XML3DRepo: a REST API for version controlled 3D assets on the web

Conference Paper · June 2013

DOI: 10.1145/2466533.2466537

CITATIONS
15

READS
110

5 authors, including:

Kristian Sons
Deutsches Forschungszentrum für Künstliche Intelligenz
22 PUBLICATIONS **269** CITATIONS
SEE PROFILE

Dmitri Rubinstein
Deutsches Forschungszentrum für Künstliche Intelligenz
22 PUBLICATIONS **200** CITATIONS
SEE PROFILE

Philipp Slusallek
Universität des Saarlandes
286 PUBLICATIONS **4,116** CITATIONS
SEE PROFILE

Anthony Steed
University College London
210 PUBLICATIONS **4,426** CITATIONS
SEE PROFILE

XML3DRepo: A REST API for Version Controlled 3D Assets on the Web

Jozef Doboš
University College London

Kristian Sons
DFKI Saarbrücken

Dmitri Rubinstein
DFKI Saarbrücken
Saarland University

Philipp Slusallek
DFKI Saarbrücken
Saarland University

Anthony Steed
University College London

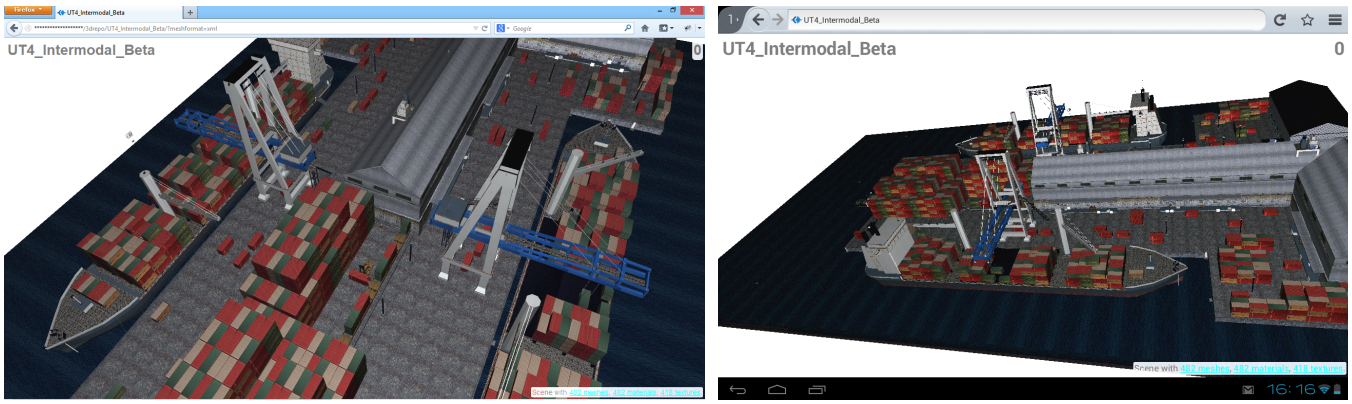

Figure 1: Urban Terror 4 *game level Intermodal Beta with 170k vertices and 418 textures is version controlled in 3D Repo and accessed as XML3D using our REST API in a Firefox web browser on Windows 8 (left) and Android (right). XML3DRepo architecture delivers requested data in the format most suitable for the current device and network, for instance, using Sequential Image Geometry for devices that support texture fetching in the vertex shader or OpenCTM for networks with low bandwidth. Model courtesy of Snipers Gaulois CTF Clan.*

Abstract

Current Web 3D technologies are not yet fully exploiting the modern design patterns for accessing online resources such as REST. XML3DRepo is a novel fusion of XML3D and 3D Repo. XML3D is an open source extension to HTML that supports interactive 3D graphics in WebGL-enabled browsers. 3D Repo is a recent versioning framework for 3D assets that provides raw access to its NoSQL database. XML3DRepo, in turn, is a server-side combination of the two technologies that stores a unified file format independent representation of 3D scenes in its repository but exposes a RESTful API for a deeper integration with other services using a variety of encodings selected between by the client application. First, we outline the overall architecture of the system and provide a simple yet powerful API definition that we believe has the potential to accommodate crowdsourcing of 3D models in the future. Next, we describe different 3D data encoding strategies for the Web and evaluate several of these for their speed and efficiency in our open source prototype implementation of the proposed API. We conclude that none of the formats strike the right balance between the number of requests, decoding overhead and the compression achieved making the proposed flexible architecture even more compelling.

CR Categories: I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types;

Keywords: XML3D, 3D Repo, revision control, REST, CRUD

1 Introduction

Originally designed for sharing of text documents, the Web has become a graphical environment ever since the famous proposal for the `<IMG>` tag by Marc Andreessen in 1993 [Pilgrim 2010]. Since then, the software surrounding the *Hypertext Markup Language (HTML)* and the *Hypertext Transfer Protocol (HTTP)* has evolved to support mainly publishing, although an often overlooked property of HTTP is the definition of several verbs suitable for creating, updating and deleting online resources, not just requesting them. Wikis [Klobas 2006] originating in the mid '90 are one of just a few examples successfully exploiting the web as an editing platform.

The recent boom in 3D adoption thanks to the introduction of WebGL [Marrin 2011] and its wide-spread support in modern Web browser has produced further pressure on 3D content creation with the desire for even more detailed and more complex scenes being developed and accessed over the Internet. With the ever increasing interest in mobile devices, the limiting factors of bandwidth and latency are reintroduced as serious risks. What is more, the prevalent paradigm in the 3D domain is to use the Web as a publishing platform but not a development one. Formats such as X3D and VRML97 are designed to be updated in place at runtime but do not provide means of persistent modification preservation on their own.

Our intention is to build a scalable open platform for 3D content creation on the Web with the support for desktop and mobile WebGL-enabled browsers. The main aim is to provide seamless encoding where assets are independent of their fixed file formats, yet tracked over time. Such a platform would enable ubiquitous access to 3D assets in a form *most preferred* by the receiving client and the opportunity to perform modifications with a confidence that the old versions will not be lost or overridden. This is especially interesting as it would create the right basis for crowdsourcing of 3D models.

Recent advancements in NoSQL technology have enabled the development of 3D Repo, an open source non-linear versioning system for 3D assets by Doboš and Steed [2012b]. This framework is

built atop of a centralised NoSQL database MongoDB [Membrey et al. 2010] and provides functionality similar to source code versioning systems including branching and merging, although only through a raw database access using Binary JSON (BSON) [10gen, Inc. 2013] queries and a small number of client applications. Unfortunately, such a framework on it its own is too complex and too specific to provide the desired widespread access to 3D data.

