Open Visualization Environment (OVE): A web framework for scalable rendering of data visualizations

Senaka Fernando[a,*], James Scott-Brown[a], Ovidiu Şerban[a], David Birch[a], David Akroyd[a], Miguel Molina-Solana[a,b], Thomas Heinis[a], Yike Guo[a,*]

[a]*Data Science Institute, Imperial College London, London, United Kingdom*
[b]*Department of Computer Science and AI, Universidad de Granada, Granada, Spain*

Abstract

Scalable resolution display environments, including immersive data observatories, are emerging as equitable and socially engaging platforms for collaborative data exploration and decision making. These environments require specialized middleware to drive them, but, due to various limitations, there is still a gap in frameworks capable of scalable rendering of data visualizations. To overcome these limitations, we introduce a new modular open-source middleware, the Open Visualization Environment (OVE). This framework uses web technologies to provide an ecosystem for visualizing data using web browsers that span hundreds of displays. In this paper, we discuss the key design features and architecture of our framework as well as its limitations. This is followed by an extensive study on performance and scalability, which validates its design and compares it to the popular SAGE2 middleware. We show how our framework solves three key limitations in SAGE2. Thereafter, we present two of our projects that used OVE and show how it can extend SAGE2 to overcome limitations and simplify the user experience for common data visualization use-cases.

Keywords: large-scale visualization, data visualization, scalable resolution display environments, SAGE2

1. Introduction

There is an ever-increasing demand to produce meaningful insight from data in very short timescales, pushing data analytics pipelines to their limits. Large multi-disciplinary projects produce and consume vast amounts of data, making it very hard to make sense of it in short periods of time. This is where collaborative group work becomes meaningful, as the ability to visualize data and incorporate collective thinking of human experts from various domains still dominates most decision-making processes [1]. Improving the efficiency of such collaborative data exploration activities is therefore a key priority for modern data science research.

Immersive analytics has the potential to support deeper, more equitable and socially engaging collaboration [2]. Academia as well as industry are building immersive data observatories to assist with the challenging task of collaborative data exploration. These environments are capable of visualizing data at resolutions that are sufficiently large to accommodate a group of up to 20 people working together. Scalable Resolution Display Environments (SRDEs) are becoming increasingly common within immersive data observatories and easier to build as displays decrease in cost. These environments support collective decision making processes through their ability to recreate scenarios, retrace approaches, and reenact situations, which are useful in emerging domains such as explainable AI [3, 4].

The Data Science Institute at Imperial College London has several SRDEs purpose-built for data visualization [5]. These allow multidisciplinary teams to explore big data collaboratively [6], in a mixture of immersive and non-immersive environments. One key challenge of operating such SRDEs is that they require specialized middleware to drive them. Whilst several such software systems have been built, most are based on OpenGL and intended to run non-web-based applications. Meanwhile, web browsers are becoming an increasingly important platform for data visualization. A few systems, such as SAGE2 [7], are based on web technologies but focus on different purposes. SAGE2 is primarily intended as a dynamic windowing environment for co-located and remote collaboration [8, 9]; this makes it unsuitable for our use-cases. We discuss the limitations of SAGE2 and other comparable frameworks further in Section 2.

To overcome these limitations, we developed Open Visualization Environment (OVE), an open-source web framework for scalable rendering of data visualizations. OVE supports many web-friendly standards and content formats, and as a result, it is capable of displaying content developed using popular visualization libraries such as D3.js [10] and Three.js [11] with few or no modifications. The compositing technique used by OVE is

*Corresponding author
Email addresses: senaka.fernando15@imperial.ac.uk (Senaka Fernando), james@jamesscottbrown.com (James Scott-Brown), o.serban@imperial.ac.uk (Ovidiu Şerban), david.birch@imperial.ac.uk (David Birch), david@davidakroyd.co.uk (
David Akroyd), mmolinas@ic.ac.uk (Miguel Molina-Solana), t.heinis@imperial.ac.uk (Thomas Heinis), y.guo@imperial.ac.uk (Yike Guo)

capable of combining multiple content of various image, video, audio, graph, geospatial, HTML and PDF formats by resizing, rearranging, joining and overlapping them into a single contiguous interactive visualization that may in many cases span multiple displays. In terms of key features, this framework is:

- Capable of displaying high-resolution content on display canvases spanning billions of pixels across multiple display devices.
- Implemented using open web technologies with out-of-the-box support for a large number of web-friendly data formats. These files can be web-hosted or stored using the *Asset Manager*.
- Designed to be seamlessly integrated with existing data analytics workflows. It provides a REST API and Python SDK, enabling bulk operations and scripted deployments.
- Designed to be hardware, platform, and web browser independent.

With OVE, we make two key contributions:

1. A highly performing and scalable middleware framework for data visualization in SRDEs using web technologies.
2. A plug-and-play supplement to the popular SAGE2 middleware to simplify the user experience for common data visualization use-cases.

This paper is organized as follows: Section 2 introduces related work, followed by a discussion of the key design features and limitations of OVE (Section 3) and its architecture (Section 4). We then compare the performance, scalability and usability of OVE and SAGE2 in Section 5, highlighting the advantages of using OVE. And thereafter, in Section 6 we explain how these two frameworks can complement each other (where necessary), and present two projects that were developed using OVE.

