Towards a Common Implementation Framework for Online Virtual Museums

Katarzyna Wilkosinska, Andreas Aderhold, Holger Graf, and Yvonne Jung

Fraunhofer IGD, Darmstadt, Germany
kasia@wilkosinska.com, {aaderhol,hgraf,yjung}@igd.fraunhofer.de

Abstract. We present a prototypical solution to a common problem in the Cultural Heritage (CH) domain. After creation, 3D models of CH artifacts need to be processed to a format suitable for presentation on multiple platforms, e.g. in a Web Browser for online virtual museum applications, to target desktop computers and mobile devices alike. The constraints of an in-browser presentation give rise to a series of optimization and conversion concerns that need to be addressed to successfully display the CH objects in a Web application. Current 3D authoring tools do not readily support this kind of optimization and conversion required for CH domain scenarios. We therefore propose a web-based service framework, which solves the problem of pipelining 3D models for interactive Web presentations. We apply open-source technologies like X3DOM, Flask, Celery, and Redis to create a *Common Implementation Framework* (CIF) that allows content designers or researchers to optimize their 3D models for the Web through a simple one-step process.

Keywords: Web3D, Virtual Museums, Content Authoring, Cultural Heritage.

1 Introduction

Internet services like Google search or online shopping are getting more and more ubiquitous. Nevertheless, most Web applications are still 2-dimensional, though 3D graphics allows new applications, such as a Web shop, to present its products as 3D models. Likewise, interactively exploring Cultural Heritage (CH) artifacts in an online Virtual Museum based on HTML5 is possible today, even on a smart phone or the iPad [15]. Recently, with WebGL [14], real-time 3D graphics in the Browser became a reality. But since WebGL is a low-level API wrapping GPU functionality, it is not very accessible for the typical Web developer who is used to build Web applications using HTML, CSS, and scripting the DOM. Therefore, the JavaScript-based open-source framework X3DOM [3] provides declarative 3D in HTML5 while utilizing modern Web technologies like Ajax, CSS3, and WebGL.

However, the given 3D models first need to be converted into a representation suitable for the presentation on the Web. Moreover, for preview or to ease further application development, the 3D objects should be directly integrated into

A. Marcus (Ed.): DUXU/HCII 2013, Part II, LNCS 8013, pp. 321–330, 2013.

an HTML template. Finally, as the 3D visualization is part of a web page, high performance is required. Here, mesh optimization is of high importance. Essentially, Scopigno et al. [23] stated that historians, curators and other professionals are seeing more and more the need for 3D data to support their work, but that the greater challenge lies in creating new tools that use 3D models to assist CH research, to plan and document restoration, and so on.

Thus, in this paper we present an online platform that combines web-based tools and services to support the development of Virtual Museums with particular focus on presentation and visualization of Cultural Heritage assets in online virtual museums. Likewise, Berndt et al. [4] proposed a full workflow pipeline from data acquisition up to interactive web-based visualization in the Cultural Heritage domain. However, they primarily discussed 3D surface reconstruction and for presentation the authors embedded the 3D objects into a PDF document. Contrariwise, the main goal of our proposed "Common Implementation Framework" (CIF) is to provide to stakeholders and application developers alike, a useful Web platform with a harmonized pool of tools based on integrated Web technologies, to support and help them in the development of their projects.

2 The CIF Platform

The design of the CIF is based on an analysis of typical modeling workflows in the Cultural Heritage domain, starting from raw data, including the perceived limitations of current technologies, such as the deployment of 3D CH visualizations and their seamless combination with other media.

2.1 Workflows in the Cultural Heritage Domain

To prepare 3D models of artifacts for display, a multitude of operations are required, like creating a 3D model from a point-cloud, cleaning, healing and optimizing that model, converting between formats and make it suitable for display on networked devices. So, when creating a virtual museum, it is desirable to show 3D representations of CH artifacts and architectural structures on a variety of devices and operating systems to address the currently ongoing divergence of application platforms. Therefore, recently Rodriguez et al. [18] for instance presented a mixed client/server architecture, where an intelligent server prepares and delivers large point clouds in such a way that even a mobile client can display the data at interactive frame rates. However, besides their special render server, data compression along with encoding/decoding is still an intricate and time consuming process. Therefore, we focus on simpler optimization schemes (cp. e.g. [3] and [13] for more details).

