DOI: 10.1111/cgf.12929
EuroVis 2016
R. Maciejewski, T. Ropinski, and A. Vilanova
(Guest Editors)

Volume 35 (2016), Number 3
STAR – State of The Art Report

State-of-the-Art Report in Web-based Visualization

F. Mwalongo[1] M. Krone[1] G. Reina[1] T. Ertl[1]

[1]Visualization Research Center, University of Stuttgart, Germany

Abstract

In this report, we review the current state of the art of web-based visualization applications. Recently, an increasing number of web-based visualization applications have emerged. This is due to the fact that new technologies offered by modern browsers greatly increased the capabilities for visualizations on the web. We first review these technical aspects that are enabling this development. This includes not only improvements for local rendering like WebGL and HTML5, but also infrastructures like grid or cloud computing platforms. Another important factor is the transfer of data between the server and the client. Therefore, we also discuss advances in this field, for example methods to reduce bandwidth requirements like compression and other optimizations such as progressive rendering and streaming. After establishing these technical foundations, we review existing web-based visualization applications and prototypes from various application domains. Furthermore, we propose a classification of these web-based applications based on the technologies and algorithms they employ. Finally, we also discuss promising application areas that would benefit from web-based visualization and assess their feasibility based on the existing approaches.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics

1. Introduction

Remote visualization has long been an important area of research motivated by the need to visualize huge amounts of data ranging from simulations over medical imaging to big data from social media or business intelligence. This need arises from insufficient computational resources at the user's side (client). The reason can either be relatively thin clients, like mobile phones, or the sheer amount of data to process. Even if computational resources at the client side were powerful enough, moving the full data set to the client quickly becomes impractical due to bandwidth, latency or local storage limitations. In certain scenarios, making the raw data available to other users might also present legal or privacy issues. Here, remote visualization can be a viable solution. However, limited bandwidth is still the main challenge for remote visualization, together with network latency issues. Different approaches and techniques have been proposed over the years to solve these problems.

Despite the technical challenges addressed by past research, remote visualization is still an active area of research due to the continuous growth of data sets and gradual improvement of infrastructure and client devices. Web-based visualization in particular has received much attention due to its ubiquity across platforms, ranging from desktop computers to smartphones, and its potential as a collaborative platform [IES*11].

Web-based approaches can allow visualization tools to be deployed across devices from single source code base. This does not only allow easy collaboration and sharing of visualization tools among different teams, but also simplifies software maintenance issues, allowing visualization researchers and domain scientists to focus more on the core issues of their research. Moreover, this deployment model allows domain scientists to access latest visualization techniques (e.g., getting access to a latest visualization technique by simply refreshing a page). This close collaboration between visualization researchers and domain scientists is also important for the visualization research community to avoid the danger of loosing its users [Lor04].

Another advantage of web-based solutions for the client-side is user convenience. Users (e.g., domain scientists) do not need to install software, they just need a browser. Since browsers are meanwhile available on virtually all computing devices, users can flexibly work from any device, anywhere, as long as there is Internet connection and their data can be accessed remotely. As many visualization solutions are currently GPU-based, shader code (being simple text files) can be hosted on a server together with the data and easily shared by multiple clients. This setup is attractive even in cases where the data is not shared. A user with data on a local machine can simply get the visualization code from a server and visualize the data locally instead of uploading the data to the server, which may be expensive for large data sets or overload the server when many users are uploading their data sets. For large data sets, moving computation (code) to the data is cheaper than moving the data to the computation [GLNS*05].

Early techniques for web-based visualization exploited the use

of VRML and Java applets in the browser, and server-side rendering [ESE00, ESEE99, EE99, HJS98, TP97, WBW96]. Other techniques have combined Java, JavaScript and Flash [JJAM11]. These approaches usually lead to poor performance due to bandwidth and latency issues. Historically, server-side rendering was favored, mainly due to limitations in browser technologies.

However, advances in web technologies have greatly improved and the current trend in web-based visualization is to exploit the power of GPUs through WebGL [Khr13, Khr11b] and HTML5 technologies for GPU-accelerated client-side rendering within the browser without plugins. GPU-based visualization techniques help to improve rendering performance and interactivity by offloading to the GPU expensive computations, which would otherwise be infeasible on the CPU using JavaScript. Client-side rendering is preferred because it avoids the round-trip network latency suffered by server-side rendering approaches that require changes in rendering or visualization parameters to be sent to the server in order to generate new images. Despite these improvements, the ability to render complex scenes in the browser can still benefit from offloading demanding computations (e.g., preprocessing) to the server side.

This report gives an overview over available techniques for web-based visualization. This includes a discussion of state-of-the-art methods for rendering as well as a survey on infrastructure technologies and optimization strategies, all of which enable interactive visualization. These methods provide the foundation for web-based visualization applications. Therefore, special attention is given not only to fast, GPU-accelerated rendering methods on the client but also to enabling server-side technologies like grid-based visualization, and the use of methods that reduce the required memory bandwidth like streaming or compression of visualization data. We also discuss available web-based visualization applications from various application domains and review how they correlate with the previously detailed techniques. Furthermore, we provide a classification that is based on this assessment, which gives a structured overview of the related work. Finally, we discuss specific application scenarios benefiting from web-based visualization and provide practical advice to solve the concrete visualization problems raised by these applications. We also use our classification for a white spot analysis concerning the use of certain technologies for specific application areas. We delineate our work explicitly from analyzing work related to collaborative visualization on the web, since this is a vast field of its own and would go beyond the scope of this manuscript.

The rest of this report is structured as follows: In section 2, we discuss technical aspects of web-based visualization. Available web-based visualization applications and prototypes are reviewed in section 3. Section 4 introduces a classification of these web-based visualization methods based on the technical aspects. They are sorted using the previously given classification and according to their targeted application area. Based on this overview of available web-based visualizations, section 5 discusses several challenging real-world data sets from different application domains where users could benefit from web-based visualization. We also review their feasibility considering the previously mentioned available technologies, techniques, and applications. Finally, section 6 concludes the paper and gives direction for future challenges concerning web-based visualization.

2. Technical Aspects

This section discusses technical aspects about visualization as a web service, grid-based, and cloud-based visualization. Local rendering in the browser, data encoding, and transfer techniques are also discussed. Remote visualization and web-based visualization in particular usually follows a client-server architecture. Grid and cloud-based visualization approaches aim at scaling the server side in order to tackle larger problems (e.g., rendering large models or complex preprocessing tasks) and support multiple users. In some scenarios where data resulting from simulations or other sources are already stored on the grid or cloud, this approach becomes the only feasible solution as it is impractical to move these huge amounts of raw data.

Web-based visualization as an approach to remote visualization tries to exploit the ubiquity of the browser across operating systems and devices and other web technologies for wider deployment and collaboration. It is also used in grid and cloud-based visualization approaches where the browser acts as a deployment platform for the client-side in a client-grid or client-cloud architecture.

The web environment is a heterogeneous platform where a variety of different tools (usually written in different programming languages and running on different platforms) have to work together harmoniously to accomplish a task. The concept of visualization as a service has received attention lately. The idea is to apply the principles of web services to visualization tools in order to enable seamless interoperability between different software tools. A common trend among many approaches for grid and cloud-based visualization is to follow a web service approach. This trend is likely due to the distributed and heterogeneous nature of these environments where different nodes may be running on different platforms and have to be interoperable. This is in contrast to cluster environments where nodes are also distributed, but in most cases homogeneous with regard to hardware and software. Moreover, most of grid and cloud-based solutions are implemented on top of a grid or cloud middleware that itself follows a service-oriented approach (e.g., Globus Toolkit [Fos05]).

Shi and Hsu [SH15] provide a survey of interactive remote rendering systems. While their work focuses on remote rendering approaches, our work provides a comprehensive survey on remote visualization in the web environment including grid-based visualizations and cloud-based ones as well as visualization as a web service. Their work also discusses latency hiding techniques for interactive remote rendering and points out local rendering as a suitable latency reduction technique for interactive user experience. In our case, latency hiding as discussed by them is only relevant for image-based visualization approaches. Given the importance of interactivity in visualization, our work pays special emphasis on local rendering techniques in the browser that exploit modern web technologies (e.g., HTML5 [W3C14] and WebGL [Khr13,Khr11b]. We also build on the classification scheme proposed by them (see Fig 1) to include infrastructure and optimization strategies (see Table 1).

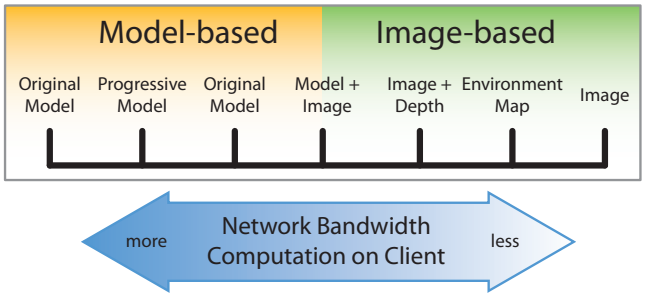

Figure 1: *Classification scheme for remote rendering proposed by Shi and Hsu [SH15].*

2.1. Visualization as a Web Service

The World Wide Web Consortium (W3C) defines a web service as *a software system designed to support interoperable machine-to-machine interaction over a network* [W3C04]. The main goal of web services is to allow seamless interoperability between different software applications and tools that need to communicate in order to accomplish a particular task. Since the services interact based on standardized interfaces without consideration of implementation details, software written in different programming languages and running on different platforms can communicate and work together smoothly without requiring manual integration by the user. This approach is especially important in heterogeneous distributed systems.

Main approaches for applying the concept of web services to visualization with regard to how the pipeline is partitioned can be grouped into three classes: in the first approach, the entire visualization pipeline is treated as a black box and an application can be viewed as a single service. The user is provided with an interface for specifying data to be visualized and interacting with the resulting visualization. Tableau Online [tab] and TIBCO Spotfire Cloud [spo] are some of the examples that fall into this category. This approach has the advantage of being simple for the end user. However, it may limit the users' choice of visualization types that can be created.

In the second approach, each of the different stages of the visualization pipeline as first described by Haber and McNabb [HM90], is implemented as an independent web service. In this approach, the entire visualization pipeline is exposed to the user who can connect different services to create pipelines in order to generate various visualizations. These services can potentially be from different providers, implemented in different languages and running on different platforms. Wang et al. [WBHW08], Zudilova-Seinstra et al. [ZSYA*08], and Charters et al. [CHM04] follow this approach in the context of grid-based visualization. Although this approach provides the maximum flexibility for the end user, it may suffer from inefficiencies due to communication overheads and excessive data copying between services.

The third approach combines several stages of the visualization pipeline into a single service. By combining some stages into a single service, inefficiencies suffered by the first approach due to communication and data movements can be avoided and still retain most of the flexibility provided by the second approach, thus allowing users to connect various visualization services to obtain more powerful visualizations. Wood et al. [WBS*08] and d'Auriol [d'A11] follow this approach.

Viewing a visualization application as a single service would provide a good abstraction when considering a visualization as an integral component in the entire data analysis pipeline (e.g., combining simulation, analysis, and visualization) rather than a separate application. On the other hand, mapping each stage of the visualization pipeline to a web service or combining some services into a single service may be more suitable for visualization application developers. These may be interested in combining different implementations of visualization techniques that provide different capabilities or exploit special hardware features for each stage of the visualization pipeline to create a new composite visualization service for end users (e.g., domain scientists).

Another important aspect is the handling of data movement between web services in a visualization pipeline. Web services can exchange data directly with each other or through a centralized data store. Koulouzis et al. [KZSB10] compare the performance of these two data movement approaches and show that direct data exchange between services provides better performance compared to employing centralized data storage.

On the implementation side, two main approaches are used, SOAP-based web services [See01] and RESTful web services [Pau14], based on an architectural style first introduced by Fielding [Fie00]. Current trends [WPR10, WP11, RAR13] favor RESTful web services rather than SOAP-based services due to their simplicity and easy integration with other web standards. Pautasso et al. [PZL08] give a detailed architectural comparison between these two approaches. Despite the popularity of RESTful web services in the business domain and cloud-based services, this approach has not yet caught much attention with regard to visualization services. We discuss visualization as a web service in the context of grid and cloud computing environments in section 2.2 and section 2.3 respectively.

Web service approaches allow different tools to communicate and exchange data in standardized ways irrespective of their implementation languages, platforms, or devices on which they are deployed. This would allow easy automation of the data analysis pipeline, where visualization is just a component. Components need only agree on a communication protocol and keep their internal implementation details to themselves. Moreover, decoupling the interface from implementation details allows each component to be independently implemented and optimized based on its specific task that it performs or special hardware features that it exploits.

Existing work on visualization as a service has focused more on applying the concept of web services to visualization software alone (i.e., interoperability between visualization system components) and less attention has been paid to interoperability between visualization services with other tools in the data analysis pipeline (e.g., simulation and analysis tools/services). The increasingly huge amounts of data and complex analyses required to understand and gain insight can make single visualization tools inadequate for the task, hence the need to combine multiple tools.

By providing programming language-independent and platform-independent interfaces to visualization services, it becomes easy to connect these tools together and potentially automate the creation of data analysis pipelines. This could be achieved through sophisticated visualization workflow engines that can automatically orchestrate different visualization services given only the data to be visualized and some user-provided visualization parameters for the desired output visualization. This would allow scientists to focus on their domain research work, and reduce time spent manually integrating software tools.

2.2. Grid-based Visualization

Grid computing is a distributed system model that allows pooling computational and storage resources to provide a high-performance problem-solving computing infrastructure [FKT01, FKNT02]. Computational resources in a grid are distributed similar to a cluster. However, while a cluster is usually built from resources that belong to a single organization, a grid combines resources from multiple organizations that may be geographically distributed [FK99]. Therefore resources in a grid environment are usually heterogeneous compared to those in a cluster, which are typically homogeneous.

Grid-based visualization is motivated by high demands from high-performance and complex simulations that use grid resources for computation and storage of the simulation results. As the simulations and other scientific instruments produce massive data, it becomes infeasible to visualize these data with only locally available computing resources. Therefore an infrastructure of similar magnitude is required for data visualization. Computational and storage resources in the grid are by nature geographically distributed, the challenge for visualization algorithms is therefore how to efficiently use this distributed infrastructure to achieve good performance while at the same time handling network bandwidth and latency issues.

Much work in grid-based visualization exploits the use of web services. This is understandable given the heterogeneous nature of the computing resources in these environments.

Koval et al. [KMS*15] build a web-based front-end for visualization of data resulting from a simulation running in a grid computing infrastructure. Data preprocessing on the grid is used to filter out data that is not relevant for the requested visualization in order to reduce the amount of data transfered to the client. Rendering is done on the client in WebGL using Three.js [Dan12] library. The motivation for client-side rendering is to minimize latency and improve interactivity.

Koulouzis et al. [KZSB10] apply a web service approach for enabling domain scientists to flexibly create visualization applications from distributed visualization pipelines in the context of medical image analysis. The use of a web service approach is motivated by the need for collaboration between geographically distributed teams and interoperability of different tools in the analysis pipeline. Orchestration of the visualization is done through a visualization workflow engine that controls the execution of the pipelines. Since services do not communicate directly, this model of interaction provides more flexibility. The implementation of the visualization is

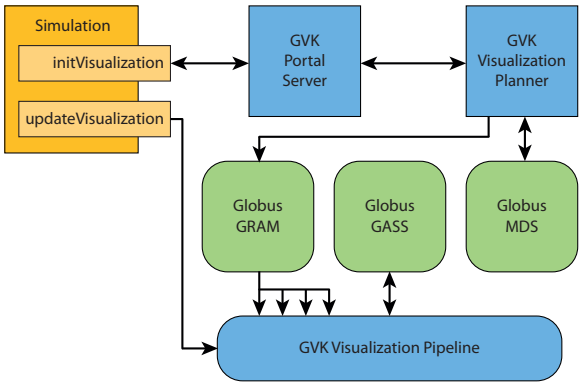

Figure 2: *Example for a visualization service using grid middleware: setup of a simulation process including in-situ visualization using the Grid Visualization Kernel (image adapted from Kanzlmueller et al. [KHRV04]).*

based on the VTK library and is composed of data read, filter, map, and render web services. Furthermore, the authors experiment with two different models of data communication between web services: centralized and distributed communication models. In the centralized model, services communicate through central data store and in the distributed model, they communicate directly. A service can act as a data producing or a data consuming service. In both models, the URIs of data resources are exchanged between services rather than the actual data. This is done for efficiency reasons.