2. Related Work

Using SRDEs for data visualization is not a new idea [12–17]. While many researchers have developed application-specific alternatives, those focused on reusability have opted to either develop new frameworks, introduce new applications for existing frameworks [7], extend existing frameworks [18], or combine multiple frameworks [19]. There are a number of competing frameworks, including Equalizer [20], Chromium [21], DisplayCluster [22], CGLX [23], CubIT [24], ContextuWall [25], and Canvus [26], but SAGE2 is currently the most widely used and has nearly 100 deployments worldwide. The popularity of SAGE2 is partly due to it being a re-development of SAGE [27], one of the oldest software frameworks for high-resolution tiled displays. The choice of web technologies also makes it easier for industry and academia to adopt it for most data-centric collaborative research projects as pointed out by Renambot et al. [7].

Equalizer, Chromium, DisplayCluster and CGLX are all developed using non-web-based OpenGL technology. Given the limitations of web browsers and also the WebGL standard compared to the latest OpenGL standard, these frameworks have superior performance and scalability when displaying high resolution 2D and 3D graphical content. But, adopting OpenGL for data visualization comes with several usability challenges. Many popular data visualization [10, 28], mapping [29, 30] and graph drawing libraries [31, 32] used by data scientists do not have direct OpenGL equivalents, introducing considerable application development overheads. Similarly, most of these frameworks do not provide Python SDKs or REST APIs, preventing seamless integration with data analytics workflows.

ContextuWall is a framework for co-located and remote collaboration using touch and mobile devices as well as displays of various sizes. It provides an ecosystem for asset management with superior collaborative annotation capabilities. Despite these features, ContextuWall only supports image formats and lacks support for video, audio, HTML and PDF formats. CubIT is a multi-user presentation and collaboration framework that was specifically designed for the Cube facility of the Queensland University of Technology (QUT). Due to the restricted scope of their use-cases, the framework only supports images, videos and text notes. It does have a useful feature befitting QUT's requirement of being able to create presentations by sequencing the display of content in these three formats, but, it is not designed as a general purpose framework and it is unsuitable for deployments involving heterogeneous display devices.

Canvus is a purpose-built commercial software developed by MultiTaction to be specifically used by customers who purchase display walls built using the touchscreens that they sell. It is a dynamic windowing environment for collaboration and supports many media formats. Canvus needs to be installed on a single computer to which the displays are connected, limiting its scalability and making it unsuitable for environments with higher resolutions such as EVL's CAVE2 [33]. Also, it neither has out-of-the-box support for non-media formats such as maps, graphs and charts, nor has an application repository with a list of extensions. Canvus also uses a customized web browser (Chromium Embedded Framework) limiting its ability to embed frameworks such as SAGE2 and ParaViewWeb [34].

In addition to specialized middleware for SRDEs, we also looked at few other examples, such as the modern game engines, Unity and Unreal Engine, which are not only used to develop games but also as platforms for developing interactive experiences, some of which are related to data visualization [35, 36]. Mechdyne Corporation's getReal3D [37] is a Microsoft Windows application and a plugin for running Unity applications on SRDEs such as CAVE2. DXR [38] and IATK [39] are toolkits for developing immersive data visualizations on Unity. While game engines are suitable for some use-cases, they have usability limitations (similar to OpenGL-based frameworks) as they are platform-dependent and require considerable development effort. Consequently, many of these frameworks are not suitable for general purpose data visualization. However, we find several useful features, such as their support for visibility and occlusion culling which improves performance as well as their modular architecture which improves scalability.

2.1. Requirements and Limitations of Existing Systems

SAGE2 is a lightweight middleware designed to run on a web browser that enables local and remote collaboration on SRDEs [7]. The drawbacks of other frameworks makes it the best among currently available options for driving SRDEs in data observatories. SAGE2 provides a web-interface through which the user can open applications and load content (images, videos, PDFs, 3D animations, or a pixel stream from a shared desktop). SAGE2 also provides features for collaborative working, including partitioning the display area and embedding remote sessions to enable multi-site collaboration. However, a major conceptual difference is that SAGE2 acts essentially as a *window manager for a seamless ultra-high resolution desktop* for a tiled display, whereas OVE provides an *ecosystem for data visualization*. This introduces a few notable limitations that prevented us using SAGE2 as-is, extending it, or supplementing it with a combination of existing libraries and frameworks. We were particularly affected by three key limitations of SAGE2:

L1: It relies on server-side libraries such as ImageMagick and FFmpeg to process and render images and videos, which limits the maximum number of pixels of a single continuous application that can be displayed in its vast display area. As a result, it cannot be used to play 4K or 8K videos.

L2: Existing content developed using popular JavaScript frameworks such as D3.js and Three.js needs to be redeveloped as SAGE2 applications (with substantial modifications) before they can be displayed, introducing significant development overheads.

L3: The *SAGE2 Pointer* must be used to interact with most applications. This is a key design decision but limits the capabilities of the framework by preventing performance optimizations such as culling.

In terms of requirements, Ni et al. present an excellent survey of a variety of challenges when working with large high-resolution display technologies [40]. While they look at various aspects covering displays, interaction devices, reconfigurability of the infrastructure etc., a few challenges are specifically relevant for middleware designed to visualize data in SRDEs:

C1: Providing a truly seamless tiled display by preventing visual discontinuity of content that cross bezels, crucial when visualizing large networks

C2: Scalability to support 1000 display tiles

C3: Parallelizing rendering (and other workloads) to improve performance

C4: Integrating SRDEs into a seamless computing environment

The design of OVE is greatly influenced by that of SAGE2, and as a result most technologies used by these systems are similar. But, OVE also takes design cues from game engines and is implemented as a modular microservices architecture addressing the challenges **C1–C4** as well as the limitations **L1–L3** that we have outlined above. The key design decisions underpinning OVE are aimed at addressing these challenges and limitations, but they also introduce some limitations to our implementation, which we discuss in Section 3.