It is also necessary to provide intuitive interaction possibilities tailored to a specific device – e.g., multi-touch on tablet vs. mouse on desktop. Additionally, showing or even adding meta data or annotations might be of benefit for experts [15], while the system should also provide means to navigate around and examine the object. Another very interesting application would create an entire immersive

3D world of a museum or an archeological excavation site, so that scientists, historians, etc. are able to get a better insight than it would be possible from photographs. Currently, there are several processes that need to be performed to achieve these goals. The first step is to scan the artifacts or historical sites to create digital 3D models. After scanning, the models need to be further refined using standard DCC tools like 3ds Max or Blender.

In order to present the models online, it is essential to have small file sizes and low memory footprints. This allows not only for faster download but also accounts for limited computing power of handheld devices and the constraints of Web applications running in a browser. Therefore, another process is usually model optimization that can be performed with a dedicated mesh optimization software such as MeshLab [5]. The output of this process, however, typically consists of another file in a standard 3D format like OBJ or PLY, which cannot be interpreted by a browser. Moreover, the mesh structure is not optimized for web-based rendering (e.g., for indexed rendering WebGL requires having geometry chunks of at most 2^{16} vertices). And finally, the optimized and converted models should be stored in a standardized format that will still be available in the future, why special proprietary WebGL-based 3D formats are obviously not useful. Otherwise, the underlying format should be declarative so that it can be easily integrated into a web service architecture. Here, using standardized Web techniques and services also enables an automated connection of existing data with related data to be visualized. In this context, [22] outline how to combine and link 3D data with other sources of information on the Web.

Therefore, we propose using X3DOM, a JavaScript-based framework for declarative 3D graphics in HTML5, that is based on the open ISO standard X3D [25] for describing the scene and runtime behavior. For data transcoding, the *aopt* tool from the InstantReality framework [8] can be used (see [12] for details). Thus, the model will be available for presentation in a HTML page. Though, using multiple command line tools chained together and applying styling and HTML markup by hand is possible for a few models and with appropriate knowledge, yet it is usually not a feasible approach for real-world scenarios. Thus, to achieve a more user friendly content preparation workflow esp. for non-experts, we have implemented a special web service as part of the Common Implementation Framework (CIF) proposed in this paper.

2.2 Proposed Methods and Components

For our prototype (cp. Figure 1), we have selected important visualization tools as well as optimization and transcoding services, which we will present within this paper, while also discussing possible guidelines for further design and development of the proposed architecture. The current CIF pipeline is basically as follows. First, the 3D model is uploaded. Then, a sequence of pre-processing steps is applied in order to optimize the model. Finally, the given model is transcoded into an X3DOM file and ready to be displayed on the Web.

In order to build a solid foundation of the CIF, it is mandatory to design a technology stack that is future proof. The technology and process choices have

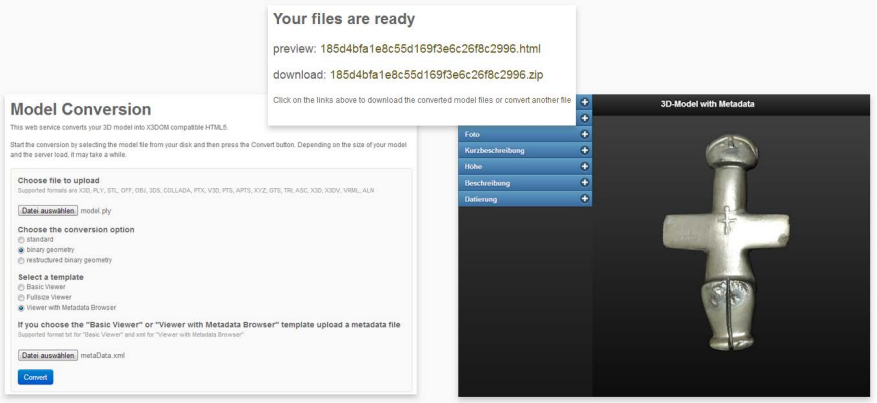

Fig. 1. Entry point for uploading and converting a 3D model (left). After the conversion is ready, a new page appears (top) with an option for online preview (right) and one for downloading a zipped package.

