The Common Implementation Framework as Service – Towards Novel Applications for Streamlined Presentation of 3D Content on the Web

Andreas Aderhold[1], Katarzyna Wilkosinska[1], Massimiliano Corsini[2], Yvonne Jung[3], Holger Graf[4], and Arjan Kuijper[4]

[1] University of Applied Sciences Darmstadt, Germany
aha@dockwerk.com, kasia@wilkosinska.com
[2] ISTI CNR, Pisa, Italy
Massimiliano.Corsini@isti.cnr.it
[3] University of Applied Sciences Fulda, Germany
Yvonne.Jung@informatik.hs-fulda.de
[4] Fraunhofer IGD, Darmstadt, Germany
{hgraf,akuijper}@igd.fraunhofer.de

Abstract. We solve a standing issue of the recently published Common Implementation Framework (CIF) for Online Virtual Museums: programmatic access to the transcoding, optimization and template rendering infrastructure of the CIF. We propose a method that enables researchers and developers to build novel systems on top of the CIF infrastructure beyond its current Cultural Heritage workflow. Therefore, we introduce a way to programmatically access the powerful backend of the CIF through a universal access layer, addressable by standards like HTTP and the JSON Data Interchange Format. In order to demonstrate our approach, we present two different use cases in which the CIF pipeline is utilized as a service through the proposed resource-based access layer: a native mobile iOS application for browsing 3D model repositories realizing just-in-time optimization of large models, and a MeshLab plugin to asynchronously convert and prepare a model for the Web.

Keywords: Web 3D, Virtual Museums, Cultural Heritage, Content Authoring, Distributed Systems.

1 Introduction

Today, the majority of web applications remain predominantly 2-dimensional. However, embedding 3D graphics is increasingly adopted by many services like geological visualization solutions (cp. e.g. [4]) or displayal of archeological artifacts [3]. In the Cultural Heritage (CH) domain 3D technology has long gained a strong foothold into the work of researchers and museums. Artifacts need to be presented on multi-display virtual museum installations as well as on a broad range of different devices to support researchers in this ambit in different ways

A. Marcus (Ed.): DUXU 2014, Part II, LNCS 8518, pp. 3–14, 2014.

[26]. Traditionally, these systems where developed with great expenses on specialized, often closed-source, platforms.

The introduction of WebGL [21] paved the way for real-time 3D graphics in the common web browser. It subsequently has become a mainstream technology deployed on a wide variety of desktop and mobile devices. However, within the CH domain, 3D models usually stem from scanner-acquired data at very high resolutions and need to be optimized aggressively to be useful on devices with limited resources like mobile phones, tablets or desktops with integrated graphic chips, esp. for connections with low bandwidth. Furthermore, to present and interactively explore 3D models in a web browser, the optimized models need to be embedded in a custom HTML5 application or web site, which can not be generalized. The spectrum of the presentation format can range from a simple website to a full virtual environment like an interactive exploration of an archeological excavation site or the presentation of a virtual reconstruction hypothesis.

Therefore, Wilkosinska et al. [29] recently proposed a system called the Common Implementation Framework for Online Virtual Museums (CIF), which presents a method to convert and optimize 3D heritage models into a form suitable to be displayed and interactively explored in a standards-based customizable web application. While the CIF is tailored to the CH domain, its transcoding and optimization backend can be utilized much more broadly and without the limitations imposed by a web front-end and a process specific to the Cultural Heritage workflow.

In this paper we outline how researchers and developers can build novel systems on top of the CIF infrastructure without being tied to a CH workflow or the web interface provided by the original CIF. We introduce a way to programmatically access the powerful optimization/transcoding and delivery backend of the CIF through a universal middleware access layer addressable by standard protocols and formats like Hypertext Transfer Protocol (HTTP) [10] and the JSON Data Interchange Format (JSON) [9]. To display the usefulness of our approach, we present two different cases of using the CIF pipeline as a service through the proposed access layer.

First, we present an iOS [2] application that allows accessing a remote repository of 3D models. In order to interactively explore 3D models on the mobile device, an optimized version of the model is generated by the CIF pipeline, and transmitted to the mobile outlet for presentation and storage. Due to the CIF and its API no large datasets need to be exchanged between the handheld device and the repository. Second, we extend the well-known MeshLab [7] desktop application by a "web export" plugin. This plugin employs the CIF API to send a 3D model to the CIF, which generates an HTML5 application presentation package according to some pre-defined templates. As a result, the users of MeshLab can easily deploy his/her work on a web site, without knowledge of WebGL or other web technologies.

