Publishing and Consuming 3D Content on the Web: A Survey

Marco Potenziani[1], Marco Callieri[2], Matteo Dellepiane[3] and Roberto Scopigno[4]

[1] *Visual Computing Lab, ISTI CNR; marco.potenziani@isti.cnr.it*
[2] *Visual Computing Lab, ISTI CNR; marco.callieri@isti.cnr.it*
[3] *Visual Computing Lab, ISTI CNR; matteo.dellepiane@isti.cnr.it*
[4] *Visual Computing Lab, ISTI CNR; roberto.scopigno@isti.cnr.it*

ABSTRACT

Three-dimensional content is becoming an important component of the World Wide Web environment. From the advent of WebGL to the present, a wide number of solutions have been developed (including libraries, middleware, and applications), encouraging the establishment of 3D data as online media of practical use. The fast development of 3D technologies and related web-based resources makes it difficult to identify and properly understand the current trends and open issues. Starting from these premises, this survey analyzes the state of the art of 3D web publishing, reviews the possibilities provided by the major current approaches, proposes a categorization of the features supported by existing solutions, and cross-maps these with the requirements of a few main application domains. The results of this analysis should help in defining the technical characteristics needed to build efficient and effective 3D data presentation, taking into account the application contexts.

Marco Potenziani, Marco Callieri, Matteo Dellepiane and Roberto Scopigno (2018), "Publishing and Consuming 3D Content on the Web: A Survey", Foundations and Trends® in Computer Graphics and Vision: Vol. 10, No. 4, pp 244–333. DOI: 10.1561/0600000083.

The version of record is available at: http://dx.doi.org/10.1561/0600000083

1

Introduction

Three-dimensional (3D) data has evolved from being merely specialized content, used just by a small community of professionals, to a completely integrated web medium, now reaching a reasonable maturity level. Although the technological foundations needed to enable this new medium to bloom have been available for a few years, users' perceptions have changed only recently and 3D web content has now started to be able to reach the wider public. In this evolutionary process, a key role was played by the democratization of 3D content creation (the availability of low-cost 3D scanning devices, improvement of 3D-from-images/structure-from-motion approaches, the consolidation of manual modeling systems) and the introduction of a series of game-changing contributions addressed to a wider range of target users (3D printing applications, 3D viewer and editing systems embedded in common operating systems, etc.).

Nowadays, these new trends are pushing 3D content toward an unexplored world, where data management, user interactions, and cross-media integration are open issues still to be solved. Obviously the novel ecosystem we are envisioning is part of the web (a democratic space "par excellence"), and in recent years has been the subject of renewed

attention concerning the integration of three-dimensional content and the development of resources specifically aimed at this.

However, despite the increased interest in recent years, the first attempts to bring 3D content online date back a long time. Indeed, web developers and 3D professionals understood very quickly the potential relevance of opening the web to 3D data, so that 3D should not stay trapped in standalone applications. A few months after the release of the first multimedia browser [able to manage just text and images; 163], Raggett [157] presented his vision for a platform-independent 3D standard for the web by proposing the Virtual Reality Modeling Language (VRML). The Web3D denomination emerged immediately after.

Unfortunately, such a prompt start was not followed by the same pace in the development of practical and consistent solutions, and the path toward an effective Web3D resulted in a long and winding process. Some major pioneering landmarks were the Macromedia Flash plug-in [44]—released in 1996, it was the direct ancestor of Adobe Flash and probably the first approach to handling fully interactive multimedia content online—and the Apple Webkit CANVAS [75], the first HTML drawing element controlled by means of JavaScript. Nevertheless, for a long time the web landscape has just been populated by a series of proprietary systems, third-party software, and closed solutions. Not having a common and recognized development standard was a strong limiting factor for the extensive publication and use of 3D content on the web.

The release of the WebGL application programming interface (API) [100] was a major breakthrough, starting the rapid growth of a new generation of applications, based on a common standard, that were able to act directly on the rendering pipeline and, above all, were supported by all common web browsers. In short, thanks to WebGL, Web3D entered in a new era. The first survey completely dedicated to web-based 3D graphics [57] demonstrated the mature status reached in this domain just four years after the introduction of WebGL.

Nowadays, the proposed Web3D approaches (considering both academic and commercial systems) are still very heterogeneous, since they adapt their data presentation strategy to the 3D content, the target

users, the publishing venue typology, the application field, and the planned outcome. The growing number of solutions has contributed to familiarizing users with the presence of 3D on the web, but it has also resulted in an extremely complex scenario, where developers and users often find it difficult to orient themselves, especially those developers with a poor awareness of the particular needs of each specific combination of 3D data and application domain requirements.

This survey presents a review aimed at coping with these needs. Our main goal is to define a schema of the available possibilities and features supported by the enabling technologies and implemented systems. This is aimed at providing the reader with a map that, depending on the application field, could help in navigating through the technical characteristics needed to build an efficient and effective Web3D presentation. Our hope is that the result of this survey could be helpful for readers interested in mastering concepts that characterize the different phases of the 3D publishing process: content creators (enhancing their awareness about the Web3D ecosystem of libraries and authoring tools), content consumers (increasing their ability to fully experience the capabilities of existing systems), and finally also researchers and developers of future solutions.

For the purpose of this review we have evaluated a heterogeneous set of software applications and the state of the art of the scientific literature. The characterization of available solutions proved to be difficult, due to the heterogeneity of the approaches proposed and the number of issues to be considered. Moreover, this survey is designed to focus not only on the current trends, but also on the big challenges that researchers and developers face when sophisticated 3D graphics have to be efficiently ported to the web.

This monograph is organized as follows. Chapter 2 provides a short recap of the evolutionary process bringing us from the early Web3D phases up to the launch of WebGL. Chapter 3 presents three grand challenges to be faced in the development of 3D web content and resources. Chapter 4 presents the categorization adopted for the analysis of the state of the art of current Web3D solutions and technologies, defining a set of features required for 3D web publishing which are described in detail in

The version of record is available at: http://dx.doi.org/10.1561/0600000083

Chapter 5. Leveraging the previous results, Chapter 6 outlines the profile of the available publishing solutions and assesses the current solutions for a representative group of application fields. Finally, Chapter 7 presents the final considerations and future challenges.

The version of record is available at: http://dx.doi.org/10.1561/0600000083

2

Web3D, from Plug-ins to WebGL

A pioneering work [146], published just a few months after the first WebGL release [100], was already asking the question "Is 3D finally ready for the web?" The paper's author stated that, despite the 16 years that had elapsed since the universally recognized Web3D foundation stone of the VRML web standard [157], 3D and the web still seemed to be two distinct worlds. However, although it was still not easy to find 3D content online, the recently introduced technology justified the expectation of a forthcoming major change. In particular, Ortiz's paper suggested that WebGL would be the "most interesting development" for the future of Web3D, as in fact it happened. But let us start from the beginning.

2.1 Early Approaches

The first attempts to publish 3D content online have led to a heterogeneous set of approaches, often very different from each other. This was mainly due to the lack of a real standard for 3D graphics on the web. Actually, two different ISO standards were available in the early phase: the already mentioned VRML [157], and X3D [207, 29]. Unfortunately, since they were designed basically as file/scene formats, they failed

to cope with all the needs of web publishing. Moreover, they needed additional software to enable visualization in web browsers. This last issue was probably the biggest hurdle to the success of that generation of solutions. Those attempts required the design (and use) of proprietary plug-ins, inaccessible to independent developers, that were poorly integrated with other web page elements.

Besides the previously mentioned Adobe Flash [44], Sun Microsystems Java applets [24] were presumably the earliest plug-in able to integrate computationally onerous content on the web. First released in 1995, these are small client-side applications executed in the Java Virtual Machine, an abstract computing machine developed for running Java bytecode in a hardware-agnostic mode. Although Java applets probably constitute one of the first attempts to access the graphical processing unit (GPU) from a web browser (well before WebGL), they were not specifically aimed at managing 3D content. A further development in 1998 was the release by Sun of Java3D [181], an API expressly designed to simplify the development of 3D web applications. Java3D provided high-level constructs to create and manipulate 3D geometries, supporting both Direct3D and OpenGL. Java3D was later discontinued, but the effort to embed 3D content online using Java has been carried on by other libraries, like JOGL [Java OpenGL; 91] and LWJGL [Lightweight Java Game Library; 119], known for being used in the development of the popular Minecraft game [149].

The whole Java ecosystem ignited and supported the design of interesting solutions at the beginning of the Web3D era, concerning both academic outcomes and software results. To cite a few: RAVE [Resource-Aware Visualization Environment 71], an applet designed for collaborative server-side visualization; COLLAVIZ [53], a framework for collaborative data analysis developed using JOGL; ParaViewWeb [94], an applet for remote 3D processing and visualization (later adapted to WebGL); and finally OSM-3D [214], an interactive 3D viewer for OpenStreetMap data developed as an applet.

Java-based applications were thus the first approaches to support 3D content publishing online, but not the only ones. Indeed, just a year after Sun's initial release, Microsoft unveiled ActiveX [126], essentially a framework for downloading multimedia content from the web. Developed

with a philosophy similar to Java applets, it does not use files compiled to bytecode, but dynamically refers to OS libraries (which share the same memory space with the browser). This solution makes ActiveX very fast in execution, but causes drawbacks related to security and OS dependency which hindered the wider adoption of this solution. Microsoft changed their approach in 2007 by releasing Silverlight [127], a client-side API for developing web applications, this time closer to the Adobe Flash philosophy. Silverlight is another solution not specifically aimed at 3D, but is able to provide a programmable graphics pipeline and basic 3D transforms, thus resulting in a step forward from the Adobe Flash plug-in, which was not able to access GPU functionalities until the release of the Stage 3D add-on [3].

Conversely to Adobe Flash, the Google O3D project [65] was designed from the beginning to support the use of GPU features (either via Direct3D or OpenGL). Released in 2009 as an open-source plug-in for creating interactive 3D applications, it was ported to WebGL just a few months after.

Finally, two approaches developed just before WebGL were the Opera Software plug-in [92] and the Canvas3D [129] project, with its supporting library C3DL [114]. These solutions, aimed at creating an OpenGL context in the HTML CANVAS element, were WebGL precursors, later becoming part of the new standard.

2.2 The WebGL Revolution

Anticipated by the aforementioned Canvas3D experiment, WebGL was finally announced by the Khronos Group in late 2009. It introduced a new standard for developing 3D applications on the web which spread rapidly in a very short period, revolutionizing the world of Web3D.

WebGL is a royalty-free API fully integrated with the HTML Document Object Model (DOM). It is based on OpenGL ES 2.0 [99], the OpenGL API for embedded systems (e.g. mobile or portable devices, possibly with lower-end computing resources, power consumption constraints, low bandwidth, reduced memory space, etc.). This makes WebGL extremely optimized and computationally light, and thus ideal for the web environment.

The main advantages of WebGL are:

- cross-browser and cross-platform compatibility (this means that a WebGL application can run on any platform without the need to rewrite source code or install additional software);
- tightly integrated with HTML content (this includes layered compositing, interaction with other HTML elements, use of the standard HTML event-handling mechanisms, etc.);
- a scripting environment that makes it easy to prototype interactive 3D graphics content (making it unnecessary to compile and link before you can view and debug the rendered graphics);
- based on familiar and widely accepted 3D graphics standards, which implies several advantages, such as the chance to use the GL shading language;
- provides access to the GPU programmable pipeline (exploiting hardware-accelerated 3D graphics in the browser environment).

Thanks to these features, WebGL was able to rapidly checkmate the other 3D web publishing approaches, contributing to the quick extinction of all other non-WebGL solutions.

Although it is designed to work strictly in conjunction with extant web technologies like HTML and JavaScript, WebGL remains quite a low-level API, so it is not easy to master without solid skills in computer graphics (CG) and programming. For this reason, the years immediately after its release have seen the proliferation of middle-level wrapper libraries, aimed at making the use of this new standard easier.

Among these middleware solutions, one of the first to be released was SpiderGL [197, 47, 48], a JavaScript library providing typical structures and algorithms for real-time rendering. SpiderGL abstracts a lot of WebGL methods without forcing the use of a specific paradigm (such as the scene graph, a hierarchical structure discussed in more detail in Section 5.2), and without preventing low-level access to the underlying WebGL graphics layer. Other mid-level libraries appearing at almost the same time as SpiderGL were WebGLU [46], supporting a set of

low-level utilities and a high-level engine for developing WebGL-based applications, and GLGE [28], which provides a declarative method for programming a 3D scene. These were followed a year later by PhiloGL [17], a framework for data visualization, creative coding, and game development; Lightgl.js [205], a low-level wrapper that abstracts much code-intensive WebGL functionality; and KickJS [140], a game-oriented engine that abstracts WebGL to make game programming easier.

At the same time as these mid-level JavaScript libraries, some focused solutions were also developed to build a bridge between WebGL and some popular 3D software applications. Among these were Inka3D [208], an exporter plug-in for Autodesk Maya [11]; J3D [51], a utility to export static scenes from Unity3D [191]; and KriWeb [122], a Blender [21] exporter written in Dart.

The version of record is available at: http://dx.doi.org/10.1561/0600000083

3

Grand Challenges for 3D on the Web

The WebGL revolution opened up many possible uses for 3D content on the web, and this impressive potential was clear right from the beginning. Nevertheless, while several characteristics of local tools could be easily mapped to web applications, some solutions were needed to handle the peculiarities of remote use characteristic of the web. In this chapter we focus on the three main challenges affecting the effectiveness of Web3D solutions, which justified some intense research. These three challenges concerned the dichotomy between a *declarative* and an *imperative* approach when defining an online 3D scene, the management and remote visualization of a (complex) 3D dataset over (possibly slow) network connections, and the policies for creating and protecting 3D content in collaborative/sharing environments. For each of these challenges we present a synthetic overview of the basic research and of the seminal experiments that led to the current wide range of solutions, also indirectly providing a preliminary outline of the open issues.

3.1 The Declarative/Imperative Dichotomy

The definition and management of a 3D scene can be done with several levels of complexity, and the more complex the scene, the harder it is to provide a structure that can be easily used by non-experts. This is also true for local 3D rendering, but it poses severe challenges in the case of online 3D content creation and interaction.

A few years after the release of WebGL, Jankowski et al. [88] presented a classification matrix that became popular in the academic world, being recalled in many later works [e.g. 95, 57]. The proposed scheme introduced a parallel between approaches to 2D and 3D graphics on the web, classifying the available techniques into two main groups: *declarative* and *imperative*. While both techniques allow the creation, modification, sharing, and interactive experience of 3D graphics on the web, they differ in their basic approach and target users. While the *declarative* approach exploits the HTML DOM to provide access to high-level 3D objects (components familiar to the web development community), the *imperative* approach uses scripting languages to offer access to the low-level functionality of the rendering stage (elements more common among the CG developer community). Although the distinction between the two approaches is nowadays becoming less and less substantial, the early years of Web3D were strongly marked by this dichotomy. This contrast was able not only to catalyze the attention of the research community, as demonstrated by Jankowski et al. [88] and similar works, but also to influence a number of design choices which still characterize Web3D solutions.

Among the main causes of this separation was the strenuous attempt to lead CG solutions (and their developers) to the native web environment approach, an effort bound to fail in the Web3D environment, a world in rapid evolution populated by solutions not confined to watertight compartments but rather inclined to overlap.

However, thanks especially to VRML [157], that represents a seminal work for the declarative Web3D approach—the declarative approach came to dominate in many solutions between the late 1990s and the early 2000s, for example with ARCO [148, 211, 204], ShareX3D [93], and X3DMMS [215]. VRML was essentially a logical markup format

for non-proprietary independent platforms, using text fields to describe a simple 3D scene (only the content and basic appearance). In 2004 VRML was superseded by X3D [207], in turn an ISO standard, which is defined as a royalty-free open standard and run-time architecture to represent and communicate interactive 3D scenes and objects using XML. It adds a lot of features to VRML (advanced scene graph, event handler, data-flow system for node interpolation, etc.), but, analogously to VRML, it was designed basically as a file/scene format, and moreover it needs external software, e.g. FreeWRL [178] or View3dscene [96], for visualization in a web browser.

Despite these evident limitations (and not always continuous support), many solutions tried the declarative approach in the early years. One of the first was the Blaxxun Contact plug-in released in 1995, which evolved into BS Contact [20] in 2002. It is essentially a scalable multi-user server environment for the VRML developer community. Another pioneering declarative system was SimVRML [159], which uses VRML for scientific simulations, followed a few years later by other interesting attempts like Orbisnap [81], a VRML viewer able to connect with remote servers running Matlab Simulink 3D animation [125]; Octaga Player [141], a solution for building interactive 3D web presentations; Cortona3D [147], a VRML viewer and authoring tool (the last two solutions were later adapted to WebGL); or InstantReality [59], a VRML and X3D framework providing support for virtual reality (VR) and augmented reality (AR).

In order to extend the browser support for X3D, X3DOM [15, 16] was proposed as a declarative system free from plug-ins, able to integrate the X3D nodes directly into the DOM content. Defining an XML namespace and using a special connector component, X3DOM builds a bridge between the DOM and X3D that allows manipulation of the 3D content simply by adding, removing, or changing DOM elements. An approach similar to X3DOM is the XML3D system [175, 176], another plug-in-free high-level approach which, instead of embedding an existing framework (X3D) in the browser context (like X3DOM does), tries to extend HTML, maximizing the use of existing features to embed the 3D content in the web page. XML3D uses XHTML and, for intensive online data processing [see, for instance, 104], relies on Xflow [103, 105], an

"extension" developed to expose system hardware and allowing data-flow programming.

X3DOM and XML3D have been the last two widespread solutions to endorse a "pure" declarative approach. Nowadays, many of the features indicated as characteristic ones in the Jankowski et al. [88] matrix, such as the scene graph construct for the declarative approach or the WebGL API for the imperative approach, no longer allow unequivocal classification of 3D web solutions. A brilliant demonstration of this is A-Frame [130], a solution that merges a declarative structure and the famous Three.js [32, 49] WebGL supporting library.

3.2 Managing 3D Data over the Internet

The efficient visualization of 3D data over the Internet has long been both one of the most important research areas and a characterizing topic of Web3D. The great number of solutions aimed at 3D web publishing available nowadays can be misleading in this regard: this topic is still an active research field, and most of the existing applications work well because they have been calibrated just to render specific types of highly optimized 3D assets. A definitive and universal answer to the many issues characterizing the effective management of complex 3D datasets online still does not exist.

The difficulties of handling interactive 3D content over a network mainly arise from insufficient computational resources (usually due to client-side limitations), poor network capabilities (usually due to limited bandwidth and latency issues), and the huge amount of data to be processed (usually due to the intrinsic complexity of the 3D content).