Another difference between the centralized model and the distributed model is that, in the centralized model, the workflow engine provides a URI of the central data store where the particular service should upload its results as an input parameter to the service. In the distributed model, each service returns a URI to the workflow engine after completing its task. Other services can then use this URI to get the results. This URI is passed to the next service in the pipeline which can then read the data directly from the provided URI.

Wood et al. [WBS*08] present a 3-layered architecture for web service-based visualization. The architecture consists of a client layer, a middleware layer, and a visualization layer. The client layer provides the user interface, the middleware layer provides an interface to the visualization layer, and the actual visualization functionality is provided by the visualization layer. The visualization layer can be built on top of a visualization toolkit like VTK. The middleware layer decouples the user interface from the actual visualization service.

The authors use a custom XML-based markup language called skML [DS05], to describe the visualization pipelines. They implement this architecture using SOAP-based web services [See01] that leverage grid services of the Globus Toolkit [Fos05] for communication between the web services and the visualization layer. An earlier work by Brodlie et al. [BDG*04] that incorporates the ability to connect the visualization system to a simulation application for computational steering also builds on similar concepts. This architecture is also used in a framework for configurable visualization systems called *ADVICE* [WSD*10].

Kanzlmueller [KHRV04] present a Grid Visualization Kernel (*GVK*) based on Globus Toolkit [Fos05]. The authors propose a set of visualization services as an extension to the grid middleware services that can be used as building blocks for creating visualization applications. Figure 2 shows how simulation applications register with the GVK as sources of data and visualization clients can subscribe through the kernel to view data produced by the simulations, linked to each other in a client-server architecture. To visualize data from a simulation, a visualization request is sent to the GVK. The GVK initializes the visualization pipeline and remains connected with the simulation. When the simulation has new data, it sends the simulation data to the GVK to update the visualization. To view the visualizations, a client (user or application interested to view the visualization) sends a visualization request to the GVK. Upon receiving the request, GVK establishes a connection that links the client with the visualization pipeline connected to a simulation that the client has chosen to view its results.

The kernel allows multiple visualization requests to be connected to a single simulation. One of the main advantages of this architecture is that it decouples the visualization clients with the simulation providing the data source. This decoupling provides much flexibility and possibilities of linking between the simulation and visualization clients.

Charters et al. [CHM04] build SOAP-based web services [STK02] on top of Globus Toolkit [Fos05] by mapping each of the visualization pipeline stages [HM90] as a separate web service. They further extend the original pipeline by introducing bidirectional communication between the stages of the pipeline. The implementation of the visualization pipeline is based on the VTK library [SML06]. A similar approach to web service based-based visualization pipeline is advocated by Zudilova-Seinstra et al. [ZSYA*08] who apply a service-oriented approach to create a visualization framework for collaborative and distributed medical data analysis.

As discussed in section 2.1, mapping each stage of the visualization pipeline as a separate web service may result into inefficiencies due to communication overheads and excessive data movements between the services.

2.3. Cloud-based Visualization

Cloud computing is a new model of distributed computing whereby IT resources are delivered and accessed using Internet technologies [AFG*10, MG11]. This new model is a result of advances in virtualization, networking, and web technologies [EPM13]. Cloud computing provides a flexible and scalable platform that can be exploited for remote and collaborative visualization. Cloud-based visualization can be considered as an evolution of grid-based visualization; they all scale the server part in the client-server remote visualization architecture and use distributed computational resources. Although grids and clouds are similar in concept, they differ in the model of management and provisioning of computational and storage resources. Management of computational and storage resources in the cloud is based on virtualization technology. That is, cloud management software depends on a virtualization layer controlled by a hypervisor to provide computational and storage resources to applications that run on the cloud through virtual machines. Foster et al. [FZRL08] provide a more detailed comparison between these two paradigms.

Main categories of cloud services are Infrastructure as a Service (IaaS), Platform as a Service (Paas), and Software as a Service (SaaS) [MG11]. Cloud computing deployment models can be categorized as public, private or hybrid clouds [MG11]. There are both commercial and open-source cloud platforms that provide different services. Several popular open-source cloud management softwares (e.g., Open Nebula [MVML12], Eucalyptus [NWG*09], CloudStack [Apa10], and OpenStack [Ope10]) can be used for building a private cloud computing environment. Apart from providing cloud infrastructure management functionality, they also provide APIs to interface with public clouds like Amazon.

A cloud-based visualization service falls into the Software as a Service (SaaS) category and can be built on top of a Platform as Service (PaaS). Most available PaaS offerings, however, currently support interpreted or byte-code based languages (e.g., JavaScript, Python, Java, C#, PHP, etc). To host a service built using a compiled language such as C++ would require deployment on a Infrastructure as a Service (IaaS).

As discussed in section 2.4, local rendering is important for interactivity in a remote visualization scenario. Since cloud services are usually accessed using devices with limited computational and storage resources (e.g., laptops, tablets, and smartphones), the main challenge in cloud-based visualization is how to efficiently transfer data between the cloud and these devices while at the same time reducing requirements on computational resources at the client. To address this challenge, cloud computing resources can be employed for demanding preprocessing tasks that extract and transfer only the data necessary to generate the required visualization at the client.

2.3.1. GPU support in the Cloud

Current visualization techniques rely on the computational power of the GPU to accelerate the rendering. For these applications to benefit from the cloud computing platform, ability to access the GPU from virtual machines in the cloud is required.s Although many hypervisors support virtualization of the majority of computational resources (e.g., CPU, disks, and I/O devices), the virtualization of the GPU is currently least supported. The main current approaches for accessing the GPU from a virtual machine is through GPU pass-through and virtual GPUs (currently provided by NVIDIA GRID or AMD Multiuser GPU) [Her13, AMD15].

In a GPU pass-through approach, a physical GPU can be assigned for exclusive use by a single virtual machine. This approach provides a near-native performance because each virtual machine has its own dedicated GPU. However, it does not scale well because it does not allow sharing of the physical GPU among multiple virtual machines. One of the positive aspects of the cloud is that if one virtual machine fails, the workload can be automatically migrated to other compute nodes. However, this is not possible when a virtual machine has been directly assigned to a physical device. With the virtual GPU approach, a physical GPU is virtualized and can be shared among virtual machines. That is, a virtual machine has no direct access to the physical GPU. This approach provides more

scalability by allowing a single physical GPU to be shared among more than one virtual machine. Performance implications of such a virtualization are quite complex though. Nvidia's approach, for example, statically assigns a dedicated subset of available memory, while compute power is time-slice-shared among guests.

Cloud management software depends on virtualization capabilities provided by hypervisors (e.g., VMware, Microsoft Hyper-V, KVM, Xen, and XenServer) for managing GPU access from the virtual machines. Most of the hypervisors support GPU pass-through, although there is no standardized way of support across all the available hypervisors. To the best of our knowledge, NVIDIA GRID GPUs are, at the time of writing, only supported in Microsoft Hyper-V, VMWare, and XenServer [NVI15b], while AMD Virtual GPUs only support VMWare [AMD15]. Of these three hypervisors that support virtual GPUs, only XenServer is open-source [Xen14]. As an open-source software, XenServer would make a preferred choice for building a cloud-based rendering platform in a research environment. Only CloudStack and OpenStack currently support the XenServer hypervisor among the open-source cloud management toolkits. This means that, currently, Cloud Stack and OpenStack are the only open-source cloud management toolkits that can be used for setting up a private cloud environment capable of hosting a cloud-based visualization service that leverages the power of the GPU. This situation is, however, likely to improve due to research being done on GPU virtualization techniques [DS09, GGS*09, GMAC10, YWO*12, TDC14, VSB14] and demand for cloud gaming services [SLNC13, HHCC13].

Currently, GPU-based cloud-side rendering is focused on gaming and on remote desktop solutions [DVS*12]. Games are rendered in the cloud and streamed as video to the client [OTO13, NVI15a]. To reduce latency, GPUs usually have built-in video encoders enabling streaming the video to the client without additionally taxing the CPU. The clients for these gaming services are usually mobile devices or game consoles and even TVs. Several studies [CWSR12, CCT*11, SSB09] have shown that latency is still a challenge for cloud-based gaming. Since interactive cloud-based visualization and gaming are both latency sensitive, cloud-based visualization faces similar challenges when employing cloud-based rendering. A hybrid approach for cloud-based visualization that combines cloud computation with client-side rendering can be an attractive approach for latency-sensitive applications. We discuss this approach in the next section.

Virtual machine-based deployments of services in the cloud are not without limitations. Since each virtual machine requires a complete operating system, memory requirements and communication overheads can be high [FFRR15]. The communication overhead can be due to I/O calls between a service running in the virtual machine and the native hardware resources or communication between services running in different virtual machines. The current trend toward widespread light-weight virtualization like containers (e.g., Docker Container [Mer14]) looks more promising in terms of performance and efficient sharing of computational resources. This is similar to operating systems specifically tuned for cloud infrastructure like Apache Mesos [HKZ*11]. Docker Containers and Apache Mesos both have GPU support (mainly NVIDIA GPUs).

Combining cloud and client side computing resources would benefit cloud-based visualization more than cloud-based rendering alone. Given the latency sensitivity of interactive visualization applications, it would be ideal to perform rendering on the client-side and offload other demanding preprocessing tasks on the cloud. Advances on GPU mobile technologies make this combination feasible. Client-side avoids network induced latency and therefore is appropriate for interactive visualization applications. Cloud computing resources are already used to run simulations [KSL*13, VPB09, EH08]. These simulations usually produce massive data that cannot be visualized interactively using a single workstation. Since the data, in this case, is already in the cloud, it becomes desirable to host the visualization services also in the cloud to avoid moving these massive data.

The requirement for GPU access in the cloud is not only beneficial for cloud-side GPU-accelerated rendering. GPU-accelerated computations on the server, for example using CUDA and OpenCL, can provide high performance while still being energy efficient. Therefore, especially mobile devices, which are not as powerful as desktop machines, can greatly benefit from offloading expensive computations to the cloud.

2.4. Local Rendering in the Browser

The main motivation for rendering at the client-side (i.e., locally in the browser) is to improve interactivity during visualization. Local rendering avoids round trip latencies suffered by server-side rendering approaches. Recent advances in web technologies including improved JavaScript performance and widespread availability of WebGL have made it possible to achieve high rendering performance in the browser. In this section we focus only on techniques that utilize HTML5 and WebGL, as these are modern standards that allow high performance rendering in the browser without requiring additional plugins.

The `<canvas ...>` element, introduced in the HTML5 standard, provides a surface for graphics drawing through JavaScript. The two most relevant rendering contexts that are supported currently are *2d* and *webgl*. The first one provides CPU-based 2D rendering capabilities. An alternative to the 2D canvas API are scalable vector graphics (SVG) usable through the `<svg ...>` HTML5 element. The second type of canvas context offers GPU-based 3D rendering capabilities through WebGL. WebGL exposes an OpenGL ES 2.0-based API for GPU access through JavaScript in the browser. Since the APIs provided by both rendering contexts are low-level, many libraries and frameworks have been built to simplify their usage and provide high-level functionality. Frameworks include X3DOM [BJK*10, BEJZ09, BJDA11] and XML3D [SKR*10] that aim at integrating 3D graphics with the HTML5 document object model (DOM). X3DOM aims at supporting 3D graphics in the browser by integrating X3D with HTML5 through bridging HTML, DOM, and XML3D models. XML3D extends the capabilities of HTML5 to provide native support for 3D graphics in the browser. These frameworks provide a declarative scene graph-based API for 3D graphics in the browser and are implemented on top of WebGL. Their integration approach is similar to that of SVG for 2D graphics. A 3D scene is described using XML and embedded in a page by including a custom tag.

Alternatively, GPU capabilities can be accessed in the browser

via other libraries built on top of WebGL. Three.js [Dan12] and Babylon.js [bab] are popular JavaScript libraries that provide a programmatic scene graph-based API on top of WebGL rather than an XML document-based scene graph as in X3DOM and XML3D. More details on graphics libraries and frameworks can be found in a recent survey on 3D web graphics by Evans et al. [ERB*14].

Modern browser offer additional capabilities to improve performance besides GPU access. *Web workers* can be used to exploit multi-core CPUs for background processing. *ArrayBuffers* [Khr11a] are typed arrays used for binary data processing. *WebSockets* [FM11] allow for full-duplex persistent communication channels between client and server. Finally, *WebRTC* [W3C11] offers peer-to-peer communication between browsers to transfer real-time audio, video, and data. The direct data communication provided by WebRTC through its *RTCDataChannel* component has the advantage of providing lower communication latencies for collaborative applications compared to WebSockets because it does not involve an intermediate server between the communicating clients. Moreover, peer-to-peer can scale better to larger numbers of concurrent users than a client-server architecture. For example, Desprat et al. [DJL15] implemented a collaborative 3D editor that uses the WebRTC protocol for low-latency peer-to-peer communication and supports direct broadcast of real-time updates between all connected clients. Andrioti et al. [ASK*15] provide implementation details on integrating WebRTC and X3DOM in the context of collaborative applications (educational gaming and video conferencing). In the following subsections, we provide an overview of the different technical approaches that exploit these web standards.

2.4.1. GPU-based Techniques

Techniques that exploit the GPU for rendering in the browser use polygon-based approaches or ray casting. Polygon-based techniques typically use a library built on top of WebGL to generate or import the meshes on the client side and upload them to the GPU for rendering. In other cases the meshes are streamed from a server to the client.

However, polygon-based techniques have some limitations. In order to get smooth surfaces, fine tessellation is required, resulting in frame rates decreasing proportionally to the number of triangles. Furthermore, high numbers of triangles not only consume considerable amounts of memory, but also require considerable CPU-GPU bandwidth each time the geometry changes. Moreover, in a browser environment, generating geometry data on the CPU can result in lower rendering performance if the rendering is stalled by said generation (see, for example, Jmol [Jmo09]). Offloading the generation of geometry to a powerful server may not always help. Network latency and the bandwidth of the network as well as of the CPU-GPU interconnect may still negatively affect the resulting performance. To alleviate these problems, a preferable approach would be to use implicit, parametric geometry rendered using GPU-based ray casting. This ensures that less data is transferred from CPU to GPU and avoids performing expensive computations in JavaScript.

GPU-based ray casting techniques use parameters to define the implicit surface of an object (e.g., center and radius in case of a sphere) are sent to the GPU, and ray-object intersections are computed in the fragment shader to generate the surface of the object. To start the rendering pipeline and generate the fragments, a proxy geometry is usually rendered. For example, a full screen quad or a tight bounding-box for the object can serve as a proxy geometry. These techniques have the advantage of providing high rendering performance and high quality images even for large data [MKB*15]. Depending on the implementation approach used, an acceleration structure may be required for efficient ray-object intersections [MKK*14]. Figure 3 shows some exemplary images produced by these techniques.

GPU-based volume ray marching is the state-of-the-art approach for direct volume rendering due to its performance and high image quality. This technique uses 3D textures (assuming the use of WebGL 2.0) or a set or atlas of 2D textures for data storage. To initialize the rendering, a proxy geometry, usually the bounding box of the volume data is rendered. Similar to OpenGL on a desktop PC, there are two approaches for implementing this technique using WebGL: multi-pass and single-pass GPU-based ray marching. In the multi-pass approach, the proxy geometry (usually the volume bounding box) is rendered into a texture in order to get the entry and exit points of the rays in the volume data. These rays are then used during the volume traversal pass in order to get the starting point and the direction of the sampling rays. In the fragment shader, the volume is sampled along the rays, classified, optionally shaded, and iteratively composited to get the final color of the pixel. Congote et al. [CSK*11] described the implementation of a multi-pass volume ray marching for WebGL 1.0. Due to lack of 3D textures in the WebGL 1.0, the authors used 2D mosaic textures to store the 3D volume using 2D textures.