been taken to support ultimate flexibility, ease of integration of third party tools, and highest possible degree of maintainability, scalability and testability. In order to build a maintainable long-lasting system, it was paramount to select components and processes that are not owned by a corporation (hence are truly open source) and which are widely used in the HTML5 domain and unlikely to be vulnerable to the bus factor. Therefore, we decided to base the CIF on modern open source web application technologies with the following technology decisions. All components in this stack (compare Figure 2) are loosely coupled and thus replaceable at any time, should the need arise to exchange a specific component within the stack.

Python a stable general purpose programming language to create the web service and integrate other tools. It is a core system component of most Linux/Unix-style operating systems and therefore already available.

Apache/mod_wsgi as web server front-end; Apache is a web server and mod_wsgi a module to run Python application services from within Apache.

Celery as the task queue to make processing asynchronous.

Redis a key-value database for storing transient data, messages, and caching.

MeshLab to reduce and clean-up triangular meshes.

InstantReality transcoder to optimize large models for the Web.

X3DOM to present interactive 3D models in a Web browser.

Tools for 3D Rendering and Optimization. As mentioned, *X3DOM* (see `http://www.x3dom.org`) is an open source framework and runtime to integrate declarative 3D content directly into HTML5 by using a so-called "polyfill" layer to mimic native Web browser support. We use X3DOM [3,13] mainly for two reasons: on the one hand, the declarative data representation fits nicely with our

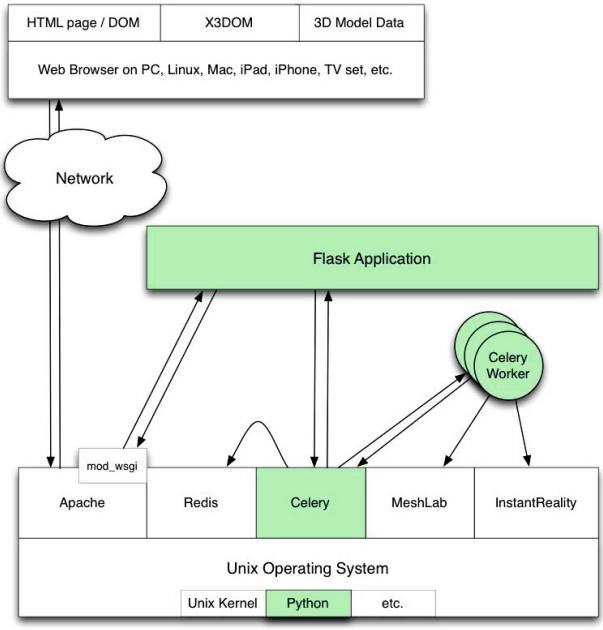

Fig. 2. The proposed CIF stack (green denotes Python-based components)

service architecture (cp. Figure 3), and on the other hand, various compressed binary formats are supported. Due to its seamless DOM integration X3DOM provides another advantage, since the application developer or content editor only needs to deal with interaction and presentation, whereas the graphics artist only needs to prepare the 3D models.

For data transcoding we have developed the *aopt* transcoder tool for optimizing 3D models with special focus on scene-graph data [12]. The transcoder is part of the *InstantReality* framework [8] that uses the X3D standard [25] as application description language. The CIF stack presented here uses aopt to convert the MeshLab optimized model into an HTML5 representation suitable for the presentation in X3DOM. The tool also allows to optimize large models for progressive loading through X3DOM (compare [3,13]). For more advanced healing and filtering processes like removing faces from non-manifolds or edge collapse mesh decimation, we use a service instance of *MeshLab* (called MeshLabServer), which is a portable open source system for processing and editing unstructured triangular 3D meshes [5].