3. Key Design Decisions and Limitations of OVE

OVE is an ecosystem for data visualization on SRDEs that provides users with performance, scalability, portability, and ease of use, above and beyond existing frameworks. OVE is designed to run independently or be embedded within other frameworks that support web application rendering such as SAGE2 and Canvus.

To allow scaling to large numbers of display tiles (**C2**), OVE was designed as a **modular middleware** based on a *compositional microservices architecture* [41] in which each component is made up of granular subcomponents that can be deployed together or separately. Deploying a large resource-intensive application downgrades the performance of most middleware that drive SRDEs, regardless of whether they are web-based or not. The ability to isolate the impact of these applications from the rest of the ecosystem based on their demand makes it possible to retain optimum performance levels and thereby ensure greater scalability of the design. Demand-driven isolation becomes straightforward when each application, and each service consumed by an application, is implemented as a separate microservice.

This architecture differentiates OVE from previously existing frameworks. For example, it is possible to entirely decouple an unused subcomponent from the core server component or have more than one instance of a component for high-availability. Like SAGE2, OVE uses WebSockets to broadcast messages between control and display clients. Interactive application instances (e.g. a WebGL animation) spanning several display tiles tend to produce large quantities of messages in shorter timespans, introducing significant bottlenecks on messaging ecosystems that uses WebSockets. Unlike SAGE2, OVE can provide such application instances with a dedicated message broker while other instances of the same application as well as other applications can share a common message broker within the core server component.

To eliminate exhaustive client-side resource consumption and improve performance, OVE supports **visibility and occlusion culling** — the display clients do not load/render content that lies outside of its viewport as well as occluded content that would not be visible to a user. Deploying 1000 applications on a SAGE2 canvas spanning 50 display tiles creates 50000 instances of the application in the best case. In OVE, visibility culling ensures that there are at most 1000 instances (one instance per application) in the best case where there is no occlusion. Occlusion culling provides a further improvement ensuring that there are at most 50 instances (one instance per display tile) in the best case when the top-most application instance occupies the entire display area covering all tiles.

It is impossible to support visibility culling in SAGE2 due to **L3**. The SAGE2 Pointer works on a *master-slave* principle where each action made by the user is directly replicated to each and every display client. This requires each and every application instance to be present on each and every display tile regardless of whether the content was within the viewport or not. As well as impacting performance, this introduces other limitations. For example, SAGE2 cannot simultaneously play more than 16 videos, even if they are distributed across multiple display tiles, as web browsers such as Google Chrome currently support a maximum of 16 active WebGL contexts.

OVE provides an independent control client and a touch/pointer sensitive overlay that can be rendered as the top-most layer of the display area. The touch/pointer sensitive overlay works on a *client-server* principle, capturing user actions and relaying them back to the control client which in return broadcasts corresponding changes to the display clients. This ensures that OVE does not have the limitation **L3**. However, one downside of supporting occlusion and visibility culling is that content would not necessarily be pre-loaded on each and every display client, which would invariably introduce overheads if the location of the application on the display changes frequently. Therefore, to meet the requirements of **C1** by preventing visual discontinuity of content, we always toggle the visibility (hide, then show) before and after moving an application from one location to another.

A **synchronized clock** is used by OVE to ensure that all clients perform actions at precisely the same time. We initially assumed that all server-side and client-side components have clocks already synchronized using protocols such as NTP, but in practice this is not always the case, especially when addressing **C4**. We therefore use the Berkeley clock synchronization algorithm proposed by Gusella and Zatti [42] to keep all active components (control and display clients and as well as server-side) in sync with the OVE core server, which acts as the master. As OVE implements a client-server architecture, the server is completely unaware of the clients that may exist, unlike in a master-slave architecture. Therefore, we made a simple enhancement to the algorithm to let the clients initiate the synchronization handshake to which the server responds. Thereafter, when a user performs an interactive operation such as starting an animation or playing a video, the control client schedules it to run at a precise time that is sufficiently far in the future for the call to reach all display clients before it is due to execute; in practice, this time is a few milliseconds beyond the synchronized clock of the control client.

Synchronized clocks would ensure videos start playing perfectly in sync, but **C1** also requires videos to continue to play at the same rate on all display clients. For various reasons, video playback can be momentarily interrupted on a web browser, causing some clients to be a few frames ahead or behind the others. To address this, OVE also includes a timekeeping subcomponent in time-critical applications (including the videos application). This pauses the playback of videos by a few frames on faster display clients and resumes the playback at a precise point in time to ensure that the videos keep playing in sync despite these web browser-specific delays. This timekeeping subcomponent runs locally on each display client against its synchronized clock without introducing additional overheads to the messaging ecosystem of the overall system. The modular architecture of OVE combined with features like this makes it possible to support truly parallel rendering of content (including high resolution images and WebGL animations achieving `60 FPS`) to meet the requirement **C3**.