In the single-pass approach, which is for example used by Mobeen and Feng [MF12], either the front or the back faces of the volume's bounding box are rasterized. Ray traversal is also performed in the fragment shader by marching along the view ray in discrete steps through the volume and compositing the sample colors to get the final pixel color. The main constraints for visualizing large volumes of data in WebGL 1.0 are texture storage limitations and limited support for dynamic looping in shaders. Mobeen and Feng [MF12] introduce a 3D texture slicing technique to address the problem of looping limitations by slicing the volume data with a plane perpendicular to the view direction, as proxy polygons. These polygons are then rasterized and finally blended to obtain the final image.

To address the problem of limited texture size, Noguera and Jiménez [NJ12] present a technique that slices the original volume and uses multiple textures to package the data for rendering. Similar to the aforementioned techniques, 3D textures are stored by tiling the volume data into a single large 2D texture.

A different approach for volume visualization using WebGL is to extract geometry on the server side and performing rendering on the client. Jiménez et al. [JLC*14] presented a method that extracts geometry of the 3D fractal dimension from volumetric data. The geometry is rendered in the client using the Three.js [Dan12] library. The visualization data itself is extracted from the original MRI data in a server-side preprocessing step using CUDA. It is encoded in various LODs and transmitted to the client in XML or JSON formats.

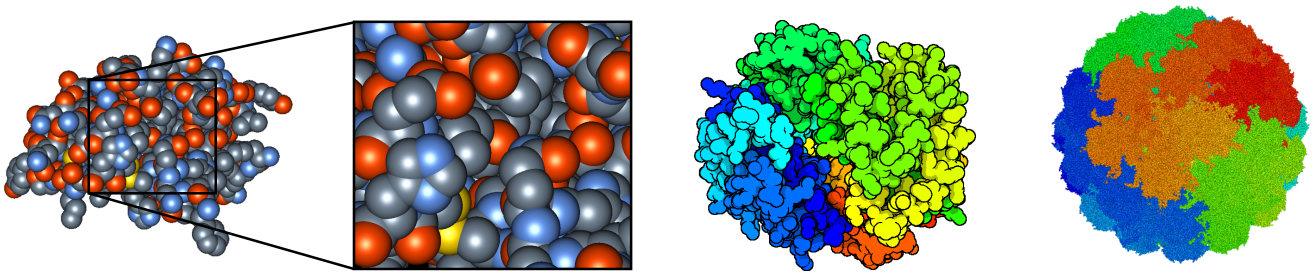

Figure 3: *Visualization of molecular structures at interactive frame rates using WebGL-based ray casting techniques. Left: an insulin protein (PDB ID: 1RWE, 823 atoms) and its close-up view with correct sphere-sphere intersections and pixel-accurate spheres [MKK*14]. Middle: a hemoglobin protein (PDB ID: 4HHB, 4,384 atoms) with toon shading effects generated through deferred shading [MKB*15]. Right: capsid of a papillomavirus (PDB ID: 3IYJ, 1.3 M atoms).*

Jacinto et al. [JKD*12] use an image-based approach that extracts slices from the original volume data and sends them to the client as JPEG or PNG images that are then uploaded to the GPU as textures for rendering. The server side uses VTK library and the client side uses Three.js [Dan12] library.

Mesh rendering. Many mesh rendering techniques use progressive meshes to improve interaction. This is achieved through progressive decoding and rendering of the different levels of detail (LODs) of the meshes. The main idea of progressive meshes [Hop96, Hop98] is to represent the mesh data in a low resolution (base mesh) together with operations (e.g., vertex split) to reconstruct a mesh from the base representation at a desired level of detail. This representation allows for efficient loading, decoding and rendering. This is because only the operations to reconstruct the model from the previously loaded one are necessary in order to get a model representation at a required level of detail. This leads to less data being transferred to the client and hence reduced bandwidth usage.

Sawicki and Chaber [SC12] implement techniques for progressive visualization of 3D Meshes in the browser. Their solution is targeted at mobile devices and focuses on efficient use of network and computational resources. They use the aforementioned a progressive mesh representation to allow streaming of different levels of detail to the client for rendering large models. The progressive mesh format is computed offline and stored on a server. A coarser model is first sent to the client and detailed models are progressively reconstructed based on *vertex split operations* that are streamed and received by the client in the background. Transferring the detailed models is regulated by the client by taking into consideration the capability of the client device and current computational workload. Rendering in the client is handled through the Three.js library [Dan12].

Figueiredo et al. [FRSVP14] describe the implementation of a system for the visualization of high-resolution mesh data via a low-resolution overview and a high-resolution detail view. The simplified low-polygon mesh is sent to the client in X3D format [BD07] and rendered in the browser using X3DOM [BJK*10]. While the simplified model is used for interaction and navigation, data for the detail view is extracted on demand from the original high-resolution mesh stored on the server.

Ponchio and Dellepiane [PD15] present a mesh representation that divides a model into patches with different levels of detail that are independently encoded and compressed on the server. The representation is optimized for fast decoding times. The patches are stored in a tree data structure that captures the dependencies of the patches. The server streams the requested patches in response to HTTP Range requests from the client with each patch requiring a separate HTTP call. The received patches are encoded as an ArrayBuffer and are decoded in a web worker and rendered using the SpiderGL library [DBPGS10]. Although issuing separate requests for each patch has scalability problems, rendering performance does not suffer as long as it is done in a separate thread and communication between a data fetching thread and a rendering thread is asynchronous.

Another progressive representation that is optimized for fast decoding times is presented by Lavoue et al. [LCD14]. The model is again stored in an ArrayBuffer, similar to the work of Ponchio and Dellepiane [PD15]. A client first loads a base mesh from a server and other levels of detail are reconstructed progressively by applying *vertex split operations* to the model that is already received and decompressed. To get the next level of detail, the client issues a single HTTP request (using an XMLHttpRequest object). The implementation uses a web worker for decoding and Three.js library [Dan12] for rendering. The decoding and the rendering thread communicate asynchronously to ensure that the two operations are decoupled.

Wen et al. [WJL14] use a *Light Weight Progressive Mesh (LPM)* representation that simplifies the original model based on similarity of components making up the model. The representation uses a scene graph structure to store only unique elements of model components and transformation information for all their instances. The unique components of the model are represented at different levels of detail to allow progressive streaming and rendering. The benefit of this approach is that the client does not require a decoding step because instanced rendering can be used directly to render the model. Again, a web worker is used to receive and progressively refine the base model as data for new levels (vertex splits) are streamed to the client. However, the transferred data uses the JSON format, which can be less efficient in terms of bandwidth and client side processing compared to an ArrayBuffer.

Techniques based on progressive meshes do not only help improve rendering and interactivity but also help reduce bandwidth usage by reducing the number of triangles transferred to the client. Combining progressive meshes with compression yields better results in terms of bandwidth and latency and hence offer significant improvements compared to when each technique is used individually. The work of Ponchio and Dellepiane [PD15] is an example of an approach that combines progressive meshes with compression.

2.4.2. SVG/2D Canvas-based Techniques

As mentioned in section 2.4, the `<canvas ...>` element provides a 2D graphics drawing API in addition to the 3D API provided through WebGL. It offers built-in functions for drawing paths and several 2D shapes and text using JavaScript.

D3.js [BOH11] is a JavaScript library for information visualization that builds on the document object model (DOM) of the browser as well as CSS and SVG. SVG (scalable vector graphics) offers an alternative approach for 2D graphics rendering using a markup language. D3.js allows users to bind data to be visualized to DOM elements and manipulate their element properties based on the attribute values of the underlying data. It exposes the native document model rather than providing its own data model, which can introduce some overhead in translating between the two models. Moreover, direct manipulation of the DOM can lead to poor performance, especially for large data sets, as the browser may be required to layout, paint, and composite whenever the DOM changes. However, by using the DOM model, seamless interoperability with other web standards is ensured.

2.5. Data Encoding and Transfer Techniques

Efficient data encoding and transfer are important for interactive visualization in the browser. Without efficient data transfer formats, bandwidth and latency issues can lead to long waiting times until data is available on the client. In case of dynamic data, this can hamper real-time data updates for smooth animations of time series. While compression reduces the amount of data to be transferred and hence save bandwidth, it cannot guarantee interactive data updates by itself. This is because overhead in compression and decompression times can outweigh the savings in transfer times.

Ideally, the entire data transfer pipeline from data encoding, transfer, decoding to rendering should be optimized to obtain balanced end-to-end performance. Most techniques focus on optimizing only one or a combination of pipeline stages. The bottleneck is usually in the decoding and rendering stages since these are performed at the client side. Various compression and decompression approaches proposed for the web focus on allowing for efficient decoding and rendering rather than only optimizing compression ratios. Limper et al. [LWS*13] investigated this taking into consideration the capabilities of different client devices and network bandwidth capacity.

Streamable and progressive mesh formats (see also section 2.4) are an established means to hide latency and improve interactivity and consequently user experience. The mesh data is usually represented at different levels of detail and each level is encoded as a separate chunk of data that can be progressively decompressed and rendered on the client. Most approaches focus on efficient CPU-based decoding. Another option is to use GPU-friendly formats, which can be uploaded directly to the GPU. In this case, the decoding is executed entirely on the GPU. This does not only improve performance by reducing decoding time on the client, but also requires less CPU-GPU bandwidth. We will discuss examples for these different categories in the following.

Lavoue et al. [LCD13, LCD14] use a CPU-based decoding technique that leverages a web worker for background processing. By decoupling the decoding thread from the main thread, rendering and user interaction performance is not affected by long decoding times. Data is stored in an ArrayBuffer for efficient processing.

Wen et al. [WJL14] present a progressive mesh compression technique that is based on identifying similar parts in the model and replacing redundant parts with instances of a single occurrence. A progressive mesh is subsequently obtained by creating continuous levels of detail (CLODs) of each part (*Lightweight Progressive Meshes*) that is then streamed and rendered progressively in the browser using WebGL.

Limper et al. [LJB*13] present two progressive mesh encoding approaches (*Progressive Binary Geometry* and *Sequential Image Geometry*) based on the idea of using images for encoding geometry data. The benefit of using standard image formats is that decompression can be handled by the browser without requiring decoding in the application code. Decoding can be performed on the CPU or on the GPU depending on the texture fetch support in the vertex shader stage. Their performance results show that Progressive Binary Geometry performs better with respect to progressive decoding. A related approach has been presented by Dworak and Pietruszka [DP15]. They describe how to encode 3D mesh data (in .OBJ format) using the PNG file format for efficient data transfer and lossless compression.

Limper et al. [LTBF14] present a general container file format called *Shape Resource Container (SRC)* optimized for streaming 3D mesh data and direct data upload to the GPU. The original mesh data is split and encoded into separate *buffer chunks* that can be transmitted progressively to the client. The geometry and texture data are transmitted in interleaved manner to allow for proper decoding of textured geometry. The different chunks encode different levels of detail of the mesh. In order to minimize the number of request calls, the buffer chunks are encoded in a single binary file. This differs from an earlier format [LJBA13] that stores each chunk as a separate file downloaded in a separate HTTP request. The implementation of the container is integrated into the X3DOM framework. By using a GPU-friendly binary buffer representation [BJFS12], the transmitted data can be directly uploaded to the GPU for rendering without incurring decoding overhead. The concept has been enhanced into a more generic approach [BMP*15] that can switch to image streaming depending on client capabilities. This also includes a hybrid stategy to enable server-side culling if required.

Sutter et al. [SSS14] present a general container format for streaming binary data that uses a chunked data representation. The chunks are independent and each consists of a header and a body part. Each chunk of data can store a different level of detail for progressive rendering or a different encoding for adaptive transmis-

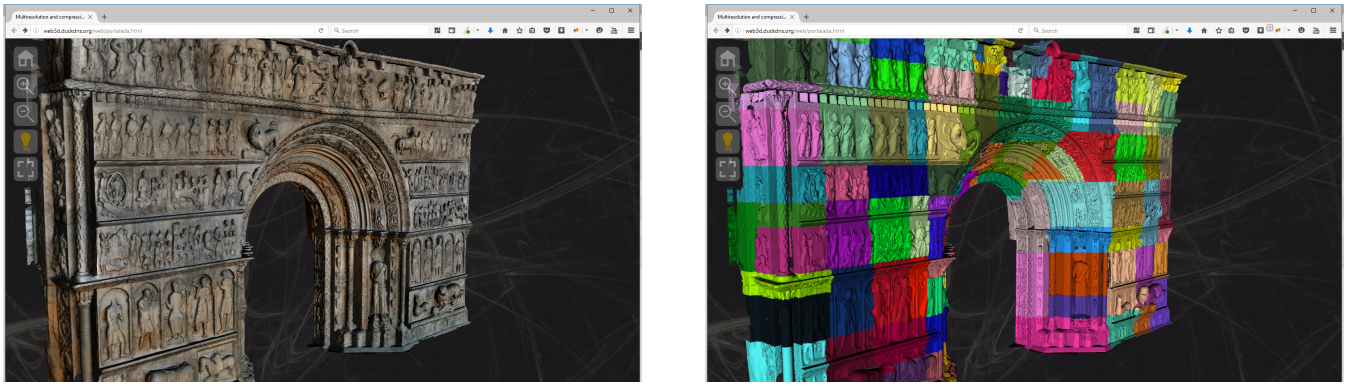

Figure 4: *Interactive visualization of a large mesh in the browser using view-dependent compression [PD15]. The right image shows the resolution of the model as seen from the view point shown in the left image. Fine details are only used for regions that are close to the camera to reduce bandwidth requirements (image created using the authors' tools: web3d.duckdns.org, last accessed 2016/04/29).*

sion depending on various network bandwidth conditions. In their implementation, chunking is used as a way to handle network connections with various network bandwidths by allowing each chunk to contain different data sizes depending on the capacity of the connection. Since data chunks are independent, they can be transmitted independently and processed in parallel. It is, however, also possible to transmit multiple resources in a single request to reduce latency. The implementation is integrated into the XML3D framework. They further provide benchmark results with other existing formats.

Ponchio and Dellepiane [PD15] present a technique for mesh rendering in WebGL that combines view-dependent levels of detail, progressive decoding, and mesh compression. Their goal is to minimize decoding time. The technique leverages HTTP range requests for progressive loading of each model level and ArrayBuffers for efficient processing in JavaScript. The technique compresses both geometry and connectivity information. Level-of-detail representations allow progressive rendering on the client where a low detail model is first rendered and progressively refined as the detailed representation becomes available. Since the different representations are independently decoded and rendered, this minimizes decoding time and hides latency, thus improving interactivity. Additionally, compression allows for efficient use of bandwidth. View-dependent level-of-detail techniques allocate more details for the models that are in view while those at the periphery are represented at lower levels of detail. Figure 4 shows an example of this view-dependent level-of-detail visualization.

Yang et al. [YSG15] present a compression technique for time-varying volumetric data that combines the S3TC texture compression standard with DEFLATE compression techniques for efficient transmission of weather data to the browser through a web socket connection (see figure 5). The technique works by encoding scalar values of the volume into RGBA image components prior to applying compression. On the client side the compressed data is inflated to get the compressed texture and uploaded directly to the GPU as video textures for rendering (WebGL has direct support for video textures). The technique is implemented as an X3D extension to the X3DOM framework.