XML3D by Sons et al. [2010], on the other hand, is an extension to HTML5 that describes interactive 3D graphics as part of a web page. Similarly to HTML, all XML3D elements belong to the *Document Object Model (DOM)* representation, and thus can easily be accessed and modified via JavaScript. Events can be attached to scene objects, e.g. using event attributes such as `onmouseover`. The similarity to HTML enables web developers to create dynamic 3D applications with little extra knowledge and due to its tight integration, powerful frameworks such as jQuery can also be applied.

REST *Representational State Transfer (REST)* [Fielding 2000] is a style of software architecture for the Web governing the behaviour of clients and servers in terms of requests for *resources* and the corresponding *responses*. In this style, a resource is any uniquely addressable piece of data (using the *uniform resource identifier (URI)* string) while its *representation* is a document which the data is returned in. A closely related concept of a persistent storage management, as is the case of a 3D repository, is formed by the functions *create*, *read*, *update* and *delete* (CRUD). When deploying a web-based architecture, a RESTful application programming interface (API) maps all of these methods to the individual HTTP 1.1 protocol verbs [Fielding et al. 1999] in order of `POST`, `GET`, `PUT` and `DELETE` respectively. In addition, it has to specify the base URI for the web service and the media types of supported representations. Therefore, the only required information for a client to interact with a server is the location of the resource and the intended action. Requesting a document representation makes this style of programming independent from the underlying storage format, a crucial component of our vision for ubiquitous access to 3D assets.

To achieve our goals, we propose a simple yet powerful API that combines the persistent versioning of 3D Repo with the ease of use of XML3D. This API enables client applications to request the desired encoding format of any resource making this a file format-independent means of access to 3D assets, see Fig. 1. In order to demonstrate the feasibility and advantages of such an approach, we have implemented several encoding strategies, namely *Extensible Markup Language (XML)*, *JavaScript Object Notation (JSON)*, *Binary JSON (BSON)*, *Sequential Image Geometry (SIG)*, *Open Compressed Triangle Mesh (OpenCTM)* and *ArrayBuffers* and evaluated them for their speed of delivery and encoding efficiency.

Contributions We begin by critically evaluating the deficiencies of the existing technologies and outline the benefits of the unification of XML3D and 3D Repo. Hence, our main contributions are:

1. A definition of a novel API that unifies XML3D and 3D Repo into a coherent RESTful platform that we call *XML3DRepo*.
2. A taxonomy of 3D data delivery strategies with a detailed description of example formats.
3. An open source implementation of various external types of 3D data delivery methods in XML3D loaded from 3D Repo. Demo server is available at `http://xml3drepo.org`.
4. Evaluation of these prototype implementations in terms of speed and efficiency and a discussion of the benefits and implications of such an approach especially in terms of the future work.

2 Related Work

Previous attempts at providing access to 3D assets have been successful to some degree. An ongoing *rest3d* initiative [Parisi and Arnaud 2011] proposes to define a REST interface shared by all online 3D resources. The suggested delivery formats are XML and JSON, although, currently considered outside of version control. Scheifer et al. [2010] demonstrated the benefits of a REST web service integration on a C++ scene graph system OpenSG. More recently, Olbrich [2012] applied XMLHttpRequests (XHR) to X3D server communication, most notably to preserve user annotations in a NoSQL database, while Schubotz and Harth [2012] devised a prototype server supporting `POST` and `GET` methods with XML3D rendering, however, without external referencing of resources. We follow in their footsteps and define a fully specified API with several example data encoding implementations where significant considerations were taken for the speed of data delivery on various platforms including mobile devices as well as versioning.

Sunglass [DeBiswas and Rao 2012], a recent MIT spin-off, provides a proprietary WebGL-based collaborative 3D modeling solution. Their paid servers can be accessed via a REST API in order to manage JSON mesh representations alongside of the original binary file formats. Version control in this system stores a linear history of binary snapshots without delta changes and with only a side-by-side visualization of revisions lacking support for differencing or merging. Autodesk's computer aided design (CAD) package AutoCAD also supports editing and sharing of CAD drawings via its online viewer and mobile apps [Autodesk 2012]. In contrast, an open source visualization system VisTrails [Bavoil et al. 2005] provides a broad support for versioning of scientific workflows. Despite the provenance history being represented as XML or stored in a relational database, the actual 3D files have to be managed locally. The spreadsheet-like workflows can be visualized using the Visualization Toolkit (VTK) and HTML renderings.

On the other hand, repositories such as Trimble 3D Warehouse (former Google Warehouse), TurboSquid or 3D Repository (3DR) by the Advanced Distributed Learning are large online libraries of 3D assets. Despite providing searchable Web interfaces to locate predefined file formats, only 3DR has a basic REST API accessible using XML and JSON [Advanced Distributed Learning 2011].

Many forms of distributed virtual worlds and games also provide some sort of networked 3D formats [Steed and Oliveira 2009]. Second Life, for example, relies on a standardized open protocol for sending and receiving 3D models as serialized XML via a REST API [Lentczner 2008]. In principle, this protocol could be repurposed for sharing of assets over the Internet, but in practice it is customised for a real-time application and not for a general-purpose data retrieval. Additionally, it is highly constrained to the types of data encodings by the platform it was designed on, hence it is not as flexible as other representations such as XML3D. In summary, most Web accessible 3D APIs support XML or JSON formats and many lack version control support altogether. Details regarding the alternative delivery encodings are described in §5.

3 Architecture Overview

The goal of our REST specification is to provide a transparent API that supports unified access to version controlled 3D resources over the Internet, but is independent from the underlying technologies and data storage. Relying on a RESTful architecture style has the effect of flattening the scene and revision graphs as the resources can now be queried individually using their unique URI or together as a collection of resources based on their common type, also referenced by a URI. For the purposes of our prototype implementation,

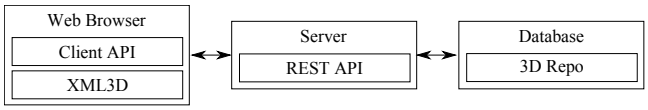

Figure 2: *High-level overview. Client connects to a server using our REST API. 3D content is dynamically fetched from the repository and delivered to the web browser as XML3D.*

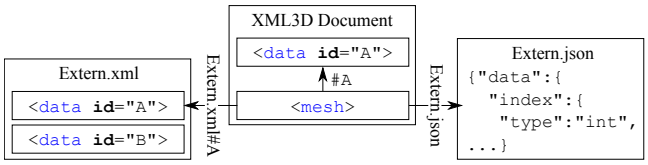

Figure 3: *Three ways of referencing resources in XML3D: Resources within external documents (Extern.xml#A), intra-document resources (#A) and entire documents as one resource (Extern.json). The mechanism—shown here for meshes—equally applies to other resources such as shaders, animations, etc.*

as shown in Fig. 2, XML3D, a declarative extension to HTML5, was selected on the client-side. Even though the rendering could be accomplished using other suitable means, XML3D provides many benefits over alternatives especially in terms of external references to resources. On the server-side, 3D Repo was used as it is specific to 3D assets and is based on a database rather than a file system.