2 Related Work

2.1 Foundation Technologies

WebGL is a gateway technology enabling developers to deploy 3D graphics through the use of a common web browser. Most major browser vendors adopted the WebGL standard early on. Recently, also the last remaining major browser without WebGL support, Microsoft Internet Explorer, gained this ability with version 11 [22]. However, WebGL is a low-level API to the GPU and not very approachable for typical web developers who are used to frontend technologies like HTML, CSS and DOM-scripting. Therefore, JavaScript libraries have been emerging, which aim to close the gap between front-end developer and computer graphics expert. For instance, one such library is X3DOM [5], a polyfill, which allows for declarative embedding of 3D graphics based on the open ISO standard X3D [28] into HTML documents. Over the past year substantial performance improvements of X3DOM enable progressive streaming of heavily optimized binary data to the client to further enhance the usability and maximize resource usage of the library [19].

Rendering a 3D scene on a web site constitutes only one challenge in bringing 3D into the browser. Within the CH domain, 3D models usually stem from scanner acquired data at very high resolutions, consisting of an abundance of vertices. Those large 3D models are non-trivial to manage, especially when constraints like memory, network bandwidth, storage capabilities and processing power are key factors in the successful adoption of an application. Therefore the raw models need to be optimized massively to be useful on devices with the limited resources mobile phones, tablets or commodity desktops with integrated graphics provide. To that end we further developed the *aopt* transcoder tool for optimizing 3D models with special focus on scene-graph data [16]. We use *aopt*, which is part of the *InstantReality* framework [14] that utilizes the X3D standard [28] as application description language, to convert the 3D scene into an HTML5 representation suitable for the presentation in X3DOM. The tool also allows to optimize large models for progressive loading through X3DOM (cf. [6] and [17,19]).

To optimize a model even further, the software package *MeshLab* is used. It allows for more advanced healing and filtering processes like removing faces from non-manifolds or edge collapse mesh decimation[7]. Within the CIF pipeline running the model through *MeshLab* is an optional pre-processing step. To this end we are using a service instance of *MeshLab* (called *MeshLabServer*). For a detailed overview on how the CIF pipline is orchestrated cp. [29].

2.2 State of the CIF

Since the introduction of the CIF prototype, further reserach has built upon the presented framework. Especially Aderhold et al. [1] described more specifically how the transcoding and optimization backend of this system can be scaled through the use of distributed processing. Moreover, the system has been improved for robustness and has been in productive use for over a year now. The

originally presented stack (cp. [29], Figure 2) was slightly modernized to a production ready version. For example, the Apache/mod_wsgi combination has been replaced with an nginx [25] and Chausette [30] setup managed by the Circus [23] socket and process manager. An output bundle for the CAD domain has been added as well as additional template related features and complete customizability of the pipeline behavior, through the use of a template preset system, is also work in progress. Finally, in gathering best practices for CH workflows, a case study has been provided by Baldassari et al. [20] by example of the online museum of "Villa Di Livia".

3 Resource-Oriented Access

While the CIF allows for easy transcoding and optimization of 3D models for Cultural Heritage applications, it is inherently tied to the workflow used by that domain. With the proposed system it is not readily possible to integrate the transcoding, optimization and templating into other applications, like batch processing scripts, native mobile applications or even yet unknown scenarios like for example an online furniture shop, where the user can customize a living room suite or cupboard and interactively try out the drawers etc.

In essence, the basic steps and operations performed to transform a model into a Web usable form are generic by design. Domain-specific behavior can be implemented through the use of highly customized output templates. In order to access the fundamental technologies underpinning the CIF, we propose a generic way to access the CIF subsystems by means of a standards-based API to facilitate the use of the CIF a service.

As more and more of the web experience moves into the browser, enabling access to applications and data through asynchronous HTTP calls is increasingly important. Therefore, we decided to extend the CIF with a resource-based interface. By exposing a middleware layer through a network-based service, the entire CIF pipeline, or only parts of it, can be accessed by any 3rd party application using a fixed set of API instructions.

This service endpoint can be accessed utilizing the standardized HTTP [10] protocol combined with the JSON [9] data exchange format. A resource-based architecture inspired by Fielding [12] is employed to make full use of the HTTP protocol as well as provide a sensible future aware and easy to understand implementation.