The latter factor is critical, since 3D models are usually complex resources in terms of elementary components (number of triangles or points), resolution of the attached texture images, and memory size. Some examples of 3D models (particularly the more professional ones) can be misleading, since we may be fooled by a highly optimized geometry or the extensive use of textures to encode geometric detail and save on explicit geometry. The more usual models produced nowadays might be composed of millions of triangles, causing severe network traffic

and resource consumption (especially when represented with naïve mono-resolution formats). In effect, a high polygon count is needed to mask the approximation of curved surfaces with a tessellated mesh (reducing the jagged appearance of edges), or to encode in a more correct manner the color discontinuities when color-per-vertex encoding is used. This problem becomes even more evident with the high-resolution screens available today and with the zooming capability provided in most 3D browsers.

As reported in the recent surveys of Shi and Hsu [169] and Mwalongo et al. [136], different approaches and techniques have been proposed over the time to solve these problems. Web-based 3D visualization in particular has received growing attention in recent years, with an increasing number of emerging applications. This success has been due to its ubiquity across platforms (from desktop computers to mobile devices), but mostly to continuous improvement of the enabling technologies, such as server-side rendering infrastructures (via grid or cloud computing) or client-side rendering techniques (via WebGL and HTML5).

Given the importance of interactivity in visualization, 3D web publishing platforms have been mainly based on the client-side rendering strategy, performing GPU-accelerated local rendering directly within modern browsers. This is also the area we will analyze in this work.

The first set of issues in the management of complex datasets resides in their size: large datasets are problematic in both the transmission and rendering phases. To cope with these two bottlenecks, all the main systems currently aimed at interactive web 3D visualization successfully exploit either *layered* (discrete sequences of data each of which represents the object at a different resolution) or *multi-resolution* (a virtually continuous set of data representing an object at increasing resolutions) data encoding. Layered representations are usually called *level of detail* (LoD) representations [118].

Despite the fact that technologies to handle large 3D models have been studied since the mid 1990s [77, 156], decisive advances have appeared only in the last decade, when it became clear that to efficiently manage 3D content online, web solutions should have to satisfy specific web requirements [116]. Thus, to take into account these peculiar needs, many researchers began to focus on technologies like

LoD/multi-resolution representations, progressive data transmission, and data compression schemes able to balance visualization latency and to perform well in space/time.

One of the first efforts in this direction was Gobbetti et al. [63], which proposes the transmission of 3D models parametrized into a quad-based multi-resolution format. Lavoué et al. [108, 109] suggested iterative simplification of 3D meshes, encoding their information in a compressed stream, thus adapting for the Internet the progressive algorithm of Lee et al. [110], enabling a low compression ratio with a small decompression time—a solution based on valence-driven progressive connectivity encoding was introduced by Alliez and Desbrun [7]. Limper et al. [115] overcame the problem of decompression time/complexity by using different quantization levels for the model vertices and transmitting them using a set of nested GPU-friendly buffers. More recently, Ponchio and Dellepiane [151, 152] presented a multi-resolution rendering algorithm (parallelizable and scalable to very large models) based on Cignoni et al. [41] that was able to combine progressive transmission of view-dependent representations and efficient geometric compression/decompression. Finally, to cope with different issues alternative directions have been proposed, focusing on improved data organization [e.g. generic formats: 117, 182] or different data types [such as point clouds; 56, 165] for streamable 3D content.

The works mentioned in this section do allow considerable performance enhancements; moreover, they represent some real turning points for many application domains that require online handling of complex 3D content and where the capability of presenting data at full accuracy (without using degraded simplified models) is a major need.

However, managing 3D content (and user interaction) over the network raises issues not related just to the sheer size of the 3D data. Since client-side 3D rendering applications are hosted by web browsers, the limitations of the client software play a relevant role in the quality of the resulting visualization. In fact, browsers were not designed to manage the data-intensive and interaction-packed throughput required by Web3D applications. The first issue is related to pure computational power and execution efficiency: web-based code is usually implemented

in JavaScript, which is not a high-performance language like C++ (which, conversely, is the most common choice for the implementation of 3D desktop applications). This situation is slowly changing as the "cloud" paradigm gains momentum, and modern JavaScript engines are incredibly fast, but this issue still has to be considered when implementing a Web3D solution.

Related to the size of 3D data are the limits on the size of the local browser's memory, cache size, and the number of HTTP requests usable simultaneously. Similarly, some web servers may only support certain types of data requests (e.g., the HTTP Range request is often used in multi-resolution schemes, but not all web servers support it).

Another set of issues is related to security. Browsers operate in a "sandbox environment": it is almost impossible to directly write temporary data to disk, and also accessing local resources while working online is not straightforward. Cross-origin data fetching is also limited, and a series of precautions and safeguards have been placed on the execution of JavaScript code. On top of this, the way these limits are enforced subtly changes across browsers.

Finally, user interaction is also affected by the presence of the browser as an "interface" to the code. 3D desktop applications are notorious for their complex inputs using multiple mouse keys and keyboard; the browser is unable to track some of these events (and others may be system-reserved). The situation is even more complex with touch-based inputs, which are supported in various flavors on different OS and browsers, sometimes "virtualized" as mouse inputs, sometimes as touch events, that are nevertheless non-uniform across browsers. The user interfaces of 3D web applications should handle these limitations with specific web-oriented interaction design.

The search for clever solutions to overcome the limitations of the browsers, as well as to define proper optimization for 3D data streaming, rendering, and compression, has shaped the management of 3D data online, steering the evolution of Web3D. However, as visualization quality standards continue to improve, these elements are still trending topics in research, and will surely also remain key components for the next generation of 3D web publishing solutions.

3.3 Production and Protection of Shared 3D Content Online

The success of the technologies mentioned in the previous section revolutionized the publishing of 3D content online, bringing new possibilities but also new challenges, some of which are central to a sharing-oriented environment such as the web. Among these new opportunities/challenges, the production and protection of shared 3D content online deserves a special mention, since these topics represent valuable examples in which it is hard to use or extend standard web frameworks to 3D graphics, but rather it is necessary to reimplement them specifically for Web3D.

For instance, the *collaborative visualization and creation* of 3D content online has always been an interesting and active research field, since WebGL does not define how and in what format the data is transferred to the client, deferring this specification to each application. Studied and analyzed since the beginning of Web3D—the survey by Mouton et al. [128] dates back to 2011—collaborative visualization represents an early component in the design of digital content creation (DCC) systems.

The first system aimed at demonstrating whether web services are capable of supporting collaborative visualization was the already mentioned RAVE [71] Java applet. Designed to determine at run time the performance of the local client devoted to the rendering, and to send to it rendering instructions depending on these capabilities (less powerful clients receive a video feed from a remotely rendered scene, while more powerful ones receive the polygonal dataset to render it locally), this system was soon followed by COLLAVIZ [53], a JOGL framework for collaborative data analysis, and ShareX3D [93], which was the first implementation of a collaborative 3D viewer based on HTTP communication.

Nowadays, leveraging network features like XMLHttpRequest (to request data from a web server) or the WebSockets API (which allows bidirectional full duplex communication), a fully fledged collaborative

visualization package can be completely presented in the web browser—Marion and Jomier's 2012 pioneering work, which proposes a collaborative prototype for scientific visualization using a WebGL/WebSocket combination, offers a brilliant demonstration.

Nevertheless, moving from collaborative visualization to collaborative creation of 3D content is not as straightforward as one might think, the major cause being that CG workflows for working on the data are still not well established. As shown in the recent work of Calabrese et al. [33], overcoming issues like the concurrent overlapping of 3D edits (at different scales) may not be simple (even in systems not designed to work online). This is the reason why over the years we have seen a proliferation of non-collaborative web DCC solutions, as in Ulbrich and Lehmann [190], or collaborative but not completely web-based applications providing offline editing and online visualization, as in Dobos and Steed [50]. Only very few solutions have been able to propose full Web3D systems for collaborative DCC. Among these, Song et al. [174] proposed 3D-CollaDesign, a web framework aimed at distributed designers concurrently designing 3D models in different domains (it uses data replication technology and lock methods to avoid the potential conflicts and to support data consistency), and Du et al. [52], a collaborative solution for assembling 3D components (it provides scene data synchronization, user and data management, and real-time expansion of the component library).

Another technical challenge, closely connected to online data management issues, concerns the policies supported for the management of intellectual property rights (IPR). The difficulties in extending general purpose techniques to shared 3D content make this a particular challenge for Web3D. The management of data protection is a core issue affecting all the media associated with web pages, but it is much more complex for 3D data since it is a more recent media format without proper international legislation able to protect it clearly.

The fact that high-resolution models are usually rendered by adopting multi-resolution and view-dependent techniques represents a form of implicit protection, since the full-resolution model is never sent for rendering in its entirety, but Web3D systems send to the client only a view-dependent model customized for the current view specifications.

Thus, fraudulent copies of high-quality geometry would need more advanced methods (reverse engineering techniques). When the 3D object is a simple mesh, however, and multi-resolution is not needed, IPR protection becomes a problem.

Current solutions are mostly based on *digital watermarking*. Watermarking consists of hiding information in a noise-tolerant carrier signal (like audio, video, or images), and is the process most often used for tracing copyright infringements. Digital watermarking systems can be grouped into different schemes depending on the information needed for watermark detection: "non-blind" or private watermarking, if the watermark detection requires the original data and possibly also the embedded watermark; "semi-blind" or semi-private watermarking, if detection requires the embedded watermark; and "blind" or public watermarking, if detection requires neither the original data nor the embedded watermark. These algorithms have been implemented with success for 1D and 2D data. Unfortunately, implementation of robust 3D watermarking is much more difficult, due to the topological complexity of 3D models and the higher number of attacks and manipulations that watermarks should be robust to when applied on 3D data (such as noise addition, smoothing, cropping, etc.).

Seminal work on 3D watermarking was performed by Ohbuchi et al. [143, 144] using a non-blind technique, robust to 3D transformations and cropping, with the watermark embedded in the geometric domain modifying either the vertex coordinates or the vertex connectivity. Other important contributions have been the works by Praun et al. [155], a detail-preserving non-blind method driven by multi-resolution theory, resistant against many local transformations such as noise addition and smoothing; Ohbuchi et al. [145], a non-blind frequency domain Fourier approach for 3D shapes, requiring manual interaction; Cayre et al. [35], again Fourier, but in this case blind, extending spectral decomposition to 3D meshes for robust watermarking; Uccheddu et al. [189], a blind method decomposing the mesh into a multi-resolution representation working in the wavelet domain; and Zafeiriou et al. [213], a blind, radial watermarking method that embeds the watermark in the distribution of distances between the mesh vertices and its center of gravity, presenting good levels of robustness at a low computational price.

Despite all these research efforts exploring a wide space of solutions (from the spatial to the spectral domain), the search for a functional and practical watermarking approach for protecting 3D content online is still a challenging issue [120]. This is mostly because these attempts have generally led to applications that are demonstrators rather than fully qualified products, but also because watermarking has led to collateral issues, such as infringement recognition. After watermarking an asset, the owner should be able to recognize if a given 3D model has been produced as a copy of that asset. But to reach this result, one has to put in place a policy for (a) the retrieval and test of possible copies, and (b) legally prosecuting the authors of those copies. Since this might be afforded by large companies, but not by standard users, it easy to understand that an ideal IPR policy should rather prevent the copying of the digital data, which is a goal that is very hard to obtain with the current technologies.

To conclude, both issues (shared content creation and protection management) are still open issues for Web3D, and solutions able to fruitfully support all their requirements are rare. However, since the policies adopted for online 3D data management have a potential impact on the features and performance of a Web3D platform/system, it is easy to predict that advancements in these fields will become more and more a key factor for the success of future 3D web publishing solutions.

The version of record is available at: http://dx.doi.org/10.1561/0600000083

4

Feature-Based Characterization of Web3D Solutions

Since the first WebGL release, the resulting Web3D growth has resulted in a myriad of different proposals, diversified depending on content, target, context, and of course, expected results. In order to define a schema of the available (or required) features, a categorization of these approaches is required.

4.1 Which Categorization of the Existing Solutions?

Chasing a representative categorization of the Web3D landscape, we have surveyed and analyzed a large set of solutions, starting with simple libraries, moving to middleware solutions, and concluding with the analysis of complex applications. The decision to cover approaches going from low-level to high-level solutions (see Figure 4.1) allowed us to evaluate and review almost the entire 3D web publishing possibilities and requirements; on the other hand, it has forced us to consider a collection of contributions inherently heterogeneous, composed of libraries, tools, frameworks, applications, etc.

Despite these solutions being characterized by different abstraction levels, they also reveal several deviations from more classical level-based classifications. This leads to fuzzy boundaries between the various

Figure 4.1: A representation of the Web3D space reviewed, covering everything from low-level to high-level solutions.

systems that make it hard to reduce them to fixed reference patterns. The idealized image of a clean "layer stack" application (where each software layer is built on top of another, and only vertical communication is possible) seems to be unable to cope with the more fluid development of web applications. As most modern Web3D systems are based on JavaScript, it is really easy to jump over layers and to interface at different levels to the various software components of the application. Many of what we would call "application layers" directly interface with the very bottom WebGL layer for some specific low-level function, but simultaneously exploit multiple "supporting layers", which may themselves have cross-dependencies. It is quite common for WebGL wrapping libraries to be designed in this way, allowing them to be used as low-level libraries, but also as basic load-and-display applications. This fluid development can make it difficult to categorize and restrict a software component to a single category, and the resulting scheme is somewhat more complex than a neat series of software layers only communicating vertically.

The stratification proposed in Figure 4.2 tries to structure and better represent the scenario introduced in Figure 4.1, enclosing the attempts built over WebGL in three macro-groups (defined not only by looking at the abstraction level of the solutions, but also considering other key factors, such as their interactions, the capabilities provided, coding needs, etc.):

Lib/Lib++ Supporting libraries for the WebGL API (low level, coding is required). The notation covers the whole low-level scenario, starting with basic libraries that do not provide high-level features, like Lightgl.js [205], a bare-bones WebGL wrapper, abstracting

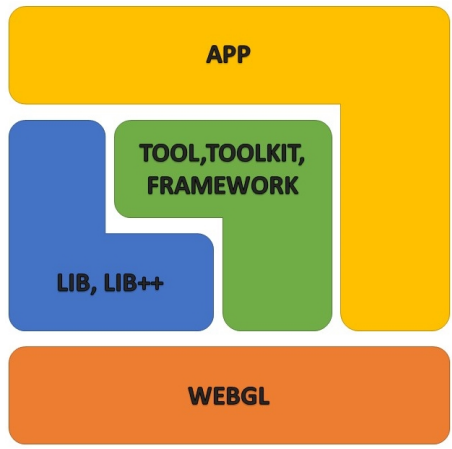

Figure 4.2: A layered representation of the software contributions used in Web3D implementations, which aims to define a map characterizing the connections and relations among the different solutions.

many code-intensive functionalities, and gradually working up to more complete ones, like Frak Engine [2], a library providing features for simplifying the creation of complex interactive 3D applications, or Three.js [32], a multi-level API allowing the developer to rely on a wide set of features, such as coding support utilities, documentation, examples, tutorials, how-tos, etc.

Tool/Toolkit/Framework Middle-level solutions. To access WebGL they often use one of the aforementioned libraries; usually provide a graphical user interface (GUI); almost always require a certain degree of coding. May range from simple tools to complex frameworks. Besides the basic features, they can provide the developer with higher-level functionality related to the 3D scene (hotspots, user interfaces, analytical tools, etc.). Examples: WebGLStudio [5, 4], a toolkit to create interactive 3D scenes directly from the browser; Clara.io [58, 79], a platform for modeling, animating, and visualizing 3D content online.

App Applications at the highest level, where authoring elements are used and coding is not needed. Typically end products supporting

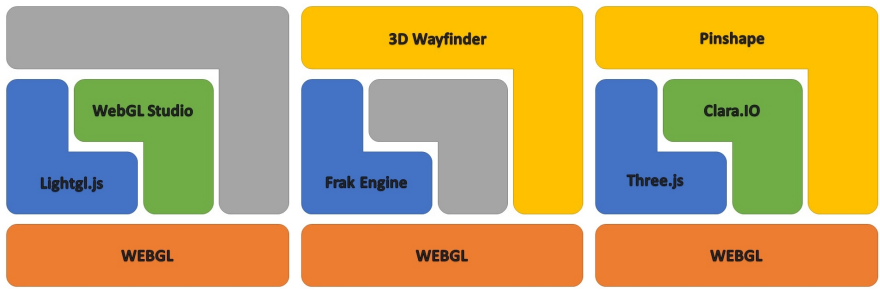

Figure 4.3: Graphical representation of different layering possibilities adapted to three real case studies. The diagrams show the interaction between (from left to right): a basic library (Lightgl.js) and a middle-level solution (WebGL Studio); a basic library (Frak Engine) and a high-level application (3D Wayfinder); a basic library (Three.js), a middle-level solution (Clara.io), and a high-level application (Pinshape).

online 3D publishing in the form of web services. Examples: 3D Wayfinder [1], an architectural application offering ways to manage content in 3D floor plans; Pinshape [124], a portal and marketplace for the 3D printing community.

Although this layered scheme fits very well with a wide range of reviewed cases (in Figure 4.3 we present just three graphical examples describing some of the cited software), on closer analysis it turns out to be not flexible enough to provide a complete characterization of all the existing solutions. For instance, it fails to clearly classify "multi-level" libraries (wrapping libraries providing, at the same time, very low-level and very high-level access), or again to correctly identify solutions like 3D publishing standalone suites to download and install (end products that act like middleware systems but at the same time provide very low-level possibilities).

To accomplish our categorization, we therefore decided to propose a different point of view: a systematic study of the *features* supported by the state-of-the-art solutions, organized in a subset of macro-groups. The reason for following a "per feature" review rather than presenting the domain by listing and describing all solutions is also related to a wish to promote a transverse analysis over the possibilities offered by the main approaches. In this way, instead of producing a linear list

of libraries/tools/apps, we aim at obtaining a more useful comparison of different philosophies and methodologies. Moreover, we also think that this decision may be the most suitable in providing researchers, developers, and final users with a full understanding of the breadth and depth of each contribution developed for publishing 3D content on the web.

4.2 Characterizing and Grouping Web3D Features

Characterizing the different approaches by their features means, first of all, defining the atomic actions and functionalities needed for publishing and interacting with 3D content online. With this aim, we have identified a large set of characterizing features that should be able to satisfactorily cover the possibilities provided, supported, or required by a Web3D solution.

This set turned to be a long list of disparate aspects, including features at different levels, for both implementation and usage:

- supported data types and associated representation schemes (3D volume, particle systems, triangle based, etc.)
- publishing modalities (developer oriented, hybrid node based, for naïve users, etc.)
- scene creation and organization tools (hierarchical structures, procedural definition, geometry instancing, etc.)
- hyperlink-based integration between 3D and other media (text, images, etc.)
- data encoding and transmission schemes (LoD formats, progressive streaming, view-dependent refinement, etc.)
- object/scene interaction paradigms provided (inspection vs navigation)
- scene customization possibilities (shaders, materials, lighting maps, etc.)