3.1 XML3D

XML3D (`http://xml3d.org`) [Sons et al. 2010] is a lean, modern and generic scene graph that is based on and extends HTML5. One of its core components is the `<data>` element essentially grouping named and typed arrays, much like the buffer data structures of modern graphics APIs, e.g. vertex arrays. Additionally, it is possible to recursively reference other `<data>` elements from within the element itself. Fig. 3 shows how XML3D references can either point to resources in the same document, to external resources or to resources residing inside external documents using URI semantics. The data concept in combination with URI references provides a very fine-grained control over the composition of a scene in terms of reuse as well as organization across multiple resources. Such a data concept applies to `<mesh>` elements defining geometry in the scene, `<shader>` elements describing material properties and `<lightshader>` elements describing lights which are all just specialized `<data>` elements. Xflow [Klein et al. 2012] extends the XML3D data composition by a declarative dataflow component. Each `<data>` element can reference an operator that takes the entries of the data block as its input parameters and computes an output from them. Hence, Xflow transforms `<mesh>`, `<shader>` and `<lightshader>` into sinks of dataflow providing functionality for dynamic meshes, morphing, animations, etc.

A similar approach is X3DOM [Behr et al. 2009], a format based on X3D [Web3D Consortium 2011] which derives from VRML97. Thus many of the VRML97 concepts can be found in X3DOM which together with XML3D forms the evaluation platform of W3C Community Group “Declarative 3D for the Web Architecture”.

We have chosen XML3D, because it enables consistent handling of external resources, a feature that was important for the evaluation of our REST API. XML3D supports external references to arbitrary data containers, hence it is possible to realize all of the proposed delivery approaches without extending XML3D. The only required action is to implement a loader plug-in for each format that decodes the resources and maps them onto a collection of data entries. It is even possible to mix and match several different delivery formats to compose a single resource. Conversely, the *Inline* node (a VRML/X3D mechanism for external documents) allows only inclusion of complete subscenes. Since there is no way to access parts of a subscene, the *Inline* node cannot be used for a fine-grained referencing [Behr et al. 2012].

3.2 3D Repo

3D Repo (`http://3drepo.org`) [Doboš and Steed 2012b] is a version control framework specifically designed for 3D assets. The repository is based on a NoSQL database MongoDB [Membrey et al. 2010] which stores individual components of 3D scenes with their associated revision histories. Due to a database being the persistent storage provider, 3D Repo avoids constraints of a file-based system, offers extensive querying functionality as well as implicit access control. Its *3D Diff* tool [Doboš and Steed 2012a] enables two and three-way differencing and merging of concurrent edits including detection of implicit and explicit conflicts based on the intersections of component bounding boxes. Models in 3D Repo are loaded using the Open Asset Import Library (Assimp) [Schulze et al. 2012] which converts the most common 3D file formats into a unified in-memory representation. Their deltas are stored as two collections (tables), one for all the scene graph components and one for all the components of a revision history, each represented as a directed acyclic graph (DAG) with a single root node. Every graph node, regardless of it belonging to a scene or a revision, is expected to specify its *unique identifier (UID)*, a functional requirement of the database, as well as a *shared ID (SID)* which is shared amongst multiple documents. In the context of a scene graph, the SID is shared by all the revisions of the same scene graph node, while in the context of a revision history, the SID is shared by a single branch, where the SID of all zeros is reserved for the trunk/master.

4 REST API

The API defined in this section provides access to 3D resources stored in a version controlled environment. Its architecture can be summarized as a two-way URI encoding where in its core lies a combination of `id` and `type` variables, the order of which determines the behaviour of the interface. In general, to address a collection of resources the `/:id/:type` ordering is used, while the `/:type/:id` combination addresses a single resource. Depending on the context, the ID is either the UID or the SID of a resource (see §3.2) and the type is a family of resources such as ‘meshes’, ‘textures’, etc. and even ‘revisions’. Each resource or a collection of resources can be requested in various encodings. This successfully decouples the storage implementation from the querying interface.

POST Posting data to a server is used for creating new repositories as well as committing revisions and performing merges.

`/xml3drepo` Creates a new empty repository with a unique name if not present. Hence, a name string input is expected.

`/xml3drepo/:name` Commits a new head revision to the trunk/master of the repository identified by its unique name. Expected input is the data to be committed and the new revision.

`/xml3drepo/:name/:id` Commits a new revision, but to a branch identified by its shared ID. If a branch does not exist, it is created. When merging, the posted revision document specifies the revisions to be merged as its parents in the DAG hierarchy of a revision history. Its SID is the one of the branch which lives on after a successful merge operation.

GET Retrieving data is the most commonly used feature of any such an API. Therefore, it has to be flexible enough to provide means of addressing collections of resources, single resources and even individual attributes (sub-parts) of those resources.

`/xml3drepo` Returns a collection of all available 3D repositories, i.e. a list of databases containing 3D scenes and revisions.
`/xml3drepo/:name` Returns the head revision of a trunk/master, i.e. all components of a scene identified by its unique name.
`/xml3drepo/:name/:id` Returns a scene similarly to the trunk/master's head, but from a revision identified by its unique ID. If a shared ID is requested, returns the head of a branch.
`/xml3drepo/:name/:id/:type` Returns a collection of resources matching the requested type that belong to a revision identified by its UID or SID. If the type is 'revisions', returns a revision resource describing the author, commit message, etc.
`/xml3drepo/:name/:type` Returns a collection of resources matching the requested type from the trunk/master's head. If the type is 'revisions', returns the entire revision graph.
`/xml3drepo/:name/:type/:id` Returns a resource matching the requested type which can belong to multiple revisions identified by UID or a collection of revisions of the same scene graph or revision history component identified by SID.
`/xml3drepo/:name/:type/:id/:attribute` Rather than the entire resource, returns a single attributed from it. This is useful for non-standard encoding formats such as Sequential Image Geometry which is composited from 8-bit sections.

HEAD Same as `GET`, however, without the body of the data. This is used for accessing the DAG structure of a scene graph or a revision history without the actual contents such as vertices or textures.