Utilized Web Technologies. The core technology of our service platform is *Flask* [20], a Python-based micro framework designed for developing Web applications conforming to the Representational State Transfer (REST) paradigm [9,24]. For scalability, we furthermore use Celery [17], an asynchronous

distributed task queue library that can be integrated well into Flask and uses message passing. Redis [21], a high-performance key-value store, suits as message broker. First of all, the dynamic nature of *Python* and ease of use, in combination with comprehensive high-quality libraries [11] creates a "best-for-fit" web application development environment in general [1,10] as well as for the CIF framework in particular. We therefore use Python as base language for the CIF and integration of other tools and components.

Flask is based on the libraries Werkzeug [19] and Jinja2. In contrast to fully fledged frameworks like Django or Zope, Flask only contains very basic functionality to build web applications and services. Essentially, it is a thin wrapper around the Python Webserver Gateway Interface Standard (WSGI) [7] – the Python specific standard of a application server communication protocol (similar to the Java Servlet Standard). Flask also acts as URL router and Model-View-Controller (MVC) facilitator as well as provides a library toolset for common web application tasks. Flask allows the developer to map Python routines to URLs in a declarative way. The following example listing illustrates this mechanism.

```
from flask import Flask
from flask import render_template

app = Flask("HelloWorldApplication")

@app.route("/hello")
def hello():
    return render_template('hello.html')

app.run()
```

This example app contains a method to render a given HTML template. Once the script is executed, the application serves itself on a given port. The decorator `@app.route` is used to instruct the Flask kernel to call the `hello` method when the URL `/hello` is accessed in the browser. The `render_template` method then parses the given template and returns a string, which forms the HTTP response body that is sent back to the browser. This idiomatic approach gives the developer flexibility in library choices while removing the pain of doing error prone HTTP input/output work manually and in turn allows for easy creation of RESTful applications and APIs without hiding actual functionality. A clean separation of concerns is easily maintained but not enforced.

The *Apache HTTP server* is the de-facto standard of delivering web sites and applications today [16]. Apache listens for HTTP connections on a port and then delivers files within a HTTP response. It also automatically spawns, scales and restarts web application processes as required. In order to interface with programs like a C application, PHP, or our CIF, it needs a special extension (called module) which facilitates communication between the web server and the web application. *Mod_wsgi* [6] is such a module, which allows to run Python application servers behind the Apache HTTP server. It should be noted that the Apache HTTP server, while very useful and easy to configure, is not always a sensible choice in production environments that need to support performance and scalability with a low memory footprint. However, due to the modular design

of the CIF stack, Apache can very easily be exchanged in favor of a more modern web server/ app server frontend (i.e., nginx/gunicorn).

Celery is a Python-based open source asynchronous task queue based on distributed message passing. It is easy to integrate in Python applications like Flask web apps or command line tools. Celery requires an additional component to send and receive messages, usually in form of an additional message broker service like RabbitMQ. The message broker can also be a database (e.g. Redis or CouchDB [2]) in order to retain task status information. We use Celery in order to make processing tasks in the pipeline asynchronous. Instead of letting the user wait on the processing, which could take up to minutes or even longer during high-load situations or for large models, the task is executed in the background. Once completed, the status is reported back to the web application which then lets the user download the converted model. This allows starting multiple conversion tasks simultaneously and checking the results at a later time. To find the tasks later on, a unique URL is created. At this URL the current status is displayed and, in case of successful conversion, download links are provided.

Redis is a high-performance key-value "no-SQL" Web 2.0 tool. In our technology stack, Redis is used by Celery for message passing and storing of tasks as well as status information. This allows appropriate feedback when a task fails. It may also be used to implement a rate-limit system or data caches. We thus also use Redis to implement a basic Denial of Service (DoS) attack vector counter measure called rate-limiting. In order to prevent abuse of the web application and bring down our servers, we limit the frequency of requests to the conversion pipeline for a specific user. To keep track of the IP addresses and requests already performed, Redis is used as well.