- supported input/output modalities (touch based, gesture based, WebVR, etc.)
- data pre-processing possibilities (client side vs server side)
- supported publication aims (pure visualization, digital content creation, etc.)
- visual schemes for informative scene enrichment (HTML labels vs clickable instances)
- data IPR management modes (watermarking, grant permission, access passwords, etc.)
- scene-level interaction elements (toolbars, 2D maps, view cube, etc.)
- annotation/hotspot authoring tools (policies for implementation and use, pros and cons)
- scene animation support (camera animations vs model animations)
- supported publishing experience (specialized analytical tools, distinctive interaction paradigms, community-aimed features, etc.)
- data hosting costs (disk space footprint, payment fees, freemium plans, etc.)
- data storage and access protection (data encryption, data center secure location, threat prevention, etc.)
- distribution terms and costs (open source, freemium, commercial, etc.).

Given the variety of the reported items, instead of building a hard-to-read table full of footnotes, we decided to structure the discussion of the following sections by grouping the listed features into a few sensible macro-groups, related to their scope and usage. This harmonization process allows us to define the following five groups (also schematized in Figure 4.4):

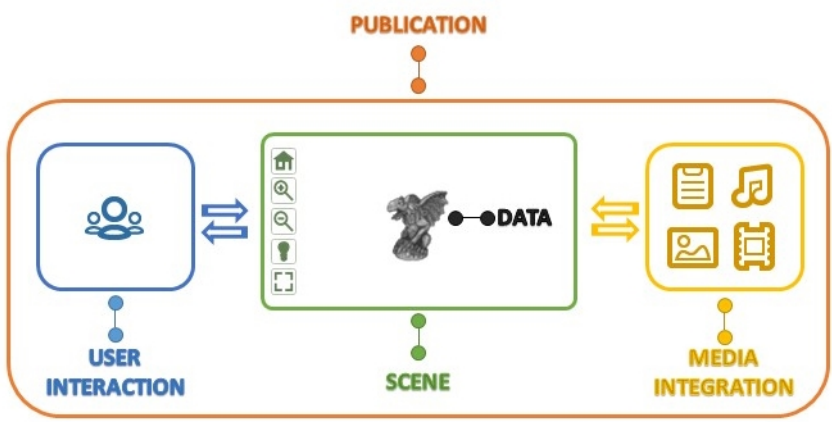

Figure 4.4: A graphical representation of the five macro-classes of features defined. The proposed scheme covers the whole ecosystem of Web3D solutions, ranging from low-level to higher-level functionalities.

Data Level Functionalities related to 3D data handling, including the *representation*, *processing*, and *transfer schemes* adopted to efficiently deploy and render 3D models. This level also introduces higher-level policies for data management (*hosting costs*, *storage and access protection*, *IPR best practices*).

Scene Level Functionalities needed to define a structured 3D *scene* as a *composition* of a number of basic components (*hierarchical structures*, *geometry instancing*, etc.), and to specify the properties of those components (e,g, scene *animations*). This level also includes the possibilities of personalizing the *scene appearance* (customizing *shaders*, *lighting maps*, etc.).

Interaction Level Functionalities concerning *tasks and paradigms* related to the final user *interactions* with the represented 3D scene/object (*inspection*, *navigation*, etc.). This level also introduces *interfaces* aimed at interactive exploration, and components required to drive and manage *specialized input devices* and *advanced output modalities*.

Integration Level Functionalities implemented to provide a *linking logic* between the 3D content and the other multimedia contents present in a web page. This level investigates the trans-media integration and embedding strategies (*annotation* and *hotspot* schemes), as well as the possibilities of exploiting *authoring tools* to make them easier to define.

Publication Level Publishing functionalities analyzed at the higher abstraction level. This level introduces features based on more general concepts, like the *kind of publication* provided to content creators (*naïve-user targeted*, *developer centric*, etc.), the *basic publishing aim* promoted (*interactive visualization*, *collaborative editing*, etc.), the class of *final experience* supported (*specialized inspection/analysis*, *community sharing*, etc.), or even the *type of use* permitted (*open source*, *commercial*, etc.) and the associated *costs*.

Obviously, each of these groups cannot be considered an absolutely isolated container. The boundaries of these groups are fuzzy, and many of the features discussed often exist only if strictly correlated with characteristics listed in a different group of features. At the same time, some other features could span more than one group. But, in the end, this simple schema, considering five areas commonly shared by almost all the Web3D solutions we analyzed, proved to be an effective way to analyze the whole landscape of 3D content published online.

The version of record is available at: http://dx.doi.org/10.1561/0600000083

5

Analysis of the Features

In the previous chapter we introduced a *per-feature* presentation and categorization approach. We have first listed a potential set of features and then we established five macro-groups, to better organize the discussion. The following sections discuss each of these macro-groups, detailing their features and providing relevant examples on how the same capability/issue has been addressed by different Web3D solutions currently available.

5.1 Data Handling

To discuss the Web3D publishing features we follow a logical order, starting from the innermost component of the Web3D tools and services, which is the *data* level. Analyzing this level we will touch both low-level features (data representation types, rendering techniques, transfer schemes, pre-processing optimizations, etc.) and higher-level functionalities (storage policies, IPR protection, etc.).

Studying the Web3D context, the first thing that catches the eye is the difficulty of managing the inherently complex data in a computationally poor environment, such as the web. Over time, this issue has led to systems tailored to the specificity of the data to be managed, proving

that the data involved in the publishing process, and the related design choices, play a central role in characterizing each Web3D solution.

So, content creators approaching Web3D publishing face a preliminary basic characterization, strongly dependent on *data types* and *representation schemes*. The visualization of different datasets requires different data representation and rendering techniques, also influencing the choice of the proper publishing channel.

For instance, *volume visualization* (very popular in medical applications, but also in geographic and meteorological visualizations) often relies on *volume ray-marching* rendering algorithms, i.e. GPU-based techniques that use textures for data storage—MEDX3DOM [42] and X Toolkit [73, 212] are two examples of medical solutions adopting this rendering approach). If, conversely, the aim is to visualize a *particle system*, it would be preferable to use the *ray-casting* technique (another GPU-based technique, that defines objects using implicit primitives, and then generates their surface by computing ray–object intersections). Examples that fruitfully exploit this approach can be found in the biomedicine/molecular visualization domain, where atoms are approximated by spheres whose centers and radii are the parameters sent to the GPU; examples include the systems by Mwalongo et al. [135] and Chandler et al. [37].

Even though data-specific rendering techniques are frequent, recent advances in web technologies (once again, JavaScript improvements and WebGL availability) have made the *triangle-based* approach more and more efficient, making it the most widely used approach, not only when "classical" mesh-based 3D models are involved, but also extending this representation to the visualization of other types of data: geospatial maps [89], city models [60], marine data [158], and many others, including the very same volumetric 3D datasets [72] and particle systems [160] mentioned previously.

However, despite the increase in performance due to the adoption of GPU-enabled rendering solutions, the efficiency of the rendering system remains tightly coupled with the intrinsic characteristics and granularity of the data, thus maintaining the importance of a careful choice of data representation and rendering strategy.

Strongly related to the adopted rendering techniques, *data encoding* and *transmission approaches* provide another basic characterization of Web3D systems. As already stated in Section 3.2, efficient data handling is fundamental to interactive web visualization, mainly because, without adequate data transfer performance, low bandwidth and latency issues can lead to long waiting times until data is available to the browser.

As we have discussed, a number of algorithms have been formalized specifically to address this bottleneck, mostly proposing *transfer formats* able to support *progressive streaming* of simplified versions of the handled 3D geometry, such us P3DW, derived from Lavoué et al. [108, 109]; SRC, derived from Limper et al. [117]; or NXS, derived from Ponchio and Dellepiane [151, 152].

A couple of examples of real Web3D approaches using these data formats are the InstantReality [59] tool already mentioned in Section 3.1, which uses the SRC format, and the 3DHOP framework [199, 154], a specialized solution for publishing high-resolution cultural heritage 3D content using the NXS format.

Some of those representation schemes also provide additional features for the progressive rendering of LoD or multi-resolution encodings. They may include network-oriented compression and decompression algorithms, that ensure efficient decoding and rendering, rather than only optimizing the compression ratio. This is exactly what happens, for instance, in the WebGL-loader [68, 40], a solution developed in the context of the Google Body project [23]: based on UTF-8 coding, delta prediction, and GZIP, this produces a compression ratio of around five bytes per triangle for encoding coordinates, connectivity, and normals. Other optimizations related to progressive rendering techniques concern the refinement criteria: they may be view dependent (refine the geometry resolution depending on dynamic camera specifications) or based on the system rendering load (provide a high-resolution representation for more static visualizations, and a lower resolution for more demanding computational conditions, as in the example presented in Figure 5.1). The Nexus toolkit [198] for adaptive multi-resolution management of 3D models (both triangle and point clouds), which provides all these features, is a perfect representative of these efficient and optimized transmission and rendering schemes.

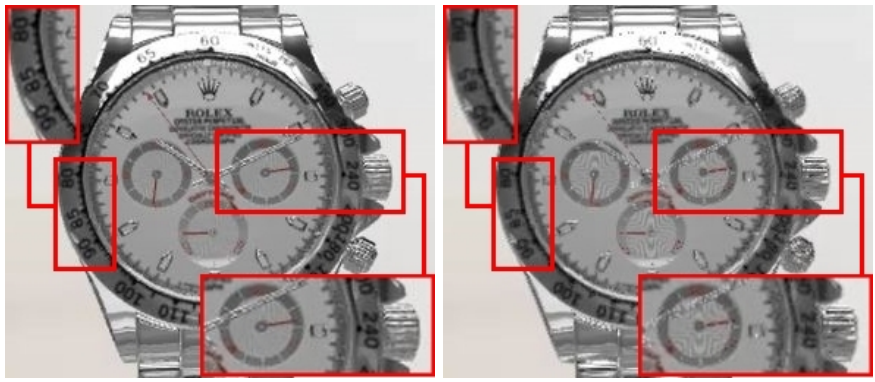

Figure 5.1: Example of selective LoD rendering driven by animations or user interactions in the ThreeKit [78] viewer, an online 3D configurator based on discrete LoD layered models, built on the top of the Clara.io [58] viewer introduced in Section 4.1. When the final user starts to interact with the model, the system switches from a higher-resolution (left) to a lower-resolution (right) representation, to keep pace with the more frequent image refresh rate required by the interactive session.

Nowadays, almost all the main mesh-oriented web 3D viewers adopt LoD/multi-resolution representation schemes (often developing proprietary transfer/streaming formats). Nevertheless, for specific cases where the original coherence/precision of the 3D data is more important than minimizing the download times, or when the kinds of primitives adopted are difficult to simplify, single-resolution data representations are still adopted.

Often, this strategy occurs in application contexts that exploit computer-aided design (CAD) solutions for assisted 3D modeling and to support production processes (this is common in fields like engineering, architecture, and mechanical design). The reasons for employing single-resolution models are various.

CAD-like tools often rely on the use of precise metric data, and this conflicts with the LoD and multi-resolution geometries where the local geometry is simplified with a loss of precision and, possibly, a change of topology. Additionally, the modeling approach more common in these domains is based on many independent components, each derived from basic geometric primitives or modeled independently, which are then

merged together. The resulting 3D mesh may present uneven triangulation, intersecting elements, and issues related to a non-clean topology. These geometries can be easily rendered without noticeable defects (e.g. interpenetration is usually masked by the visible surface closer to the viewer), but create a lot of problems when an incremental simplification or LoD/multi-resolution construction algorithm is applied to them, due to the topological issues. This often prevents the automatic creation of coherent LoD/multi-resolution representations. Moreover, LoD/multi-resolution structures are static and do not allow the modification of the geometry/topology that is needed in these tools. Every time the mesh is edited, the LoD/multi-resolution structure has to be reconstructed. This introduces issues in efficiency (reconstruction is usually a complex task) and in resources (the amount of data to be processed is very hard to handle efficiently in a web browser, because of the limitations in the browser cache memory). Again, the previously mentioned read/write protection of web browsers represents an important hurdle for these frequent and resource-expensive computations.

Consequently, Web3D systems devoted to online 3D modeling such as Autodesk Tinkercad [12], a browser-based 3D modeling tool aimed at 3D printing, usually rely on single-resolution representations. This allows the final user to edit the 3D model in real time and with full precision, but also requires considerable data transmission/download throughput and may slow down the interactivity. Even if the 3D geometries modeled can be considered relatively simple (in terms of number of vertices and triangles), they can easily reach sizes of the order of tens or hundreds of megabytes, mostly because of the assets associated with these models (textures, materials, etc.), but also because of the complexity of the virtual scenes modeled (typically characterized by a high number of 3D model instances).

It should be mentioned explicitly that adopting a single-resolution geometry does not reduce the importance of employing (lossless) compression and optimization strategies focused on minimizing the size of the 3D data transmitted. Anyway, the limitations of these data formats still affect the performance of Web3D CAD applications, especially because they clash with the data management limitations typical of all the main browsers (see Section 3.2).

However, despite the mentioned critical factors in adopting single-resolution approaches and the related well-known issues [162], online applications aimed at assisted 3D modeling are forced to rely on those representations, mainly because they need to perform frequent dynamic 3D geometry editing actions, which conflicts with the usually relevant processing times of high-quality LoD/multi-resolution construction algorithms.

In fact, the main drawback with LoD/multi-resolution approaches is that they require some heavy 3D *data pre-processing* before data can be published on the web. Data pre-processing, however, is also often necessary for non-mesh highly specialized viewers like the ones mentioned at the beginning of this section, which, relying on highly optimized and custom-tailored rendering, generally require data conversion to specific optimized formats. This step can be performed locally (on the client device), or remotely (usually on the server that hosts the Web3D publishing service). Both approaches have pros and cons.

In *client-side* conversion the data optimization is performed on the client, and thus the content creator needs to download and run additional software to convert the 3D data into a layered or multi-resolution format. Converting the data on the data-owner device prevents the need to upload the often huge single-resolution 3D model—the progressive encoding is usually compressed, and thus the final encoding is much smaller than the original data file. The aforementioned 3DHOP [199] framework adopts this approach.

In *server-side* conversion the data processing is instead run on the server. In this case the whole single-resolution data file must be uploaded, and usually no clues to the data conversion process are provided. On the other hand, in choosing this approach the content creators do not have to be aware of the technicalities, do not have to bother ensuring availability of the required resources (memory space or processing power), and do not have to install the required software tools for data conversion. Therefore, this approach increases the perceived degree of automation of the web publishing process. For these reasons this latter option is typically preferred to the first one by end-products that support online 3D publishing in the form of one-click publishing services. Representative examples of solutions adopting this approach

are the Visual Media Service [201, 153], a web service providing easy online presentation of complex visual media assets (it converts data to the multi-resolution Nexus open-source format), and the Sketchfab platform [171] for community sharing of 3D models (it converts data to a rendering-optimized and compressed proprietary representation).

However, especially if uncontrolled and undocumented (as is often the case in commercial systems), data conversion can affect data integrity. This may be perceived as quality degradation and can represent a problem in some specialized application fields (such as medical diagnostics data visualization and analysis).

The migration of 3D data toward web-friendly representation formats (simplified and/or compressed) is fundamental when storage and data transmission requirements are critical issues, as in the case of service-oriented platforms. So, also considering the peculiarities of 3D data, the *cost of online models* (measured by disk space footprint) can easily become a relevant discriminant in the choice of the most appropriate publishing approach. Usually, commercial Web3D platforms that offer publishing space for sharing 3D creations propose freemium or payment plans diversified on the required storage space or on the provided rendering quality, e.g. PlayCanvas [54], a game engine for creating, publishing, and sharing interactive 3D applications on the web. But disk space and bandwidth efficiency are also central issues when working with open-source solutions (which usually let the developer embed the 3D viewer in a web page for free). In fact, in this case the server storage and bandwidth resources are generally outsourced to an external fee-based service.

The policies adopted for 3D data encoding have a potential impact on another important Web3D feature, the *IPR management* issue. Exploiting LoD/multi-resolution and view-dependent techniques may represent not only an advantage in terms of performance, but also a sort of implicit data protection. Since with these approaches the full 3D model is never sent for rendering in its entirety (in fact, Web3D systems send to the client only model chunks customized for the current view specifications), a fraudulent copy of the original data would need more advanced methods (reverse engineering techniques). But when LoD/multi-resolution is not needed and the 3D object used has a

"plain" geometry, then IPR protection becomes a problem. This crucial point, common to all media-hosting services, has in recent years also become relevant in the Web3D world, driven by the explosion of sharing platforms for 3D printing: Thingiverse [172], Shapeways [168], Threeding [183], MyMiniFactory [137], etc. As stated in Section 3.3, IPR protection of online 3D data is a hot topic for Web3D; 3D watermarking can be a (partial?) solution, but the current approaches to building a defense are often based on a set of progressive barriers. Many solutions try to protect the uploaded content from unauthorized replication (by other users), for example giving the opportunity to make published material private (not indexed, or accessible only through temporary links) or not downloadable. In some cases the adoption of proprietary file formats (preferably LoD/multi-resolution, as we have seen), or the use of password protection for every model, can add additional security to the management of the uploaded 3D content. The ShareMy3D [9] platform for publishing (and storing) 3D meshes online was an example of system offering all these features (recently acquired by Cognite, this platform is no longer available).

The characterization of the security policies of Web3D approaches should also take into account the more general and higher-level area of *data storage and access protection* (i.e. related to the cloud servers used). Nowadays, such infrastructure should be compliant with specific ISO standards for information management security (ISO/IEC 27001) and able to provide a number of security mechanisms, including physical *data center secure locations* (perimeter fencing, patrolled security guards, biometric entry authentication, laser beam intrusion detection, etc.), *data encryption* (automatic encryption under the 256-bit Advanced Encryption Standard, regularly rotated set of master keys, etc.), and *threat prevention* (uninterruptible power and backup systems, fire/flood detection and prevention, etc.). However, an in-depth discussion of these last points is outside the scope of this survey.

5.2 Scene Setup

In Section 5.1 we analyzed the core features of 3D data management. But what is presented to the user is often not just a single 3D model,

but a more complex set of entities. Consequently, most 3D viewers rely on the concept of a *3D scene*. A scene is basically a "container" defined in three-dimensional space, used by the 3D application to arrange all the entities that are needed to represent, manage, display, and interact with a three-dimensional environment. To this extent, it may be considered the atomic unit of a 3D publishing project. A scene contains one or more 3D entities (mesh models, but also particle systems, impostors, volumetric data), arranged in a reference space, plus all their related assets (textures, materials, shaders, animations), but also contains all the entities used to compute their appearance (such as lights), one or more cameras (the point of view of the user), and all the other 2D and non-dimensional entities used for interaction and rendering.