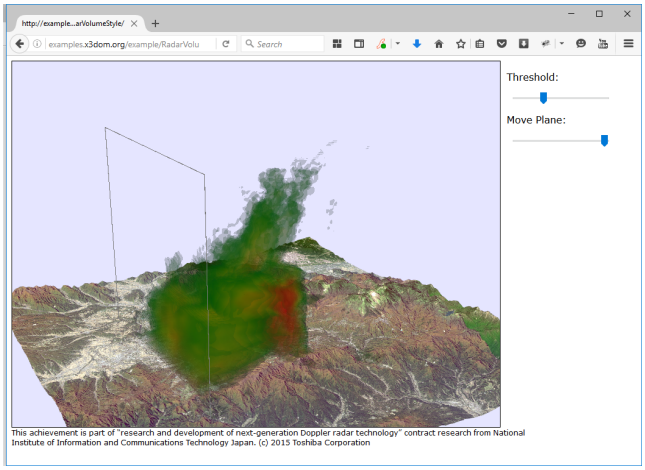

Figure 5: *Volume visualization of meteorological data [YSG15]. The volumetric data is compressed before sending it to the client to reduce data transfer times. (image created using the online demo: http://examples.x3dom.org/example/RadarVolumeStyle/)*

3. Web-based Visualization Applications

In this section, we give an overview of web-based visualization applications from various application domains. These applications use the methods described in section 2.

3.1. Particle Data Visualization

Molecular visualization in the browser has been popular from the early beginnings of the web. The visualization of molecular data as calotte model is an example for particle-based visualization where each atom is depicted as a sphere. The size of each sphere represents the van-der-Waals radius of the corresponding element. When using polygon-based rendering, these spheres have to be tessellated into triangles that are then rasterized.

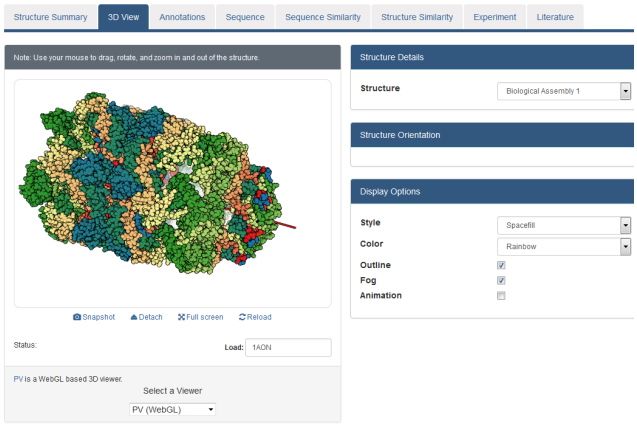

Figure 6: *Interactive visualization of the chaperonin complex (PDB ID: 1AON, 58,674 atoms) on a webpage. (image produced using PV protein viewer [pvp])*

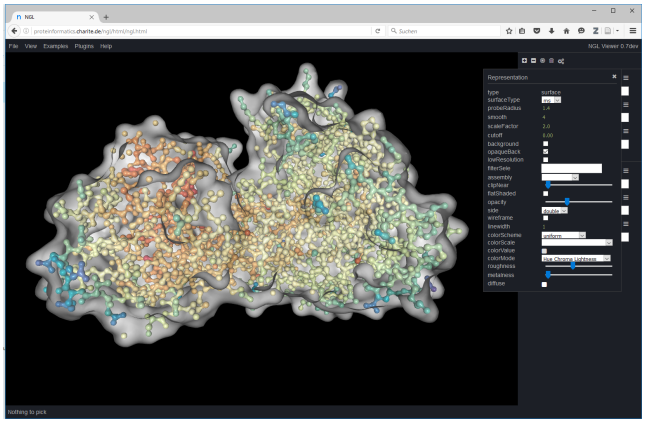

Figure 7: *Protein visualization using hyperballs and a semi-transparent molecular surface in the NGL Viewer [RH15] (PDB ID: 1VIS). (Image created using the official NGL website: http://proteinformatics.charite.de/ngl/html/ngl.html)*

Jmol [Jmo09], implemented as a Java applet and its recent re-implementation in JavaScript, JSMol [JSm13] are some of the most popular tools used for visualizing protein structures. The main limitation of Jmol is that it requires users to install a browser plugin. While JSmol alleviates the plugin problem, it still lacks the capability of handling large molecules. Both Jmol and JSMol use polygon rendering. Callieri et al. [CADB*10] also use polygon-based rendering for the visualization of molecular data via the SpiderGL library [DBPGS10]. Additional recent examples of web-based molecular viewers are NGL Viewer [RH15], 3DMol.js [RK15], and PV [pvp]. Figure 7 shows a visualization produced by the NGL Viewer. They all leverage WebGL for visualization of molecular structures in the browser. The RCSB protein data bank [BWF*00] web site, for example, uses PV as one of its integrated protein viewers. Figure 6 shows one of the visualizations produced using PV. iView [HLW14] is another protein viewer that also incorporates virtual reality features. Li et al. [HLW14] presented a visualization tool for protein-ligand complexes that was built using Three.js [Dan12]. Beside the aforementioned calotte model, most of these viewers support other commonly used visualizations for molecular data (e.g., molecular surfaces or ball-and-stick models), which are derived from the particle positions. These additional representations are all the same based on a polygonal representation and rendered using the same techniques.

Visualization of dynamic molecular data (*trajectories*) from molecular simulations is also an important feature for domain scientists. This capability has up to recent years been confined to desktop solutions. As discussed in section 2.4.1, two main approaches can be used for rendering spheres: GPU-based ray casting and mesh rendering. GPU-based ray casting techniques offer better results in terms of rendering performance and image quality.

Mwalongo et al. [MKK*14] present a client-server approach for visualization of static molecular data sets involving a grid acceleration structure to speed up ray-sphere intersection calculations. They implement two different approaches for generation of the acceleration structure as a preprocessing step. Their results show that generating the acceleration structure in the server gives better performance compared to generating it on the client-side using JavaScript. This setup can be generalized to any visualization pipeline where a computationally demanding preprocessing step is offloaded to the server/cloud and rendering is performed on the client. They also follow the concept of decoupling the rendering loop from the server processing and network transfers to avoid stalling the rendering thread.

Although the use of acceleration structures created in a preprocessing step works fine for static data, it can become a bottleneck for dynamic data. Solutions that avoid this preprocessing step become important for dynamic data. Such an approach is used by Mwalongo et al. [MKB*15]. Their solution is capable of visualizing time-varying molecular data from molecular dynamics simulations with over one million atoms at interactive frame rates (see figure 3). The application achieves this performance by using GPU-based ray casting for rendering instead of traditional polygon-based rendering. Moreover, the implementation decouples fetching data from the server from the rendering loop through a web worker to ensure interactivity. Persistent and bidirectional communication between client and server is achieved through the use of a WebSocket to avoid overhead of establishing new network connections for each new request.

Chandler et al. [COJ15] present a WebGL remote visualization solution for smoothed particle hydrodynamics (SPH) simulation data. The server side preprocesses data by creating an octree data structure for storing a multi-resolution model for each time step. Streaming of data to the client is done for each individual time step. The client initially receives a low-resolution model from the server and then high resolution models are incrementally loaded as the rendering progresses. This is done to avoid long latencies before the first view of the model is rendered. The received octree data structure is encoded as a texture and uploaded to the GPU during rendering. Since the solution uses a WebGL 1.0 implementation

Figure 8: *Example of medical volume rendering in WebGL using volume ray marching. (image source: [MF12] © 2012 IEEE. Reprinted by permission.)*

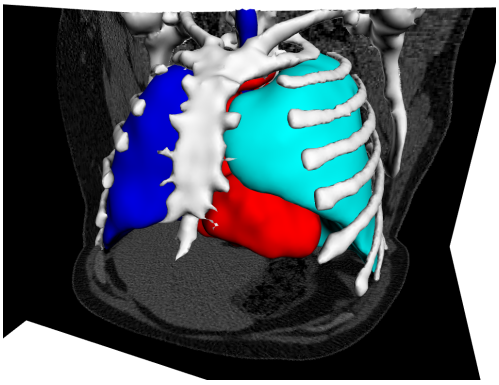

Figure 9: *Example of the web-based medical volume rendering presented by Jacinto et al. [JKD*12]. The isosurface meshes were extracted on the server using Marching Cubes and rendered on the client using WebGL. (image source: [JKD*12] © 2012 Association for Computing Machinery, Inc. Reprinted by permission.)*

that does not support 3D textures, the textures are packaged as 2D textures and address translation is used to access the data in the shaders. Moreover, fetching and processing of data in the client is done in a web worker to avoid stalling the rendering or UI interactions in the main thread.

3.2. Volume Visualization

As mentioned in section 2.4, Congote et al. [CSK*11] presented an early work on volume visualization using GPU-based ray marching in WebGL. They use a multi-pass GPU-based volume ray casting approach to visualize volumetric data from medical imaging and weather radar scans. Their work forms a basis for the implementation of MedX3D in X3DOM—MEDX3DOM [Con12]. MedX3D [JAC*07] is an extension of the X3D standard [BD07] to support web-based volume visualization. Other volume visualization systems for large medical data were introduced by Mobeen and Feng [MF12] (see figure 8) and Noguera and Jiménez [NJ12]. Both systems also run efficiently on mobile device browsers.

Jiménez et al. [JLC*14] presented a client-server application for the analysis of the 3D fractal dimension of MRI data.

Jacinto et al. [JKD*12] presented a client-server application for medical volume segmentation and rendering. The segmentation uses an image-based approach that extracts slices from the original volume data on the server and sends them to the client as JPEG or PNG images, where they can be annotated to trigger the segmentation on the server. Isosurfaces are extracted from the segmented volume and sent to the client for visualization. The server side uses the VTK library and the client side uses the Three.js [Dan12] library for rendering the isosurface meshes (see figure 9). One targeted application is the fast extraction of knee bones for prosthetic design.

A medical visualization application that exploits augmented reality capabilities has been shown by Virag et al. [VSTA14]. The system is designed using a client-server architecture, but also takes advantage of WebRTC for accessing a camera and transmitting the imagery to remotely allow for a second opinion from another physician. The camera feed is also used as input for the JSARToolKit, which takes care of marker tracking. That way, a virtual model of a patient equipped with said markers can be synchronized with the visualization.

Hou et al. [HSZ15] presented an image-based technique to visualize medical volumes. They used a server-side GPU-accelerated rendering and stream each generated frame to the client for display. The rendering engine on the server uses CUDA and the VTK library for rendering the data to a framebuffer object (FBO) that is then read back and sent to the client. The advantage of this approach is that it benefits from powerful computational and storage resources on the server side. However, by doing the rendering on the server, smooth interaction on the client is affected due to round trip network latency.

3.3. Geospatial data Visualization

Visualization of geospatial data, including spatial-temporal data (4D), on the web covers various kinds of data like marine data [RWW14], raster-images [JaL16], weather forecasting data [DPD*15], and city models [GVB*15]. Common among all modern web-based geospatial visualization is the use of WebGL for hardware-accelerated rendering and other modern HTML5 technologies. Some approaches [RWW14, GVB*15] use WebGL directly and others [MYB14, KPS15] use declarative frameworks like X3DOM. X3DOM implements a component from the X3D standard for visualization of geospatial data [PM15].

The overview-and-detail visualization for mesh data by Figueiredo et al. [FRSVP14] described in section 2.4.1 was used for the visualization of geomorphological structures of an underground cave in the browser. A 3D mesh representing the surface of the cave was generated from a detailed point cloud and stored on a server at different levels of detail. The overview can be used to navigate and select stalactites or stalagmites and request a high-resolution representation of the filtered neighborhood of these relevant structures for detail analysis.

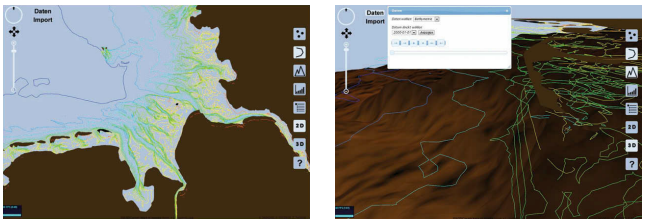

Figure 10: *Examples of a web-based geospatial visualization application. (image source: [RWW14])*

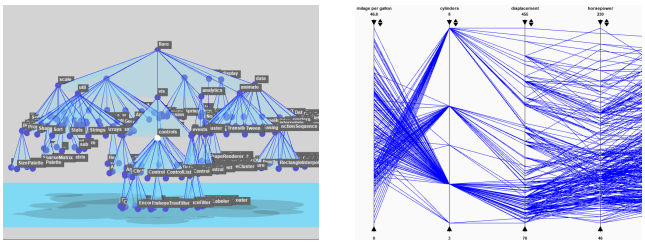

Figure 11: *Examples of web-based information visualization using FluidDiagrams [AW14]. (image created from examples on the author's homepage* `http://digitalwright.net/msc/examples/`*, last accessed 2016/04/29)*

Resch et al. [RWW14] describe a prototypical implementation of a web application for visualization of 3D time-dependent geospatial data to support non-experts in decision making (see figure 10). They further discuss various user requirements including usability, representing 4D geo-data for such applications, their implementation aspects and performance. Moreover, the authors highlight challenges of representing spatial-temporal data in web-based visualization tools.

Jenny et al. [JaL16] introduce hardware-accelerated projection techniques for raster images used in web-based geospatial visualization. These techniques allow users to control the center of the map at any location, giving them the flexibility to view the map at different global scales with minimal distortions.

Gaillard et al. [GVB*15] present a client-server framework for visualizing large city models and discuss the importance of progressive meshes for high performance rendering. They use progressive textures in order to render large city models containing gigabytes of data at interactive frame rates. Multiple resolution textures are generated from the data and stored on the server and then transmitted to the client for rendering using WebGL.

3.4. Information Visualization

Sarikaya and Gleicher [SG15] present a WebGL-based implementation of the Splatterplots technique [MG13] for interactive visualization of large two-dimensional point data sets. Splatterplots address the problem of overdraw in regular scatter plots that arise when visualizing large point data sets. Computations that were performed on CPU in the original implementation [MG13] were moved to GPU using *render-to-texture* techniques in order to attain better performance.

Liu et al. [LJH13] present techniques for visualization of geolocated data in WebGL. They use a client-server architecture where the server preprocesses the data before streaming them to the client for rendering. The preprocessing involves pre-computing data tiles from data cubes and then encoding them as image files. These images are transferred to the client and uploaded to the GPU as textures. On the client side, the authors employ a two-pass approach for rendering: The first pass is a computation step. It uploads the data tiles to the GPU and computes summary values. The results are written to a framebuffer object (FBO). The second pass is a rendering pass that uses the textures generated in the first pass to render the final images. The focus of their techniques is to achieve *interactive and perceptual scalability* for visualization of large data sets. In order to achieve this scalability, they use binned aggregation techniques to reduce the amount of data to be visualized and exploit GPU acceleration for parallel computation and rendering. Data reduction techniques combined with the use of data tiles allow efficient visual analysis of large data sets.

3.5. Generic toolkits

Several generic web-based visualization toolkits exist, covering a range of application domains including information visualization, protein visualization, medical visualization, and geospatial visualization. Most of these toolkits discussed in this section use client-side rendering that employs modern browser-based web technologies like WebGL.

FluidDiagrams [AW14] is a prototypical toolkit that offers GPU-accelerated implementations of various information visualization techniques (e.g., bar charts, scatter plots, line charts, hyperbolic browser, parallel coordinates, and cone trees) using the Three.js [Dan12] library (see figure 11). The authors demonstrate that their WebGL-based implementation is capable of scaling to large data sets and achieves higher frame rates compared to those based on SVG (e.g., D3 [BOH11]) or the 2D Canvas API [W3C15].

Goecks et al. [GET*13] discuss various server and client web-framework components for simplifying the creation of visual analysis applications for high-throughput genomic data. They further describe how the components are used to create several concrete applications. The components are implemented and integrated into the Galaxy web platform [GNTT10].

ManyEyes [VWvH*07] was a public information visualization website that allows users to upload their data, visualize them, and finally annotate the visualizations while sharing them with others.

Popular commercial cloud-based information visualization solutions include Tableau Online [tab] and TIBCO Spotfire Cloud [spo]. These solutions provide Software as a Service (SaaS) tools for creating various interactive visualizations using local or online data and allow sharing the visualizations amongst users. These commercial solutions are usually designed to scale to very large data sets and many concurrent users.