PUT Idempotence of `PUT` guarantees that sending a request multiple times has the same effect as sending it only once. However, once in a versioned repository, the resource can never change, only become superseded by a newer revision of itself. Therefore, updating a resource via `PUT` commits a new revision to the database, although, posting all the new resources in a single commit is preferred. Furthermore, `PUT` can be utilised for requesting changes in the state of the repository and resources, e.g. locking. Nevertheless, most existing APIs listed in §2 do not support `PUT` requests.

`/xml3drepo/:name/:id/:type` Commits a new revision of resources identified by the type to a branch identified by its shared ID. If the type is 'locks', acquires a lock on the branch.
`/xml3drepo/:name/:type/:id` Commits a new revision of a resource identified by its UID or SID to the trunk/master. If the UID is not at its head revision, the request fails with a conflict. If the type is 'locks', acquires a lock on the resource.

DELETE By the nature of versioning, deleting data does not actually remove it from the database, merely commits a new revision where it is marked as deleted. Thus, the data can still be accessed via older revisions if needed. Similarly to `PUT`, it is preferred to commit all the deletes in a single revision rather than one by one.

`/xml3drepo/:name/` Force remove a repository identified by its unique name from the database. This operation cannot be reverted as the corresponding collections (tables) are dropped.
`/xml3drepo/:name/:id/:type` Commits a new revision to a branch identified by its shared ID where all the resources identified by the type are marked as deleted. If the type is 'locks', releases the lock from a branch.
`/xml3drepo/:name/:type/:id` Commits a new revision where the resource identified by its UID or SID is marked as deleted. If the type is 'locks', releases the lock from a resource.

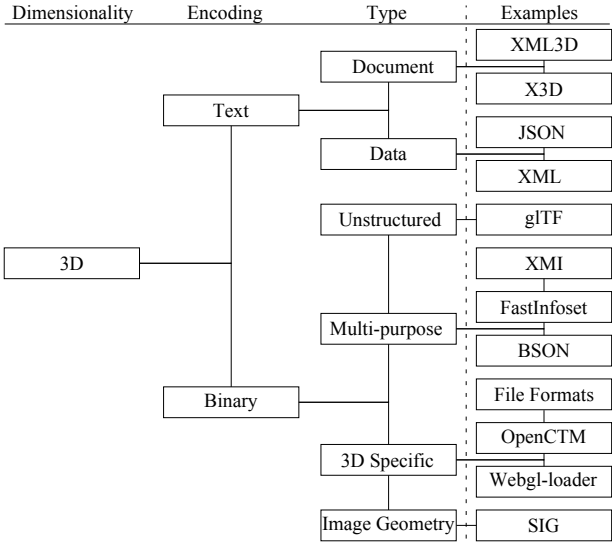

Figure 4: *Taxonomy of 3D data representations for the web.*

To overcome the missing `DELETE` functionality of some web browsers, it is customary to generate a hidden field value 'delete' and use `POST` or `PUT` instead [Richardson and Ruby 2007]. When receiving a form with this parameter, the server overrides the actual HTTP request and fulfils the desired action.

Status Codes Various HTTP/1.1 status codes can be received when using the API. The most important ones are the `201 Created` when successfully committed, `400 Bad Request` when invalid syntax is used, `406 Not Acceptable` when a requested encoding cannot be achieved and `409 Conflict` when committing changes that are in conflict with the head revision for which a list of conflicting entities is returned ([Fielding et al. 1999] p. 66).

5 Data Encoding

In the HTTP protocol, it is the client who requests the most appropriate representation of a resource depending on its intended application. For example, there exist several encoding formats for web pages, e.g. HTML, XHTML, etc., each providing its own set of advantages. For 3D, the distinction of encodings is even more important as very different component types and file sizes make up the structure of a 3D scene. As recently demonstrated by Jung et al. [2013], by relying on a quantization it is even possible to render a 91M polygon model in a web browser. This section, therefore, discusses various representations that are evaluated for 3D data delivery, their benefits and drawbacks and how the support within XML3D is realized. Fig. 4 shows a taxonomy of possibilities.

5.1 Text Formats

Text-based formats have the advantage of being human-readable, although, string representations are usually larger than binary and need to be parsed in order to be utilised. In general, we distinguish document-based and pure data-based formats of text encoding.

Documents Document-based formats typically represent the whole scene, maybe even with some run-time information. An obvious approach is to encode geometry, shader and animation resources directly in the document. This has the advantage of resources being available to the DOM API at parse-time. However,

the data is in a string format resulting in longer processing and interaction being only possible once the parsing has finished. Including resources as an attribute or a character component of XML can be found in COLLADA [Khronos Group 2008] and X3D. In the context of XML3D, a document is the HTML page containing one or more XML3D scenes. There, the resources can be encoded either internally or referenced externally providing a URI. Another approach is to include only those resources needed in the DOM for modification at runtime and reference all static ones externally.

Data Various formats can be used to encode external resources that are referenced from within a document. Text-based encodings can either contain a single resource or a collection of resources, e.g. all shaders of a scene. The browser then provides means to load external resources during runtime; XMLHttpRequest (XHR), the main component of the Asynchronous JavaScript and XML (AJAX) architecture, defines an API to transfer data between the client and a server. However, XHR is not restricted to XML which together with JSON is the most commonly used external format mainly because Web browsers expose native parsing capabilities for both. JSON gained popularity especially due to its missing end tags that make it smaller for documents with lots of structure but a few data entries. For 3D, the difference is negligible as demonstrated in Tab. 1. However, JSON does not offer any natural way of addressing its elements. In contrast, browsers provide multiple means of accessing elements in DOM—the in-memory representation of an XML document. These include CSS Selectors and XPath. A common approach to address the elements of HTML is to use the URL fragment which refers to the element as its *id* attribute. This is why XML, unlike JSON, can represent not just individual resources but also their collections. JSON and XML compress well with deflate and gzip compressions that are available in all major Web browsers trading-off the decoding time in favor of bandwidth. Nevertheless, both formats suffer from issues that come with any generic string representation. For the WebGL API to access such data, the strings have to be deserialized into Typed Arrays [Khronos Group 2012].

5.2 Binary Formats

Unstructured Buffers Binary data can be transmitted via XHR as ArrayBuffer [Khronos Group 2012], a buffer introduced with Typed Arrays that is essentially a byte array. It is possible to generate a specific view on it, e.g. interpret every four bytes as one float entry and thus derive a `Float32Array` from it, similarly to the 3D Repo storage. Obviously, such a buffer has no structure, thus one approach is to request a buffer per vertex attribute as proposed in [Behr et al. 2012] as Binary Geometry and in glTF [Robinet et al. 2012]. The drawback is that these come with a large amount of XHR requests which may lead to a reduced performance in high-latency networks. This can be moderated to some degree by interleaving multiple vertex attributes into a single ArrayBuffer.