This section presents the *3D scene setup* possibilities provided by the various Web3D approaches, including scene appearance (color, reflectivity, transparency, etc.), logical and spatial organizational structures, composition customizations, etc. Many applications require a 3D scene to be organized into some kind of *hierarchal structure*, which is used to arrange the different elements of the scene in a particular spatial or logical disposition.

Scene graphs are the most popular among this kind of structure: they organize the scene as a complex graph that includes all the elements of the scene as a set of nodes, possibly resembling an identifiable rooted tree. Scene graph approaches are frequently used in 3D rendering engines, and are generally based on a logical, rather than spatial, arrangement of the data nodes. Many scene graphs also provide *geometry instancing*: a system in which the scene graph nodes represent entities or objects in the scene that refer to a single copy of the data (made up of a 3D mesh, textures, materials, etc.) kept in memory just once. This allows the memory budget to be reduced and the rendering speed increased, but also facilitates the organization and management of physical interactions, collision detection, and large-scale animations. For these reasons, scene graphs are particularly useful in managing increasingly complex 3D scenes, such as those of modern 3D games.

Although there exist specific libraries [e.g. OSGJS: 150] developed to port the scene graph concept in WebGL, nowadays many Web3D approaches are based on proprietary inbuilt scene graph managers. This

tendency follows the idea expressed in the previous chapter of tailoring the data management to the specific characteristics of the target data and the specific application, sacrificing generality for more efficiency. Examples of solutions adopting this approach are SceneJS [97], an open-source low-level scene-graph-based engine for 3D visualization, or, at a higher level, the already mentioned Unity3D [191] engine (a freemium multi-platform authoring tool, initially designed for 3D games but currently used for general purpose interactive 3D creations and installations).

However, scene graphs are not the only strategy followed in the Web3D environment and, in some specific situations, these logical hierarchal structures may also complicate the content creator's life, for instance enforcing a higher level of hierarchy when instead a much finer or spatially aware level of control over the execution flow is needed. Therefore, some solutions, such as the already introduced Lightgl.js [205] or Stackgl [177], a WebGL software ecosystem inspired by the Unix philosophy, do not explicitly provide a scene graph, and give the content creator the freedom to build it on top of a basic functionality layer, just like the previously mentioned WebGLStudio [4] system does, featuring scene graphs despite being based on Lightgl.js.

A frequent alternative to the support of the scene graph concept is to enable the *procedural definition* of the 3D scene. This approach exposes several fundamental components, combined in a procedural way to form more complex entities. Solutions adopting this composition scheme (which also includes a geometry instancing system) provide a more flexible way to create content-rich scenes and represent 3D objects, offering a performance-oriented data structure that can be effectively used in applied research or algorithm prototyping. An excellent example of this kind of approach are those systems in which the goal is not to present a large 3D scene, but rather to find a way to visualize efficiently very complex 3D data, such as the Potree system [164], an open-source viewer for large point clouds, that uses multi-resolution octree-based algorithms for displaying at interactive rates enormous unprocessed 3D point clouds.

A structured 3D scene can also include the definition of the *visual representation appearance* of its 3D content and, in some cases, it may

also allow for complete run-time control of the rendering appearance at object or scene level. It may be argued that, in some contexts, the appearance (material, texture, shader) should be considered part of the 3D model data. However, more often than not, in Web3D applications the shaders, materials, and textures are assets that are associated with the 3D model only at the level of the scene, and mostly at publishing time; moreover, the possibility of changing the rendering appearance of the scene elements in real-time is certainly scene-level functionality. This "soft" link between 3D data and appearance certainly derives from the nature of the WebGL rendering pipeline, which is completely based on shaders.

Customization of the 3D scene representation appearance involves the ability of a Web3D solution to modify the GPU rendering pipeline, personalizing the adopted *shaders*, *materials* (mapping enhancements that allow objects to simulate various types of realistic materials), and *lighting maps* (techniques used to create different rendering effects by defining different types of light sources).

For web publishing software, the more this rendering stage is programmable/configurable, the easier it is to achieve complex visual effects, at the cost of requiring a much deeper knowledge of rendering concepts from the content creator (shifting the emphasis from web programming to CG programming expertise).

Some Web3D solutions provide a complete set of customizable parameters, comparable to the classic standalone 3D creation suites. BabylonJS [34], a relatively new open-source JavaScript/TypeScript 3D engine oriented toward game design, is a good example of this. It allows developers to fully personalize the rendering/shading process by creating custom shaders, materials, and lighting, and providing explicit features to set diffuse, ambient, and specular lightning, as well as opacity, reflection, mirror, emissive, specular, bump, and lightmap textures, unlimited lights (points, directional, spots, hemispheric), the Fresnel term for diffuse, opacity, emissive, and reflection, etc.

In these kinds of solutions, usually the setup of the 3D scene appearance is defined simultaneously with the geometrical scene definition. Some other Web3D applications also provide the possibility to change the initial settings in real time during the visualization, using external

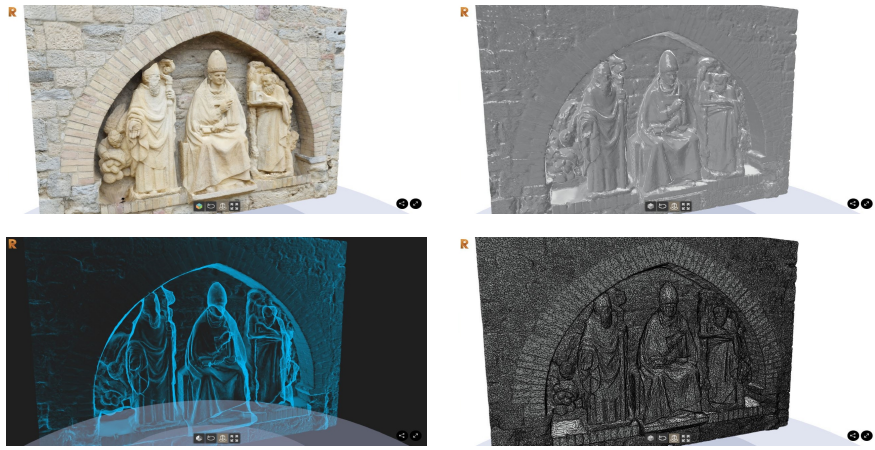

Figure 5.2: Four rendering possibilities provided by Autodesk ReMake [14]. In clockwise order starting from the upper left: textured, solid, wireframe, and X-ray mode. Interacting with the viewer controls, the final user can select in real time to visualize one of these options.

variables or textures for modifying the algorithms defined in the shaders. Obviously, in this case the set of actions allowed to the final user is considerably smaller than in the previous case.

The already mentioned Sketchfab [171] publishing platform offers the possibility to choose between a few predefined shading methods, each one with a series of customizable parameters. The content creator, at upload time, besides configuring the optimal shading method for their 3D model, may also choose which method and parameters will be available to the content consumers when exploring the scene/object.

Conversely, the Smithsonian X3D [173] viewer (developed by Autodesk for this cultural institution) has a single shading pipeline but offers to the final user a lot of real-time-editable shading parameters, material properties, and control over multiple light positions and colors.

ReMake [14], again by Autodesk, an end-to-end solution for creating (offline) and publishing (online) 3D scenes, provides a single shading method, and only offers the chance to switch on the fly between a predefined set of rendering modes (see Figure 5.2).

As stated above, a 3D scene also contains the specifications for a *viewpoint* or *camera*. This element is not only important for rendering

the scene, but also to specify other characterizing features such as camera *animations*. These may be simple predefined views, smooth view interpolation between predefined positions, or precomputed camera paths. Even if this kind of animation seems to be relatively simple (when compared with complex or large-scale animations of the 3D scene objects), they transform a simple 3D viewer into an effective publishing tool. Thanks to their simplicity and high effectiveness, they are provided by many systems. The availability of this feature allows the creation of bookmarked views or points of interest (POIs) inside the 3D scene. These predefined views can be linked to other web media/components outside the canvas element, or also interconnected in a smooth, animated, and immersive view path inside the 3D scene: the Archilogic [10] platform (focused on 3D architecture and interior design) provides a functional implementation of this kind of guided presentation.

Of course, cameras are not the only animation that is available as an asset in a 3D scene, and the more complete platforms (particularly those oriented toward gaming) provide to the users features to animate all the elements in the scene. An exhaustive review of animation possibilities on the web was recently presented by Ahire et al. [6]. However, for the sake of conciseness, the main approaches to 3D online publishing (such as the Sketchfab platform) support just the following animation modalities:

- *skeleton-based*: the skinned 3D surface of the model is connected to an animated multi-joint structure, generally used for character animations;
- *rigid*: animate model translation, rotation, and scale, generally used for mechanical animations;
- *morph to target*: morph shapes from one state to another, generally used for facial animations.

The availability of camera and object animations is a fundamental requirement when evaluating different Web3D solutions, because by means of these features the content creator can cross the borders of the 3D rendering container, introducing elements of *digital storytelling*

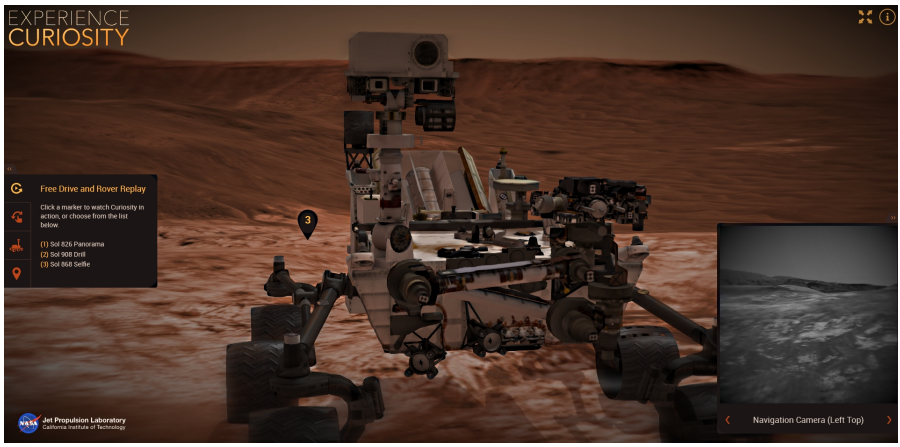

Figure 5.3: The interactive 3D web application developed by NASA using exclusively open-source software (including the commercially licensed Blend4Web). This application was released in 2015 to celebrate the third anniversary of the Curiosity rover landing on Mars. The end user can control both the rover and the camera, driving the rover across the bookmarked views placed on the virtual scene (for example, marker number 3 in the figure).

and moving the published content toward an interactive and integrated experience. "Experience Curiosity" [138], a serious 3D online game shown in Figure 5.3, is a brilliant and simple storytelling example achieved by combining camera and 3D model animations; developed by NASA's Jet Propulsion Laboratory using the Blend4Web [186] publishing solution, it was successfully presented at the WebGL session at SIGGRAPH 2015 [102].

5.3 User Interaction

Strongly connected to the scene, the *interaction level* is one of the most characteristic features of a Web3D solution. While the features at this level may be tailored to specific types of 3D data/content/application, we can also envision in this case some common elements and some characterizing features. User interaction in a Web3D solution happens at different levels. It includes features supporting interaction paradigms that work at object or scene level (trackballs, first-person controllers, manipulators, etc.), but it is also present as interface elements focused

on the interaction between the web page and the 3D layer (toolbars, hypertext-based functions, etc.), or it may even concern the interaction of the Web3D environment with novel technologies and input/output devices (touch surfaces, VR devices, gyro sensors, etc.).

Interaction with 3D Content

In the last two decades, several research groups have studied effective interaction with 3D objects or scenes [74, 25, 26, 86, 87], leading to design approaches able to cover most of the relevant combinations of data and application fields. Nowadays, many of these interaction approaches have been transposed online, implemented in the various available Web3D solutions.

The interaction in a 3D environment can be characterized in terms of *universal interaction tasks*: the process of getting around a virtual environment is one of these. In the variety of existing interaction paradigms deriving from this task, *exploration* approaches addressing general movements are the most exploited. In particular, two main exploration techniques are widely supported in 3D web systems:

- interactive *inspection* (rotate/pan/zoom), where the user interaction moves the object/scene in front of the camera;
- interactive *navigation* (walking/driving/flying), where the user moves the camera through the scene.

This dichotomy is often called *world in hand* (WIH) vs *camera in hand* (CHI) [206, 61].

Interactive inspection Rotating, panning, and zooming are the basic view movements, used in almost every 3D software. These operations, applicable at object level and at scene level, are also extremely common in the Web3D world because, being easy to map to two-dimensional operations, they work well with common pointing devices (such as a mouse). This paradigm generally implements the concept of a *virtual sphere* containing the object to be manipulated, also known as a *trackball*, a seminal technique designed for moving 3D objects around, based on the Chen et al. [38] Virtual Sphere and the Shoemake [170] ArcBall.

This interaction method is perfect for solutions aimed at the inspection of a single object or of simple scenes—like the viewer developed by University College, London for the 3D Petrie Museum [192] project—where the object/scene can be fully explored with the camera from the outside, looking toward a center of interest.

On the top of this basic behavior, it is possible to implement more sophisticated and customized interactive inspection modes, perhaps directly related to the characteristics of the object to be inspected, or to the application field. The 3D web viewer developed, for instance, in the Visionary Cross Project [196, 112], an international cross-disciplinary project aimed at exploiting recent developments in digital humanities to study a key group of Anglo-Saxon texts and monuments, adopts a specific trackball that allows the inspection of an elongated object present in the virtual scene only through constrained panning movements.

However, despite most systems providing an interpretation of the virtual sphere, a standardization of its components is still missing. In fact, in moving from one 3D-enabled application to another the behavior of the trackball does not usually match, and small details (such as key/button mapping or inertia) are often different.

Interactive navigation On the other hand, when the scene or the 3D model represents a 3D environment, it may be useful to let the camera viewpoint travel inside the scene, bringing the point of view to relevant and natural positions. Therefore, the other common approach for allowing the end user to navigate a 3D environment is the *walking/driving/flying paradigm*.

One interpretation of this approach, directly derived from the videogame development world, corresponds to *first-person perspective* (FPP) navigation, in which the view position of the player is usually controlled with arrow keys (for this reason, it is also known as the WASD paradigm) and the view direction is controlled with the mouse. This is implemented by all the game-oriented Web3D solutions, like the previously introduced PlayCanvas [54], but despite being versatile and powerful, it requires multiple simultaneous inputs, and it is often deemed too complex for some classes of users.

Considering that these interaction paradigms are highly specialized on the handled dataset, and on the experience that the publication wants to convey, from the beginning of Web3D multiple options have been provided by the various solutions. As an example, VRML [157] already provided the possibility to set the navigation node (the element containing information describing the physical characteristics of the viewer's avatar and viewing model) with at least four different types of user interactions: "WALK" (interactive navigation with gravity), "EXAMINE" (interactive inspection), "FLY" (interactive navigation without gravity), and "NONE" (interactions disabled, system in guided tour mode).

Interactive inspection and navigation modalities are widely distributed because of their effectiveness for general cases, but many other more specialized paradigms are available on the web. For example, 3D map software, like the popular Google Earth application [66], usually provides *POI logarithmic movement* (a targeted interaction for smooth shifting, also known as go-to/fly-to), while CAD systems and 3D editing software like WebGLStudio [4] generally use specified coordinates (x, y, z points supplied by the user via dialog boxes, or other kinds of 2D interface) for both position and orientation displacement.

Specific coordinate movements are also the basis for the *selection and manipulation* task, an interaction technique different from the approaches presented so far, consisting in choosing an object and specifying its position, orientation, and scale through explicit and direct translation, rotation, and scaling tools, called *manipulators* [18, 139, 43, 179]. Especially used during scene construction in 3D modeling software like Autodesk Maya [11] and game development environments like Unity3D [191], this paradigm results in very efficient designs of 3D scenes with multiple objects, in which users have to repeatedly realign and adjust different parts. Nowadays the implementation of this manipulation technique almost always relies on visible graphic representations of the operations on (or the state of) an object displayed with the 3D model and able to control it via user interactions [180].

Interaction with 3D content may also happen on another level, i.e. characterizing the system through the *application/system control*

task, which describes the interactions in a virtual environment in terms of communication between a user and a system, which is not part of the virtual environment. This technique, widely exploited in 2D "point-and-click" *WIMP* (windows, icons, menus, pointer) graphical user interfaces, has also been adapted to 3D applications, where control interface components are placed in a conventional 2D interface presented in screen space on a 2D plane called a *HUD* (head-up display). This WIMP application control approach splits the interface into two parts with very different interaction metaphors, since the navigation and manipulation functionality is accessible through the 3D scene while the rest of the environmental controls can only be interacted with through the screen space interface overlaying, or side-by-side with, the 3D scene.

While the variety of all these possibilities is a strength, it makes the learning curve steeper for more naïve users, who usually find it difficult to manipulate and interact with 3D environments. Quite often, this is the result of a user interface that is not properly designed, and is unable to efficiently associate these paradigms with the represented scene, or is unable to conform the interaction to what is expected in the publishing target ecosystem.

Web-Based Interfaces

In an ideal world, the selection of the best interaction mode should be tightly connected with the 3D content handled and with the publishing purpose. In a Web3D system, however, it should also be tailored to the characteristics of the web, possibly adopting techniques that are optimal for the *hypertext-based* web interface.

From a research point of view, and focusing on the more recent years, Jankowski has probably been the most active in trying to bring together hypertext technology and interactive 3D graphics. He aimed at clarifying some of the foundations of 3D web user interface design [83], focusing on an understanding of the fundamental tasks users may be engaged in while interacting with web-based 3D virtual environments, and then introducing his interface management approach. His work proposes a dual-mode user interface [84, 85] that follows the usual hypertext-based interaction mode, with the 3D scene embedded in the

hypertext, but also exploits an immersive 3D mode, which moves the hypertextual references directly into the 3D scene.

Indeed, although hypertext-based web interfaces are optimal in linking together web and 3D (implementing a step toward the essential integration between 3D and other media, as we will see in Section 5.4), in some situations they may not be enough to provide full interaction with the 3D scene. In these cases toolbars and other clickable graphical commands/elements/buttons immersed in the virtual scene (or, as we have already seen, superimposed in screen space) are a practical solution. This is nowadays provided by a large number of systems: Autodesk Tinkercad [12], Clara.io [58], Archilogic [10], etc. They usually make available a set of actions to help interactions with the 3D scene, orienting the user in using the adopted exploration paradigm (without preventing the joint exploitation of the hypertextual approach).