The X Toolkit [HRA*] is a WebGL-based framework for interactive visualization of medical data sets in the browser. The

toolkit supports various data formats and visualizations from the neuroimaging domain. Additionally, it supports various data formats for surfaces and volumes. Using this toolkit, Haehn et al. [HKBR*14] developed a web-based application for collaborative proofreading of brain data that supports 2D and 3D visualizations.

Cesium.js [Gra12] is a general library for geospatial data visualization that relies on WebGL for fast rendering. It supports various standard data formats as well as the glTF format [Khr15] for the exchange of 3D data between client and server.

Jomier et al. [JJAM11] discuss the integration of ParaviewWeb [JAG10] with a medical and scientific data management system (MIDAS) [JAM*09] for visualization of large data sets. The system uses a remote rendering approach where images are rendered on the server and streamed to the client for display. Their integration setup uses a client-server architecture between the data management system and the visualization server. The user selects what data to visualize through a web-based user interface of the data management system. Subsequently, the visualization request is forwarded to the visualization server.

Badam and Elmqvist developed Polychrome [BE14], an application framework that allows synchronized interaction with web-based visualization. This mainly focuses on the interaction side of visualization and can be used for online collaboration as well as UI synchronization across devices or remote control. Interaction mode extensions to X3D/X3DOM concerning navigation and manipulation for web-based CAD applications are also described by Mouton et al. [MPJ*14]. However, as mentioned in the introduction, we will not further elaborate on interaction and collaborative systems as this is out of scope for this state-of-the-art report.

4. Classification for Web-based Visualization

We propose a classification of web-based visualization applications that enables the assessment of the technical foundations and applied methods of the respective visualization. Our classification is based primarily on the infrastructural distribution of storage and rendering effort. The next level is divided by the technical and algorithmic approach to rendering, which is tightly coupled with the type of data that is visualized. The lower levels specify the optimizations applied to data transfer and rendering. We also included miscellaneous features, namely the employment of compression algorithms and the exploitation of GPU acceleration, as both can have major influence on the interactivity of the resulting approach. These features are treated separately from the other lower levels since they provide a generic and data-agnostic means for accelerating the visualization.

Table 1 gives an overview of the web-based visualization applications introduced in section 3. The table classifies the applications based on the aforementioned criteria. The resulting overview shows which technical approaches and infrastructures were used by which application domain. We sorted the available visualization applications by the targeted application area to provide direct access to the available techniques based on the scenario a practitioner has at hand. In the following section, we will discuss visualization scenarios that would potentially benefit from web-based visualization and assess their practicability based on this classification of already available methods and applications.

5. Discussion

There are a number of relatively obvious benefits to web-based visualization, especially from the perspective of potential users. When there is no need for installing specific software to gain access to visualization, there is basically no hurdle putting it to use. Such a configuration has no restrictions reaching out to potential users and is thus optimal for dissemination and education. Due to the ubiquity of browsing capabilities via smartphones and tablets, users with Internet access over the air can access any visualization on the go. This development is only slightly hampered by the development speed of WebGL and the GPU capabilities of mobile devices. With the mobile scenario in mind, it is effectively a benefit if users do not store the data they want to visualize on their workstation, but on some centralized storage system. That way, it makes no difference from which client visualization and analysis will be performed. The nearer the storage to the visualization service, the better, be it on the machine serving the visualization web pages or some dedicated storage system closely coupled with the web server. Scaling the storage space to grid or cloud does affect the bandwidth and latency to the web server component, however the infrastructure on that end will rarely be the limiting factor, especially when also considering over-the-air clients. Additionally, it must be considered that time-dependent data automatically increases linearly in size with the number of time steps compared to the static case.

Once the visualization ecosystem is configured that way, some limiting factors that typically affect low-cost clients can be removed automatically. Large-scale spatial data e.g. from current simulations can be stored much easier on server-class hardware or specialized storage systems since these are far easier to scale than client storage. In that case, problems arising from long download times or limited local storage on the client side are automatically mitigated. Concrete examples from scientific visualization follow.

Considering the use case of flow visualization, the data need not necessarily be excessively large to be taxing to visualize, since many flow visualization techniques require long integrations, for example when computing streak- or streamlines [SD11, KSWE16] or FTLE separatrices [BSDW12, SUEW12]. If the integration is performed on the server, the base representation of the vector field needs to be transmitted once per configuration, using any optimization as per section 2.5. Post-classification is cheap enough to be applied on the client, so the user can interact with the color map and highlight values of special interest. Additionally, line geometry needs to be transmitted, which can be optimized as shown by Ponchio and Dellepiane [PD15]. WebGL enables rendering even a large number of lines interactively. One popular application that is also interesting to the general public as well as scientists is represented by weather data like forecast simulations or real-world measurements. Here, pre-rendered visualizations could be replaced by user-explorable ones where the user can dynamically choose the region of interest, important regions of the depicted attributes (only rain or wind worse than a determined amount, as comfort is a very personal measure) and freely navigate the simulated time.

A second use case is found in the visualization of particles, for

Table 1: *Web-based visualization tools and methods from various application domains. The visualizations are classified according to the criteria given in section 4 (bullets). They are sorted according to the application domain in order to provide quick overview of the methods that were applied in these fields. Within one application domain, the visualizations are sorted by year to show potential trends.*

Reference	Infrastructure				Technique				Transfer			Optimization			Misc.		Application
	Web Service	Grid-Based	Cloud-Based	Local	Volume	Raycasting	Triangle Mesh	SVG/2D Canvas	Full	Progressive	Simplified	Latency	Bandwidth	Rendering	Compression	GPU	
[SC13]				•													3D Models
[LJB*13]				•			•			•	•	•	•	•	•	•	3D Models
[LCD14]				•			•			•	•	•	•	•	•	•	3D Models
[FRSVP14]				•			•			•	•	•	•	•		•	3D Models
[LTBF14]				•			•			•	•	•	•	•		•	3D Models
[WJL14]				•			•			•		•	•	•		•	3D Models
[GVB*15]				•			•			•	•	•	•	•		•	3D Models
[PD15]							•			•	•	•			•	•	3D Models
[BDG*04]	•	•					•										Computational Steering
[KHRV04]	•	•					•				•						Computational Steering
[d'A11]	•	•	•														Computational Steering
[KMS*15]	•	•	•				•				•	•	•			•	Computational Steering
[RWW14]				•			•									•	Geovisualization
[JaL16]				•			•									•	Geovisualization
[DPD*15]				•				•									Geovisualization
[KPS15]				•			•			•	•					•	Geovisualization
[MYB14]				•			•									•	Geovisualization
[PM15]				•			•									•	Geovisualization
[spo]			•														Information Visualization
[tab]			•														Information Visualization
[GET*13]				•				•									Information Visualization
[LJH13]				•												•	Information Visualization
[AW14]				•			•									•	Information Visualization (Toolkit)
[SG15]				•			•		•							•	Information Visualization (Toolkit)
[CADB*10]				•			•									•	Molecular Visualization
[HLW14]				•			•		•							•	Molecular Visualization
[MKK*14]				•		•					•	•	•	•		•	Molecular Visualization
[MKB*15]				•		•					•	•	•	•		•	Molecular Visualization
[pvp]				•			•									•	Molecular Visualization
[RH15]				•			•								•	•	Molecular Visualization
[RK15]				•			•									•	Molecular Visualization
[CHM04]	•	•					•										Particle and Volume Visualization
[COJ15]				•	•	•				•	•	•		•	•	•	Particle Visualization (SPH)
[KZSB10]	•	•			•		•				•						Volume and Flow Visualization
[CSK*11]			•		•	•										•	Volume Vis. (medical and meteorological)
[Con12]			•		•	•										•	Volume Vis. (medical)
[JJAM11]				•	•						•						Volume Visualization (medical)
[JKD*12]				•	•		•				•	•				•	Volume Visualization (medical)
[MF12]				•	•	•								•		•	Volume Visualization
[NJ12]				•	•											•	Volume Visualization
[JLC*14]				•	•		•				•	•	•	•		•	Volume Visualization (medical)
[HRA*]				•	•		•									•	Volume Visualization (medical)
[HKBR*14]				•			•								•	•	Volume Visualization (medical)
[HSZ15]	•			•	•						•		•			•	Volume Visualization (medical)
[YSG15]				•	•								•		•	•	Volume Visualization (meteorological)

example molecular dynamics for materials [GBM*12] or chemistry [LBPH10], and SPH for fluids in general [AIAT12] or cosmological simulations [FSW09]. Here, the simulated data can consist of hundreds of millions of particles [FAW10, KWN*14], which without careful preprocessing or specific hardware cannot be rendered locally any more. This calls for remote aggregation to some LOD representation [MAPV15] and transmission of the resulting small subsets of the data for local rendering, which scales to the order of 10^6 [GRE09]. As the simulations that generate such data sets run for considerable time on large clusters, also the scenario of ubiquitous simulation monitoring is an important application. The user can periodically connect to a running simulation for a rough overview that helps decide whether the run is converging or exhibiting some major problems. This is useful to reduce costs for misconfigured or otherwise faulty simulations. Another typical use case is post-mortem analysis of the simulation data. Here, the aforementioned issue of data set sizes that scales linearly with the length of the simulation becomes increasingly important since MD simulations usually require long time scales while maintaining sufficiently small time differences. Consequently, data transfer has to highly optimized in order to maintain interactive updates.

Volume visualization is also a common means for analyzing medical data at different scales as well as materials. Besides the technical data size issues, remote access is a great benefit in the medical scenario, as it allows multiple experts from disjunct locations to view patient data and discuss their findings. Hence, the web platform with chat and discussion functionality can be valuable, especially since many services are becoming available that allow for remote conferencing without installing a client, like Google hangouts, Discord, or Skype. Furthermore, collaborative web-based visualizations that allow users to make and share annotations in real time could significantly reduce the time to treatment, which can often be a life-saving factor.

Abstract data, respectively web-based information visualization techniques constitute the other large portion of possible application areas. Ensuing from the medical visualization scenario, the single data set visualization as described by Jiménez et al. [JLC*14] could be complemented by visualizing patient data, for example at the gene level [VKB*15], to cross-reference similarities in CT/MRI data with genetic features of the patients. This way, a holistic approach to patient data analysis can be devised, potentially revealing patterns that allow to predict illnesses from genetic predisposition to optimize treatment.

Abstract data currently is mostly associated with so-called big data, i. e. data that is produced by a large number of users utilizing some kind of service. These services are usually web-based themselves, like search engines, social networks, other online communities, and web stores. This means that data is already in the cloud and as such naturally consumed and analyzed in this self same environment. One simple example of such an analysis is the web-based piwik system (`http://piwik.org/`), which collects site access and navigation patterns and displays them using jqplot (`http://www.jqplot.com/`) and tabular layouts. The analysis is used to check for impact and analyze so-called conversion rate, i. e. the conversion from a normal site user to a paying customer, and how it might have been triggered. Another example would be the real-time aggregation and analysis of hot topics across twitter, YouTube and flickr provided by ScatterBlogs [BTW*11, BTH*13] to obtain situational awareness. Since this data is already located in the cloud, a cloud-based analysis solution is a natural choice. The same is of course true for other social media networks such as facebook, where a lot of effort is put into data warehousing and analytics infrastructure [TSA*10].

Recently, open data initiatives are appearing in a lot of areas. A prominent example is government data, which is offered by many countries (e.g., `https://www.govdata.de/` for Germany or `https://www.data.gov/` for the U.S.). These sites offer a huge amount of data from diverse sectors, for example reaching from finance and economy to air quality and college scores. Consequently, these data are again of interest to the general public as well as experts, for example economists. Visualizations are needed to conveniently access these data. Web-based visualization applications are especially suitable in this field since they grant fast and convenient access to this data without the need to install complex software. Here, the amount of data that has to be transferred to the end user is often not very large; the main challenge is rather providing simple mechanisms for extracting the sought information as well as providing a comprehensible visualization for users with different levels of expertise.

These aforementioned application scenarios are only a few examples of application areas that would benefit from web-based visualization or that are already using it and would benefit from improvements. As discussed in section 2, the technological foundations for building interactive high-quality visualization applications are already available today. The visualization applications and prototypes discussed in section 3 can act as references for future developments. On the one hand, our classification of these applications given in section 4 facilitates this by providing an overview of technologies that were successfully applied for different application domains. On the other hand, Table 1 also shows white spots where technical aspects have not yet been exploited or where technical problems have not yet been solved for the respective domain. One obvious example is the absence of efficient optimization methods to reduce latency, bandwidth, or rendering times for information visualization. It remains to be investigated if the currently available optimization methods are not suitable for this domain or if optimization was not yet necessary due to the nature of the data. However, with the growing availability of big data, we estimate this to be an area that will gain importance in the near future.

Another condition that is apparent in Table 1 is the scarce use of grid- and cloud-based infrastructures. Although there are promising examples for the successful application of these infrastructures, the problem of cost remains. Naturally, the cost for acquiring and operating the infrastructure rises with the number of users. This might prevent providers of web-based visualization applications from choosing this direction. As discussed below in section 6, cloud-based visualization might be a promising direction for future applications.

Another issue is that the bandwidth available for data transfer does not improve with the same rate with which data set sizes grow. As mentioned above, moving large amounts of data is a yet unsolved problem. This problem is especially prominent in case

of mobile use. Compression methods and optimization methods that either reduce the amount of data that has to be transferred or use progressive data models can only counteract this to a certain point. Therefore, although visualization technologies for the web have made huge advances in recent years, using server-side rendering where only the final images are streamed will probably still be a viable alternative to client-based rendering in the near future, even though these methods have their own inherent limitations and drawbacks. Consequently, developers designing new web-based visualization applications have to choose carefully from the pool of available methods to provide the best user experience.

6. Conclusions and Challenges

6.1. Conclusions

Motivated by rapidly growing data sets from simulations, sensors, and other digital information sources, remote visualization is gaining a renewed interest and strives to leverage powerful computation resources to enable scientists and engineers to make sense of the massive data sets.

The main challenges for remote visualization are still bandwidth and latency. Of the two main challenges for remote visualization, bandwidth has received the most attention through various compression techniques (e.g., LZ4 [lz4] and ZFP [Lin14]) and video encoding techniques. Latency has not been adequately addressed. This makes latency reduction and latency-tolerant techniques an important agenda for research in interactive remote visualization.

There has been much improvement on the computational resources available on the client side. It is now common for mobile devices (e.g., smart phones, tablets, and laptops) to be equipped with multi-core CPUs and graphics cards that can handle computational tasks that a few years ago were restricted to desktop computers. As the trend towards collaborative research and mobile researchers accelerates, visualization techniques that harness the power of mobile devices and server-side technologies (e.g., cloud computing) will become more attractive.

Remote visualization with remote rendering and image/video streaming is based on the assumption that the client machines do not have enough computational power to perform rendering at interactive frame rates. However, this is now changing due to improvements on CPU and GPU technology on mobile devices and networking technology. Hybrid visualization approaches, where expensive preprocessing steps are offloaded to the server or cloud and rendering performed in the client, are more promising because they exploit the entire spectrum of the available computational resources. Moreover, rendering on the client side addresses the problem of network latency, which is crucial for interactive visualization.

As the data sets continue to grow, even if it becomes possible to visualize all the data, the images generated may become hard to comprehend due to limitations of the human visual system [HE12, KU00]. Rather than trying to visualize all the data at once, query-based visualization techniques [SWR*12, GABJ08, SSWB05] that extract only a subset of the data relevant for the task at hand will gain importance. By selecting only a subset of the data, less bandwidth is used and computation requirements for local rendering are minimized.

Visualization of large data sets may require data management capabilities similar to those available in databases. Building visualization algorithms on top of a data management and analysis platform would spare the visualization algorithms from data management issues and challenges of the heterogeneity of the data sources. The visualization layer would focus only on mapping and rendering steps of the visualization pipeline. Data access and filtering can be handled by a separate data service layer that hides the differences of data sources and presents a uniform data access interface to the visualization layer. Scientific databases like SciDB [SBZB13] that combine data management and analysis capabilities can be exploited for remote visualization of large data sets.

