D Technology. Web3D
1133 '10; New York, NY, USA: ACM. ISBN 978-1-4503-0209-8; 2010, p.
1134 165–74. URL: <http://doi.acm.org/10.1145/1836049.1836075>.
1135 doi:10.1145/1836049.1836075.
- 1136 [2] Microsoft. Microsoft ActiveX Controls. [http://msdn.microsoft.com/en-us/library/aa751968\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa751968(VS.85).aspx); 2013.
- 1137 [3] Raggett D. Extending WWW to support platform independent virtual
1138 reality. Technical Report 1995;.
- 1139 [4] Don Brutzmann LD. X3D: Extensible 3D Graphics for Web Authors.
1140 Morgan Kaufmann; 2007.
- 1141 [5] Khronos Group . WebGL - OpenGL ES 2.0 for the Web. 2009.
- 1142 [6] Khronos Group . Khronos: Open Standards for Media Authoring and
1143 Acceleration. 2009.
- 1144 [7] Khronos Group . OpenGL ES - The Standard for Embedded Accelerated
1145 3D Graphics. 2009.
- 1146 [8] Evans A, Romeo M, Bahrehmand A, Agenjo J, Blat J. 3d
1147 graphics on the web: A survey. Computers & Graphics
1148 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>.
- 1149 [9] Jankowski J, Ressler S, Sons K, Jung Y, Behr J, Slusallek P. Declar-
1150 ative integration of interactive 3d graphics into the world-wide web:
1151 Principles, current approaches, and research agenda. In: Proceedings
1152 of the 18th International Conference on 3D Web Technology. Web3D
1153 '13; New York, NY, USA: ACM. ISBN 978-1-4503-2133-4; 2013,
1154 p. 39–45. URL: <http://doi.acm.org/10.1145/2466533.2466547>.
1155 doi:10.1145/2466533.2466547.
- 1156 [10] Behr J, Eschler P, Jung Y, Zöllner M. X3dom: a dom-based html5/x3d
1157 integration model. In: Proceedings of the 14th International Conference
1158 on 3D Web Technology. Web3D '09; New York, NY, USA: ACM. ISBN
1159 978-1-60558-432-4; 2009, p. 127–35. URL: <http://doi.acm.org/10.1145/1559764.1559784>.
1160 doi:10.1145/1559764.1559784.
- 1161 [11] Sons K, Klein F, Rubinstein D, Byelozorov S, Slusallek P. Xml3d: In-
1162 teractive 3d graphics for the web. In: Proceedings of the 15th Inter-
1163 national Conference on Web 3D Technology. Web3D '10; New York, NY,
1164 USA: ACM. ISBN 978-1-4503-0209-8; 2010, p. 175–84. URL: <http://doi.acm.org/10.1145/1836049.1836076>.
1165 doi:10.1145/1836049.1836076.
- 1166 [12] Kay L. SceneJS. <http://www.scenejs.com>; 2009.
- 1167 [13] Brunt P. GLGE: WebGL for the lazy (web site). <http://www.glge.org/>; 2010.
- 1168 [14] Dirksen J, editor. Learning Three.js: The JavaScript 3D Library for
1169 WebGL. Packt Publishing; 2013.
- 1170 [15] DeLillo B. WebGLU: A utility library for working with WebGL. <http://webglu.sourceforge.org/>; 2009.
- 1171 [16] Jankowski J, Decker S. A dual-mode user interface for accessing 3d con-
1172 tent on the world wide web. In: Proceedings of the 21st international con-
1173 ference on World Wide Web. WWW '12; New York, NY, USA: ACM.
1174 ISBN 978-1-4503-1229-5; 2012, p. 1047–56. URL: <http://doi.acm.org/10.1145/2187836.2187977>.
1175 doi:10.1145/2187836.2187977.
- 1176 [17] Callieri M, Leoni C, Dellepiane M, Scopigno R. Artworks narrating
1177 a story: a modular framework for the integrated presentation of three-
1178 dimensional and textual contents. In: Web3D, 18th International Confer-
1179 ence on 3D Web Technology. 2013, p. 167–75.