Two representative examples of this class of additional elements that enable improved interaction are *overview-plus-detail* components (allowing simultaneous display of an overview and a detailed view of an informative space) and *orientation widgets* (which help to address the problem of disorientation, especially in WIH interfaces). Both of these are supported and exploited in a good number of Web3D solutions.

Two-dimensional *minimaps* are a good example of the first group of overview-plus-detail elements. Often superimposed in a corner of 3D viewers to provide interactive visualization of a planar dataset (terrains, floor plans, geographical systems, etc.), these alternative media support easier transitions between different locations of interest, providing improved self-localization of the user in the space represented; see the Potree [164] example in Figure 5.4.

An example of an orientation widget could be the *3D ViewCube* element. Introduced by Khan et al. [98], this cube-shaped component is both an orientation indicator and a controller, and is generally placed in a corner of the scene window; the Smithsonian X3D [173] is an example of a 3D web viewer using this interface component (see Figure 5.5).

Additional interface elements like these can be also exploited by Web3D solutions to develop and provide specialized features addressed to specific application domains, some of which often require more technical or analytical presentation tools. A system designed to present

The version of record is available at: http://dx.doi.org/10.1561/0600000083

Figure 5.4: The Potree [164] basic viewer showing a running example of a 2D minimap (in the top left corner of the image). This additional interface element is useful to georeference the 3D model, but also to provide visual feedback of the camera (represented by the triangle shape in the map) position and orientation in the virtual space. Minimap elements refresh their position every time the user interacts with the 3D scene.

Figure 5.5: Smithsonian X3D [173] is an example of a 3D web viewer using the ViewCube interface component (in this case presented at the top right corner of the 3D rendering window). The 3D cube widget rotates synchronously with the 3D model; it has the primary purpose of providing the user with feedback on the orientation of the 3D scene. The component can also be used to directly drive the scene interactions (by rotating the 3D cube, the user applies a corresponding rotation to the 3D scene).

cultural heritage 3D artefacts, for instance, usually needs features able to support annotations and metadata, as, for example, in the Aton front-end software [195], or even technical measurements and visualization of sections like those provided by the aforementioned Smithsonian X3D platform [173] or the 3DHOP framework [199]. A solution addressing the handling of 3D chemical structures instead, for example, should possibly provide tools for visualizing vibrations, orbitals, schematic shapes, and symmetry operations, as well as features for assisting the management of molecules, crystals, and materials—as in JSmol [90], a browser-based biomedical viewer.

A general issue with these screen space controllers is often related to the user immersion in the 3D context [74]. Indeed, requiring a switch from interacting directly with 3D objects to indirectly interacting with them, these interfaces may easily lead to loss of user engagement (a relevant problem in more immersive 3D applications). To overcome the "cognitive distance" due to mapping 3D tasks and 2D control widgets in a 3D space, van Dam [194] introduced new user interfaces, not dependent on classical 2D widgets such as menus and icons, called "post-WIMP," which in recent years have also reached the Web3D world [19]. These interaction techniques strongly rely on the latest input/output (I/O) devices and technologies such as gesture and speech recognition, eye/head/body tracking, etc.

Specialized I/O Devices

Modern 3D web environments exploit common general-purpose hardware devices like the mouse and the keyboard to support user interaction, but also more recent *input/output technologies*, like touch or multi-touch input surfaces. The extremely rapid market penetration of the latter devices through the widespread diffusion of smartphones and tablets, is redefining the way people interact with digital content. Now, thanks to the last generation of mobile devices equipped with browsers supporting WebGL, this technological evolution is also beginning to involve the 3D medium.

Nevertheless, most of the interaction methods in the Web3D environment derive from (older) PC interfaces. For this reason they are

undergoing a radical redesign process [121] in order to also enable touch-based input devices. However, if it is true that these novel technologies open up new possibilities, they also lead to new challenges and issues.

For instance, touch inputs favor direct and fast interaction with the manipulated content, but at the same time introduce some constraints: with respect to a mouse and keyboard, they are much less expressive (even considering gestures, they are a long way from the multiple keys and modifiers of the traditional input devices) and have lower precision (mouse positioning can be pixel perfect; finger touch cannot). Moreover, they also require creating versions of the source code to enable switching between multiple input/output device configurations, which is an open issue particularly relevant for systems, like Web3D software, required to operate on very different hardware platforms; the Seo et al. [167] work with super multi-view autostereoscopic displays is confirmation of that.

Nevertheless, touch-based systems represent just the tip of the iceberg of new I/O technologies, especially considering the increasing number of novel devices, sometimes specifically designed for 3D applications, that are making it practically mandatory to redefine the obsolete WIMP paradigm. Among these, there are new devices supporting *gesture-based interfaces* by means of fingers, hands, or body motions, which often do not require touch contact to generate input signals, or devices for enabling *immersive virtual and augmented reality*, or again devices providing access to the wide set of sensors (gyros, accelerometers, etc.) available nowadays.

All these new I/O methods, after being successfully tested in research applications—like Wingrave et al. [209], which presents techniques for exploiting the motion-sensing capabilities of a console controller, the Nintendo Wiimote, to enable a 3D user interface—and standalone devices—like Gravity Sketch [70], a 3D creation tool focused on mobile and VR platforms that offers an innovative and intuitive design experience—are now arriving on the web with device-mapping JavaScript libraries. A brilliant example of that can be found in Kwan [107], which describes how to use the Leap Motion device [30], a low-cost commercial hardware device supporting hand/finger tracking, inside the Three.js [32] ecosystem.

Among these libraries, a good number are still working at the level of a draft specification for the web. This is the case for the DeviceOrientation [132] and DeviceMotion [131] event handlers for accessing orientation and motion sensors directly within the browser, or for the WebVR API [133] for providing support in generating *stereo-pair images* (two offset images that combine in the brain to give the perception of 3D depth) for virtual reality devices like the Oculus Rift [142] and HTC Vive [80] head-mounted displays.

However, despite that, the chance to exploit new devices and I/O methodologies has been taken by several platforms, and it is not unusual today to have access to Web3D implementations able to support them. Some examples are Patches [202], an online programming node-based editor expressly designed for building WebVR experiences, or Parallax [184], a cross-platform Java 3D software development kit (SDK) that provides in its demo the possibility to switch between several different 3D effects: *anaglyph*, which encodes each eye's image with filters of different colors to allow perception of a three-dimensional scene by composition), *parallax barrier*, which consists of a series of spaced slits allowing each eye to see a different set of pixels to create a sense of depth through parallax, and the already mentioned *stereo pairs*.

5.4 Multimedia Integration

Over the years many initiatives approached the integration of 3D data in the standard web publishing ecosystem. Several of them failed to reach their goal, mostly because they were focused on managing 3D content alone, keeping it isolated from other multimedia layers. Nowadays, people working on 3D data online are aware of the need for complete integration of the 3D content in the web environment. Consequently, they are also aware of the importance of establishing bidirectional channels able to logically and functionally link the 3D layer with the other media typically composing a web page. Following this idea, this section will describe all the features aimed at transforming a simple embedding of 3D data online in a real integration of 3D content on the web, discussing and comparing the strategies adopted by some

pioneering Web3D solutions to breach the HTML CANVAS borders like, for instance, hotspot deployment, annotation possibilities, etc.

In designing a web page it is possible to simply spatially arrange the various media inside it, but this would be a very limited and trivial take on the "multimedia" approach. While the arrangement of more classical media like text and images is fluid and shaped by the analogy with the composition/layout of a physical page, less static media like videos and 3D are often enclosed in rigid boundaries (or on dedicated single-page viewers). In particular, the integration of 3D is problematic, because the view area represents a window over a 3D space, while the rest of the web page has a 2D nature.

A way to effectively connect the different media on the page is to try to exploit hypertextual links to connect them, thus achieving a *trans-media storytelling* behavior. Hyperlinks can be considered the game-changing component of the web, able both to transform a linear and passive supply of information in an interactive navigation and to convert from spatial to logical the relationship between the elements on the page.

To enable this vision, a Web3D viewer should define a set of elements that may be useful to connect its components, events, and states with the rest of the web page, to provide the following, for instance.

- A 3D analog of a hypertextual link (e.g. a clickable geometry, a shape, or a marker) visible/highlighted in the 3D scene, that should be easily visually identified by the user.

- A series of exposed functions able to change the state of the viewer (e.g. a `change_model_visibility()` method, to modify the visibility of an object in the 3D scene). These functions may be called by other components of the web page to drive view/camera controls (maybe using the camera animations introduced in Section 5.2), data behavior, visual appearance, etc.

- A series of exposed functions able to read the current state of the viewer, or access some of its data (e.g. an `is_model_visible()` method, to read the visibility information of an object in the 3D scene). These functions may be used by other parts of the

web page to display 3D information contained (or created) in the viewer.

- A series of function hooks to track the events generated by the viewer (e.g. an `on_change_model_visibility()` method, to check if an object in the 3D scene changes its visibility state).

Through these elements a publishing system can make available different strategies to integrate 3D and web content, for example proposing *hyperlink-based schemes* that let the web page manage the 3D scene from outside the CANVAS. Exploiting this technique, the aforementioned functions would be called by HTML elements, making it possible to control the viewer behavior or composition from the web page content. For instance, in the case of a single-model 3D scene associated with external textual information, it could be possible to connect the action of clicking a text area that describes a detail of the presented 3D element to a change in the position of the element, with the aim of showing the detail described. Or again, in case of a 3D model gallery published in a web page also containing pictures of these models, it may be possible to link a mouse-over event, triggered when the mouse pointer enters one of these pictures, to the retrieval of the related 3D model from the gallery. Among the solutions allowing this level of multimedia integration are two previously introduced systems, 3DHOP [199] and the discontinued ShareMy3D [9]. This strategy relies on a basic set of JavaScript functions to drive the 3D scene through web page components external to the CANVAS area. But obviously, this kind of hyperlink-based scheme is not the only way to integrate 3D content and other media in a web page.

Nowadays, the dominant approach, an alternative to the one previously presented, pursues an inverse path. It consists in bringing the other media into the 3D scene, rather than migrating the viewer commands outside of it. This technique makes use of the Web3D *immersive interfaces* introduced in Section 5.3, designed to embed or to superimpose graphical elements directly as floating elements in the 3D scene. As stated by Jankowski and Decker [84], in these interfaces 3D graphics are the main information carrier, able to provide to the content-consumer

both the freedom to navigate the 3D environment and the hypertextual-like data (directly immersed in it).

The implementation of this approach for multimedia integration relies on the possibility of attaching, at the scene level, information to the 3D environment and 3D objects. This mechanism is the basis of *annotation/hotspot* systems, features currently offered by many Web3D solutions such as A-Frame [130], BabylonJS [34], Blend4Web [186], PlayCanvas [54], etc. These systems provide notes to "stick" on 3D models, useful for adding information to a specific part (see Figure 5.6). Each note usually has a position, a camera position, a title, some multimedia content, and, sometimes, also an order in a numerical list. Users can view the attached information by interactively recalling these notes during navigation. In response to this action, the additional content is presented (typically in screen space) adjacent to the associated object.

The connection between multimedia data and a 3D scene can be achieved by adopting different strategies. One of these consists in placing in the 3D environment *labels* (or other hyperlink-based 2D elements) connected to objects of interest via placeholders (usually anchors to specific 3D points in scene space). Generally these clickable components are HTML (or HTML derived). This allows Web3D solutions to exploit, without much effort, the native interaction methods provided by the markup language. Another possibility is to transform the 3D objects in the scene into *clickable instances*. This strategy does not need placeholders, but to drive the user interaction it requires that the 3D system provides a set of event handlers specifically related to the 3D environment (this feature is usually only provided by the more complete Web3D solutions). Implementations of the labels and clickable instances approaches can be respectively found in Cl3ver [82], a software-as-a-service (SaaS) application to edit and display 3D content online (see Figure 5.6), and WhitestormJS [31], a JavaScript framework for simplifying 3D web publication deployment—it adds physics and post-effects to the Three.js [32] technology.

One of the biggest weak points of the multimedia integration methods introduced in this section concerns their usability, intended as ease of setup in the publishing stage. All the methods presented require the content creator to spend a significant amount of time and effort in

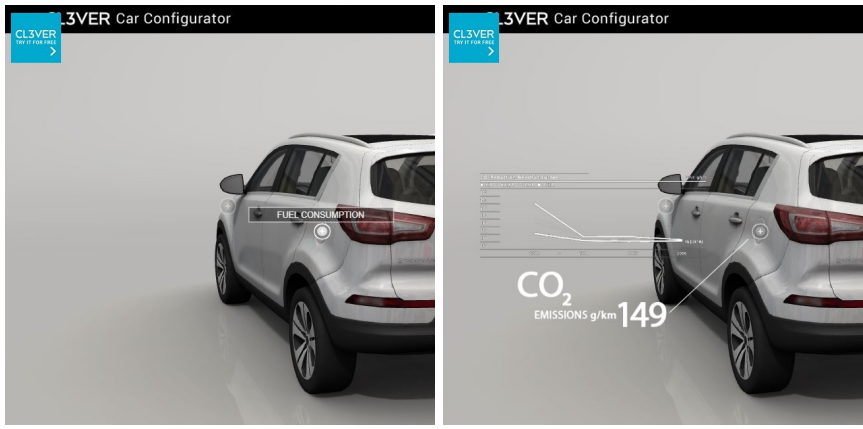

Figure 5.6: Multimedia integration via an annotation system in the Cl3ver solution [82]. In this example, clicking on the circular 2D graphical placeholder anchored to a specific 3D point in the scene space (left) presents the user with additional information in screen space as a superimposed HTML element (right).

uploading, linking, and configuring the additional layers of information. While this may not be a problem for a professional content creator, a casual one may find this work overwhelming.

Many systems, in an attempt to allow the coexistence of multimedia integration and democratization of use (both pillars of modern Web3D), have resorted to specific *authoring tools*, able to interactively guide the connections between 3D and other media step by step. This approach seems to meet the intent perfectly, and has been fruitfully implemented by systems like the already mentioned Autodesk ReMake [14] or Sketchfab [171]. In these solutions the authoring tool covers all the areas related to web publishing. However, the implementation and maintenance of an authoring system are cumbersome tasks, and require a *server infrastructure* able to run them, narrowing the feasibility of this approach to the solutions deployed as services (or, alternatively, to standalone offline software equipped with a wizard).

How to support these needs remains an open issue for all the other self-publishing or serverless approaches. More generally, the overall informative enrichment of the 3D layer is still today a very active research field. Indeed, in recent years, several works have been addressed

at improving the technology behind interconnection systems between 3D and other media (not always specifically aimed at the web world). Russell et al. [161], for instance, have explored the possibilities provided by automatic annotations; Lehmann and Döllner [111] have tried to increase the potentiality of labeling 3D content in virtual worlds (not necessarily online); while more recently Seo et al. [166] have proposed a method to enhance the perception of annotations by rendering them with a 3D layout context and a camera perspective common to the related object.

5.5 Publishing Context

The characterization discussed so far has been based mostly on specific technical choices of the visualization system, considering the published data as the core of the Web3D environment. Nevertheless, to get a complete overview of the ecosystem it is necessary to take a step backward, and to look at the online *publishing* phase. This section addresses the main aspects of the publication process, including the publication modalities available (target coding, node-based, full graphical user interface, etc.); the primary purposes supported by the publishing (pure visualization, assisted content creation, collaborative editing, etc.); the kind of final experience promoted (specialized analysis, interactive presentation, social sharing, etc.); and the type of licenses adopted and the resulting permitted uses (open source, commercial, freemium, etc.).

One of the preliminary discriminating points of the 3D web publishing process is related to the *target content creator* expertise level. This results in some software systems addressed at naïve users and others designed for skilled developers. At publishing time, it is possible to choose between different development styles, ranging from *pure coding proposals* to node-based editing and block programming, until arriving to full GUI/wizard approaches.

The list of systems where the viewer is created by coding includes a wide set of software, like for instance MathBox [210], a library for rendering presentation-quality mathematical diagrams in the browser, or LayaAir [39], a dedicated open-source API for games and multimedia routine modules. These tools are usually aimed at developing more

complex or highly specialized applications (like online games or analytical tools). On one hand such solutions allow for more customization and greater control of the resulting publication, but on the other hand they require knowledge of one or more coding languages (JavaScript, HTML, TypeScript, etc.), and for this reason are generally targeted at specialized/professional users.

Less demanding, from the creation/setup point of view, are node-based solutions such as Goo Create [64], a complete 3D authoring platform for cloud-based 3D content creation, or CopperLicht [8], an open-source 3D library equipped with a full 3D world authoring editor. These systems do not require writing code (even if they often include sections where this is still possible) but provide the content creator with a drag-and-drop interface for visual programming, where the scene, its behavior, and the interactions are defined by assembling and configuring predefined elements.

Finally, there are solutions which do not require the use of a programming environment, like Koru [27], an authoring software that helps to prepare 3D models and to export the result online, or Kokowa [216], a publishing platform for non-programmers to create and share 3D and VR spaces. In these tools, a GUI is used as a wizard setup to drive the publishing step by step (typical of SaaS systems). These approaches do not require any kind of coding, and are thus ideal for naïve users; the drawback is that they have a more general purpose orientation and do not allow full personalization of the published viewer.

An alternative categorization of Web3D systems can be based on the *basic publication aim* they promote, in other words their primary purpose/target for being published. This means differentiating, for example, solutions more focused on *digital content creation* (DCC) from others specifically addressed at *pure visualization*, characterizing all of them depending on the degree of DCC support provided (intended as real-time object/scene editing possibilities).