RESTful services [Pau14] have become a predominant approach for implementing web services due to their simplicity and for effectively leveraging the web infrastructure for scalability and performance. The introduction of WebGL [Khr11b, Khr13] in the web browsers combined with modern web technologies introduced in HTML5 [W3C14] standard have given the browser the capability to become a preferred platform for deployment of interactive graphics applications. Given the ubiquity of browsers across different devices, visualization tools can harness all the computational resources available to support collaborative visualization for research teams that are geographically distributed. By combining mobile, web, and cloud computing technologies, complex problems can be tackled by bringing together experts across the globe to work on a problem and, thus, accelerate scientific discovery.

6.2. Challenges

The main challenge for cloud-based visualization is limited support for GPU access from the virtual machines. Although the rendering part can be shifted to client devices (the majority of which are currently fitted with GPUs), still GPU access in the cloud is important for visualization techniques that depend on technologies like CUDA and OpenCL for more demanding computations (e.g., preprocessing). As discussed in section 2.3, a preferred approach for interactive cloud-based visualization is cloud computation with client-side rendering. Interactive cloud-based visualization can benefit by leveraging GPU acceleration in the cloud the same way cloud gaming is benefiting from GPU acceleration (e.g., rendering and video encoding done on the GPU) [OTO13, SLNC13, CCT*11].

Another challenge is that existing visualization tools were not designed for virtualized and elastic computing infrastructure. For these applications to be deployed in the cloud and take advantage of its elasticity benefits (ability to scale down and up depending on current workload), redesigning them into the so called *Cloud-Native Applications* [ABLS13] would be required. Moreover, visualization tools will need to be easily integrated with other tools (e.g., simulation and analysis tools) to support seamless analysis workflows and spare researchers from manual integration of different tools. Service-oriented software techniques can also prove helpful in this regard. Different tools running on different platforms and written in different programming languages should be able to communicate and share their data based on standard communication protocols.

References

[ABLS13] ANDRIKOPOULOS V., BINZ T., LEYMANN F., STRAUCH S.: How to adapt applications for the cloud environment. *Computing 95*, 6 (2013), 493–535. doi:10.1007/s00607-012-0248-2. 17

[AFG*10] ARMBRUST M., FOX A., GRIFFITH R., JOSEPH A. D., KATZ R., KONWINSKI A., LEE G., PATTERSON D., RABKIN A., STOICA I., ZAHARIA M.: A view of cloud computing. *Commun. ACM 53*, 4 (#apr# 2010), 50–58. doi:10.1145/1721654.1721672. 5

[AIAT12] AKINCI G., IHMSEN M., AKINCI N., TESCHNER M.: Parallel surface reconstruction for particle-based fluids. *Computer Graphics Forum 31*, 6 (2012), 1797–1809. doi:10.1111/j.1467-8659.2012.02096.x. 16

[AMD15] Amd: Virtualization, 2015. [Online; accessed 15.04.2016]. URL: http://www.amd.com/en-us/solutions/professional/virtualization. 5, 6

[Apa10] APACHE: Apache CloudStack Open Source Cloud Computing. https://cloudstack.apache.org/, 2010. [Online; accessed 10.08.2015]. 5

[ASK*15] ANDRIOTI H., STAMOULIAS A., KAPETANAKIS K., PANAGIOTAKIS S., MALAMOS A. G.: Integrating webrtc and x3dom: bridging the gap between communications and graphics. In *Proceedings of the 20th International Conference on 3D Web Technology* (2015), ACM, pp. 9–15. 7

[AW14] ANDREWS K., WRIGHT B.: FluidDiagrams: Web-Based Information Visualisation using JavaScript and WebGL. In *EuroVis - Short Papers* (2014), Elmqvist N., Hlawitschka M., Kennedy J., (Eds.), The Eurographics Association. doi:10.2312/eurovisshort.20141155. 13, 15

[bab] BabylonJS - 3D engine based on WebGL/Web Audio and JavaScript. (last accessed 2015/12/15). URL: http://www.babylonjs.com/. 7

[BD07] BRUTZMAN D., DALY L.: *X3D: Extensible 3D Graphics for Web Authors*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. 8, 12

[BDG*04] BRODLIE K., DUCE D., GALLOP J., SAGAR M., WALTON J., WOOD J.: Visualization in grid computing environments. In *In Proceedings of IEEE Visualization* (2004), pp. 155–162. 4, 15

[BE14] BADAM S. K., ELMQVIST N.: Polychrome: A cross-device framework for collaborative web visualization. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces* (New York, NY, USA, 2014), ITS '14, ACM, pp. 109–118. doi:10.1145/2669485.2669518. 14

[BEJZ09] BEHR J., ESCHLER P., JUNG Y., ZÖLLNER M.: X3dom: a dom-based html5/x3d integration model. In *Proceedings of the 14th International Conference on 3D Web Technology* (2009), ACM, pp. 127–135. 6

[BJDA11] BEHR J., JUNG Y., DREVENSEK T., ADERHOLD A.: Dynamic and interactive aspects of x3dom. In *Proceedings of the 16th International Conference on 3D Web Technology* (2011), ACM, pp. 81–87. 6

[BJFS12] BEHR J., JUNG Y., FRANKE T., STURM T.: Using images and explicit binary container for efficient and incremental delivery of declarative 3d scenes on the web. In *Proceedings of the 17th International Conference on 3D Web Technology* (2012), Web3D '12, pp. 17–25. doi:10.1145/2338714.2338717. 9

[BJK*10] BEHR J., JUNG Y., KEIL J., DREVENSEK T., ZOELLNER M., ESCHLER P., FELLNER D.: A scalable architecture for the html5/x3d integration model x3dom. In *Proceedings of the 15th International Conference on Web 3D Technology* (New York, NY, USA, 2010), Web3D '10, ACM, pp. 185–194. doi:10.1145/1836049.1836077. 6, 8

[BMP*15] BEHR J., MOUTON C., PARFOURU S., CHAMPEAU J., JEULIN C., THÖNER M., STEIN C., SCHMITT M., LIMPER M., DE SOUSA M., FRANKE T. A., VOSS G.: webvis/instant3dhub: Visual computing as a service infrastructure to deliver adaptive, secure and scalable user centric data visualisation. In *Proceedings of the 20th International Conference on 3D Web Technology* (New York, NY, USA, 2015), Web3D '15, ACM, pp. 39–47. doi:10.1145/2775292.2775299. 9

[BOH11] BOSTOCK M., OGIEVETSKY V., HEER J.: D3: Data-driven documents. *IEEE Trans. Vis. Comput. Graphics* (2011). 9, 13

[BSDW12] BACHTHALER S., SADLO F., DACHSBACHER C., WEISKOPF D.: Space-time visualization of dynamics in lagrangian coherent structures of time-dependent 2d vector fields. *International Conference on Information Visualization Theory and Applications* (2012), 573–583. 14

[BTH*13] BOSCH H., THOM D., HEIMERL F., PUTTMANN E., KOCH S., KRUGER R., WORNER M., ERTL T.: ScatterBlogs2: Real-Time Monitoring of Microblog Messages through User-Guided Filtering. *IEEE Transactions on Visualization and Computer Graphics 19*, 12 (2013), 2022–2031. doi:10.1109/TVCG.2013.186. 16

[BTW*11] BOSCH H., THOM D., WORNER M., KOCH S., PUTTMANN E., JACKLE D., ERTL T.: ScatterBlogs: Geo-spatial document analysis. In *2011 IEEE Conference on Visual Analytics Science and Technology (VAST)* (2011), pp. 309–310. doi:10.1109/VAST.2011.6102488. 16

[BWF*00] BERMAN H. M., WESTBROOK J., FENG Z., GILLILAND G., BHAT T. N., WEISSIG H., SHINDYALOV I. N., BOURNE P. E.: The protein data bank. *Nucleic Acids Research 28*, 1 (2000), 235–242. URL: http://www.pdb.org, doi:10.1093/nar/28.1.235. 11

[CADB*10] CALLIERI M., ANDREI R. M., DI BENEDETTO M., ZOPPÈ M., SCOPIGNO R.: Visualization methods for molecular studies on the web platform. In *Proceedings of the 15th International Conference on Web 3D Technology* (New York, NY, USA, 2010), Web3D '10, ACM, pp. 117–126. doi:10.1145/1836049.1836067. 11, 15

[CCT*11] CHEN K.-T., CHANG Y.-C., TSENG P.-H., HUANG C.-Y., LEI C.-L.: Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM International Conference on Multimedia* (New York, NY, USA, 2011), MM '11, ACM, pp. 1269–1272. doi:10.1145/2072298.2071991. 6, 17

[CHM04] CHARTERS S. M., HOLLIMAN N. S., MUNRO M.: Visualization on the grid: A web service approach. In *Proceedings UK eScience third All-Hands Meeting* (2004), pp. 202–209. 3, 5, 15

[COJ15] CHANDLER J., OBERMAIER H., JOY K. I.: WebGL-Enabled Remote Visualization of Smoothed Particle Hydrodynamics Simulations. In *Eurographics Conference on Visualization (EuroVis) - Short Papers* (2015), Bertini E., Kennedy J., Puppo E., (Eds.), The Eurographics Association. doi:10.2312/eurovisshort.20151116. 11, 15

[Con12] CONGOTE J.: Medx3dom: Medx3d for x3dom. In *Proceedings of the 17th International Conference on 3D Web Technology* (2012), ACM, pp. 179–179. 12, 15

[CSK*11] CONGOTE J., SEGURA A., KABONGO L., MORENO A., POSADA J., RUIZ O.: Interactive visualization of volumetric data with webgl in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology* (New York, NY, USA, 2011), Web3D '11, ACM, pp. 137–146. doi:10.1145/2010425.2010449. 7, 12, 15

[CWSR12] CHOY S., WONG B., SIMON G., ROSENBERG C.: The brewing storm in cloud gaming: A measurement study on cloud to end-user latency. In *Proceedings of the 11th Annual Workshop on Network and Systems Support for Games* (Piscataway, NJ, USA, 2012), NetGames '12, IEEE Press, pp. 2:1–2:6. 6

[d'A11] D'AURIOL B. J.: Serviceable visualizations. *The Journal of Supercomputing 61*, 3 (2011), 1089–1115. doi:10.1007/s11227-011-0687-0. 3, 15

[Dan12] DANCHILLA B.: Three.js framework. In *Beginning WebGL for HTML5*. Apress, 2012, pp. 173–203. doi:10.1007/978-1-4302-3997-0_7. 4, 7, 8, 11, 12, 13

[DBPGS10] DI BENEDETTO M., PONCHIO F., GANOVELLI F., SCOPIGNO R.: Spidergl: A javascript 3d graphics library for next-generation www. In *Web3D 2010. 15th Conference on 3D Web technology* (2010). URL: http://vcg.isti.cnr.it/Publications/2010/DPGS10. 8, 11

[DJL15] DESPRAT C., JESSEL J.-P., LUGA H.: A 3d collaborative editor using webgl and webrtc. In *Proceedings of the 20th International Conference on 3D Web Technology* (2015), ACM, pp. 157–158. 7

[DP15] DWORAK D., PIETRUSZKA M.: *New Research in Multimedia and Internet Systems*. Springer International Publishing, Cham, 2015, ch. Fast Encoding of Huge 3D Data Sets in Lossless PNG Format, pp. 15–24. doi:10.1007/978-3-319-10383-9_2. 9

[DPD*15] DIEHL A., PELOROSSO L., DELRIEUX C., SAULO C., RUIZ J., GRÖLLER M. E., BRUCKNER S.: Visual analysis of spatio-temporal data: Applications in weather forecasting. *Computer Graphics Forum 34*, 3 (May 2015), 381–390. doi:10.1111/cgf.12650. 12, 15

[DS05] DUCE D. A., SAGAR M.: skML a Markup Language for Distributed Collaborative Visualization. In *EG UK Theory and Practice of Computer Graphics* (2005), Lever L. M., McDerby M., (Eds.), The Eurographics Association. doi:10.2312/LocalChapterEvents/TPCG/TPCGUK05/171-178. 4

[DS09] DOWTY M., SUGERMAN J.: Gpu virtualization on vmware's hosted i/o architecture. *SIGOPS Oper. Syst. Rev. 43*, 3 (July 2009), 73–82. doi:10.1145/1618525.1618534. 6

[DVS*12] DEBOOSERE L., VANKEIRSBILCK B., SIMOENS P., DE TURCK F., DHOEDT B., DEMEESTER P.: Cloud-based desktop services for thin clients. *Internet Computing, IEEE 16*, 6 (Nov 2012), 60–67. doi:10.1109/MIC.2011.139. 6

[EE99] ENGEL K., ERTL T.: Texture-based volume visualization for multiple users on the world wide web. In *Virtual Environments*, Gervautz M., Schmalstieg D., Hildebrand A., (Eds.), Eurographics. Springer Vienna, 1999, pp. 115–124. doi:10.1007/978-3-7091-6805-9_12. 2

[EH08] EVANGELINOS C., HILL C. N.: Cloud computing for parallel scientific hpc applications: Feasibility of running coupled atmosphere-ocean climate models on amazon's ec2. In *In The 1st Workshop on Cloud Computing and its Applications (CCA* (2008). 6

[EPM13] ERL T., PUTTINI R., MAHMOOD Z.: *Cloud Computing: Concepts, Technology & Architecture*, 1st ed. Prentice Hall Press, Upper Saddle River, NJ, USA, 2013. 5

[ERB*14] EVANS A., ROMEO M., BAHREHMAND A., AGENJO J., BLAT J.: 3d graphics on the web: A survey. *Computers & Graphics 41* (2014), 43 – 61. doi:10.1016/j.cag.2014.02.002. 7

[ESE00] ENGEL K., SOMMER O., ERTL T.: A framework for interactive hardware accelerated remote 3d-visualization. In *Data Visualization 2000*, de Leeuw W., van Liere R., (Eds.), Eurographics. Springer Vienna, 2000, pp. 167–177. doi:10.1007/978-3-7091-6783-0_17. 2

[ESEE99] ENGEL K., SOMMER O., ERNST C., ERTL T.: Remote 3d visualization using image-streaming techniques. In *In ISIMADE - 11 TH Internationl Conference on Systems Research, Informatics and Cybernetics* (1999), pp. 91–96. 2

[FAW10] FRAEDRICH R., AUER S., WESTERMANN R.: Efficient High-Quality Volume Rendering of SPH Data. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1533–1540. doi:10.1109/TVCG.2010.148. 16

[FFRR15] FELTER W., FERREIRA A., RAJAMONY R., RUBIO J.: An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on* (March 2015), pp. 171–172. doi:10.1109/ISPASS.2015.7095802. 6

[Fie00] FIELDING R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000. AAI9980887. 3

[FK99] FOSTER I., KESSELMAN C. (Eds.): *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. 4

[FKNT02] FOSTER I., KESSELMAN C., NICK J. M., TUECKE S.: The physiology of the grid: An open grid services architecture for distributed systems integration. URL: http://toolkit.globus.org/alliance/publications/papers/ogsa.pdf. 4

[FKT01] FOSTER I., KESSELMAN C., TUECKE S.: The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl. 15*, 3 (Aug. 2001), 200–222. doi:10.1177/109434200101500302. 4

[FM11] FETTE I., MELNIKOV A.: The WebSocket Protocol. http://www.rfc-editor.org/info/rfc6455, 2011. Online; accessed 2015.03.09. 7

[Fos05] FOSTER I.: Globus toolkit version 4: Software for service-oriented systems. In *Proceedings of the 2005 IFIP International Conference on Network and Parallel Computing* (Berlin, Heidelberg, 2005), NPC'05, Springer-Verlag, pp. 2–13. doi:10.1007/11577188_2. 2, 4, 5