Through this API complete control of the individual steps in the processing pipeline can be offered. In contrast to other HTTP based protocols like SOAP, using elements characteristic to REST style architectures allows us to fully exploit the potential of HTTP. In SOAP for example, mainly the HTTP POST verb is used and return status codes are largely ignored. Additionally by implementing a concise JSON-based request/response content, in contrast to the verbose XML messages used in SOAP, we minimize the amount of data to be transferred over the network. This part is especially important for mobile devices which often have a limited network bandwidth capacity and/or throughput.

3.1 Entities, Resources and Operations

To use the CIF through a generic API a new vocabulary needs to be derived to address resources within the system. What was formerly handled by an HTML web interface with an implicit narrative in the domain language of the CH domain, now needs a more generic exposition to be useful in other domains. Therefore, the following entities in the system are defined:

Assets are files processed by the pipeline, for example model files, metadata, templates, textures, images, etc.

Application Bundles define how the transcoded and optimized model(s) will be delivered. A bundle consists of one or more "templates", images, JavaScript, etc. that form a blueprint HTML application. Additionally, a bundle can contain specific settings, which allows for very fine grained and sophisticated configuration of the pipeline behavior. Those two powerful mechanisms allow creating highly domain specific application bundles that range from simple catalog style applications to whole virtual environments like an interactive virtual museum.

Buckets represent a container for a single asset or a collection of assets. In the simplest form this container represents a directory on a disk containing a set of files. However, there are no conceptual limitations of where and how the contents of a bucket are stored. For example it could also be a database or bigdata file system.

Jobs are asynchronously executing sets of operations triggered with a specific payload and input/output options. Usually the input of a job consists of a URI pointing to a bucket or a single web resource. The output of job can be a URI to a bucket or other web resource.

Tasks are small, side-effect free, independent computing units which perform a single specific function, for example transcode a model or unzip compressed assets. Tasks are triggered during the lifecycle of a job. Tasks can also start subtasks which can run in parallel or block the parent task until completed. Tasks run distributed and can be spread over multiple machines to expedite processing (cp. [1]).

Payloads represent sets of assets to be processed by the pipeline. Usually this is the content of a bucket, identified by a URI pointing to the bucket container. It could also be a URL pointing to a single resource on the Web, for example.

Furthermore we distinguish between *collections* and *items* – both of which can represent a resource. A single item is usually part of a collection. However, there is no artificial limit on how nesting can occur. A specific *task* item for instance can be part of a *tasks* collection. Or a single model file (item) is part of a bucket (collection). On each resource different operations can be performed like *retrieve*, *create*, *delete*, *update*, *status* etc.

3.2 HTTP and JSON

For the CIF, we decided to address resources through the well understood HTTP protocol. Each resource is addressed by a unique URI (Uniform Resource Identifier) combined with an HTTP verb detailing the required operation to be performed on the resource. Optionally an HTTP body containing JSON data is provided with requests that need to modify a resource. Using HTTP verbs, the following CRUD operations can be performed:

GET Retrieve a resource (e.g., `GET /tasks` or `GET /tasks/4711`)
POST Create a new resource within a collection (`POST /buckets`)
PUT Update an existing resource (`PUT /buckets/123/herkules.ply`)
DELETE Remove a resource (`DELETE /buckets/123/herkules.ply`)

The next example illustrates the use of our proposed API through HTTP requests. We demonstrate a typical workflow to obtain a fully optimized model, embedded in an HTML application, by providing a locally stored large model. To illustrate the working of the API on the lowest level, we make use of the `cURL` [27] utility, a command-line HTTP client, which can be also be used in batch processing scripts. Additionally, Figure 1 outlines the entire API session.