To make it less likely that a browser will interrupt playback due to being late to buffer a video (which is a common occurrence), OVE also waits for a certain percentage of the video to buffer before enabling the play button on the control client. These features, together with the synchronized clock, eliminate the need for server-side synchronization of video playback, overcoming the limitation **L1** as that affects SAGE2, and allowing the playback of 4K and 8K videos. The videos application does not create a distinction between local, remote or streaming videos (unlike SAGE2, which plays YouTube videos using its web browser application rather than its video player application). Therefore, unlike SAGE2, OVE also synchronizes 8K YouTube videos (which is the highest resolution supported by YouTube at the time of writing) spanning multiple display tiles meeting the requirements of **C1**.

OVE loads content for each application by URL, which is different to most contemporary frameworks where users are expected to upload assets to a server-side storage. For use-cases where assets need to be versioned, stored securely, or shared among a limited set of users, OVE also provides an **asset manager**, as a separate modular microservice. The asset manager can connect to multiple local and external stores making it possible to distinguish between private, shared and public assets. Immovable data (stored on Neo4J/MongoDB databases) and live streaming data (streaming audio/video and WebRTC based video conferencing and screensharing sessions) are not stored on the asset manager and are accessed directly as in the case of SAGE2. Furthermore, multiple assets (of various types) can be grouped together and stored under a project structure described by an optional manifest file (`project.json`) instructing OVE how they should be displayed as a single composite visualization. Another feature of the asset manager is to coordinate one or more microservices (called *Workers*) to pre-process assets on-demand or as they are uploaded. Workers can extract zip files, automatically convert high resolution image files into Deep Zoom Images (DZIs) or pre-run layout algorithms on files containing graph data. This addresses **L1** as it avoids the need for additional server-side libraries.

Compositing visualizations is the most notable capability of OVE, which is achieved on the server-side through a project model (described using a `project.json`) and the use of iFrames on the display clients to isolate application instances from each other, allowing the use of multiple JavaScript libraries without conflicts. OVE makes use of a number of web frameworks and JavaScript libraries [11, 29–32, 43–49] to render content. The choice of iFrames makes it possible to embed existing content with few or no modifications, meeting **L2**. D3.js based content is rendered with no modifications using the Tuoris [43] library whilst Three.js content require a simple modification to include the `Distributed.js` library provided by OVE. By using borderless iFrames with transparent backgrounds, it is possible to seamlessly join or layer multiple fragments of content of various types as desired.

One of the key benefits of OVE is the possibility to breakdown a complex visualization into portions and render them individually; this increases the rendering speeds as well as the reliability of the system. A simpler example will be rendering a graph with four separate iFrames containing (i) the vertices and edges, (ii) the labels, (iii) the axes and legends and (iv) the annotations. In this example, the vertices and edges would be rendered using one instance of the networks application while the labels can be rendered using another. If the axes and legends were drawn in SVG, they can be rendered using the SVG application and the annotations can be made using either the HTML application or the whiteboard application. OVE also natively supports semi-transparent overlaying by controlling the opacity of iFrames, making it not only possible to overlay semi-transparent text and graphics but also videos, maps, WebGL animations and even DZIs.

Deployment complexity is perhaps the most substantial limitation of OVE, as it consists of several microservices. To simplify the installation experience, the binary release of OVE is a multi-container docker application along with a docker-compose description that is capable of downloading and running the containers in a couple of steps. Those who prefer not to use Docker or who want to extend and improve OVE can compile the source code and deploy the resulting Node.js applications using the PM2 [50] daemon process manager. In addition to that, as explained previously, OVE components (and subcomponents) can be deployed together or as a set of isolated instances. To further simplify the management of the installation, the OVE docker application also includes (a) a lightweight NGINX load balancer which reduces the number of exposed ports to a handful and (b) a status UI which provides availability and uptime details of each component. It is also possible to deploy a minimum subset of OVE components on systems with very low resources — a system that is only rendering images requires activating just the OVE core and the images application backend components, while all other microservices can remain deactivated.

Lastly, OVE is a **framework based on web technologies** similar to SAGE2, Canvus, CubIT and ContextuWall. This is a huge benefit in terms of data visualization, but makes the framework inferior to some other OpenGL based frameworks such as Equalizer, Chromium, DisplayCluster and CGLX which are much more performant and capable of rendering immersive 3D experiences at a much higher quality. We see this as a limitation, but this is a trade-off that we willingly made.

4. OVE Architecture

OVE is a modular middleware providing users with an ecosystem of built-in applications and specialized services that can be used along with popular JavaScript frameworks to design and develop interactive composite data visualizations suitable for SRDEs with minimum effort. OVE is implemented as a collection of loosely coupled components (i.e. microservices) which can be deployed together or in isolation. These components integrate with each other by passing messages using WebSockets and REST APIs. The architecture of OVE is shown in Figure 1 and explained in detail below.

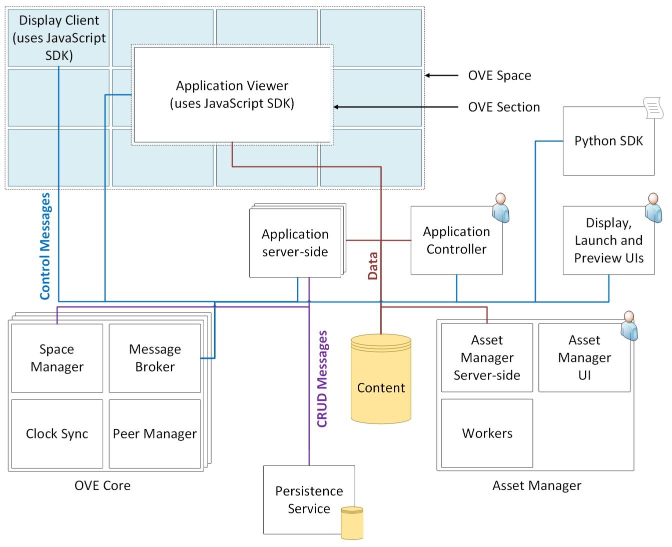

Figure 1: **The architecture of OVE.** This figure describes the key components of OVE and distinguishes between the control message flows, CRUD messages used for persistence, and data flows.