- 1180 [18] Russell BC, Martin-Brualla R, Butler DJ, Seitz SM, Zettlemoyer L. 3D
1181 Wikipedia: Using online text to automatically label and navigate recon-
1182 struction

- 1188 structured geometry. *ACM Transactions on Graphics (SIGGRAPH Asia*
1189 *2013)* 2013;32(6). 1259
- 1190 [19] Smithsonian I. Smithsonian X 3D. <http://3d.si.edu/>; 2011. 1260
- 1191 [20] Autodesk . Project Memento. <http://memento.autodesk.com/>; 2011. 1261 [36] Zhang Y, Xiong H, Jiang X, Shi J. A Survey of Simplification and
1192 [21] Lavoué G, Chevalier L, Dupont F. Streaming compressed 3d data on the 1262
1193 web using javascript and webgl. In: Proceedings of the 18th International 1263
1194 Conference on 3D Web Technology. Web3D '13; New York, NY, USA: 1264
1195 ACM. ISBN 978-1-4503-2133-4; 2013, p. 19–27. 1265
- 1196 [22] Limper M, Jung Y, Behr J, Alexa M. The pop buffer: Rapid progres- 1266 [37] Wimmer M, Scheiblaue C. Instant points: Fast rendering of unpro-
1197 sive clustering by geometry quantization. *Computer Graphics Forum* 1267
1198 *2013*;32(7):197–206. 1268
1199 [23] Limper M, Wagner S, Stein C, Jung Y, Stork A. Fast delivery of 3d 1269
1200 web content: A case study. In: Proceedings of the 18th International 1270
1201 Conference on 3D Web Technology. Web3D '13; New York, NY, USA: 1271
1202 ACM. ISBN 978-1-4503-2133-4; 2013, p. 11–7. 1272 [38] Ganovelli F, Scopigno R. Ocmc: out-of-core mesh editing made practi-
1203 [24] Limper M, Thöner M, Behr J, Fellner DW. SRC - a streamable format 1273
1204 for generalized web-based 3d data transmission. In: Polys NF, Chesnais 1274
1205 A, Gobbetti E, Döllner J, editors. The 19th International Conference on 1275 [39] Koller D, Turitzin M, Levoy M, Tarini M, Crocchia G, Cignoni P, et al.
1206 Web3D Technology, Web3D '14, Vancouver, BC, Canada, August 8-10, 1276
1207 2014. ACM. ISBN 978-1-4503-3015-2; 2014, p. 35–43. URL: [http://](http://doi.acm.org/10.1145/2628588.2628589)
1208 doi.acm.org/10.1145/2628588.2628589. doi:10.1145/2628588. 1277
1209 2628589. 1278
- 1210 [25] Sutter J, Sons K, Klusallek P. Blast: A binary large structured transmis- 1279 [40] UnityTechnologies . Unity: Create the games you love with unity. More
1211 sion format for the web. In: Proceedings of the Nineteenth International 1280
1212 ACM Conference on 3D Web Technologies. Web3D '14; New York, NY, 1281 [41] Sketchfab . Publish and find the best 3d content. More info on:
1213 USA: ACM. ISBN 978-1-4503-3015-2; 2014, p. 45–52. URL: [http://](http://doi.acm.org/10.1145/2628588.2628599)
1214 doi.acm.org/10.1145/2628588.2628599. doi:10.1145/2628588. 1282
1215 2628599. 1283 [42] GeorgiaTech . The ply file format. More info on:
1216 [26] Hoppe H. Progressive meshes. In: Proceedings of the 23rd Annual Confe- 1284
1217 rence on Computer Graphics and Interactive Techniques. SIGGRAPH '96; New York, NY, USA: ACM. ISBN 0-89791-746-4; 1996, p. 99–108. 1285 [43] Google . Google Code WebGL Loader. <https://code.google.com/p/webgl-loader/>; 2011.