For a number of systems, the presentation feature is predominant with respect to the digital content design component (very basic and often only related to the customization of the scene). Kubity [106], a cloud-based 3D player mainly intended to enable end users to experience models in AR/VR online, is a perfect example of this. In many other

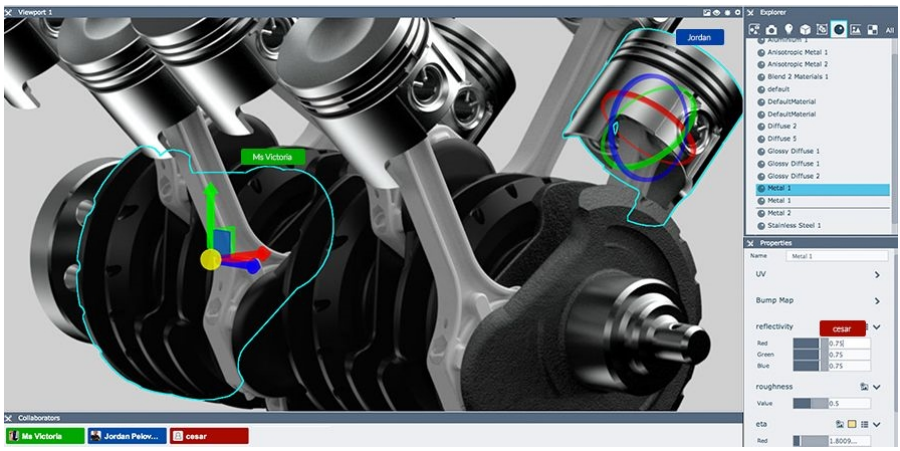

Figure 5.7: Collaborative DCC implementation in the Lagoa [45] Web3D system. This cloud-based solution aimed at CAD design guaranteed performant real-time multi-user synchronized scene editing.

more technical and specialized cases, the ratio between presentation and creation support is inverted. In these latter solutions the web viewer has the primary purpose of supporting the content creation process, whose final result may not even be public online publication (as in the case of the MeshLabJS [200] editing platform). Such systems are often aimed at technical users and usually provide support for collaborative DCC. A good example of one of these was the Lagoa platform [45], a cloud-based system for online CAD design, shown in Figure 5.7; the company was recently acquired by Autodesk to help in the development of its own, similar, product, Fusion 360 [13]. Of course, between these two extremes there exists a full set of hybrid systems in which the proportion of the ingredients is more balanced. This group includes solutions focused on web visualization, but at the same time able to provide technical features for assisting digital content creation, for instance supporting modeling and animation—the already introduced Clara.io platform [58] being a representative example.

An additional categorization of modern Web3D software may be driven by their specialization. Nowadays these systems are more and more designed to satisfy specific needs, or to focus on particular goals, than following the idea of general-purpose visualization. This can lead

to a characterization based on the *target uses* of the analyzed platforms; that can be applied not only to the basic aims of a publishing solution, but also to its ultimate target purposes, i.e. the type of *final experience* supported.

For example, as already stated in Section 5.3, solutions aimed at specialized technical fields generally provide *specific presentation or analysis components*. For example, systems aimed at *geographic information system* (GIS) visualization, like Cesium [36], a JavaScript library for 3D globe and map creation, often assist the development with a set of features (geo-vector formats, map projections, Columbus view, large distance and coordinate handling) hard to find in other contexts. Likewise, solutions designed for creating games and similar interactive 3D applications, such as Unity3D [191] or the Unreal Engine [55], another multi-platform game engine equipped with a WebGL exporter, offer very specialized components like a physics engine, networking, or even spatial audio. In Section 5.3 we also saw that characterizing features connected to specialized types of experiences may be related not only to particular presentation or analysis elements, but also to *distinctive interaction paradigms*. For example, a Web3D solution designed to present architectural models will exploit the first-person navigation mode, to let the user walk through the 3D scene. The already introduced Archilogic [10] system is an example of that category.

Interpreting the concept of supported experience in the broadest sense, some side elements of a publishing platform can also contribute to this level of characterization. For instance, solutions aimed at online sharing of 3D creations often include social media tools (with linking or embedding capabilities) and virtual spaces (model galleries or showcases) for supporting content dissemination. These *community-oriented features*, external to and seemingly disconnected from the rendering context, actually represent a characterizing factor as important as the others in the virtual web environment, able to influence the choice (and also the success) of a Web3D platform; the Sketchfab [171] case can be considered a representative example.

A final high-level classification of the available solutions can be drawn by considering the *costs and distribution licenses* of the reviewed

systems. As for any other software product, the usage terms and costs are important characterizing factors.

For example, the choice between a *client-based* and a *server-based* platform can influence the costs of a specific solution. As we have seen in Section 5.1, the need for server-based capabilities (storage, computation, user management, database support) for some of the existing solutions often leads to non-trivial expenses (for content creators interested in self-publishing), or to the payment of a subscription (for end users of commercial services). From the point of view of users, current trends are generally toward combinations of basic services for free (with some usage restrictions) and more specialized features that become optional and fee-based (with fewer or no restrictions). The PlayCanvas system [54] is just one of many adopting this policy.

Some online platforms are offered free of charge, even if they require high maintenance costs, because they are used as a "beachhead" to promote ancillary products or other paid services. As an example, this is the case for Thingiverse [172], created and maintained by a 3D printer manufacturer to provide its users with usable content, thus promoting the sales of 3D printers, or ReMake [14], created and maintained by Autodesk as a way to introduce their software to possible clients.

Fortunately, thanks to its wide landscape, the Web3D field has solutions offering all the different possibilities, with *open-source* [e.g. CopperLicht: 8], *freemium* [e.g. Sketchfab: 171], or *commercial* [e.g. Cl3ver: 82] solutions, and licenses like Apache [e.g. BabylonJS: 34], MIT [e.g. A-Frame: 130], or GPL [e.g. Blend4Web: 186].

The version of record is available at: http://dx.doi.org/10.1561/0600000083

6

Discussion

As discussed in Chapter 4, a single, organic description of the Web3D panorama is complex, due to the necessity of building a low- to high-level categorization of the available solutions. For this reason, we have tried to look at this heterogeneous landscape from two different points of view. In the previous chapter we presented an analysis of the different features, organizing them into functional macro-classes. Conversely, in this chapter we try to work orthogonally, synthesizing the topics introduced in Chapter 5 into a general reference scheme, with the aim of using it to outline the profile of the various existing approaches/systems, connecting each of them with their characterizing features.

These same features will finally be projected on a representative number of application fields that benefit more than others from Web3D technologies, in order to assess the ideal approach for each of them, and also to enrich our classification pattern.

6.1 Classification

In order to provide a map enabling the assessment of the Web3D solutions introduced so far, we propose a classification based on the features previously discussed. This reference scheme, connecting these

features to fully qualified products, is also useful to evaluate how these characteristics might apply to real-world 3D web systems. Thus, our classification is primarily based on a representative selection of the characterizing points introduced in Section 4.2, gathered in a table, to provide an overview of the amount of support given to each of them by the reviewed systems.

Ideally, we could have mapped each feature onto each of the Web3D solutions reviewed so far, but we decided not to follow this avenue mainly because of the difficulty in representing in a visually accessible table the large number of combinations eventually obtained (approximately a hundred, since we would intersect around twenty features with the approximately fifty systems introduced).

Moreover, consideration of the effective usefulness of explicitly referring to existing solutions has led us to avoid this kind of representation, since it would be condemned to become quickly obsolete in a technological environment in rapid evolution such as Web3D, where new solutions are systematically released and old ones disappear quickly—suffice it to say that just during the drafting of the present review three different software systems, Lagoa [45], Autodesk ReMake [14], and ShareMy3D [9], became unavailable for different reasons.

We therefore decided to proceed in a different way, and, exploiting the schematic representation proposed in Figures 4.2 and 4.4, we decided to simplify and synthesize the reviewed approaches and the features harmonizing them into a representative number of reference classes (Lib/Lib++, Tool/Toolkit/Framework, App) and characterizing levels (Data, Scene, Interaction, Integration, Publication). Table 6.1 shows the results of this classification, giving an overview of the surveyed software based on the aforementioned criteria. The scheme outlines which publishing feature or technique is supported by which application domain, with the amount of support expressed as a range between a couple of values selected among: none [–], low [*], medium [**], and full [***] support.

Even though this schematic representation leaves out some very broad (but no less important) features, like licensing and cost factors, it is still able to exhaustively present the options that a practitioner has at hand when 3D content for the web has to be created, providing an

Table 6.1: Web3D characterizing features mapped onto the various publishing approaches. The solutions are classified according to a representative selection of the criteria given in the bulleted list in Section 4.2. The software systems and features are grouped as in Figures 4.2 and 4.4, respectively, to provide a quick overview of the methodologies and options available to a content creator. Each cell shows the amount of support given by a group of systems to each feature level, expressed as a range between "none" and "full." The elements are sorted by supported features in order to show potential trends. The gray cells represent the overall behavior of standalone end products (full GUI applications mainly working locally, characterized by access and customization features similar to low-/middle-level approaches).

		Lib, Lib++	Tool, Toolkit, Framework	App
Data	Representation types/schemes Encoding/transmission formats Pre-processing possibilities	**/***	*/***	–/**
Scene	Spatial/logical definition modes Appearance customization features Animation possibilities	**/***	*/***	–/**
Interaction	Tasks and paradigms Interface-based components Specialized I/O modalities	*/***	*/***	*/**
Integration	Hyperlink-based schemes Informative enrichment tools Authoring elements	–/**	*/**	*/***
Publication	Content creation facilities Real-time DCC modalities Specialized high-level elements	–/*	*/**	**/***

implicit understanding of the level of complexity/difficulty involved in obtaining high-quality results.

Moreover, analysis of the proposed scheme shows that it is possible to glimpse some interesting trends. In particular, scrolling through the table from the features point of view (row-major order from top to bottom), it attests that low-level functionalities (mostly Data and Scene related) are usually better handled by libraries, APIs, or local solutions, while Integration/Publication features are generally more broadly supported by higher-level systems (mainly thanks to GUIs or authoring interfaces). At the same time, moving to the systems point of view (column-major order from left to right), the table points out that, while the Lib/Lib++ and Tool/Toolkit/Framework support curve decreases going from top to bottom (i.e. from low-level to high-level features), the App trend does the opposite, significantly increasing the amount of support given to each characterizing level moving from top to bottom.

Finally, this schematic visualization is also important in confirming the point stated in Section 4.1 concerning the problems in clearly classifying the wide range of solutions in a universal fixed scheme. Seeking to represent in the table, for instance, standalone software aimed at two-step 3D web publishing (local model/scene processing and online exporting), we would get an unusual characterization. The level of support for this class of solutions (essentially full GUI applications providing customization possibilities equal to middle-level systems and data/scene access comparable to low-level approaches) could in fact be ported to the proposed scheme only by following a transversal representation. The gray cells in the table just serve to highlight those particular cases in which we can convey all these commercial end products that have to be downloaded and installed, like for example Unity3D [191] or Autodesk ReMake [14].

6.2 Application Fields

In order to specialize the reference scheme introduced in Section 6.1, we apply its features to some representative application fields. As well as providing an overview of the best-fit systems for each specific field of

application, we also want to define the full set of features that content creators could have at hand while creating specialized 3D web content. We notice that for some of these fields the solutions all tend to be of the same type, while other fields present more heterogeneous sets of solutions.

Cultural Heritage

Cultural heritage (CH) applications, and related solutions, are mainly related to the publication of content (with some degree of personalization). Usually the 3D data involved in this kind of publishing are high-resolution 3D models coming from real-world acquisitions (digitized with photogrammetry/structure from motion approaches or active scanning devices). Handling these datasets online (large size, huge complexity) requires the adoption of *multi-resolution representations* and *performant data transfer formats*, so a *pre-processing* phase is usually needed. Since the 3D models to be published often represent copyrighted objects, *IPR protection* and *infrastructure security* are relevant in CH systems that offer cloud services. Other central elements concern the possibility to *integrate informative content* into the 3D layer, and to *support innovative I/O devices* (these two aspects are of fundamental importance when building virtual museums). *Camera animations* can also be really useful, while *model animations* are not a strict requirement. Effective interactive 3D scene *inspection* and *navigation* are both mandatory features (for architecture and artwork 3D models). Conversely, *scene customization and editing* tools are not so important, due to the need to convey the proper message and to pursue high fidelity in the visualization; they can also be dangerous since they could produce model appearance modifications.

Cultural heritage is characterized by a heterogeneous set of application cases and requirements, and by a low economic value since most of the applications are implemented on a low budget. As an application field that does not attract much interest from commercial companies, there are few ready-to-use tools specifically designed for this domain. For this reason, the CH community very often has to use tools developed for other purposes and domains, such as games or animation—immediate

examples are Unity3D [191] and Unreal Engine [55]. This does not prevent some systems not specifically designed for CH, such as Sketchfab [171] or Autodesk ReMake [14], actually fitting the CH application needs quite well (to the point where Sketchfab now has a dedicated section to support web publishing by cultural institutions). Also, academic solutions that do not explicitly refer to CH, like Potree [164] and the Visual Media Service [201], are extensively used to publish CH 3D models. However, ad hoc CH-oriented solutions have been proposed, too. Mostly initiated by cultural or academic institutions, they may be either "black box" systems of restricted use, like the 3D viewers of the Petrie Museum [192] or the Smithsonian [173], or open solutions freely available and accessible on the web, like the 3DHOP framework [199].

Biomedical

Biomedical applications are typically focused on visualization, enabling the rendering of particle systems (by *ray-casting*) and volumes (by *ray-marching*). This could direct the choice of the appropriate publishing platform to systems capable of handling these datasets on the web (even if, as we have seen, triangle-based techniques could also be exploited). Due to the technical nature of these publications, the presence of *specialized analytical tools* (enabling visual and numerical data analysis, such as unit cell operations, computation of distances and angles, torsion angle measurements, etc.) plays a key role, as well as the possibility to use *interaction paradigms* tailored to inspecting this specific 3D content. For the same reason Web3D solutions providing interactive (or collaborative) *annotation systems* may be preferable. Finally, *animation* features are also extremely useful in biomedical presentations, particularly for didactic and dissemination purposes.

Among the systems presented so far, besides the more research-oriented proposals like X3DMMS [215] and MEDX3DOM [42], it is worth mentioning a couple of open-source projects like X Toolkit [212]—a framework for visualizing and interacting with medical imaging data, it provides a simple API offering native support for neuroimaging file formats—and JSmol [90], a molecular viewer for chemical structures in 3D with features for molecules, crystals, materials, and biomolecules.

GIS, Maps, and Architecture

This publishing field covers a heterogeneous set of Web3D systems, all somehow dedicated to the creation of map-based applications (at different levels of detail and complexity). Almost all these approaches require the handling of datasets structured as *LoD trees* for progressive view-dependent refinement, that probably need to be *georeferenced* (particularly in web GIS solutions). Map applications often make extensive use of *geometry instancing* for repeated objects in the 3D scene, and often exploit *camera animations* supporting "spatial" storytelling (the recent Voyager function in the popular Google Earth [66] application implements this feature). Of course, the *navigation component* (usually first-person/walkthrough) is equally important, enabling the interactive exploration of these datasets (especially for architectural models). Finally, this domain also requires a peculiar set of *specific presentation/analysis features*, like individual object picking, atmospheric element drawing, precision handling of large view spaces (avoiding z-fighting) and large world coordinates (avoiding jitter), or timeline controls for the simulation of time-varying phenomena.

Nowadays, GIS visualization and analysis on the web can be exploited using a number of interactive systems specialized for the vast volumes of geospatial data and able to provide vector graphics, surface models, and 3D buildings. There are both commercial solutions, such as GeoWeb3D [62] and GeoBrowser 3D [69], and open-source resources, like the OpenWebGlobe SDK [193] and the aforementioned Cesium [36]. Map-based Web3D approaches can be aimed not just at geographical visualization, like the popular Google Maps API [67], but also at visual explanatory generic-data analysis, like Deck.gl [188]—complex visualizations of large datasets rendered as a stack of visual layers—or Seerene [76], which provides map-based interactive analysis of source code. Map-driven visualization is also useful in platforms addressed at architecture, such as Archilogic [10] and 3D Wayfinder [1], both providing floor plans and 3D building interior and exterior exploration.

CAD

Computer-aided design solutions span a wide space, from simple to professional systems, mainly depending on the complexity of the operations and task required. The Web3D publishing solutions reflect this. Models exploited in this application field are 3D meshes, usually characterized by relative structural simplicity (in terms of the number of triangles/faces), since they are generated (and serve) in design and modeling processes. For this reason, and mostly to ensure coherence in the geometric editing, they generally rely on *single-resolution formats*. A central feature of this kind of application is certainly the *support for DCC*, both in *specialized tools* (kinematic assembly, geometric constraints, distances, angle offsets, etc.) and higher-level features like *collaborative editing* and *version control*. For more technical uses (for instance in industrial pipelines for photorealistic publication) it is fundamental to also be able to steer the 3D object/scene *appearance editing* (shaders, materials, lighting, etc.), as well as providing access to the proper *authoring tools* able to drive these specialized operations.

The landscape of solutions addressing CAD includes systems ranging from those targeted at unskilled users [e.g. Autodesk Tinkercad: 12] to those intended for professionals [e.g. Lagoa: 45]. Some of the proposed approaches are oriented more toward editing/modeling, like Autodesk ReMake [14], while others are more focused on the publishing stage, e.g. Koru [27]. Commercial solutions, like ThreeKit [78], and free software, like OpenJSCAD.org [134], a JavaScript web interface for programmatic modeling, are both provided.

3D Printing

Web applications for 3D printing are, generally, easy-to-use platforms, where the aim of publication is mostly focused on supporting user-uploaded sharing (or selling) of printable content. These systems, mostly implemented as services, provide *hosting services* and a number of related high-level features like *IPR management* strategies, *user profiling*, and *community-oriented tools*. Since printable objects must be a single simple item, Web3D printing solutions often require bare-bones viewers aimed at *pure visualization* of a *fixed-resolution* model.

3D printing models have gained momentum in recent years. Due to this explosion of interest, a lot of SaaS sharing platforms have been released where end users can interactively visualize content before downloading it. These systems are generally marketplaces for uploading and selling 3D object files [e.g. Pinshape: 124], but there also exist solutions implemented as open virtual spaces where they can be shared for free, like MyMiniFactory [137] or Thingiverse [172]. The Threeding platform [183], in addition to allowing the sharing of 3D printing models and files (both free and paid), also provides a service for on-demand 3D printing (for users without a 3D printer). The Shapeways [168] startup is instead completely based on the concept of being a printing (and selling) service, providing users with the possibility of uploading 3D printable files and printing the objects (in over 55 materials and finishes) for themselves or for others. However, it is not just web service platforms that are characterized by 3D printing features. Online CAD systems, for instance, often provide simplified creation features to make it possible to print the designed models—Autodesk Tinkercad [12], Leopoly [113], and BlocksCAD [22] belong to this category. Also, some less specialized solutions may provide support for standard (STL) 3D printer file formats and the preparation of 3D models for printing, like, for instance, Autodesk ReMake [14].

Games

Web3D solutions aimed at game development are generally systems able to handle elaborate 3D scenes made of a large number of modeled geometries. For that reason they often provide features focused on *complex scene composition* (hierarchical geometry instancing). Particular emphasis is placed on components for customizing *scene appearance* (rendering/shading processes), *animation* (both camera and model), and *exploration* (mainly virtual environment navigation). Of course, building a game experience also plays a central role, requiring access to *specialized features* like physics, networking, and audio control.

Several solutions can be exploited or adapted to Web3D game development. Almost all the approach typologies are available, from low-level engines like PhiloGL [17] or KickJS [140] to GUI-based applications

like Goo Create [64] or CopperLicht [8], and from more general systems like BabylonJS [34] or Blend4Web [186] to highly specialized software like PlayCanvas [54] or LayaAir [39]. Even if in this specific field the most popular solutions remain systems not specifically designed for Web3D, like Unity3D [191] or the Unreal Engine [55]—both are just equipped with a WebGL exporter—it is easy to find web-targeted applications like, for instance, Turbulenz [187], a modular 2D/3D framework focused on HTML5 game development for desktops and mobile devices, or Voxel.js [203], a game-building toolkit that makes it easier to create 3D voxel Minecraft-style games.