4.1. *OVE core*

The OVE core is a platform-independent non-blocking Node.js application. This can be scaled by introducing more than one instance of OVE core, making it highly available and capable of supporting a large volume of requests. It has a client-server architecture, and the server has four subcomponents for, (a) space management, (b) message brokering, (c) clock synchronization and (d) peer management. The client-side is designed to run on a modern web browser and has two subcomponents: (e) display clients and (f) a JavaScript SDK.

Improving the client-side performance is critical for the scalability of OVE core, and it works best when each display client (opened in a web browser) spans a moderate number of pixels — for instance a resolution of `1920x1080`. Many smaller clients will increase the communication overhead, and similarly, few larger (higher resolution) clients will increase the time taken by the web browser to render/paint content. The display canvas (an OVE space) can span multiple display tiles to provide an extremely large overall resolution. Within an OVE space, each application instance is loaded in its own container (an OVE section), which is a rectangular region within a space. Sections may overlap with each other and can be grouped together to represent a composition. The space management subcomponent is responsible for defining spaces, creating sections and grouping them for which it provides a REST API to be used by other OVE components.

The Message Broker (MB) subcomponent uses WebSockets and operates in a broadcast model. Several types of messages are exchanged through the MB, for synchronizing clocks, describing the creation or deletion of sections, and executing application-specific operations such as playing videos or panning a map (Section 4.2). To minimize the use of network bandwidth, OVE uses two levels of filtering before transmitting the messages over WebSockets: (a) at each publisher, it filters out each message to only include the deltas of what changed, and (b) at the broker it introspects message headers and only passes on the messages relevant to each consumer. In addition to reducing the number of application instances displayed on each client (through visibility and occlusion culling), OVE is capable of minimizing the message volume received by each client using the filtering described above and thereby significantly outperform SAGE2 in terms of client-side resource consumption (which we demonstrate in Section 5).

The peer management subcomponent of OVE makes it possible to introduce additional instances of OVE core for high-availability and for scalability (such as having a dedicated MB for an instance of a resource-intensive application). It helps distribute load among server-side peers and improves the reliability of the system. The client-side of OVE core consumes services provided by the server-side subcomponents. The JavaScript SDK provides a simplified interface for display clients and OVE applications to consume these services; it (i) obtains geometry information of the respective display tile, (ii) communicates among client-side peers within a single web browser or across many, (iii) obtains the time from the synchronized clock, and (iv) reads and writes application state.

4.2. Applications

Each OVE application embeds a base-library exposing a REST API for application state management at the server-side as well as client-side code to consume the JavaScript SDK provided by OVE core. The majority of the functionality provided by an application is implemented in this base library, and it is possible to introduce new OVE applications (similar to SAGE2) by simply embedding and extending this library. In addition to the read/write APIs, the server-side also provides APIs to transform the state of an application corresponding to a zoom or pan operation. Such operations can be simultaneously triggered across multiple application instances, and therefore, the interactivity of a composite visualization is not just limited to the top-most layer.

The server-side of a typical application extends these common APIs with application-specific functionality such as starting or stopping playback of audio and video or filtering a graph. Whilst the server-side of an OVE application is made up of a single subcomponent the client-side contains two subcomponents: (a) a viewer (for displaying content) and (b) a control client (for controlling what is displayed). Each viewer is designed to be deployed into an OVE section that spans one or more display clients of OVE core. Each control client is a web application that is designed to be opened in a separate web browser (which can be on a desktop, tablet or a mobile phone). Applications can also have a third client-side subcomponent corresponding to the various players (or libraries) used for content-specific rendering. These are designed as JavaScript libraries that can be consumed by and embedded within viewers and control clients.

Audio: plays an audio file or stream
Charts: displays the result of applying a Vega-Lite [32] specification to a dataset
HTML: displays a web-page, which may contain an interactive visualization created with JavaScript
Images: displays an image (either from a single image file, or from a source of image tiles such as a DZI)
Maps: displays a map, which can be a combination of several raster and vector layers
Networks: displays a node-link diagram of a network
PDF: displays a PDF file
QR Code: draws a QR code for a provided URL
SVG: displays SVG content rendered using the Tuoris [43] library, which may contain an interactive visualization
Videos: plays a video (from a single video file, video streaming service or source of video tiles)
WebRTC: displays a shared screen or a videoconference using the OpenVidu [49] library
Whiteboard: allows annotations to be drawn and displayed

Table 1: **List of built-in applications.**

To cover requirements for data visualization, OVE provides twelve built-in applications, each dedicated to the display (or playback) of a particular kind of content, as shown in Table 1.

OVE also provides three special applications: the **Alignment** application supports bezel corrections to ensure precise positioning to accommodate (a) bezel width, (b) misalignment of screens due to uneven floor surfaces and physical displacement caused by wear and tear, and (c) heterogeneous screens and display devices. The **Replicator** application makes it possible to embed a replica of another space (which may even belong to a remote OVE deployment) in an OVE section. The **Controller** application is used to render a touch/pointer sensitive overlay as the top-most layer of a display area.