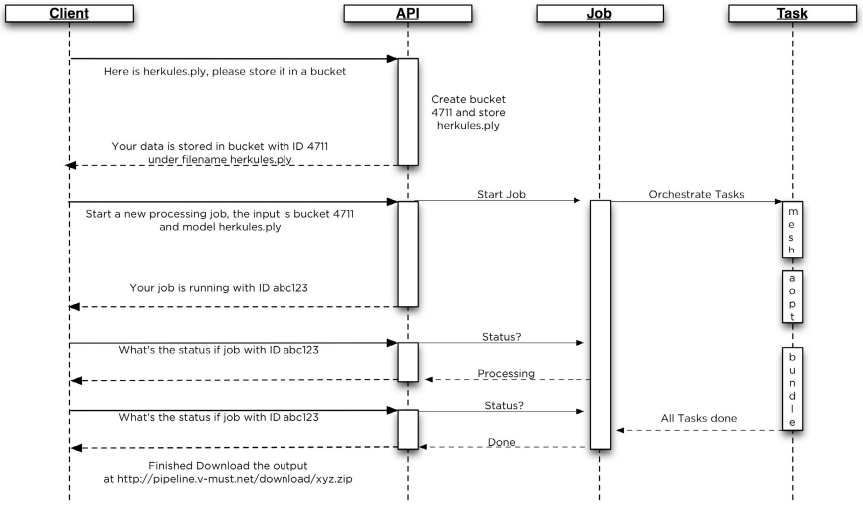

Fig. 1. Sequence of a typical API session to convert a 3D model to a representation suitable for web display

First a large model stored locally needs to be uploaded to a bucket:

```
$ curl -X POST
       -H "X-Filename:herkules.ply"
       -H "Content-Type: application/octet-stream"
       --data-binary @fixtures/herkules.ply
```

```
      http://pipeline.v-must.net/api/v1/buckets

# Response:
HTTP/1.0 201 CREATED
Content-Type: application/json
{
  "bucket_id": "4711",
  "filename": "herkules.ply",
  "message": "Bucket and file created."
}
```

The response contains a bucket ID, `4711` for sake of this example, as well as a file name. We retain this information and use it to start processing of the file just uploaded:

```
$ curl -X POST
       -H "Content-Type: application/json"
       -d '{"payload":"bucket://4711","payload_filename":"herkules.ply"}'
       http://pipeline.v-must.net/api/v1/jobs

# Response:
HTTP/1.0 201 CREATED
Content-Type: application/json
{
  "message": "Task added and started",
  "task_id": "abc123"
  "job_url": "http://pipeline.v-must.net/api/v1/jobs/abc123",
  "progress_url": "http://pipeline.v-must.net/api/v1/stream/abc123/"
}
```

The response contains, amongst other data, a `job_url` field. This URL can subsequently be used to poll the server for status information about the conversion process.

```
$ curl -X GET http://pipeline.v-must.net/api/v1/jobs/abc123
HTTP/1.1 102 PROCESSING

# in case of success:
HTTP/1.0 200 OK
Content-Type: application/json
{
  "state": "SUCCESS"
  "message": "Conversion Ready.",
  "download_url": "http://pipeline.v-must.net/download/abc123/abc123.zip",
  "preview_url": "http://pipeline.v-must.net/preview/abc123/index.html"
}
```

As soon as the conversion finished, the response contains JSON data with download and preview links, which can be used to display the web page in a browser or download the whole HTML application in a single archive.

For an exploded technical overview of the HTTP calls involved in this workflow, please refer to [15].

4 Use Cases

4.1 Mobile 3D Repository Browser

Cultural Heritage repositories are an important tool to help historians and researchers in their work. For example Doerr et al. [8] describes the design and implementation of such a repository for large CH models, and Koller at al. [18]

recognizes the need for interoperability of large CH archives, however does not specify details on how this challenge could be met. Using mobile devices that connect to CH repositories provide great potential for researchers and historians. A device like a tablet computer could be used for efficient communication, on-site exploration, augmented reality applications, and much more. Commonly, within a 3D model archive a large number of big models are stored for preservation. Exploring or even annotating, modifying or storing these models on a mobile device over the internet is currently not feasible. Large models take a long time to download and potentially require a massive amount of storage space a mobile device can not provide.