[FRSVP14] FIGUEIREDO M., RODRIGUES J., SILVESTRE I., VEIGA-PIRES C.: Web3d visualization of high detail and complex 3d-mesh caves models. In *Information Visualisation (IV), 2014 18th International Conference on* (July 2014), pp. 275–280. doi:10.1109/IV.2014.15. 8, 12, 15

[FSW09] FRAEDRICH R., SCHNEIDER J., WESTERMANN R.: Exploring the millennium run - scalable rendering of large-scale cosmological datasets. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1251–1258. 16

[FZRL08] FOSTER I., ZHAO Y., RAICU I., LU S.: Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08* (Nov 2008), pp. 1–10. doi:10.1109/GCE.2008.4738445. 5

[GABJ08] GOSINK L., ANDERSON J., BETHEL E., JOY K.: Query-driven visualization of time-varying adaptive mesh refinement data. *Visualization and Computer Graphics, IEEE Transactions on 14*, 6 (Nov 2008), 1715–1722. doi:10.1109/TVCG.2008.157. 17

[GBM*12] GROTTEL S., BECK P., MÜLLER C., REINA G., ROTH J., TREBIN H.-R., ERTL T.: Visualization of electrostatic dipoles in molecular dynamics of metal oxides. *Visualization and Computer Graphics, IEEE Transactions on 18*, 12 (2012), 2061–2068. doi:10.1109/TVCG.2012.282. 16

[GET*13] GOECKS J., EBERHARD C., TOO T., NEKRUTENKO A., TAYLOR J.: Web-based visual analysis for high-throughput genomics. *BMC Genomics 14*, 1 (2013), 1–11. doi:10.1186/1471-2164-14-397. 13, 15

[GGS*09] GUPTA V., GAVRILOVSKA A., SCHWAN K., KHARCHE H., TOLIA N., TALWAR V., RANGANATHAN P.: Gvim: Gpu-accelerated virtual machines. In *Proceedings of the 3rd ACM Workshop on System-level Virtualization for High Performance Computing* (New York, NY, USA, 2009), HPCVirt '09, ACM, pp. 17–24. doi:10.1145/1519138.1519141. 6

[GLNS*05] GRAY J., LIU D. T., NIETO-SANTISTEBAN M., SZALAY A., DEWITT D. J., HEBER G.: Scientific data management in the coming decade. *SIGMOD Rec. 34*, 4 (Dec. 2005), 34–41. doi:10.1145/1107499.1107503. 1

[GMAC10] GIUNTA G., MONTELLA R., AGRILLO G., COVIELLO G.: A gpgpu transparent virtualization component for high performance computing clouds. In *Euro-Par 2010 - Parallel Processing*, D'Ambra P., Guarracino M., Talia D., (Eds.), vol. 6271 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2010, pp. 379–391. doi:10.1007/978-3-642-15277-1_37. 6

[GNTT10] GOECKS J., NEKRUTENKO A., TAYLOR J., TEAM T. G.: Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology 11*, 8 (2010), R86. doi:10.1186/gb-2010-11-8-r86. 13

[Gra12] GRAPHICS A.: Cesium - WebGL Virtual Globe and Map Engine, 2012. (last accessed 2016/04/15). URL: https://cesiumjs.org/. 14

[GRE09] GROTTEL S., REINA G., ERTL T.: Optimized Data Transfer for Time-dependent, GPU-based Glyphs. In *IEEE Pacific Visualization Symposium* (2009), pp. 65–72. 16

[GVB*15] GAILLARD J., VIENNE A., BAUME R., PEDRINIS F., PEYTAVIE A., GESQUIÈRE G.: Urban data visualisation in a web browser. In *Proceedings of the 20th International Conference on 3D Web Technology* (New York, NY, USA, 2015), Web3D '15, ACM, pp. 81–88. doi:10.1145/2775292.2775302. 12, 13, 15

[HE12] HEALEY C. G., ENNS J.: Attention and visual memory in visualization and computer graphics. *Visualization and Computer Graphics, IEEE Transactions on 18*, 7 (July 2012), 1170–1188. doi:10.1109/TVCG.2011.127. 17

[Her13] HERRERA A.: NVIDIA GRID: graphics accelerated VDI with the visual performance of a workstation, 2013. [Online; accessed 10.07.2015]. URL: http://www.nvidia.com/content/grid/vdi-whitepaper.pdf. 5

[HHCC13] HUANG C.-Y., HSU C.-H., CHANG Y.-C., CHEN K.-T.: Gaminganywhere: An open cloud gaming system. In *Proceedings of the 4th ACM Multimedia Systems Conference* (New York, NY, USA, 2013), MMSys '13, ACM, pp. 36–47. doi:10.1145/2483977.2483981. 6

[HJS98] HENDIN O., JOHN N. W., SHOCET O.: Medical Volume Rendering Over the WWW Using VRML and Java. In *Medicine Meets Virtual Reality* (Amsterdam, 1998), Westwood, (Ed.), IOS Press and Ohmsha, pp. 34–40. 2

[HKBR*14] HAEHN D., KNOWLES-BARLEY S., ROBERTS M., BEYER J., KASTHURI N., LICHTMAN J. W., PFISTER H.: Design and evaluation of interactive proofreading tools for connectomics. *Visualization and Computer Graphics, IEEE Transactions on 20*, 12 (2014), 2466–2475. 14, 15

[HKZ*11] HINDMAN B., KONWINSKI A., ZAHARIA M., GHODSI A., JOSEPH A. D., KATZ R., SHENKER S., STOICA I.: Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2011), NSDI'11, USENIX Association, pp. 295–308. 6

[HLW14] HONGJIAN LI KWONG-SAK LEUNG T. N., WONG M.-H.: iview: an interactive webgl visualizer for protein-ligand complex. *BMC Bioinformatics 15* (2014), 56. URL: http://istar.cse.cuhk.edu.hk/iview/, doi:10.1186/1471-2105-15-56. 11, 15

[HM90] HABER R. B., MCNABB D. A.: Visualization idioms: A conceptual model for scientific visualization systems. In *Visualization in Scientific Computing*. IEEE Computer Society Press, 1990, pp. 74–93. 3, 5

[Hop96] HOPPE H.: Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 99–108. doi:10.1145/237170.237216. 8

[Hop98] HOPPE H.: Efficient implementation of progressive meshes. *Computers & Graphics 22*, 1 (1998), 27 – 36. doi:http://dx.doi.org/10.1016/S0097-8493(97)00081-2. 8

[HRA*] HAEHN D., RANNOU N., AHTAM B., GRANT E., PIENAAR R.: Neuroimaging in the browser using the x toolkit. *Frontiers in Neuroinformatics*, 101. doi:10.3389/conf.fninf.2014.08.00101. 13, 15

[HSZ15] HOU X., SUN J., ZHANG J.: A web-based solution for 3d medical image visualization. vol. 9418, pp. 941810–941810–8. doi:10.1117/12.2075982. 12, 15

[IES*11] ISENBERG P., ELMQVIST N., SCHOLTZ J., CERNEA D., MA K.-L., HAGEN H.: Collaborative visualization: Definition, challenges, and research agenda. *Information Visualization 10*, 4 (Oct. 2011), 310–326. doi:10.1177/1473871611412817. 1

[JAC*07] JOHN N., ARATOW M., COUCH J., EVESTEDT D., HUDSON A., POLYS N., PUK R., RAY A., VICTOR K., WANG Q.: Medx3d: standards enabled desktop medical 3d. *Studies in health technology and informatics 132* (2007), 189–194. 12

[JAG10] JOURDAIN S., AYACHIT U., GEVECI B.: Paraviewweb, a web framework for 3d visualization and data processing. *IADIS International Conference on Web Virtual Reality and Three-Dimensional Worlds* (07 2010). 14

[JaL16] JENNY B., ŠAVRIČ B., LIEM J.: Real-time raster projection for web maps. *International Journal of Digital Earth 9*, 3 (2016), 1–15. doi:10.1080/17538947.2014.1002867. 12, 13, 15

[JAM*09] JOMIER J., AYLWARD S. R., MARION C., LEE J., STYNER M.: A digital archiving system and distributed server-side processing of large datasets. 14

[JJAM11] JOMIER J., JOURDAIN S., AYACHIT U., MARION C.: Remote visualization of large datasets with midas and paraviewweb. In *Proceedings of the 16th International Conference on 3D Web Technology* (New York, NY, USA, 2011), Web3D '11, ACM, pp. 147–150. doi:10.1145/2010425.2010450. 2, 14, 15

[JKD*12] JACINTO H., KÉCHICHIAN R., DESVIGNES M., PROST R., VALETTE S.: A web interface for 3d visualization and interactive segmentation of medical images. In *Proceedings of the 17th International Conference on 3D Web Technology* (New York, NY, USA, 2012), Web3D '12, ACM, pp. 51–58. doi:10.1145/2338714.2338722. 8, 12, 15

[JLC*14] JIMÉNEZ J., LÓPEZ A., CRUZ J., ESTEBAN F., NAVAS J., VILLOSLADA P., DE MIRAS J. R.: A web platform for the interactive visualization and analysis of the 3d fractal dimension of {MRI} data. *Journal of Biomedical Informatics 51* (2014), 176 – 190. doi:http://dx.doi.org/10.1016/j.jbi.2014.05.011. 7, 12, 15, 16

[Jmo09] JMOL: Jmol: an open-source Java viewer for chemical structures in 3D. http://www.jmol.org/, 2009. [Online; accessed 01.04.2014]. 7, 11

[JSm13] JSMOL: JSmol: JavaScript-Based Molecular Viewer From Jmol . http://sourceforge.net/projects/jsmol/, 2013. [Online; accessed 02.04.2014]. 11

[Khr11a] KHRONOS: Typed Array Specification. http://www.khronos.org/registry/typedarray/specs/latest/, 2011. Online; accessed 2016.04.13. 7

[Khr11b] KHRONOS: WebGL 1.0 specification. http://www.khronos.org/registry/webgl/specs/latest/1.0/, 2011. [Online; accessed 21.02.2014]. 2, 17

[Khr13] KHRONOS: WebGL 2.0 specification. http://www.khronos.org/registry/webgl/specs/latest/2.0/, 2013. [Online; accessed 21.02.2014]. 2, 17

[Khr15] KHRONOS: glTF 1.0 Specification. https://github.com/KhronosGroup/glTF/tree/master/specification, 2015. [Online; accessed 28.01.2016]. 14

[KHRV04] KRANZLMÜLLER D., HEINZLREITER P., ROSMANITH H., VOLKERT J.: Grid-enabled visualization with gvk. In *Grid Computing*, Fernandez Rivera F., Bubak M., Gomez Tato A., Doallo R., (Eds.), vol. 2970 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2004, pp. 139–146. doi:10.1007/978-3-540-24689-3_18. 4, 5, 15

[KMS*15] KOVAL Y., MENDRUL H., SALNIKOV A., SLIUSAR I., SUDAKOV O.: Interactive dynamical visualization of big data arrays in grid. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference on* (Sept 2015), vol. 1, pp. 153–156. doi:10.1109/IDAACS.2015.7340718. 4, 15

[KPS15] KIM J.-S., POLYS N., SFORZA P.: Preparing and evaluating geospatial data models using x3d encodings for web 3d geovisualization services. In *Proceedings of the 20th International Conference on 3D Web Technology* (2015), ACM, pp. 55–63. 12, 15

[KSL*13] KOHLHOFF K. J., SHUKLA D., LAWRENZ M., BOWMAN G. R., KONERDING D. E., BELOV D., ALTMAN R. B., PANDE V. S.: Cloud-based simulations on Google Exacycle reveal ligand modulation of GPCR activation pathways. *Nature Chemistry 6*, 1 (Dec. 2013), 15–21. doi:10.1038/nchem.1821. 6

[KSWE16] KARCH G. K., SADLO F., WEISKOPF D., ERTL T.: Visualization of 2D unsteady flow using streamline-based concepts in space-time. *Journal of Visualization 19*, 1 (2016), 115–128. doi:10.1007/s12650-015-0284-z. 14

[KU00] KASTNER S., UNGERLEIDER L. G.: Mechanisms of visual attention in the human cortex. *Annual Review of Neuroscience 23* (2000), 315–341. 17

[KWN*14] KNOLL A., WALD I., NAVRATIL P., BOWEN A., RENDA K., PAPKA M. E., GAITHER K.: Rbf volume ray casting on multicore and manycore cpus. *Computer Graphics Forum 33*, 3 (2014), 71–80. doi:10.1111/cgf.12363. 16

[KZSB10] KOULOUZIS S., ZUDILOVA-SEINSTRA E., BELLOUM A.: Data transport between visualization web services for medical image analysis. *Procedia Computer Science 1*, 1 (2010), 1727 – 1736. {ICCS} 2010. doi:http://dx.doi.org/10.1016/j.procs.2010.04.194. 3, 4, 15

[LBPH10] LINDOW N., BAUM D., PROHASKA S., HEGE H.-C.: Accelerated visualization of dynamic molecular surfaces. *Computer Graphics Forum 29*, 3 (2010), 943–952. doi:10.1111/j.1467-8659.2009.01693.x. 16

[LCD13] LAVOUÉ G., CHEVALIER L., DUPONT F.: Streaming compressed 3d data on the web using javascript and webgl. In *Proceedings of the 18th International Conference on 3D Web Technology* (New York, NY, USA, 2013), Web3D '13, ACM, pp. 19–27. doi:10.1145/2466533.2466539. 9

[LCD14] LAVOUÉ G., CHEVALIER L., DUPONT F.: Progressive streaming of compressed 3d graphics in a web browser, 2014. (last accessed 2016/02/02). URL: http://liris.cnrs.fr/~glavoue/travaux/conference/SigTalk2014.pdf. 8, 9, 15

[Lin14] LINDSTROM P.: Fixed-rate compressed floating-point arrays. *IEEE Transactions on Visualization and Computer Graphics 20*, 12 (2014), 2674–2683. doi:10.1109/TVCG.2014.2346458. 17

[LJB*13] LIMPER M., JUNG Y., BEHR J., STURM T., FRANKE T., SCHWENK K., KUIJPER A.: Fast, progressive loading of binary-encoded declarative-3d web content. *Computer Graphics and Applications, IEEE 33*, 5 (Sept 2013), 26–36. doi:10.1109/MCG.2013.52. 9, 15

[LJBA13] LIMPER M., JUNG Y., BEHR J., ALEXA M.: The pop buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum 32*, 7 (2013), 197–206. doi:10.1111/cgf.12227. 9

[LJH13] LIU Z., JIANG B., HEER J.: immens: Real-time visual querying of big data. *Computer Graphics Forum 32* (2013). 13, 15

[Lor04] LORENSEN B.: On the Death of Visualization. In *Position Papers NIH/NSF Proc. Fall 2004 Workshop Visualization Research Challenges* (2004). 1

[LTBF14] LIMPER M., THÖNER M., BEHR J., FELLNER D. W.: Src - a streamable format for generalized web-based 3d data transmission. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies* (New York, NY, USA, 2014), Web3D '14, ACM, pp. 35–43. doi:10.1145/2628588.2628589. 9, 15

[LWS*13] LIMPER M., WAGNER S., STEIN C., JUNG Y., STORK A.: Fast delivery of 3d web content: A case study. In *Proceedings of the 18th International Conference on 3D Web Technology* (New York, NY, USA, 2013), Web3D '13, ACM, pp. 11–17. doi:10.1145/2466533.2466536. 9

[lz4] LZ4 - Extremely Fast Compression. (last accessed 2016/04/15). URL: http://cyan4973.github.io/lz4/. 17

[MAPV15] MUZIC M. L., AUTIN L., PARULEK J., VIOLA I.: cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. In *EG Workshop on Visual Computing for Biology and Medicine* (2015). doi:10.2312/vcbm.20151209. 16

[Mer14] MERKEL D.: Docker: Lightweight linux containers for consistent development and deployment. *Linux J. 2014*, 239 (Mar. 2014). 6