4.3. Specialized services and UIs

The OVE **persistence service** supplements the space management functionality of the core and the state management functionality of applications. In situations where there are more than one server-side instance of either the core or an application, this optional service provides replication to ensure that multiple instances always have a single copy of its configuration, which would otherwise be stored in-memory.

The **asset manager** of OVE (discussed in Section 3) is made up of three subcomponents: (a) a server-side component that takes care of interaction with object stores and provides APIs for the client-side, (b) a UI at the client-side, and (c) a collection of content-specific workers. Where necessary, the specialized project loader service is used to parse manifest files (`project.json`) and launch composite visualizations using content stored on the asset manager.

OVE also provides three **UIs** to (a) display the status of system on a per-component basis, (b) launch new applications and load content into them, and (c) preview an entire space (at a lower resolution) on a single web browser. The Python SDK provided by OVE can be used as an alternative to both the launcher UI and the project loader service.

5. Performance and Scalability Analysis

Throughout this paper we made several claims about the superior performance and scalability of OVE compared to contemporary frameworks. We explained how OVE addresses three key limitations in SAGE2 (**L1**–**L3**), as well as four challenges relevant to data visualization middleware for SRDEs (**C1**–**C4**). The key design decisions behind OVE and its architecture (discussed in the previous sections) address these limitations and challenges. This section presents results from a series of tests on our implementation of OVE. These tests were conducted to validate our design. Using these, we measure the performance and prove the scalability of OVE. Thereafter, we compare the results from OVE and SAGE2, tested in an identical environment.

We ran our tests in the immersive data observatory in Imperial College London's Data Science Institute, which has been showcased in previous publications [5, 51]. The client-side is made up of 64 Full HD (`1920x1080` pixels) Samsung UD46D-P professional video wall displays, arranged in a cylindrical layout in 4 rows and 16 columns; this gives the SRDE a total pixel width of `30720` and height of `4320`, so the total display resolution is `132.7` megapixels. This system is powered by 32 compute nodes, each of which has 12 CPUs[1] and `32GB` of memory. In our tests each display client was always opened in a separate Google Chrome browser window, and therefore, we had a total of 64 web browsers (1 per screen and 2 per compute node). Some of our test cases used more than 64 display clients, where we opened more than 2 web browsers per compute node. In terms of the server-side, each test case had its own dedicated OVE (or SAGE2) environment, which was an `Ubuntu 18.04` virtual machine with 4 virtual CPUs and `8GB` of memory. **T6** (noted below) used several instances of OVE that were installed in separate virtual machines of the same specification.

We conducted six different types of tests, each of which has one or more test cases. We ran each test case 10 times to reduce the effect of random errors on our measurements and increase the reliability of our results.

T1: Displaying more than one application

- **T1a:** Playback of 1–16 videos on OVE and SAGE2
- **T1b:** Playback of 1–1000 videos on OVE

T2: Displaying content of different dimensions

- **T2a:** Playback videos of various resolutions
- **T2b:** Playback of an `8K` video made up of 16 full HD tiles

T3: Rendering a DZI across 1–1000 display clients on OVE and SAGE2

T4: Displaying a single application on a SRDE with 1000 display clients: we displayed a single PDF file on a space spanning 1000 web browsers and compared OVE vs SAGE2

T5: Managing 10–1000 display clients without deploying any applications

T6: Running all OVE components on a single server vs separating OVE core, the MB subcomponent and the server-side subcomponent of an OVE application (this test is a variant of **T3**)

We present the results of these tests in Figures 2, 3, 4 and 6. The network bandwidth consumption is reported in `Mbps` and the CPU and RAM consumption are expressed as a percentage. On the client-side, 100% CPU represents 12 cores and 100% RAM represents `4GB`. On the server-side, 100% CPU represents 4 virtual processors and 100% RAM represents `8GB`.

5.1. Suitability for displaying hundreds of applications at a time

We measured server-side and client-side performance as well as the client-side network consumption in **T1**. The 1000 videos we used were sourced from traffic cameras in London[2] and were of the Common Intermediate Format (CIF) with a resolution of `352x288` pixels and a file size of around `200KB`. SAGE2 was only capable of playing 16 videos at a time due to a limitation in web browsers (noted in Section 3), and therefore, we ran two test cases **T1a** and **T1b**. In the first test case we displayed all 16 videos on a single display client; in the second we evenly distributed up to 1000 videos among 64 display clients.

The results of this test (Figure 2) show that the client-side CPU, RAM and network bandwidth consumption is directly proportional to the number of videos displayed on a client in both OVE and SAGE2. The two systems consume similar amounts of CPU and RAM, but SAGE2 consumed 20–30 times more network bandwidth than OVE. The amount of network bandwidth consumed by SAGE2 is much larger than the file size of the video, introducing a significant stress on the infrastructure. This combined with **L1** makes SAGE2 unsuitable for playing videos of resolutions higher than Full HD. The two levels of message filtering at both the publisher and the broker (Section 4.1), the support for visibility and occlusion culling (Section 3), as well as the non-reliance on server-side libraries, contribute to the significant reduction of network bandwidth consumption in OVE.