The version of record is available at: http://dx.doi.org/10.1561/0600000083

7

Conclusions

Web3D is certainly an intriguing world. Its story has changed suddenly, evolving from a slow and stagnant past to a rapidly evolving dynamic present. It is important to note that immediately after the introduction of WebGL the Web3D domain was perceived as a quasi-mature market by software companies. This generated instantaneous and conspicuous investment resulting, in a very short time span, in a series of commercial-grade systems and software solutions. The sudden availability of many nice tools is very different from what happened in the past with other kinds of media, where their web exploitation followed a trial-and-error approach by players of many different sizes and wealth (from underdogs to large corporations). This rapid growth has shaped the direction of evolution and development of this field and has created de facto standards in terms of formats, interface paradigms, and available features. This market-driven nature is also demonstrated by most of the available tools and systems not having a reference scientific publication, even though they present innovative technical solutions.

As stated at the beginning of this survey, the fluidity of the web context makes it difficult to outline a clear and rigid classification of the Web3D denizens. To overcome this issue, we organized our analysis

by isolating some of the prominent and recurring features that are used in the different tools and applications, then by grouping them by their scope and functionality. Users may find this characterization helpful in choosing a system suitable for publishing their data, but also in understanding the underlying technology and having an outline of the available features and mechanisms that are nowadays considered standard. Developers can find in this review an overview of the different issues and algorithmic/software solutions, with a mapping to several application fields.

Looking at the wide variety of tools and solutions now available in the Web3D panorama, it may seem that every need has already been covered by existing tools. However, on closer inspection, many "missing links" appear obvious. The rapid development of the field has left out many niches for specialist and technical users, and has restricted some of the analyzed features to specific fields and associated implementations, neglecting others. This review may be helpful for readers interested in finding these empty spots.

7.1 Research Directions

The evolution of the Web3D ecosystem has exploited the results of various research fields, from geometry processing to rendering, from data compression and streaming to web protocols, from human–computer interfaces to metadata and ontologies. It is impractical to list all the possible research directions of such a wide field.

Clearly, the design of future Web3D technologies will offer space for further development of basic geometry processing algorithms and methods, as well as improvements in data management/compression/streaming methods—see, for example, the recent stable release of the glTF asset delivery format [101]. User interaction with a 3D space is still a field where things can (hopefully) improve, maybe in conjunction with the technological advance of new output devices (HDM style) and the development of new 3D input devices. However, these are trivial forecasts.

Looking at more interesting aspects, the IPR problem described in Section 3.3 is still largely unexplored. Most of the available solutions and

approaches are simple retargeting and hijacking of existing approaches that have been developed for non-web platforms or non-3D media. Here there will be plenty of space for new approaches and developments. This will not only be a "technical" issue, but will also affect the commercial approach, legislative considerations, and user interaction/access aspects.

Another promising field is the integration of Web3D with novel devices. Most Web3D solutions already support mobile devices, but mostly with basic behavior. Truly exploiting their hardware and, more importantly, the different behavior that users have when interfacing with a mobile device, is still a direction to be explored. The same is true for VR/AR output devices (which are also still evolving in their PC-based incarnations). Natural interaction devices and the new generation of low-cost 3D measuring devices (integrated into mobile devices) also need to be mapped effectively to Web3D applications.

Considering instead the Web3D access paradigm, we have witnessed another trend. As in many other fields, big software houses are experimenting with using the web as a platform for complete software solutions, offering cloud-based versions of their software. This has multiple advantages: easier and stronger management of licenses, instantaneous software updates, and the possibility of relying on cloud computation on dedicated servers and thus having the Web3D application as just a front end to a remote service. Examples of this trend are the SketchUp software [185], currently in beta with a fully online version, or the Autodesk ReMake/ReCap Photo tool [14], where the web platform is the front end to a cloud-based 3D geometry processing software pipeline.

Finally, the integration of other multimedia elements like images or videos into a 3D environment is currently represented only by simple, non-extensible solutions. We believe that a tool providing features to allow immersive, integrated visualization of multiple data types (not just 3D) would open outstanding opportunities in terms of user experience and entertainment.

The version of record is available at: http://dx.doi.org/10.1561/0600000083

Acknowledgments

The research leading to these results has been partially supported by the European Union H2020 Programme under grant agreement no. 654119 (EC "PARTHENOS" project), and by the International Bilateral Joint Lab CNR–CNRS ("Mass3DxCH," 2017–2019).

References

[1] 3D Technologies R&D. 3D Wayfinder, 2012. https://3dwayfinder.com.

[2] 3D Technologies R&D. Frak Engine, 2012. https://github.com/evanw/lightgl.js.

[3] Adobe. Stage 3D, 2011. http://www.adobe.com/devnet/flashplayer/stage3d.html.

[4] J. Agenjo. WebGL Studio, 2013. http://webglstudio.org.

[5] J. Agenjo, A. Evans, and J. Blat. WebGLStudio: A pipeline for WebGL scene creation. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 79–82, New York, NY, USA, 2013. ACM.

[6] A. L. Ahire, A. Evans, and J. Blat. Animation on the web: A survey. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, pages 249–257, New York, NY, USA, 2015. ACM.

[7] P. Alliez and M. Desbrun. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 195–202, New York, NY, USA, 2001. ACM.

[8] Ambiera. CopperLicht, 2010. http://www.ambiera.com/copperlicht.

[9] F. Anfinsen and K. Hope. ShareMy3D, 2015. https://web.archive.org/web/20160125053538/https://www.sharemy3d.com/.

[10] Archilogic. Archilogic, 2014. https://archilogic.com.

[11] Autodesk. Maya, 1998. https://www.autodesk.com/products/maya.

[12] Autodesk. Tinkercad, 2011. https://www.tinkercad.com.

[13] Autodesk. Fusion 360, 2013. https://www.autodesk.com/products/fusion-360.

[14] Autodesk. ReMake, 2015. http://remake.autodesk.com.

[15] J. Behr, P. Eschler, Y. Jung, and M. Zöllner. X3DOM: A DOM-based HTML5/X3D integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, Web3D '09, pages 127–135, New York, NY, USA, 2009. ACM.

[16] J. Behr, Y. Jung, J. Keil, T. Drevensek, M. Zoellner, P. Eschler, and D. Fellner. A scalable architecture for the HTML5/X3D integration model X3DOM. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 185–194, New York, NY, USA, 2010. ACM.

[17] N. Belmonte. PhiloGL, 2011. http://www.senchalabs.org/philogl.

[18] E. A. Bier. Skitters and jacks: Interactive 3D positioning tools. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, I3D '86, pages 183–196, New York, NY, USA, 1987. ACM.

[19] J. Biström, A. Cogliati, and K. Rouhiainen. Post-WIMP user interface model for 3D web applications. 2005. Helsinki University of Technology.

[20] Bitmanagement Software. BS Contact, 2002. http://www.bitmanagement.com.

[21] Blender Foundation. Blender, 1995. https://www.blender.org.

[22] BlocksCAD. BlocksCAD, 2017. https://www.blockscad3d.com.

[23] A. Blume, W. Chun, D. Kogan, V. Kokkevis, N. Weber, R. W. Petterson, and R. Zeiger. Google Body: 3D human anatomy in the browser. In *ACM SIGGRAPH 2011 Talks*, SIGGRAPH '11, pages 19:1–19:1, New York, NY, USA, 2011. ACM.

[24] E. S. Boese. *An Introduction to Programming with Java Applets*. Jones & Bartlett Learning, Burlington, MA, USA, 2009.

[25] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. An introduction to 3D user interface design. *Presence*, 10:96–108, 2001.

[26] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. Poupyrev. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[27] Boxshot. Koru, 2016. http://boxshot.com/koru.

[28] P. Brunt. GLGE – WebGL for the lazy, 2010. http://www.glge.org.

[29] D. Brutzmann and L. Daly. *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann, Burlington, MA, USA, 2007.

[30] M. Buckwald and D. Holz. Leap Motion, 2010. https://www.leapmotion.com.

[31] A. Buzin. WhitestormJS, 2015. https://whs.io.

[32] R. Cabello. Three.js, 2010. http://threejs.org.

[33] C. Calabrese, G. Salvati, M. Tarini, and F. Pellacini. cSculpt: A system for collaborative sculpting. *ACM Transactions on Graphics*, 35(4):91:1–91:8, 2016.

[34] D. Catuhe and D. Rousset. BabylonJS, 2013. https://www.babylonjs.com.

[35] F. Cayre, P. Rondao-Alface, F. Schmitt, Benoît Macq, and H. Maître. Application of spectral decomposition to compression and watermarking of 3D triangle mesh geometry. *Signal Processing: Image Communication*, 18(4):309–319, 2003.

[36] Cesium Consortium. Cesium, 2011. https://cesiumjs.org.

[37] J. Chandler, H. Obermaier, and K. I. Joy. WebGL-enabled remote visualization of smoothed particle hydrodynamics simulations. In E. Bertini, J. Kennedy, and E. Puppo, eds, *Eurographics Conference on Visualization (EuroVis) – Short Papers*. The Eurographics Association, Geneva, Switzerland, 2015.

[38] M. Chen, S. J. Mountford, and A. Sellen. A study in interactive 3-D rotation using 2-D control devices. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, pages 121–129, New York, NY, USA, 1988. ACM.

[39] X. Cheng-Hong. LayaAir, 2015. http://www.layabox.com.

[40] W. Chun. WebGL models: End-to-end. In P. Cozzi and C. Riccio, eds, *OpenGL Insights*, pages 431–454. A K Peters/CRC Press, Natick, MA, USA, 2012. https://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGLInsights-WebGLModelsEndToEnd.pdf.

[41] P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, F. Ponchio, and R. Scopigno. Batched multi triangulation. In *Proceedings IEEE Visualization*, pages 207–214, conference held in Minneapolis, MI, USA, October 2005. IEEE Computer Society Press. http://vcg.isti.cnr.it/Publications/2005/CGGMPS05.

[42] J. Congote. MedX3DOM: MedX3D for X3DOM. In *Proceedings of the 17th International ACM Conference on 3D Web Technology*, Web3D '12, page 179, New York, NY, USA, 2012. ACM.

[43] B. D. Conner, S. S. Snibbe, K. P. Herndon, D. C. Robbins, R. C. Zeleznik, and A. van Dam. Three-dimensional widgets. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, I3D '92, pages 183–188, New York, NY, USA, 1992. ACM.

[44] D. H. Curtis. *Flash Web Design: The Art of Motion Graphics*. New Riders Publishing, Thousand Oaks, CA, USA, 2000.

[45] T. Da Costa. Lagoa, 2011. https://web.archive.org/web/20180209211707/http://home.lagoa.com:80.

[46] B. P. DeLillo. WebGLU development library for WebGL. In *ACM SIGGRAPH 2010 Posters*, SIGGRAPH '10, page 135:1, New York, NY, USA, 2010. ACM.

[47] M. Di Benedetto, F. Ponchio, F. Ganovelli, and R. Scopigno. SpiderGL: A JavaScript 3D graphics library for next-generation WWW. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 165–174, New York, NY, USA, 2010. ACM.

[48] M. Di Benedetto, F. Ganovelli, and F. Banterle. Features and design choices in SpiderGL. In P. Cozzi and C. Riccio, eds, *OpenGL Insights*, pages 583–604. A K Peters/CRC Press, Natick, MA, USA, 2012.

[49] J. Dirksen. *Learning Three.js: The JavaScript 3D Library for WebGL*. Packt Publishing, Birmingham, UK, 2013.

[50] J. Dobos and A. Steed. 3D revision control framework. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 121–129, New York, NY, USA, 2012. ACM.

[51] B. Drozdz. J3D – Unity3D to Three.js exporter, 2011. https://github.com/drojdjou/J3D.

[52] P. Du, Y. Song, and L. Deng. A real-time collaborative framework for 3D design based on HTML5. In *Proceedings of the 20th International IEEE Conference on Computer Supported Cooperative Work in Design*, CSCWD '16, pages 215–220, New York, NY, USA, 2016. IEEE.

[53] F. Dupont, T. Duval, C. Fleury, J. Forest, V. Gouranton, P. Lando, T. Laurent, G. Lavoué, and A. Schmutz. Collaborative scientific visualization: The COLLAVIZ framework. In *Proceedings of the Joint Virtual Reality Conference of EuroVR–EGVE–VEC*, Eurographics Association, Geneva, Switzerland, 2010.

[54] W. Eastcott, D. Evans, V. Kalpias-Illias, J. Rooney, and M. Mihejevs. PlayCanvas, 2011. https://playcanvas.com.

[55] Epic Games. Unreal Engine, 2014. https://www.unrealengine.com.

[56] A. Evans, J. Agenjo, and J. Blat. Web-based visualisation of on-set point cloud data. In *Proceedings of the 11th European Conference on Visual Media Production*, CVMP '14, pages 10:1–10:8, New York, NY, USA, 2014. ACM.

[57] A. Evans, M. Romeo, A. Bahrehmand, J. Agenjo, and J. Blat. 3D graphics on the web: A survey. *Computers & Graphics*, 41(0):43–61, 2014.

[58] Exocortex Technologies. Clara.io, 2013. https://clara.io.

[59] Fraunhofer. Instant Reality, 2009. http://www.instantreality.org.

[60] J. Gaillard, A. Vienne, R. Baume, F. Pedrinis, A. Peytavie, and G. Gesquière. Urban data visualisation in a web browser. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, pages 81–88, New York, NY, USA, 2015. ACM.

[61] F. Ganovelli, M. Corsini, S. Pattanaik, and M. Di Benedetto. *Introduction to Computer Graphics: A Practical Learning Approach*. Chapman & Hall/CRC Press, London, UK, 2014. http://vcg.isti.cnr.it/Publications/2014/GCPD14.

[62] Geoweb3d Inc. Geoweb3d, 2012. http://www.geoweb3d.com.

[63] E. Gobbetti, F. Marton, M. B. Rodriguez, F. Ganovelli, and M. Di Benedetto. Adaptive quad patches: An adaptive regular structure for web distribution and adaptive rendering of 3D models. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 9–16, New York, NY, USA, 2012. ACM.

[64] Goo Technologies. Goo Create, 2012. https://github.com/GooTechnologies/goojs.

[65] Google. O3D, 2009. https://code.google.com/archive/p/o3d/.

[66] Google. Google Earth, 2011. https://www.google.com/earth.

[67] Google. Google Maps, 2011. https://enterprise.google.com/maps.

[68] Google. WebGL Loader, 2011. https://code.google.com/p/webgl-loader.

[69] GraphiTech. Geo Browser 3D, 2016. http://geobrowser3d.com.

[70] Gravity Sketch Ltd. Gravity Sketch, 2014. https://www.gravitysketch.com.

[71] I. J. Grimstead, N. J. Avis, and D. W. Walker. Rave: The resource-aware visualization environment. *Concurrency and Computation: Practice and Experience*, 21(4):415–448, 2009.

[72] D. Haehn, S. Knowles-Barley, M. Roberts, J. Beyer, N. Kasthuri, J. Lichtman, and H. Pfister. Design and evaluation of interactive proofreading tools for connectomics. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2466–2475, 2014.

[73] D. Haehn, N. Rannou, B. Ahtam, E. Grant, and R. Pienaar. Neuroimaging in the browser using the X Toolkit. In *Frontiers in Neuroinformatics Conference Abstract: 5th INCF Congress of Neuroinformatics*, 2014. https://www.frontiersin.org/10.3389/conf.fninf.2014.08.00101/event_abstract.

[74] C. Hand. A survey of 3D interaction techniques. *Computer Graphics Forum*, 16:269–281, 1997.

[75] I. Hickson. Extending HTML. 2004. http://ln.hixie.ch/?start=1089635050&count=1.

[76] M. Hildebrandt and J. Bohnet. Seerene, 2015. https://www.seerene.com.

[77] H. Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 99–108, New York, NY, USA, 1996. ACM.

[78] B. Houston. ThreeKit, 2015. https://threekit.com.

[79] B. Houston, W. Larsen, B. Larsen, J. Caron, N. Nikfetrat, C. Leung, J. Silver, H. Kamal-Al-Deen, P. Callaghan, R. Chen, and T. McKenna. Clara.io: Full-featured 3D content creation for the web and cloud era. In *ACM SIGGRAPH 2013 Studio Talks*, SIGGRAPH '13, pages 8:1–8:1, New York, NY, USA, 2013. ACM.

[80] HTC. Vive, 2016. https://www.vive.com.

[81] HUMUSOFT. Orbisnap, 2005. http://www.orbisnap.com.

[82] D. Iborra and V. Nordstrom. Cl3ver, 2013. https://www.cl3ver.com.

[83] J. Jankowski. A taskonomy of 3D web use. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 93–100, New York, NY, USA, 2011. ACM.

[84] J. Jankowski and S. Decker. A dual-mode user interface for accessing 3D content on the World Wide Web. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pages 1047–1056, New York, NY, USA, 2012. ACM.

[85] J. Jankowski and S. Decker. On the design of a dual-mode user interface for accessing 3D content on the World Wide Web. *International Journal of Human–Computer Studies*, 71(7–8):838–857, 2012.

[86] J. Jankowski and M. Hachet. A survey of interaction techniques for interactive 3D environments. In M. Sbert and L. Szirmay-Kalos, eds, *Eurographics 2013 – State of the Art Reports*. The Eurographics Association, Geneva, Switzerland, 2013.

[87] J. Jankowski and M. Hachet. Advances in interaction with 3D environments. *Computer Graphics Forum*, 34(1):152–190, 2015.

[88] J. Jankowski, S. Ressler, K. Sons, Y. Jung, J. Behr, and P. Slusallek. Declarative integration of interactive 3D graphics into the World-Wide Web: Principles, current approaches, and research agenda. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 39–45, New York, NY, USA, 2013. ACM.

[89] B. Jenny, B. Šavrič, and J. Liem. Real-time raster projection for web maps. *International Journal of Digital Earth*, 9(3):215–229, 2016.

[90] Jmol Development Team. JSmol – JavaScript-based molecular viewer From Jmol, 2013. http://sourceforge.net/projects/jsmol.

[91] JogAmp. JOGL – Java OpenGL, 2004. http://jogamp.org/jogl/www.

[92] T. Johansson. Taking the canvas to another dimension, 2007. https://web.archive.org/web/20071117170113/http://my.opera.com/timjoh/blog/2007/11/13/taking-the-canvas-to-another-dimension.