Multi-purpose Binary Formats A second approach for structured data is to use a generic binary format. There are several competing standards for binary encoded XML, for instance *XML Metadata Interchange (XMI)* [Object Management Group 2011] and *FastInfoset (FI)* [Telecom. Standardization Sector 2005]. Most binary XML formats use dictionary compression for element and attribute names and a binary representation for XML data types. XMI adds a deflate compression to the data. Binary encoding of X3D is based on FI and exploits its capabilities of referencing predefined dictionaries and custom compression methods that make FI a hybrid between a generic XML encoding and a domain-specific compressed format. By exploiting these capabilities, very high compression rates within a generic format [Stocker and Schickel 2011] can be achieved. Unfortunately, no readily available JavaScript implementation of FI exists, hence it is not used in our evaluation.

A binary representation of JSON is BSON [10gen, Inc. 2013]. It has no dictionary compression but provides means of encoding the structure and data types in a binary way. This enables efficient parsing and traversal of such documents. Additionally, BSON contains extensions that support data types that are not part of the JSON specification. One of these is a binary blob for which it is the application's responsibility to deserialize it. BSON has a special place in XML3DRepo as it is the internal format of MongoDB.

Binary 3D Formats A third class of formats comes with its own domain-specific schema to represent structured 3D data in a binary form. Many open and proprietary file formats are available, some of which also apply compression exploiting the knowledge of the data properties. Two formats that are of particular interest to 3D web delivery, mainly because a JavaScript decoder is available, are OpenCTM [Geelnard 2009] and Webgl-loader [Chun 2013]. Both use classical compression schemas such as Delta and ZigZag encodings. Webgl-loader additionally exploits the variable-length encoding of UTF-8 to capture values with 1 to 3 bytes.

In general, structured binary formats suffer from the need to decode binary data into JavaScript. Decoding is required, because the internal representation is not compatible with Typed Arrays. The time required for decoding is significant in mobile devices and other clients with slower JavaScript engines. One strategy to overcome the blocking of the UI is to shift the decoding into a Worker thread.

Geometry in Images Encoding geometry in images is a special kind of binary format. This approach is especially useful here as there is no need to modify the data in JavaScript; The image gets decoded in the browser core and can be directly uploaded to the GPU where it serves as a data buffer. Sequential Image Geometry (SIG) [Behr et al. 2012] extends this approach as vertex arrays get split into chunks of 8-bits of different relevance and are distributed into a sequence of images. Such an approach also supports quantization by omitting images with less relevant bits. Progressive loading can be achieved if the images arrive in the correct order. However, the main drawback of SIG is that it requires data fetching from images in a vertex shader, a functionality not yet supported on many mobile devices. Even support for four texture units would already be occupied by position coordinates with 16-bits (two images), normals with 8-bits (one image) and texture coordinates with 8-bits (one image). If none or too few texture units are available, it is still possible to read the image in JavaScript and create Typed Arrays from it. Obviously, images are not designed to represent 3D data and thus a Portable Network Graphics (PNG) format, for example, does not achieve high compression rates for 3D [Behr et al. 2012].

6 Implementation

As shown in Fig. 5, a prototype implementation of the proposed REST API was developed using the *node.js* [Dahl 2009] server framework which is gaining in popularity especially due to its non-blocking event-driven JavaScript execution. Middleware *Express* and templating engine *Jade*, both available via node's internal package manager *npm*, were used to provide routing mechanism and HTML rendering respectively. In addition, our JavaScript port of the 3D Repo interface closely follows its C++ counterpart so that both UID and SID (see §3.2) are realised using the *universally unique identifier (uuid)* as defined by ITU-T [2008]. Therefore, any resource can be directly addressed via its UID, e.g. `/xml3drepo/UT4_Baeza/meshes/4e992f02-3777-41ad-b777-91ad377791ad.xml`, although strictly speaking,

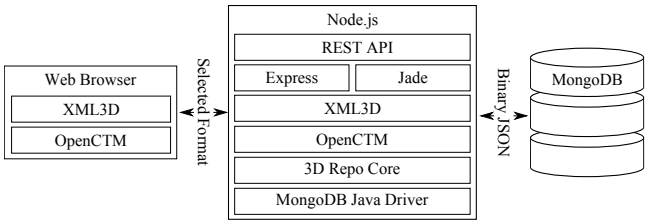

Figure 5: *System components. Remote client connects to a node.js server using our REST API. Dynamic web pages are coded using the Jade templating engine and delivered via Express middleware. While MongoDB (hence 3D Repo) supports only BSON responses, our REST API returns 3D data in a variety of encodings.*

the format extension '.xml' can be omitted as the HTTP Accept header would specify the desired format in the request [Fielding et al. 1999]. Nevertheless, it can be present for convenience or specified as a query parameter, e.g. `?meshformat=xml`. If a conflicting header and a format are requested, the header gets preference.

However, revisions in the original 3D Repo were assigned an incremental unsigned integer. In XML3DRepo, these integers were replaced with uuids to make the system ready for distributed access. In the future, users could run a local instance of the server application to record offline modifications and synchronise to a centralised repository where, just like in Git, the integration happens remotely.

XML3D Support for Delivery Formats There are currently two implementations of XML3D: A native implementation based on Google's Chrome web browser and a Polyfill implementation based on WebGL and JavaScript [Sons et al. 2013]. The latter was used for 3D Repo integration as it runs in most WebGL-enabled browsers including some mobile variants and offers a plug-in registration for external file format loaders. These can register themselves for a specific *Internet media type (MIME)*, e.g. `application/json` for JSON. When multiple loaders are registered for the same type, each is queried for support of the downloaded data block using its `hasSupportFor` method and the first one in order of registration is used. Externalizing the resources in XML3D offers progressive loading which frees the rendering and moves the control of the visualization process to the user level. If progressive loading is not desired, the content can be hidden until fully loaded.

Such a flexible approach allows us to implement XML, JSON, BSON, SIG, OpenCTM and ArrayBuffers for delivery formats. In our system, XML and JSON rely on the native parsing capabilities of the web browsers, while BSON requires a custom loader for its deserialization. OpenCTM comes with a JavaScript decoder [Geelnard 2009] that was wrapped into `xml3d.js` loader. SIG, however, requires generation of an implicit vertex array buffer that acts as a set of texture coordinates for the input textures. These coordinates depend on the number of vertices in a mesh as well as the resolution of the image. Xflow can be used to create the texture coordinates from these parameters, hence we deliver a mesh node that references an Xflow graph that in turn references the images. All Xflow graphs were clustered into a single external document so that only one additional HTTP request is generated. With the ability to query individual attributes from our REST API (see `GET` definition in §4), it is possible to retrieve individual images per 8-bit sections of the vertex and normal arrays as required. ArrayBuffers were implemented similarly to Binary Geometry [Behr et al. 2012].