On the server-side, the amount of RAM consumed was constant on both frameworks irrespective of the number of videos

[1] Intel Core i7-5820K CPU at 3.30GHz per core

[2] `https://tfl.gov.uk/traffic/status`

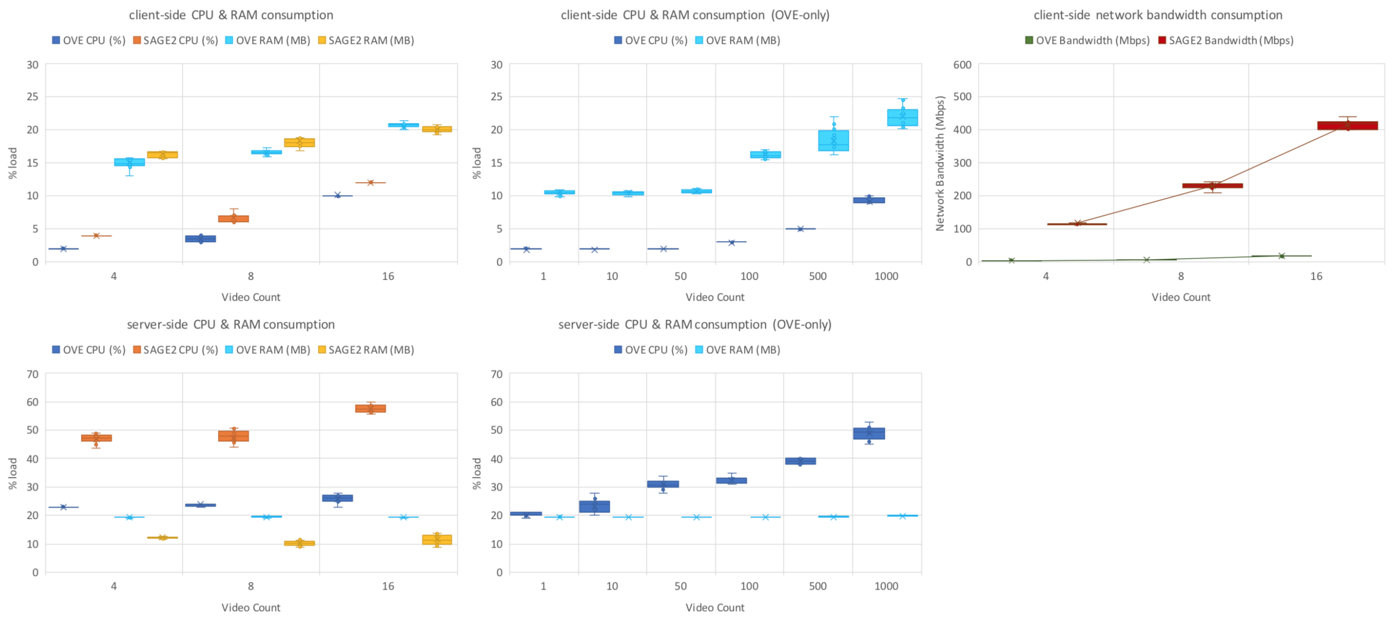

Figure 2: **CPU, RAM and network bandwidth consumption when displaying more than one application.** These graphs show results from **T1**. SAGE2 is only capable of playing 16 videos at a time, therefore, we had a separate test case **T1b** to validate the scalability of OVE when playing 1–1000 videos. We find that SAGE2 consumes 20–30 times more network bandwidth on the client-side and twice the amount of CPU than OVE. On the server-side, the amount of CPU consumed by OVE increases with the number of videos, and it consumes twice as much RAM as SAGE2, but using **T6** (seen in Figure 6), we show that these issues are addressed by the modular microservices architecture of OVE.

File Type	URL	File Size
CIF video	`https://tfl.gov.uk/traffic/status`	200KB
720p video	`https://www.nasa.gov/sites/default/files/files/Apollo_11_moonwalk_montage_720p.mov`	44MB
HD video	`http://ultravideo.cs.tut.fi/video/HoneyBee_1920x1080_30fps_420_8bit_AVC_MP4.mp4`	7.2MB
4K video	`http://ultravideo.cs.tut.fi/video/Bosphorus_3840x2160_30fps_420_8bit_AVC_MP4.mp4`	29MB
8K video	`https://upload.wikimedia.org/wikipedia/commons/1/1e/First_8K_Video_from_Space_-_Ultra_HD_VP9.webm`	491MB
DZI	`http://www.in2white.com/about/`	—

Table 2: **URLs for the list of videos and the DZI used in our tests**. The DZI that we used has a resolution of 365 gigapixels.

displayed. OVE consumes double the amount of RAM (as it is made up of many microservices, whereas SAGE2 is a monolith) but results from **T6** prove that this is not a limitation. OVE consumed almost constant amount of CPU for smaller video counts (1–50) while we saw this number gradually increase to almost double the amount when we played 1000 videos. On the other hand, SAGE2 consumed twice as much CPU as OVE from the start (when playing 4 videos), and this kept increasing even for smaller video counts (8–16) unlike OVE. The gradual increase of CPU consumption on OVE as we increased the number of videos played to a 1000 was due to the overheads introduced by the MB. This impact can be dealt with by isolating the MB from OVE core (which is also covered by **T6**). Therefore, our results show that OVE is scalable and is suitable for displaying hundreds of applications, unlike SAGE2 which had several bottlenecks.