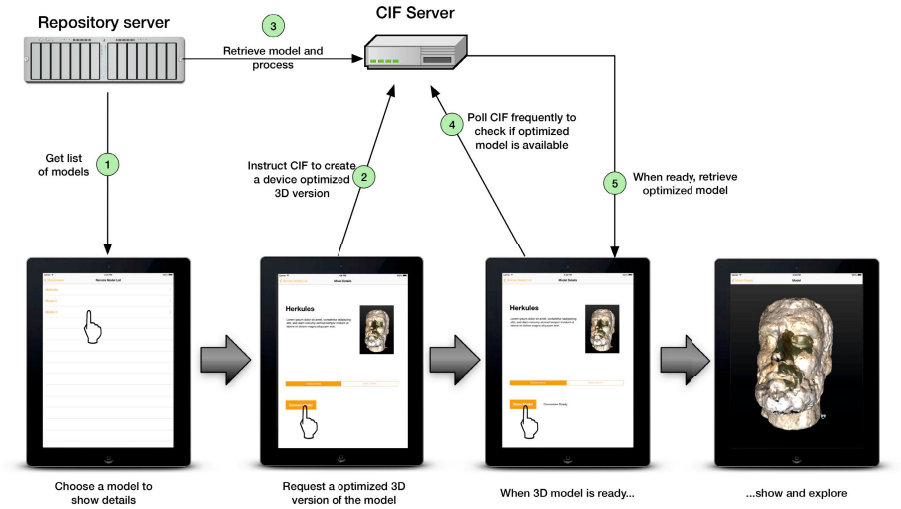

Fig. 2. Basic operation of the repository browser application. A a list of models and metadata is retrieved from a 3D repository (1). Details are shown in a separate view and a device optimized version can be requested through the CIF API (2). The CIF servers retrieve the requested model from the remote location and start to create an optimized version (3). The mobile device is polling the CIF servers to determine if the conversion process has finished (4). Once the model is ready, it is retrieved (5) and displayed to be explored on the mobile device.

To help overcoming those limitations, we also created a prototypical mobile repository browser application, which utilizes the pipeline API for just-in-time transcoding and optimization of remotely stored models. For optimal performance, our *iOS* application makes use of a pipeline API feature which allows to convert a model stored on another server accessible by the pipeline cluster (i.e., the 3D model archive). In order to trigger the conversion of a remotely stored model, the iOS application sends the location of the large model to the pipeline servers. The pipeline servers subsequently download the remote model, optimize it for the mobile device and provide a preview as well as a downloadable bundle for the device to further process as needed (cp. Figure 2).

4.2 MeshLab Web Export Filter

Another interesting example of use of the CIF API regards the extension of the well-known geometry processing software *MeshLab* for exporting 3D models for the Web. This permits any user to deploy his/her content on a web site, even if the user has no knowledge of 3D graphics or web programming.

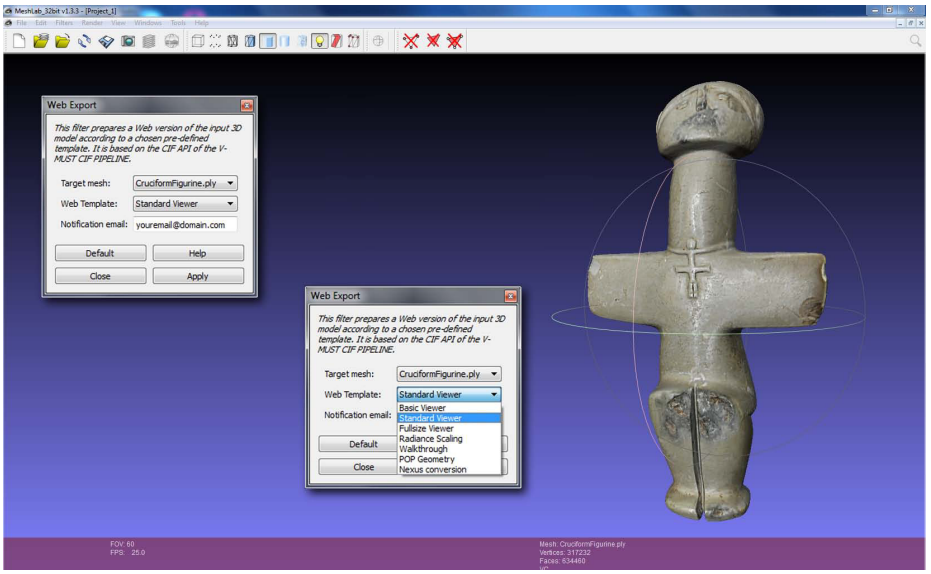

Fig. 3. The MeshLab plugin allows the user to send models stored in different MeshLab layers to the pipeline. Furthermore, she can select a target template and leave an email address to which the pipeline sends a notification email with download and preview links upon completion of the transcoding process.