[MF12] MOVANIA M., FENG L.: High-performance volume rendering on the ubiquitous webgl platform. In *High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on* (June 2012), pp. 381–388. doi:10.1109/HPCC.2012.58. 7, 12, 15

[MG11] MELL P. M., GRANCE T.: *SP 800-145. The NIST Definition of Cloud Computing*. Tech. rep., Gaithersburg, MD, United States, 2011. 5

[MG13] MAYORGA A., GLEICHER M.: Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics 19*, 9 (Sept. 2013), 1526–1538. doi:10.1109/TVCG.2013.65. 13

[MKB*15] MWALONGO F., KRONE M., BECHER M., REINA G., ERTL T.: Remote visualization of dynamic molecular data using webgl. In *Proceedings of the 20th International Conference on 3D Web Technology* (New York, NY, USA, 2015), Web3D '15, ACM, pp. 115–122. doi:10.1145/2775292.2775307. 7, 8, 11, 15

[MKK*14] MWALONGO F., KRONE M., KARCH G., BECHER M., REINA G., ERTL T.: Visualization of molecular structures using state-of-the-art techniquesin webgl. In *Proceedings of the Nineteenth International ACM Conference on 3DWeb Technologies* (New York, NY, USA, 2014), Web3D '14, ACM, pp. 133–141. doi:10.1145/2628588.2628597. 7, 8, 11, 15

[MPJ*14] MOUTON C., PARFOURU S., JEULIN C., DUTERTRE C., GOBLET J.-L., PAVIOT T., LAMOURI S., LIMPER M., STEIN C., BEHR J., JUNG Y.: Enhancing the plant layout design process using x3dom and a scalable web3d service architecture. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies* (2014), Web3D '14, pp. 125–132. doi:10.1145/2628588.2628592. 14

[MVML12] MORENO-VOZMEDIANO R., MONTERO R., LLORENTE I.: Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer 45*, 12 (Dec 2012), 65–72. doi:10.1109/MC.2012.76. 5

[MYB14] MCCANN M., YOO B., BRUTZMAN D.: Integration of x3d geospatial in a data driven web application. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies* (2014), ACM, pp. 145–145. 12, 15

[NJ12] NOGUERA J. M., JIMÉNEZ J.-R.: Visualization of very large 3d volumes on mobile devices and webgl. In *20th WSCG International Conference on Computer Graphics, Visualization and Computer Vision 2012* (2012). 7, 12, 15

[NVI15a] NVIDIA: NVIDIA GRID PC Streaming Service, 2015. [Online; accessed 09.07.2015]. URL: http://shield.nvidia.com/grid-game-streaming. 6

[NVI15b] NVIDIA: Shared Virtual GPU (vGPU) Technology, 2015. [Online; accessed 09.07.2015]. URL: http://www.nvidia.com/object/virtual-gpus.html. 6

[NWG*09] NURMI D., WOLSKI R., GRZEGORCZYK C., OBERTELLI G., SOMAN S., YOUSEFF L., ZAGORODNOV D.: The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on* (May 2009), pp. 124–131. doi:10.1109/CCGRID.2009.93. 5

[Ope10] OPENSTACK: OpenStack Open Source Cloud Computing Software. https://www.openstack.org/, 2010. [Online; accessed 10.08.2015]. 5

[OTO13] OTOY: The Future of Cloud GAMING, 2013. [Online; accessed 09.07.2015]. URL: http://www.otoy.com/cgc/cloudgaming_2013.pdf. 6, 17

[Pau14] PAUTASSO C.: Restful web services: Principles, patterns, emerging technologies. In *Web Services Foundations*, Bouguettaya A., Sheng Q. Z., Daniel F., (Eds.). Springer New York, 2014, pp. 31–51. doi:10.1007/978-1-4614-7518-7_2. 3, 17

[PD15] PONCHIO F., DELLEPIANE M.: Fast decompression for web-based view-dependent 3d rendering. In *Proceedings of the 20th International Conference on 3D Web Technology* (New York, NY, USA, 2015), Web3D '15, ACM, pp. 199–207. doi:10.1145/2775292.2775308. 8, 9, 10, 14, 15

[PM15] PLESCH A., MCCANN M.: The x3d geospatial component: X3dom implementation of geoorigin, geolocation, geoviewpoint, and geopositioninterpolator nodes. In *Proceedings of the 20th International Conference on 3D Web Technology* (2015), ACM, pp. 31–37. 12, 15

[pvp] pv - WebGL protein viewer. (last accessed 2016/01/27). URL: https://github.com/biasmv/pv, doi:http://dx.doi.org/10.5281/zenodo.20980. 11, 15

[PZL08] PAUTASSO C., ZIMMERMANN O., LEYMANN F.: Restful web services vs. "big"' web services: Making the right architectural decision. In *Proceedings of the 17th International Conference on World Wide Web* (New York, NY, USA, 2008), WWW '08, ACM, pp. 805–814. doi:10.1145/1367497.1367606. 3

[RAR13] RICHARDSON L., AMUNDSEN M., RUBY S.: *RESTful Web APIs*. O'Reilly Media, Inc., 2013. 3

[RH15] ROSE A. S., HILDEBRAND P. W.: Ngl viewer: a web application for molecular visualization. *Nucleic Acids Research 43*, W1 (2015), W576–W579. arXiv:http://nar.oxfordjournals.org/content/43/W1/W576.full.pdf+html, doi:10.1093/nar/gkv402. 11, 15

[RK15] REGO N., KOES D.: 3dmol.js: molecular visualization with webgl. *Bioinformatics 31*, 8 (2015), 1322–1324. doi:10.1093/bioinformatics/btu829. 11, 15

[RWW14] RESCH B., WOHLFAHRT R., WOSNIOK C.: Web-based 4d visualization of marine geo-data using webgl. *Cartography and Geographic Information Science 41*, 3 (2014), 235–247. arXiv:http://dx.doi.org/10.1080/15230406.2014.901901, doi:10.1080/15230406.2014.901901. 12, 13, 15

[SBZB13] STONEBRAKER M., BROWN P., ZHANG D., BECLA J.: Scidb: A database management system for applications with complex analytics. *Computing in Science & Engineering 15*, 3 (2013), 54–62. doi:http://dx.doi.org/10.1109/MCSE.2013.19. 17

[SC12] SAWICKI B., CHABER B.: 3d mesh viewer using html5 technology. *Przeglad Elektrotechniczny (Electrical Review), ISSN* (2012), 0033–2097. 8

[SC13] SAWICKI B., CHABER B.: Efficient visualization of 3d models by web browser. *Computing 95*, 1 (2013), 661–673. doi:10.1007/s00607-012-0275-z. 15

[SD11] SUTHON P., DALTON C.: Streakline visualization of the structures in the near wake of a circular cylinder in sinusoidally oscillating flow. *Journal of Fluids and Structures 27*, 7 (2011), 885 – 902. doi:10.1016/j.jfluidstructs.2011.03.003. 14

[See01] SEELY S.: *SOAP: Cross Platform Web Service Development Using XML*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001. 3, 4

[SG15] SARIKAYA A., GLEICHER M.: Using webgl as an interactive visualization medium: Our experience developing splatterjs. In *Proceedings of the Data Systems for Interactive Analysis Workshop* (Oct 2015), Chang R., Scheidegger C., Fisher D., Heer J., (Eds.), IEEE. DSIA '15. URL: http://graphics.cs.wisc.edu/Papers/2015/SG15. 13, 15

[SH15] SHI S., HSU C.-H.: A survey of interactive remote rendering systems. *ACM Comput. Surv. 47*, 4 (May 2015), 57:1–57:29. doi:10.1145/2719921. 2, 3

[SKR*10] SONS K., KLEIN F., RUBINSTEIN D., BYELOZYOROV S., SLUSALLEK P.: Xml3d: Interactive 3d graphics for the web. In *Proceedings of the 15th International Conference on Web 3D Technology* (New York, NY, USA, 2010), Web3D '10, ACM, pp. 175–184. doi:10.1145/1836049.1836076. 6

[SLNC13] SHEA R., LIU J., NGAI E.-H., CUI Y.: Cloud gaming: architecture and performance. *Network, IEEE 27*, 4 (July 2013), 16–21. doi:10.1109/MNET.2013.6574660. 6, 17

[SML06] SCHROEDER W., MARTIN K., LORENSEN B.: *The Visualization Toolkit (4th ed.): An Object-oriented Approach to 3D Graphics*. Kitware, 2006. 5

[spo] TIBCO Spotfire Cloud. (last accessed 2015/12/15). URL: http://spotfire.tibco.com/products/spotfire-cloud. 3, 13, 15

[SSB09] SUSELBECK R., SCHIELE G., BECKER C.: Peer-to-peer support for low-latency massively multiplayer online games in the cloud. In *Network and Systems Support for Games (NetGames), 2009 8th Annual Workshop on* (Nov 2009), pp. 1–2. doi:10.1109/NETGAMES.2009.5446229. 6

[SSS14] SUTTER J., SONS K., SLUSALLEK P.: Blast: A binary large structured transmission format for the web. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies* (New York, NY, USA, 2014), Web3D '14, ACM, pp. 45–52. doi:10.1145/2628588.2628599. 9

[SSWB05] STOCKINGER K., SHALF J., WU K., BETHEL E.: Query-driven visualization of large data sets. In *Visualization, 2005. VIS 05. IEEE* (Oct 2005), pp. 167–174. doi:10.1109/VISUAL.2005.1532792. 17

[STK02] SNELL J., TIDWELL D., KULCHENKO P.: *Programming Web Services with SOAP*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002. 5

[SUEW12] SADLO F., ÜFFINGER M., ERTL T., WEISKOPF D.: On the finite-time scope for computing Lagrangian coherent structures from Lyapunov exponents. In *Topological Methods in Data Analysis and Visualization II*. Springer Berlin Heidelberg, 2012, pp. 269–281. doi:10.1007/978-3-642-23175-9_18. 14

[SWR*12] SANDERSON A. R., WHITLOCK B., RÜBEL O., CHILDS H., WEBER G., PRABHAT M., WU K.: A System for Query Based Analysis and Visualization. In *EuroVA 2012: International Workshop on Visual Analytics* (2012), Matkovic K., Santucci G., (Eds.), The Eurographics Association. doi:10.2312/PE/EuroVAST/EuroVA12/025-029. 17

[tab] Tableau Online. (last accessed 2015/12/15). URL: https://www.tableau.com/products/cloud-bi. 3, 13, 15

[TDC14] TIAN K., DONG Y., COWPERTHWAITE D.: A full gpu virtualization solution with mediated pass-through. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2014), USENIX ATC'14, USENIX Association, pp. 121–132. 6

[TP97] TRAPP J., PAGENDARM H.-G.: A Prototype for a WWW-based Visualization Service. In *Visualization in Scientific Computing*, Lefer W., Grave M., (Eds.), Eurographics. Springer Vienna, 1997, pp. 21–30. doi:10.1007/978-3-7091-6876-9_3. 2

[TSA*10] THUSOO A., SHAO Z., ANTHONY S., BORTHAKUR D., JAIN N., SEN SARMA J., MURTHY R., LIU H.: Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (2010), SIGMOD '10, pp. 1013–1020. doi:10.1145/1807167.1807278. 16

[VKB*15] VEHLOW C., KAO D. P., BRISTOW M. R., HUNTER L. E., WEISKOPF D., GÖRG C.: Visual analysis of biological data-knowledge networks. *BMC bioinformatics 16*, 1 (2015), 1. 16

[VPB09] VECCHIOLA C., PANDEY S., BUYYA R.: High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on* (Dec 2009), pp. 4–16. doi:10.1109/I-SPAN.2009.150. 6

[VSB14] VU L., SIVARAMAN H., BIDARKAR R.: Gpu virtualization for high performance general purpose computing on the esx hypervisor. In *Proceedings of the High Performance Computing Symposium* (San Diego, CA, USA, 2014), HPC '14, Society for Computer Simulation International, pp. 2:1–2:8. 6

[VSTA14] VIRAG I., STOICU-TIVADAR L., AMĂRICĂI E.: Browser-based medical visualization system. In *Applied Computational Intelligence and Informatics (SACI), 2014 IEEE 9th International Symposium on* (May 2014), pp. 355–359. doi:10.1109/SACI.2014.6840092. 12

[VWvH*07] VIEGAS F., WATTENBERG M., VAN HAM F., KRISS J., MCKEON M.: Manyeyes: a site for visualization at internet scale. *Visualization and Computer Graphics, IEEE Transactions on 13*, 6 (Nov 2007), 1121–1128. doi:10.1109/TVCG.2007.70577. 13

[W3C04] W3C: Web Services Architecture, 2004. [Online; accessed 09.07.2015]. URL: http://www.w3.org/TR/ws-arch/. 3

[W3C11] W3C: WebRTC 1.0: Real-time Communication Between Browsers. https://www.w3.org/TR/webrtc/, 2011. Online; accessed 2016.04.13. 7

[W3C14] W3C: HTML5: A vocabulary and associated APIs for HTML and XHTML. http://www.w3.org/TR/html5/, 2014. [Online; accessed 01.04.2014]. 2, 17

[W3C15] W3C: HTML Canvas 2D Context. https://www.w3.org/TR/2dcontext/, 2015. [Online; accessed 20.01.2016]. 13

[WBHW08] WANG H., BRODLIE K. W., HANDLEY J. W., WOOD J. D.: Service-oriented approach to collaborative visualization. *Concurrency and Computation: Practice and Experience 20*, 11 (2008), 1289–1301. doi:10.1002/cpe.1295. 3

[WBS*08] WOOD J., BRODLIE K., SEO J., DUKE D., WALTON J.: A web services architecture for visualization. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on* (Dec 2008), pp. 1–7. doi:10.1109/eScience.2008.51. 3, 4

[WBW96] WOOD J., BRODLIE K., WRIGHT H.: Visualization over the world wide web and its application to environmental data. In *Proceedings of the 7th Conference on Visualization '96* (Los Alamitos, CA, USA, 1996), VIS '96, IEEE Computer Society Press, pp. 81–ff. 2

[WJL14] WEN L., JIA J., LIANG S.: Lpm: Lightweight progressive meshes towards smooth transmission of web3d media over internet. In *Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry* (New York, NY, USA, 2014), VRCAI '14, ACM, pp. 95–103. doi:10.1145/2670473.2670475. 8, 9, 15

[WP11] WILDE E., PAUTASSO C.: *REST: From Research to Practice*, 1st ed. Springer Publishing Company, Incorporated, 2011. 3

[WPR10] WEBBER J., PARASTATIDIS S., ROBINSON I.: *REST in Practice: Hypermedia and Systems Architecture*, 1st ed. O'Reilly Media, Inc., 2010. 3

[WSD*10] WOOD J., SEO J., DUKE D., WALTON J., BRODLIE K.: Flexible delivery of visualization software and services. *Procedia Computer Science 1*, 1 (2010), 1719 – 1726. {ICCS} 2010. doi:http://dx.doi.org/10.1016/j.procs.2010.04.193. 4

[Xen14] XENSERVER: XenServer Open Source Virtualization Platform. http://xenserver.org/, 2014. [Online; accessed 10.08.2015]. 6

[YSG15] YANG Y., SHARMA A., GIRIER A.: Volumetric texture data compression scheme for transmission. In *Proceedings of the 20th International Conference on 3D Web Technology* (New York, NY, USA, 2015), Web3D '15, ACM, pp. 65–68. doi:10.1145/2775292.2775323. 10, 15

[YWO*12] YANG C.-T., WANG H.-Y., OU W.-S., LIU Y.-T., HSU C.-H.: On implementation of gpu virtualization using pci pass-through. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on* (Dec 2012), pp. 711–716. doi:10.1109/CloudCom.2012.6427531. 6

[ZSYA*08] ZUDILOVA-SEINSTRA E., YANG N., AXNER L., WIBISONO A., VASUNIN D.: Service-oriented visualization applied to medical data analysis. *Service Oriented Computing and Applications 2*, 4 (2008), 187–201. doi:10.1007/s11761-008-0031-6. 3, 5