[93] S. Jourdain, J. Forest, C. Mouton, B. Nouailhas, G. Moniot, F. Kolb, S. Chabridon, M. Simatic, Z. Abid, and L. Mallet. ShareX3D, a scientific collaborative 3D viewer over HTTP. In *Proceedings of the 13th International Symposium on 3D Web Technology*, Web3D '08, pages 35–41, New York, NY, USA, 2008. ACM.

[94] S. Jourdain, U. Ayachit, and B. Geveci. ParaViewWeb: A web framework for 3D visualization and data processing. In *Proceedings of the IADIS International Conference on Visual Communication*, pages 502–506, 2010. IADIS.

[95] Y. Jung, J. Behr, and H. Graf. X3DOM as carrier of the virtual heritage. In *Proceedings of the 4th International Workshop on 3D Virtual Reconstruction and Visualization of Computer Architectures*, 2011. ISPRS.

[96] M. Kamburelis. view3dscene, 2004. https://castle-engine.sourceforge.io/view3dscene.php.

[97] L. Kay. SceneJS, 2010. http://scenejs.org.

[98] A. Khan, I. Mordatch, G. Fitzmaurice, J. Matejka, and G. Kurtenbach. ViewCube: A 3D orientation indicator and controller. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, I3D '08, pages 17–25, New York, NY, USA, 2008. ACM.

[99] Khronos Group. OpenGL ES – The standard for embedded accelerated 3D graphics, 2003. https://www.khronos.org/opengles.

[100] Khronos Group. WebGL – OpenGL ES for the web, 2009. https://www.khronos.org/webgl.

[101] Khronos Group. glTF – GL transmission format, 2015. https://www.khronos.org/gltf.

[102] Khronos Group. WebGL section at SIGGRAPH 2015, 2015. https://www.khronos.org/news/events/2015-siggraph.

[103] F. Klein, K. Sons, D. Rubinstein, S. Byelozyorov, S. John, and P. Slusallek. Xflow: Declarative data processing for the web. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 37–45, New York, NY, USA, 2012. ACM.

[104] F. Klein, D. Rubinstein, K. Sons, F. Einabadi, S. Herhut, and P. Slusallek. Declarative AR and image processing on the web with Xflow. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 157–165, New York, NY, USA, 2013. ACM.

[105] F. Klein, K. Sons, D. Rubinstein, and P. Slusallek. XML3D and Xflow: Combining declarative 3D for the web with generic data flows. *IEEE Computer Graphics and Applications*, 33(5):38–47, 2013.

[106] Kubity. Kubity, 2013. https://www.kubity.com.

[107] E. Kwan. Touch with WebGL and Leap Motion, 2015. https://developer-archive.leapmotion.com/gallery/touch-with-webgl-leap-motion.

[108] G. Lavoué, L. Chevalier, and F. Dupont. Streaming compressed 3D data on the web using JavaScript and WebGL. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 19–27, New York, NY, USA, 2013. ACM.

[109] G. Lavoué, L. Chevalier, and F. Dupont. Progressive streaming of compressed 3D graphics in a web browser. In *ACM SIGGRAPH 2014 Talks*, SIGGRAPH '14, pages 43:1–43:1, New York, NY, USA, 2014. ACM.

[110] H. Lee, G. Lavoué, and F. Dupont. Rate-distortion optimization for progressive compression of 3D mesh with color attributes. *Visual Computing*, 28(2):137–153, 2012.

[111] C. Lehmann and J. Döllner. Annotating 3D content in interactive, virtual worlds. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 67–70, New York, NY, USA, 2013. ACM.

[112] C. Leoni, M. Callieri, M. Dellepiane, D. P. O'Donnell, R. Rosselli Del Turco, and R. Scopigno. The dream and the cross: A 3D scanning project to bring 3D content in a digital edition. *Journal on Computing and Cultural Heritage*, 8(1):5:1–5:21, 2015.

[113] Leopoly Ltd. Leopoly, 2015. https://leopoly.com.

[114] C. Leung. C3DL – Canvas 3D JS library, 2008. https://github.com/cathyatseneca/c3dl.

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

[116] M. Limper, S. Wagner, C. Stein, Y. Jung, and A. Stork. Fast delivery of 3D web content: A case study. In *Proceedings of the 18th International Conference on 3D Web Technology*, Web3D '13, pages 11–17, New York, NY, USA, 2013. ACM.

[117] M. Limper, M. Thöner, J. Behr, and D. W. Fellner. SRC – A streamable format for generalized web-based 3D data transmission. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies*, Web3D '14, pages 35–43, New York, NY, USA, 2014. ACM.

[118] D. P. Luebke. *Level of Detail for 3D Graphics*. Morgan Kaufmann Publishers, Burlington, MA, USA, 2003.

[119] LWJGL. Lightweight Java Game Library, 2007. https://www.lwjgl.org.

[120] B. M. Macq, P. Rondao-Alface, and M. Montañola Sales. Applicability of watermarking for intellectual property rights protection in a 3D printing scenario. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, pages 89–95, New York, NY, USA, 2015. ACM.

[121] L. Malomo. Generalized trackball and 3D touch interaction. Masters thesis, Università degli Studi di Pisa, 2013.

[122] D. Malyshau. Kri-Web – Functional 3D engine for the web, 2012. https://code.google.com/archive/p/kri-web.

[123] C. Marion and J. Jomier. Real-time collaborative scientific WebGL visualization with WebSocket. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 47–50, New York, NY, USA, 2012. ACM.

[124] L. Matheson, N. Schwinghamer, and A. Yanes. Pinshape, 2013. https://pinshape.com.

[125] MATLAB. Simulink 3D animation, 2002. https://www.mathworks.com/products/3d-animation.html.

[126] Microsoft Corporation. ActiveX, 1996. https://msdn.microsoft.com/en-us/library/aa751972(VS.85).aspx.

[127] Microsoft Corporation. Silverlight, 2007. https://www.microsoft.com/silverlight.

[128] C. Mouton, K. Sons, and I. J. Grimstead. Collaborative visualization: Current systems and future trends. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 101–110, New York, NY, USA, 2011. ACM.

[129] Mozilla. Canvas 3D, 2007. https://wiki.mozilla.org/Canvas:3D.

[130] Mozilla. A-Frame, 2015. https://aframe.io.

[131] Mozilla. Device Motion Event, 2016. https://developer.mozilla.org/en-US/docs/Web/API/DeviceMotionEvent.

[132] Mozilla. Device Orientation Events, 2016. https://developer.mozilla.org/en-US/docs/Web/API/DeviceOrientationEvent.

[133] Mozilla. WebVR, 2016. https://developer.mozilla.org/en-US/docs/Web/API/WebVR_API.

[134] R. K. Mueller, J. Gay, and M. Moissette. OpenJSCAD, 2013. https://openjscad.org.

[135] F. Mwalongo, M. Krone, M. Becher, G. Reina, and T. Ertl. Remote visualization of dynamic molecular data using WebGL. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, pages 115–122, New York, NY, USA, 2015. ACM.

[136] F. Mwalongo, M. Krone, G. Reina, and T. Ertl. State-of-the-art report in web-based visualization. *Computer Graphics Forum*, 35(3):553–575, 2016.

[137] MyMiniFactory. MyMiniFactory, 2013. https://www.myminifactory.com.

[138] NASA Jet Propulsion Laboratory. Experience Curiosity, 2015. https://eyes.nasa.gov/curiosity.

[139] G. M. Nielson and D. R. Olsen, Jr. Direct manipulation techniques for 3D objects using 2D locator devices. In *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, I3D '86, pages 175–182, New York, NY, USA, 1987. ACM.

[140] M. Nobel-Jørgensen. KickJS – A WebGL game engine for modern web browsers, 2011. http://www.kickjs.org.

[141] Octaga Visual Solutions. Octaga Player, 2006. http://www.octagavs.com/solutions/web.

[142] Oculus VR. Ocolus Rift, 2016. https://www.oculus.com/rift.

[143] R. Ohbuchi, H. Masuda, and M. Aono. Watermarking three-dimensional polygonal models. In *Proceedings of the 5th ACM International on Multimedia* New York, NY, USA, 1997. ACM.

[144] R. Ohbuchi, H. Masuda, and M. Aono. Watermarking three-dimensional polygonal models through geometric and topological modifications. *IEEE Journal on Selected Areas in Communications*, 16(4):551–560, 1998.

[145] R. Ohbuchi, A. Mukaiyama, and S. Takahashi. A frequency-domain approach to watermarking 3D shapes. *Computer Graphics Forum*, 21: 373–382, 2002.

[146] S. Ortiz. Is 3D finally ready for the web? *Computer*, 43(1):14–16, Jan 2010.

[147] G. A. Pachikov. Cortona 3D, 2006. http://www.cortona3d.com.

[148] M. Patel, M. White, K. Walczak, and P. Sayd. Digitisation to presentation – Building virtual museum exhibitions. In *Proceedings of Vision, Video, Graphics*, Southend-on-sea, UK, 2003. IMA.

[149] M. Persson. Minecraft, 2009. https://minecraft.net.

[150] C. Pinson. OSGJS, 2011. http://osgjs.org.

[151] F. Ponchio and M. Dellepiane. Fast decompression for web-based view-dependent 3D rendering. In *Proceedings of the 20th International Conference on 3D Web Technology*, pages 199–207, 2015. ACM.

[152] F. Ponchio and M. Dellepiane. Multiresolution and fast decompression for optimal web-based rendering. *Graphical Models*, 88:1 – 11, 2016.

[153] F. Ponchio, M. Potenziani, M. Dellepiane, M. Callieri, and R. Scopigno. The ARIADNE Visual Media Service. In *Proceedings of the 43rd Computer Applications and Quantitative Methods in Archaeology Conference*, pages 433–442, 2015.

[154] M. Potenziani, M. Callieri, M. Dellepiane, M. Corsini, F. Ponchio, and R. Scopigno. 3DHOP: 3D heritage online presenter. *Computer & Graphics*, 52:129–141, 2015.

[155] E. Praun, H. Hoppe, and A. Finkelstein. Robust mesh watermarking. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 49–56, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[156] E. Puppo and R. Scopigno. Simplification, LoD and multiresolution principles and applications. In D. Fellner and L. Szirmay-Kalos, eds, *Eurographics 1997*. The Eurographics Association, Geneva, Switzerland, 1997.

[157] D. Raggett. *Extending WWW to support platform independent virtual reality*. Technical report, 1994. https://www.w3.org/People/Raggett/vrml/vrml.html.

[158] B. Resch, R. Wohlfahrt, and C. Wosniok. Web-based 4D visualization of marine geo-data using WebGL. *Cartography and Geographic Information Science*, 41(3):235–247, 2014.

[159] J. F. Richardsoon. SimVRML, 2002. https://sourceforge.net/projects/simvrml.

[160] A. S. Rose and P. W. Hildebrand. NGL viewer: A web application for molecular visualization. *Nucleic Acids Research*, 43(W1):W576–W579, 2015.

[161] B. C. Russell, R. Martin-Brualla, D. J. Butler, S. M. Seitz, and L. S. Zettlemoyer. 3D Wikipedia: Using online text to automatically label and navigate reconstructed geometry. *ACM Transactions on Graphics*, 32(6):193:1–193:10, 2013.

[162] J. R. Sánchez, D. Oyarzun, and R. Díaz. Study of 3D web technologies for industrial applications. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 184–184, New York, NY, USA, 2012. ACM.

[163] B. R. Schatz and J. B. Hardin. NCSA Mosaic and the World Wide Web: Global hypermedia protocols for the internet. *Science*, 265(5174): 895–901, 1994.

[164] M. Schuetz. Potree, 2013. http://potree.org.

[165] M. Schuetz. Rendering large point clouds in web browsers. In *Central European Seminar on Computer Graphics 2015*, 2015.

[166] D. Seo, B. Yoo, and H. Ko. Webized 3D experience by HTML5 annotation in 3D web. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D '15, pages 73–80, New York, NY, USA, 2015. ACM.

[167] D. Seo, B. Yoo, J. Choi, and H. Ko. Webizing 3D contents for super multiview autostereoscopic displays with varying display profiles. In *Proceedings of the 21st International Conference on Web3D Technology*, Web3D '16, pages 155–163, New York, NY, USA, 2016. ACM.

[168] Shapeways. Shapeways, 2013. https://www.shapeways.com.

[169] S. Shi and C. Hsu. A survey of interactive remote rendering systems. *ACM Computing Surveys*, 47(4):57:1–57:29, 2015.

[170] K. Shoemake. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the Conference on Graphics Interface '92*, pages 151–156, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[171] Sketchfab. Sketchfab, 2014. https://sketchfab.com.

[172] Z. Smith and B. Pettis. Thingiverse, 2011. https://www.thingiverse.com.

[173] Smithsonian Institution. Smithsonian X3D, 2011. http://3d.si.edu.

[174] Y. Song, W. Wei, L. Deng, P. Du, Y. Zhang, and D. Nie. 3D-CollaDesign: A real-time collaborative system for web 3D design. In *Proceedings of the 19th IEEE International Conference on Computer Supported Cooperative Work in design*, pages 407–412, New York, NY, USA, 2015. IEEE.

[175] K. Sons, F. Klein, D. Rubinstein, S. Byelozyorov, and P. Slusallek. XML3D: Interactive 3D graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology*, Web3D '10, pages 175–184, New York, NY, USA, 2010. ACM.

[176] K. Sons, C. Schlinkmann, F. Klein, D. Rubinstein, and P. Slusallek. XML3D.js: Architecture of a polyfill implementation of XML3D. In *Proceedings of the 6th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, pages 17–24, 2013.

[177] Stackgl. Stackgl, 2015. https://github.com/stackgl/stackgl.github.io.

[178] J. A. Stewart. FreeWRL, 1998. http://freewrl.sourceforge.net.

[179] P. S. Strauss and R. Carey. An object-oriented 3D graphics toolkit. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, pages 341–349, New York, NY, USA, 1992. ACM.

[180] P. S. Strauss, P. Issacs, and J. Shrag. The design and implementation of direct manipulation in 3D: SIGGRAPH 2002 course notes. 2002.

[181] Sun Microsystems. JAVA3D – The Java 3D API, 1998. http://www.oracle.com/technetwork/articles/javase/index-jsp-138252.html.

[182] J. Sutter, K. Sons, and P. Slusallek. Blast: A binary large structured transmission format for the web. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies*, Web3D '14, pages 45–52, New York, NY, USA, 2014. ACM.

[183] Threeding. Threeding, 2013. https://www.threeding.com.

[184] M. Toschlog. Parallax, 2012. http://parallax3d.org.

[185] Trimble Inc. SketchUp, 2006. https://www.sketchup.com.

[186] Triumph LLC. Blend4web, 2014. https://www.blend4web.com.

[187] Turbulenz. Turbulenz, 2009. http://biz.turbulenz.com.

[188] Uber. Deck.gl, 2015. https://uber.github.io/deck.gl.

[189] F. Uccheddu, M. Corsini, and M. Barni. Wavelet-based blind watermarking of 3D models. In *Proceedings of the 2004 Workshop on Multimedia and Security*, pages 143–154, New York, NY, USA, 2004. ACM.

[190] C. Ulbrich and C. Lehmann. A DCC pipeline for native 3D graphics in browsers. In *Proceedings of the 17th International Conference on 3D Web Technology*, Web3D '12, pages 175–178, New York, NY, USA, 2012. ACM.

[191] Unity Technologies. Unity3D, 2005. https://unity3d.com.

[192] University College London. 3D Petrie Museum, 2009. http://www.ucl.ac.uk/3dpetriemuseum.

[193] University of Applied Sciences Northwestern Switzerland. OpenWebGlobe, 2011. https://github.com/OpenWebGlobe.

[194] A. van Dam. Post-WIMP user interfaces. *Communications of the ACM*, 40(2):63–67, 1997.

[195] Virtual Heritage Lab. Aton front-end, 2015. http://osiris.itabc.cnr.it/scenebaker/index.php/projects/aton.

[196] Visionary Cross. The Visionary Cross project, 2015. http://vcg.isti.cnr.it/cross.

[197] Visual Computing Lab. SpiderGL – 3D graphics for next-generation WWW, 2010. http://vcg.isti.cnr.it/spidergl.

[198] Visual Computing Lab. Nexus – Multiresolution visualization, 2013. http://vcg.isti.cnr.it/nexus.

[199] Visual Computing Lab. 3DHOP – 3D Heritage Online Presenter, 2014. http://3dhop.net.

[200] Visual Computing Lab. MeshLabJS, 2014. http://www.meshlabjs.net.

[201] Visual Computing Lab. ARIADNE – Visual Media Service, 2015. https://ariadne1.isti.cnr.it.

[202] Vizor. Patches, 2014. https://patches.vizor.io.

[203] Voxel.js. Voxel.js, 2013. http://voxeljs.com.

[204] K. Walczak, W. Cellary, and M. White. Virtual museum exhibitions. *IEEE Computer*, 39:93–95, 2006.

[205] E. Wallace. LightGL – A lightweight WebGL library, 2011. https://github.com/evanw/lightgl.js.

[206] C. Ware and S. Osborne. Exploration and virtual camera control in virtual three-dimensional environments. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, I3D '90, pages 175–183, New York, NY, USA, 1990. ACM.

[207] Web3D Consortium. What is X3D graphics?, 2004. http://www.web3d.org/what-x3d-graphics.

[208] J. Wilhelmy. Inka3D, 2011. http://www.inka3d.com.

[209] C. A. Wingrave, B. Williamson, P. Varcholik, J. Rose, A. Miller, E. Charbonneau, J. N. Bott, and J. J. LaViola. The Wiimote and beyond: Spatially convenient devices for 3D user interfaces. *IEEE Computer Graphics and Applications*, 30(2):71–85, 2010.

[210] S. Wittens. MathBox, 2012. https://gitgud.io/unconed/mathbox.

[211] R. Wojciechowski, K. Walczak, M. White, and W. Cellary. Building virtual and augmented reality museum exhibitions. In *Proceedings of the Ninth International Conference on 3D Web Technology*, Web3D '04, pages 135–144, New York, NY, USA, 2004. ACM.

[212] XTK Developers. X Toolkit API, 2012. https://github.com/xtk.

[213] S. Zafeiriou, A. Tefas, and I. Pitas. Blind robust watermarking schemes for copyright protection of 3D mesh objects. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):596–607, 2005.

[214] A. Zipf. OSM-3D, 2010. http://www.osm-3d.org.

[215] F. Zollo, L. Caprini, O. Gervasi, and A. Costantini. X3DMMS: An X3DOM tool for molecular and material sciences. In *Proceedings of the 16th International Conference on 3D Web Technology*, Web3D '11, pages 129–136, New York, NY, USA, 2011. ACM.

[216] P. Zuspan, J. Finkelstein, and M. Finkelstein. Kokowa, 2015. https://www.kokowa.co.