A specific plugin has been designed for this purpose. The MeshLab user, after doing the desired processing on the 3D content, can use this *Web Export* plugin to prepare its content for the Web. The plugin requires to select the 3D model to convert (since multiple 3D models can be loaded in different layers in MeshLab), to select the application template according to which the 3D content will be prepared for the Web, and to enter a notification email to get the results, i.e. the converted model. The notification email contains also the link where to download the web package prepared by the CIF. Obviously, the selection of a different application template causes that a different web package is generated. Figure 3 shows this plugin in action. As can be also seen, the converted output does not necessarily to be in X3D/X3DOM format, but can be also any other 3D format convenient for the web, like the Nexus format for multi-resolution meshes [24], which now is also available as a WebGL/SpiderGL implementation on the client side.

From an implementation point of view, since MeshLab uses extensively the *Qt* framework, the Web Export plugin simply performs the necessary HTTP requests using the *Qt Network* module that comes along with Qt.

5 Conclusion and Future Work

With this publication we solved a standing issue of the Common Implementation Framework for Online Virtual Museums. We have shown how the CIF can be utilized as a backend service for a wider audience than just the CH domain through the introduction of a unified middleware access layer based on top of the HTTP and JSON standards. We applied a REST [12] inspired architectural style to access entities within the system as resources. During this process we established a vocabulary generic enough to be applicable to various domains that use 3D models as well as narrow enough to be useful in practice. Applying well understood standards like HTTP as a transfer channel and the light-weight JSON as data exchange format, on top of this vocabulary, we achieved a system which can be wielded by 3rd party workflows, applications, programming languages, or even shell scripts.

To demonstrate the usefulness as well as the technical possibility, we have shown two use cases where the pipeline API can be utilized to improve usability and optimize workflow for end-users. First, a novel mobile application has been created that allows to browse an archive of possibly large 3D models and convert a model, just-in-time, to a device optimized form for immediate display and/or storage on the device. Second, the well-known MeshLab application has been extended by a plugin, which allows the user to export 3D models to a Web site using the CIF as a backend service.

While the CIF matured considerably the past year, there are many areas which remain unexplored. First and foremost, the powerful Application Bundle and Template mechanism, as well as more specific details of the application generation layer, still need to be considered in more detail. Another standing issue is how and where the input and output is stored and organized. Even though we introduced the bucket as a storage container and organization concept in this paper, the advance of cloud-based storage offerings provides a good opportunity for further research.

The entire topic of scalability, which has been touched upon in [1], is another work in progress. For example, the automation of a process to dynamically allocate cloud resources and startup/shutdown machines on-demand for a given set of model conversions and/or based on pre- or auto-determined criterias like time, budget, and model complexity has not yet been thoroughly explored.

While the API presented here is heavily influenced by Fielding's REST architectural patterns, it is not truly RESTful in the original sense (see also [11]). The approach we used is coherent, pragmatic and used by many web services today and achieves a Level 2 rating in the Richardson Maturity Model (cp. [13]). A future interesting area of research might be to modify the CIF API to adhere strictly to REST principles.

Finally, the presented iOS application is in its early stages and leaves many areas of improvement. For example, caching of optimized models and metadata has not yet been realized (which is a requirement for offline usage). Furthermore, additional features like the annotation of models, sharing and social media integration, geo-location related features may present further interesting areas of research.

Free Software. The results of this research as well as prior efforts are all available in form of free and open source software projects.

- MeshLab: `http://meshlab.sourceforge.net`
- CIF Pipeline: `http://github.com/aha/pipeline`
- iOS App: `http://github.com/annakasia79/modelviewer`

Acknowledgments. Portions of this research were carried out within the FP7-funded EU project *V-MusT*. The 3D model of the cruciform figurine shown in Figure 3 has been kindly provided by the Cyprus Institute. Permission of use has been granted by the Department of Antiquities, Cyprus.