Caching Caching is a vital part of any server-side system. In order to prevent too many open connections from the node.js application to MongoDB, the connection to each database is cached once opened and via a callback deleted from the cache when dropped. This is especially important as hundreds of requests are expected for any scene. Furthermore, given that the 3D data is version controlled, the resources can never change. Even if updated or deleted, they are still accessible using their unique ID. Therefore, this UID together with the encoding format can act as a caching key.

7 Evaluation and Discussion

Performance of various 3D delivery formats was measured in order to provide indicative values for both the cumulative CPU decoding time and the overall download time in a variety of model sizes and web browsers. The 3D models used in our experiments, see Fig. 6, are readily available game levels. These are real-world examples of 3D data that is commonly downloaded over the Internet. Textures were excluded as the overhead for each model would be the same regardless of the mesh delivery format. Data for these models were collected using the built-in Developer Tools in Chrome 25.0.1364.172, Firebug add-on [Hewitt 2006] in Firefox 19.02 and the built-in Dragonfly tool in Opera 12.14, all with caching disabled. Safari 5 and Internet Explorer 10 were not tested as they do not support WebGL on Windows yet. An XPC Shuttle SX58H7 with Intel Core i7-920 at 2.67GHz with 4GB RAM running Windows 7 was used as a client while Intel Xeon at 2.67GHz with 2GB RAM running CentOS 6.2 and node.js v0.8.19 acted as a remote server over a local network. To mitigate the effects of data handling in between node.js and DB, all experiments were done with the MongoDB's C++ BSON parser/serializer instead of JavaScript.

Despite using only 8-bits per normal array in our implementation, SIG 32-bit still requires 4 textures for the vertex array definition alone. Although being rendered without normals, we include the measurements for completeness. Tab. 1 shows the different compression rates achieved by individual encoding formats as well as the number of requests made by the browsers. As expected, the difference in the sizes of XML and JSON is very small due to the data being mostly made up of large arrays, so the contribution of end tags in XML is diminished. However, BSON is noticeably larger due to its explicit array indices. This is why 3D Repo relies on unstructured binary entries for its internal representation of meshes.

To suppress fluctuations in the network and CPU performance, Fig. 7 shows median values for uncompressed encodings measured across 5 trials. The overall download times, depicted as bars, are composed of the time required to load the DOM definition and the external resources. Yellow dots show the cumulative CPU decoding overhead per format. Compressed timings are very similar and available as supplemental materials. SIG decodes in native code, hence achieves zero CPU overhead, however, the number of HTTP requests increases with the precision, significantly hampering the download time despite the smaller overall amount of data being transmitted. The lower-bit quantizations cause visual degradation especially in the areas of high frequency details. Furthermore, the SIG format did not work in Opera. When compiling a vertex shader with texture fetches it fails due to a DirectX-specific error despite reporting available texture units. In contrast, the advantage of OpenCTM compression becomes distinctive with larger sizes, although, Firefox would consume more CPU to decode this format.

Clearly, the best format can only be selected by the properties of the network connection and client performance. Since the client performance is not a bottleneck nowadays, the network is the key factor for selecting the most suitable delivery format. High latency requires less requests and low bandwidth requires more compression. With XML3DRepo the desired delivery format can be dynamically chosen by the client making this a very flexible setup.

Figure 6: *Game levels used in our experiments. From left to right: UT4 Baeza with over 93k vertices, 31k faces and 196 meshes, UT4 Paris v2 with over 129k vertices, 43k faces and 291 meshes and UT4 Intermodal Beta (see Fig. 1) with over 170k vertices, 56k faces and 482 meshes. Textures were excluded. Models downloaded from* `http://www.snipersgaulois.com/downloads.More.php`.

Format	UT4 Baeza			UT4 Paris v2			UT4 Intermodal Beta		
	Raw [MB]	Gzip [MB]	Requests	Raw [MB]	Gzip [MB]	Requests	Raw [MB]	Gzip [MB]	Requests
XML	8.9	1.4	408	7.7	1.4	598	11.9	2.4	980
JSON	8.8	1.3	408	7.6	1.4	598	11.9	2.3	980
BSON	10.4	2.8	408	10.9	3.0	598	15.7	4.6	980
SIG 8-bit	1.7	1.0	800	2.2	1.2	1180	2.5	1.7	1944
SIG 16-bit	2.3	1.6	1192	2.5	1.6	1762	3.2	2.4	2908
SIG 24-bit	2.7	1.9	1584	2.9	2.1	2322	3.9	3.3	3872
SIG 32-bit	3.3	2.5	1976	3.6	2.9	2926	4.8	4.0	4836
OpenCTM	1.6	0.8	408	1.7	0.9	598	2.1	1.3	980
ArrayBuffers	3.7	1.2	1192	4.1	2.7	1762	5.9	4.0	2908

Table 1: *Overall download size of uncompressed (Raw) vs. compressed (Gzip) encodings in MB and the total number of requests required.*

8 Conclusions

We have presented a novel REST API for a consistent online addressing of version controlled 3D assets. Despite the XML3DRepo name, the repository itself stores only a unified scene graph representation of common 3D formats encoded as Binary JSON that is independent of XML3D formatting. However, our client/server implementation demonstrates that the resource-based approach supports delivery of requested assets in a variety of encodings several of which were measured on geometry data and materials.

We have shown that XML3D, due to its consistent approach to external resources, works very well with the proposed API and offers transparent use of various representations. Thus, all delivery aspects can be delegated to the XML3DRepo framework and the application logic can start at a high level of abstraction. In this paper, it was demonstrated that the approach proposed here "holds" for six considerably different delivery formats. The advantages over a simple file server are in the ability to transparently query entire scenes, individual components and even their attributes in a representation that is independent of the data storage.

Our results also show that there is currently no single delivery format that fits all devices, networks and applications. None of the measured formats provides a good trade-off in between the number of requests, the required decoding time and its compression ratio. For instance, OpenCTM offers considerable size reduction but comes with a slow decoding and a compression schema that is not applicable to all sorts of 3D resources (e.g. animations). On the other hand, generic formats such as JSON and XML are likely to be the most suitable for many 3D resources, but require parsing and offer only generic compression in HTTP. Direct use of ArrayBuffers and Image Geometry suffers from a large amount of necessary requests. Additionally, SIG is limited by the number of applicable textures on the rendering device. With these restrictions in place, a delivery framework such as XML3DRepo becomes even more important as it provides means of adapting resources to the network and device capabilities as well as the application needs.