1218 [27] Gobbetti E, Marton F, Rodríguez MB, Ganovelli F, Di Benedetto M. 1286
1219 Adaptive quad patches: An adaptive regular structure for web distribution 1287 [44] Ponchio F, Dellepiane M. Fast decompression for web-based view-
1220 and adaptive rendering of 3d models. In: Proceedings of the 17th Interna- 1288
1221 tional Conference on 3D Web Technology. Web3D '12; New York, NY, 1289
1222 USA: ACM. ISBN 978-1-4503-1432-9; 2012, p. 9–16. 1290
1223 [28] Lee H, Lavoué G, Dupont F. Rate-distortion optimization for progres- 1291 [45] Malzbender T, Gelb D, Wolters H. Polynomial texture maps. In: Proceed-
1224 sive compression of 3d mesh with color attributes. *The Visual Computer* 1292
1225 *2012*;28(2):137–53. 1293
1226 [29] Alliez P, Desbrun M. Progressive compression for lossless transmission 1294 [46] Mudge M, Malzbender T, Chalmers A, Scopigno R, Davis J, Wang O,
1227 of triangle meshes. In: Proceedings of the 28th Annual Conference on 1295
1228 Computer Graphics and Interactive Techniques. SIGGRAPH '01; New 1296
1229 York, NY, USA: ACM. ISBN 1-58113-374-X; 2001, p. 195–202. 1297
1230 [30] Evans A, Agenjo J, Blat J. Web-based visualisation of on-set point cloud 1298
1231 data. In: Proceedings of the 11th European Conference on Visual Media 1299
1232 Production. CVMP '14; New York, NY, USA: ACM. ISBN 978-1- 1300
1233 4503-3185-2; 2014, URL: [http://doi.acm.org/10.1145/2668904.](http://doi.acm.org/10.1145/2668904.2668937)
1234 [2668904.2668937](http://doi.acm.org/10.1145/2668904.2668937). doi:10.1145/2668904.2668937. 1301
- 1235 [31] Rossignac J, Borrel P. Multi-resolution 3d approximations for render- 1302
1236 ing complex scenes. In: Falcidieno B, Kunii T, editors. Modeling in 1303
1237 Computer Graphics. IFIP Series on Computer Graphics; Springer Berlin Heidelberg. ISBN 978-3-642-78116-2; 1993, p. 455– 1304
1238 65. URL: http://dx.doi.org/10.1007/978-3-642-78114-8_29. 1305
1239 doi:10.1007/978-3-642-78114-8_29. 1306
- 1240 [32] Funkhouser TA, Séquin CH. Adaptive display algorithm for interactive 1307
1241 frame rates during visualization of complex virtual environments. In: 1308
1242 Proceedings of the 20th Annual Conference on Computer Graphics and 1309
1243 Interactive Techniques. SIGGRAPH '93; New York, NY, USA: ACM. 1310
1244 ISBN 0-89791-601-8; 1993, p. 247–54. URL: [http://doi.acm.org/](http://doi.acm.org/10.1145/166117.166149)
1245 [10.1145/166117.166149](http://doi.acm.org/10.1145/166117.166149). doi:10.1145/166117.166149. 1311
- 1246 [33] Shaffer E, Garland M. A multiresolution representation for massive 1312
1247 meshes. *IEEE Transactions on Visualization and Computer Graph- 1313
1248 ics* 2005;11(2):139–48. URL: [http://dx.doi.org/10.1109/TVCG.](http://dx.doi.org/10.1109/TVCG.2005.18)
1249 [2005.18](http://dx.doi.org/10.1109/TVCG.2005.18). doi:10.1109/TVCG.2005.18. 1314
- 1250 [34] Cignoni P, Ganovelli F, Gobbetti E, Marton F, Ponchio F, Scopigno R. 1315
1251 Batched multi triangulation. In: Proceedings IEEE Visualization. Con- 1316
1252 ference held in Minneapolis, MI, USA: IEEE Computer Society Press; 1317
1253 2005, p. 207–14. URL: [http://vcg.isti.cnr.it/Publications/](http://vcg.isti.cnr.it/Publications/2005/CGMPS05)
1254 [2005/CGMPS05](http://vcg.isti.cnr.it/Publications/2005/CGMPS05). 1318
- 1255 [35] Borgeat L, Godin G, Blais F, Massicotte P, Lahanier C. Gold: In- 1319
1256 teractive display of huge colored and textured models. *ACM Trans*
1257 *Graph* 2005;24(3):869–77. URL: [http://doi.acm.org/10.1145/](http://doi.acm.org/10.1145/1073204.1073276)
1258 [1073204.1073276](http://doi.acm.org/10.1145/1073204.1073276). doi:10.1145/1073204.1073276. 1320