References

1. Aderhold, A., Jung, Y., Wilkosinksa, K., Graf, H., Fellner, D.W.: Distributed 3d model optimization for the web with the common implementation framework for online virtual museums. In: Proceedings Digital Heritage 2013, vol. 2, pp. 719–726. IEEE and Eurographics (2013)
2. Apple, Inc.: The ios developer library (2013), `http://developer.apple.com/library/ios/`
3. Arnold, D.: 3d-coform: Tools and expertise for 3d collection formation. In: Proceedings of EVA, pp. 94–99 (2009)
4. Baumann, P., et al.: Earthserver - european scalable earth science service environment (2011), `http://www.earthserver.eu`
5. Behr, J., Jung, Y., Drevensek, T., Aderhold, A.: Dynamic and interactive aspects of x3dom. In: Proceedings Web3D 2011, pp. 81–87. ACM, New York (2011)
6. Behr, J., Jung, Y., Franke, T., Sturm, T.: Using images and explicit binary container for efficient and incremental delivery of declarative 3d scenes on the web. In: Proceedings Web3D 2012, pp. 17–25. ACM, New York (2012)
7. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: Meshlab: an open-source mesh processing tool. In: Sixth Eurographics Italian Chapter Conference, pp. 129–136 (2008)
8. Doerr, M., Tzompanaki, K., Theodoridou, M., Georgis, C., Axaridou, A., Havemann, S.: A repository for 3d model production and interpretation in culture and beyond. In: Proceedings of the 11th International Conference on Virtual Reality, Archaeology and Cultural Heritage, VAST 2010, pp. 97–104 (2010)
9. Ecma International: Standard ecma-404, the json data interchange format, 1st edn. (2013)
10. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol – http/1.1 (1999)

11. Fielding, R.: Rest apis must be hypertext-driven (2008), http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
12. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis, University of California, Irvine (2000)
13. Fowler, M.: Richardson maturity model (2010), http://martinfowler.com/articles/richardsonMaturityModel.html
14. Fraunhofer Gesellschaft: Instant Reality (2013), http://www.instantreality.org
15. Fraunhofer Gesellschaft: Pipeline Documentation (2013), http://pipeline.readthedocs.org
16. Jung, Y., Behr, J., Graf, H.: X3dom as carrier of the virtual heritage. In: Remondino, F. (ed.) Intl. Society for Photogrammetry and Remote Sensing (ISPRS): Proceedings of the 4th ISPRS International Workshop 3D-ARCH 2011: 3D Virtual Reconstruction and Visualization of Complex Architectures (2011)
17. Jung, Y., Limper, H.P., Schwenk, K., Behr, J.: Fast and efficient vertex data representations for the web. In: Proceedings of the 4th Intl. Conf. on Information Visualization Theory and Applications, pp. 601–606. SciTePress (2013)
18. Koller, D., Frischer, B., Humphreys, G.: Research challenges for digital archives of 3d cultural heritage models. J. Comput. Cult. Herit. 2(3), 7:1–7:17 (2010)
19. Limper, M., Jung, Y., Behr, J., Sturm, T., Franke, T., Schwenk, K., Kuijper, A.: Fast, progressive loading of binary-encoded declarative-3d web content. IEEE Computer Graphics and Applications 33(5), 26–36 (2013)
20. Lucci Baldassari, G., Demetrescu, E., Pescarin, S., Eriksson, J., Graf, H.: Behind livias villa: A case study for the devolution of large scale interactive "in-site" to "on-line" application. In: Marcus, A. (ed.) DUXU/HCII 2013, Part IV. LNCS, vol. 8015, pp. 238–247. Springer, Heidelberg (2013)
21. Marrin, C.: WebGL specification, http://khronos.org/registry/webgl/specs/latest/ (2012)
22. Microsoft: Webgl api for internet explorer (2014), http://msdn.microsoft.com/en-us/library/ie/dn302469v=vs.85.aspx
23. Mozilla: Circus – A Process and Socket Manager (2012), http://circus.readthedocs.org
24. Ponchio, F.: Multiresolution structures for interactive visualization of very large 3D datasets. Ph.D. thesis, Clausthal University of Technology (2008), http://vcg.isti.cnr.it/nexus/
25. Reese, W.: Nginx: The high-performance web server and reverse proxy. Linux J. 2008(173) (2008)
26. Scopigno, R., Callieri, M., Cignoni, P., Corsini, M., Dellepiane, M., Ponchio, F., Ranzuglia, G.: 3d models for cultural heritage: Beyond plain visualization. Computer 44(7), 48–55 (2011)
27. Stenberg, D., Fandrich, D., Tse, Y.: curl groks urls, http://curl.haxx.se/
28. Web 3D Consortium: X3d international standards (2013), http://web3d.org/x3d/specifications/
29. Wilkosinska, K., Aderhold, A., Graf, H., Jung, Y.: Towards a common implementation framework for online virtual museums. In: Marcus, A. (ed.) DUXU/HCII 2013, Part II. LNCS, vol. 8013, pp. 321–330. Springer, Heidelberg (2013)
30. Ziade, T.: Chaussette WSGI Server (2012), http://chaussette.readthedocs.org