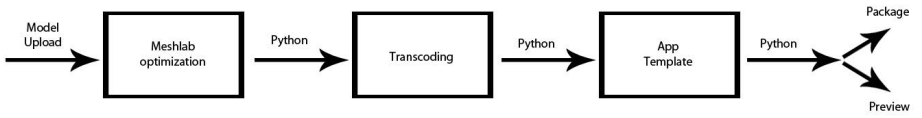

Fig. 3. The CIF workflow: a Python-based pipeline architecture for 3D model conversion and application generation

3 The Conversion Pipeline

Our proposed platform should provide the missing link between model creation and the presentation in a web browser and web services that automatically combine raw data and pre-prepared application templates into interactive 3D visualizations for the Web. The application templates define the desired application type, e.g. walk-through or standard 3D scene exploration, a user interface, etc. Our integrated online service platform thereby allows for enabling data transfer and automatic workflows for the creation and deployment of 3D Virtual Museum assets. This way, a simple method to integrate 3D visualizations into a Web application and to easily build prototypes is offered, while interactive elements can

be easily added afterwards by the application developer or directly specified by providing appropriate application templates in advance.

As mentioned, the heart of the CIF is a Flask application (section 2.2), a lightweight Python component which orchestrates user input, processing, and output. It also performs mapping of URLs to Python routines – the so-called URL mapping. The first step is to upload the raw model into our application as seen in Figure 1 (left). This is achieved via a web form that is rendered by the flask application. At this point the user can also select which optimizations need to be performed on the model and which template he wants to select for the subsequent HTML presentation. After sending the form via the *Convert* button, the model file is uploaded to the web application and the conversion starts.

The entire conversion is handled by a Celery worker in the background, which allows the web app to continue while the processing happens. A unique URL is generated and sent back to the user, which informs about the current status of the process. This URL can also be used to come back later, if it should take too long to wait or the user wished to start another conversion. A Celery worker (Figure 2) performs the optimization of the model, converts it to X3DOM and creates a compressed package of the generated model data. In case of an error, the result page shows more details. Otherwise, a download link as well as a preview link are provided (see Figure 1, top) to allow downloading the whole package or directly viewing the results in the browser (Figure 1, right).

As shown in Figure 3, the CIF pipeline consists of several steps. First, the user logs in using an auth mechanism, which not only allows uploading models but also to mark them for conversion at a later time, re-process the model with different settings without uploading it again, or emailing the converted model to the user. After uploading the model, the mesh reduction and optimization starts (e.g., removal of duplicate faces, vertices or degenerate triangles). Depending on the selection of optimization parameters, a filter is created which is passed to MeshLab in a subsequent call. In case of a successful optimization, the model is being converted to an X3D/X3DOM compatible format using the previously discussed transcoder tool, where the input meshes are also further optimized and restructured if necessary. During this process, status updates of the current conversion progress are provided as feedback. Then the model is embedded into an HTML template. Currently, a basic and full-size viewer are integrated here as well as a viewer with additional meta-data browser. The final step is compressing everything to an easily downloadable zip package and generating a preview URL, which can be used to immediately view the results in the browser.

4 Conclusion and Future Work

The CIF presented here is available on GitHub (`https://github.com/x3dom/pipeline`) and provides a new way to rapidly develop online virtual museum applications using 3D technology. Moreover, through our proposed platform the application developer is able to quickly create small prototypes to find better solutions for larger projects. Therefore, it becomes a simple task to rapidly create

a prototypical web application with embedded 3D graphics, which then can be extended with further interactive elements or just published on the web as-is. The user does not need to have deep knowledge of the tools involved, nor is it required to use all those tools manually to optimize a model.

The presented framework is still of prototypical nature but lays the foundation for further work. By choosing interchangeable open source components for building the technology stack, as well as architecting the software by standards like the MVC pattern and modularization of functionality, parts of the system can be extended, customized and refined further. Even if the need should arise to replace certain layers of the stack, this is entirely possible with minimal effort. There are many more areas in which the CIF can be improved. One such optimization is to provide the user with a broader variety of application template options that allow to choose much more different designs into which the converted model will be embedded. Alongside a greater selection of preset templates, the user should also be able to upload his own template files in order to fully customize the appearance of the optimized and HTML-ified 3D model.