Future Work Our intention is to continue developing this new architecture into a truly scalable open source platform at `http://xml3drepo.org`. We believe that it is possible to convert the existing 3D Repo into a fully distributed system where clients will connect to a gateway proxy providing seamless access to version controlled 3D data from different locations. We plan to implement a dynamic system based on the capabilities of the receiving client and the properties of the network connection. Such a system would automatically establish the best format for data delivery based on predetermined heuristics. Furthermore, we plan to add more functionality to the API including spatial and semantic queries, proximity-based data retrieval with camera position queries as well as user authentication and general search capabilities. In addition, a thin layer above this structure could enable collaborative editing of 3D assets directly in web browsers.

In the future, a new format will be required that would be generic enough to represent all kinds of 3D resources, yet efficient in its representation and with a reasonable amount of HTTP requests. One option is to base this format on Typed Arrays and provide a domain-specific compression using Xflow, although attempts trying to solve the interoperability issues already exist, e.g. [Berthelot et al. 2011] or even COLLADA and CAD STEP (ISO 10303) formats. Similarly to the encoding and compression of geometry data when downloading the resources, it would also be possible to compress the data before committing revisions back to the repository. On one hand, it might be possible to compress the data in a web browser. On the other, given the assumption that there are always going to be significantly many more read than write requests, it might be more suitable to upload raw 3D models for a more powerful server to process. Both, however appealing, require further investigation and we leave this open for future work.

Acknowledgements This research was partly sponsored by the UK EPSRC-funded EngD. Centre in Virtual Environments, Imaging and Visualisation (EP/G037159/1), the EU projects VERVE and FI-CONTENT, the Intel Visual Computing Institute, and Arup.

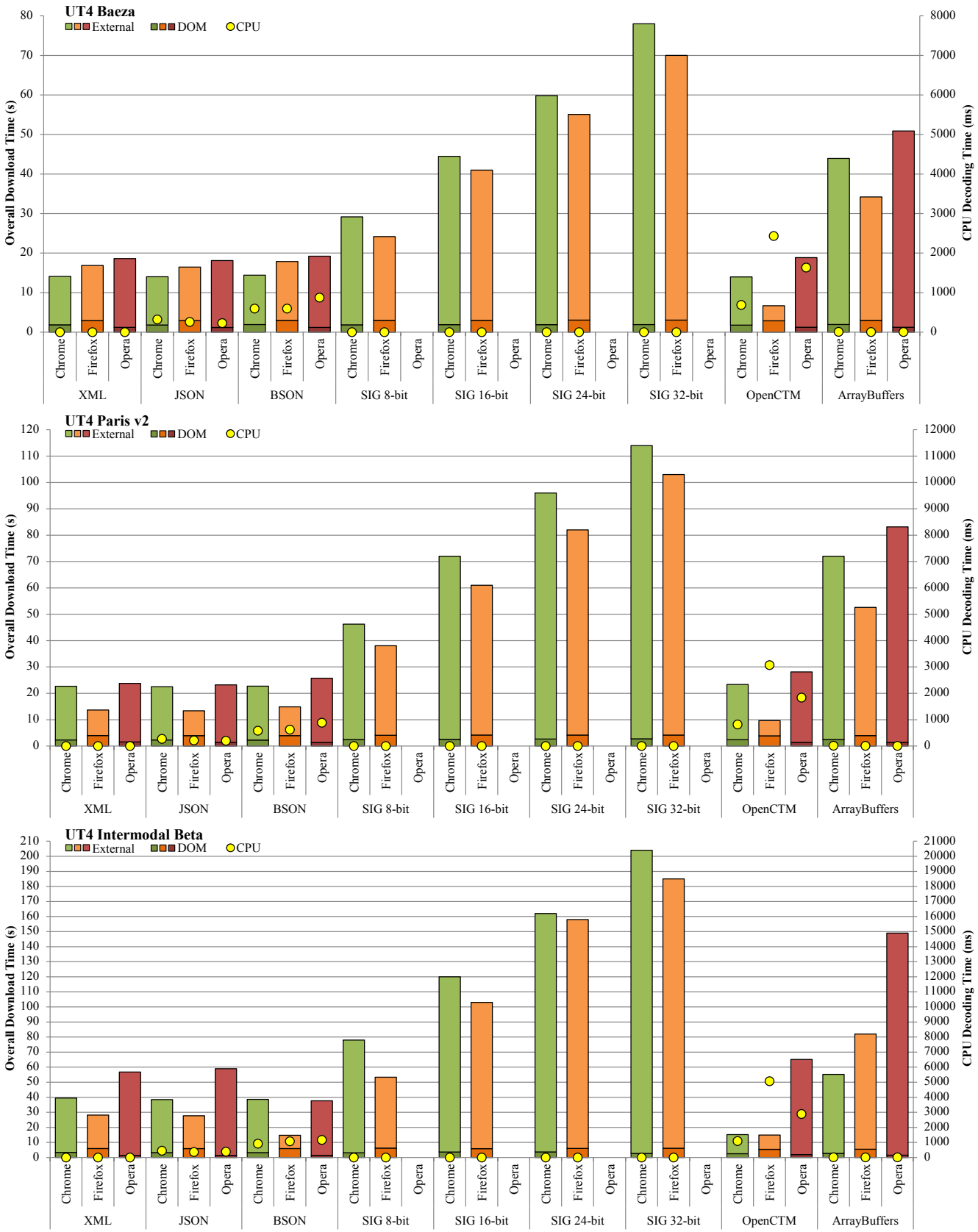

Figure 7: *Median values from 5 trials of 3 game levels. Overall download time (left Y-axis) consists of a DOM definition and the external references processing while the CPU time (ten folds less, right Y-axis) defines the amount of cumulative CPU milliseconds required to decode the encoding format. Measured on XPC Shuttle SX58H7 with Intel Core i7-920 CPU at 2.67GHz with 4GB RAM running Windows 7.*

References

10GEN, INC., 2013. BSON–Binary JSON specification. URL: http://bsonspec.org/.

ADVANCED DISTRIBUTED LEARNING, 2011. 3d repository api documentation, September. URL: https://github.com/adlnet/3D-Repository/wiki/API-Documentation.

AUTODESK, 2012. AutoCAD WS. https://www.autocadws.com.

BAVOIL, L., CALLAHAN, S. P., CROSSNO, P. J., FREIRE, J., AND VO, H. T. 2005. Vistrails: Enabling interactive multiple-view visualizations. In *IEEE Visualization 2005*, 135–142.