At the moment the CIF only supports interaction via a web interface. This means that the user has to upload a 3D model and select optimization parameters, templates, etc. through an HTML-based interface. While this is a good way to work with less tech savy content creators or editors or for quick tests and just a few models to be converted, it might pose a challenge to high-volume scenarios that demand a more automated workflow. One possible solution to this is accessing the conversion pipeline programmatically, through a standardized web services API. Such an API would allow automating the conversion processes through a scripting language or enables plugin writers to create e.g. a 3ds Max plugin to automatically convert and/or optimize a model through the pipeline via the exposed web service. It also allows exposing this service to other web applications, which can build upon the pipeline, like for example a web-based editing utility that allows to further edit the generated 3D app visually. Such an API would also allow integration or chaining of other, third party pipelines or provide batch processing solutions for a large collection of models.

Acknowledgements. The described work was carried out in the project *v-must*, which has received funding from the European Community's Seventh Framework Programme (FP7 2007/2013) under grant agreement 270404.

References

1. Python programming language – official website. Website, `http://python.org/`
2. Apache Software Foundation: The Apache CouchDB Project (2012), `http://couchdb.apache.org/`
3. 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)

4. Berndt, R., Buchgraber, G., Havemann, S., Settgast, V., Fellner, D.W.: A publishing workflow for cultural heritage artifacts from 3D-reconstruction to internet presentation. In: Ioannides, M., Fellner, D., Georgopoulos, A., Hadjimitsis, D.G. (eds.) EuroMed 2010. LNCS, vol. 6436, pp. 166–178. Springer, Heidelberg (2010)
5. 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)
6. Dumpelton, G.: Python wsgi adapter module for apache (2013), http://code.google.com/p/modwsgi
7. Eby, P.J.: Python web server gateway interface v1.0. Website (2003), http://python.org/dev/peps/pep-0020/
8. FhG: Instant Reality (2012), http://www.instantreality.org/
9. Fielding, R.T.: Architectural styles and the design of network-based software architectures. Ph.D. thesis (2000)
10. Foundation, P.: Python application domains (2012), http://www.python.org/about/apps/ (accessed October 18, 2012)
11. Hellmann, D.: The Python Standard Library by Example, 1st edn. Addison-Wesley Professional (2011)
12. 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)
13. 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)
14. Marrin, C.: WebGL specification (2012), https://www.khronos.org/registry/webgl/specs/latest/
15. Michaelis, N., Jung, Y., Behr, J.: Virtual heritage to go. In: Proceedings Web3D 2012, pp. 113–116. ACM, New York (2012)
16. Netcraft.com: January 2013 web server survey (2013), http://news.netcraft.com/archives/2013/01/07/january-2013-web-server-survey-2.html
17. Rocco, M.: Celery Task Queue (2012), http://celeryproject.org
18. Rodriguez, M.B., Gobbetti, E., Marton, F., Pintus, R., Pintore, G., Tinti, A.: Interactive exploration of gigantic point clouds on mobile devices. In: The 14th Intl. Symposium on Virtual Reality, Archaeology and Cultural Heritage (2012)
19. Ronacher, A.: Werkzeug (python wsgi utlity library) (2011), http://werkzeug.pocoo.org/
20. Ronacher, A.: Flask (python microframework) (2012), http://flask.pocoo.org/
21. Sanfilipo, S.: Redis Remote Dictionary Server (2012), http://redis.io/
22. Schubotz, R., Harth, A.: Towards networked linked data-driven web3d applications. In: Dec3D. CEUR Workshop Proceedings, vol. 869. CEUR-WS.org (2012)
23. 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)
24. W3C: Relationship to the world wide web and rest architectures (2004), http://www.w3.org/TR/ws-arch/#relwwwrest
25. Web3D Consortium: Extensible 3d (X3D) (2011), http://www.web3d.org/x3d/specifications/