BEHR, J., ESCHLER, P., JUNG, Y., AND ZÖLLNER, M. 2009. X3DOM: a DOM-based HTML5/X3D integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, ACM, NYC, NY, USA, Web3D '09, 127–135.

BEHR, J., JUNG, Y., FRANKE, T., AND STURM, T. 2012. Using Images and Explicit Binary Container for Efficient and Incremental Delivery of Declarative 3D Scenes on the Web. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, NYC, NY, USA, Web3D '12, 17–25.

BERTHELOT, R. B., ROYAN, J., DUVAL, T., AND ARNALDI, B. 2011. Scene graph adapter: an efficient architecture to improve interoperability between 3d formats and 3d applications engines. In *Proceedings of the 16th International Conference on 3D Web Technology*, ACM, NYC, NY, USA, Web3D '11, 21–29.

CHUN, W., 2013. http://code.google.com/p/webgl-loader.

DAHL, R., 2009. node.js. URL: http://nodejs.org.

DEBISWAS, K., AND RAO, N., 2012. Sunglass. http://sunglass.io.

DOBOŠ, J., AND STEED, A. 2012. 3D Diff: an interactive approach to mesh differencing and conflict resolution. In *SIGGRAPH Asia 2012 Technical Briefs*, ACM, SA '12, 20:1–20:4.

DOBOŠ, J., AND STEED, A. 2012. 3D revision control framework. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, NYC, NY, USA, Web3D '12, 121–129.

FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T., 1999. Hypertext Transfer Protocol – HTTP/1.1.

FIELDING, R. T. 2000. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine. AAI9980887.

GEELNARD, M., 2009. Open compressed triangle mesh format. URL: http://openctm.sourceforge.net/, December.

HEWITT, J., 2006. Firebug. URL: http://getfirebug.com.

JUNG, Y., LIMPER, M., HERZIG, P., SCHWENK, K., AND BEHR, J. 2013. Fast and efficient vertex data representations for the web. In *Proceedings of the International Conference on Computer Graphics Theory*, GRAPP '13, 601–606.

KHRONOS GROUP, 2008. COLLADA - 3D Asset Exchange Schema. URL: http://www.khronos.org/collada/, March.

KHRONOS GROUP, 2012. Typed Array Specification. URL: https://www.khronos.org/registry/typedarray/specs/latest/.

KLEIN, F., SONS, K., RUBINSTEIN, D., BYELOZYOROV, S., JOHN, S., AND SLUSALLEK, P. 2012. Xflow - Declarative Data Processing for the Web. In *Proceedings of the 17th International Conference on Web 3D Technology*, ACM, Web3D '12, 37–45.

KLOBAS, J. E. 2006. *Wikis: Tools for Information Work and Collaboration (Information Professional)*. Chandos Publishing (Oxford) Ltd, June. ISBN-10: 1843341786.

LENTCZNER, M., 2008. Second life grid open grid protocol. URL: http://wiki.secondlife.com/wiki/SLGOGP_Draft_1.

MARRIN, C., 2011. Webgl specification v 1.0, February. URL: https://www.khronos.org/registry/webgl/specs/1.0.

MEMBREY, P., PLUGGE, E., AND HAWKINS, T. 2010. *The Definitive Guide to MongoDB: The NoSQL Database for Cloud & Desktop Computing*, first ed. APRESS ACADEMIC.

OBJECT MANAGEMENT GROUP, 2011. Mof 2 xmi mapping (xmi) v2.4.1, Aug. URL: http://www.omg.org/spec/XMI/.

OLBRICH, M. 2012. Accessing http interfaces within x3d script nodes. In *Proceedings of the 17th International Conference on 3D Web Technology*, ACM, NY, USA, Web3D '12, 139–142.

PARISI, T., AND ARNAUD, R., 2011. 3D REST 3D specification v0.2, April. URL: http://rest3d.org.

PILGRIM, M. 2010. *HTML5: Up and Running*. O'Reilly Media.

RICHARDSON, L., AND RUBY, S. 2007. *Restful web services*, first ed. O'Reilly Media.

ROBINET, F., PARISI, T., AND OZZI, P., 2012. glTF. URL: https://github.com/KhronosGroup/collada2json/wiki/glTF.

SCHIEFER, A., BERNDT, R., ULLRICH, T., SETTGAST, V., AND FELLNER, D. W. 2010. Service-oriented scene graph manipulation. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, NYC, NY, USA, Web3D '10, 55–62.

SCHUBOTZ, R., AND HARTH, A. 2012. Towards networked linked data-driven web3d applications. In *Dec3D*, CEUR-WS.org, J. Behr, D. P. Brutzman, I. Herman, J. Jankowski, and K. Sons, Eds., vol. 869 of *CEUR Workshop Proceedings*.

SCHULZE, T., GESSLER, A., KULLING, K., NADLINGER, D., KLEIN, J., SIBLY, M., AND GUBISCH, M., 2012. Open asset import library v3.0. http://assimp.sourceforge.net.

SONS, K., KLEIN, F., RUBINSTEIN, D., BYELOZYOROV, S., AND SLUSALLEK, P. 2010. XML3D: Interactive 3D Graphics for the Web. In *Proceedings of the 15th International Conference on Web 3D Technology*, ACM, Web3D '10, 175–184.

SONS, K., SCHLINKMANN, C., KLEIN, F., RUBINSTEIN, D., AND SLUSALLEK, P. 2013. xml3d.js: Architecture of a Polyfill Implementation of XML3D. In *6th Workshop on Software Engineering and Architectures for Realtime Interactive Systems*.

STEED, A., AND OLIVEIRA, M. 2009. *Networked Graphics: Building Networked Games and Virtual Environments*. Elsevier.

STOCKER, H., AND SCHICKEL, P. 2011. X3D binary encoding results for free viewpoint networked distribution and synchronization. In *Proceedings of the 16th International Conference on 3D Web Technology*, ACM, NYC, NY, USA, Web3D '11, 67–70.

TELECOM. STANDARDIZATION SECTOR. 2005. Information technology - Generic applications of ASN.1: Fast infoset. Rec, ITU, May. X.891, ISO/IEC 24824-1:2007.

TELECOM. STANDARDIZATION SECTOR. 2008. Generation and registration of Universally Unique Identifiers (UUIDs). Rec, ITU, Aug. X.667, ISO/IEC 9834-8.

WEB3D CONSORTIUM, 2011. Extensible 3D (X3D). ISO/IEC 19775/19776/19777, URL: http://web3d.org/x3d/specifications.