5.2. *Suitability for displaying ultra-high resolution content*

We tested SAGE2 and OVE to understand their suitability for displaying ultra-high resolution content. In **T2a** we used videos with resolutions up to 8K and in **T3** we used a 365 gigapixel DZI (Table 2 lists the videos and image that we used in our tests). **L1** prevents playback of 4K and 8K video resolutions on SAGE2, therefore to compensate for this limitation we tested videos of a much lower resolution (CIF format). **T2a** reveals that in both frameworks, the client-side CPU, RAM and network bandwidth consumption increased proportional to the resolution of the content displayed. While the RAM consumption of the frameworks were similar, we found that SAGE2 consumes 4 times more CPU and the amount of network bandwidth used by SAGE2 to play a 720p video (with a resolution of 1280x720) was similar to what OVE required to play an 8K video. This reconfirms that SAGE2 consumes 20–30 times the bandwidth required by OVE, regardless of the video resolution.

In **T3** we displayed a DZI across 1–1000 web browsers. Unlike when playing videos (where the web browser needs to consume the entire file) rendering images (including DZIs) benefit from visibility culling in OVE. Therefore, we find that OVE uses constant CPU and RAM to display DZIs regardless of the number of display clients. Unlike in OVE, on SAGE2, the amount of CPU and RAM keeps increasing with the number of display clients. This makes it impossible for SAGE2 to display DZIs spanning hundreds of display clients and therefore limits the total resolution at which content can be displayed. Results are presented in Figure 3.

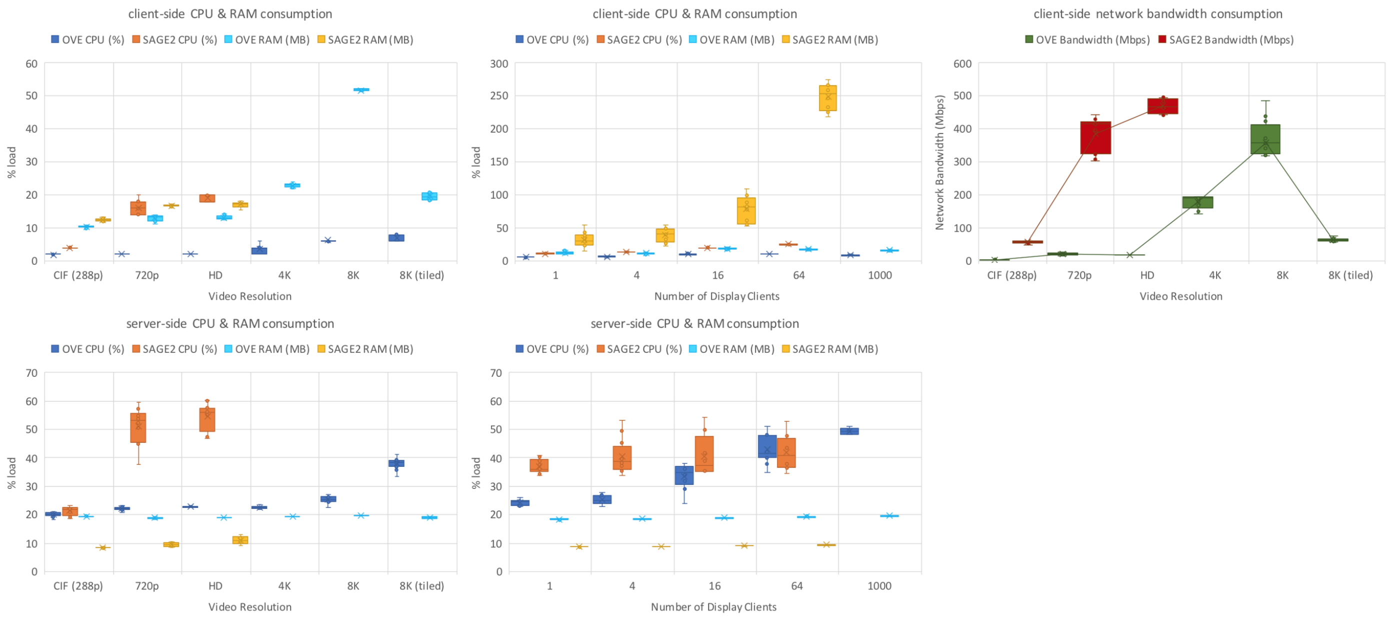

Figure 3: **CPU, RAM and network bandwidth consumption when displaying content of different dimensions.** These graphs show results from **T2** and **T3**. We prove that OVE is capable of displaying content of various dimensions spanning hundreds of display clients while pointing out two significant scalability bottlenecks in SAGE2: (a) it is unable to support playback of videos beyond Full HD quality, and (b) it fails to render the DZI we chose across 100 display clients or more.

On the server-side, for both **T2a** and **T3**, OVE and SAGE2 consume a fixed amount of memory. OVE consumes a fixed amount of CPU regardless of the video resolution, but the amount of CPU consumed by SAGE2 keeps increasing proportional to the resolution of the video. However, SAGE2 consumes fixed amount of CPU when rendering a DZI, but on OVE the amount of CPU consumed increases proportionally to the number of display clients involved due to the overheads introduced by the MB. As noted previously, this bottleneck can be overcome by deploying dedicated MBs for resource-intensive application instances and isolating them from the rest of the ecosystem. Importantly, the scalability and performance characteristics of the two frameworks varied based on the content that was displayed. OVE was superior to SAGE2 in all fronts except for server-side memory where it needed twice the amount required by SAGE2.

In order to play videos of even higher resolutions such as 16K and 32K OVE supports playing tiled videos meeting the requirements of **C3**. In this scenario, each high resolution video would be broken down into tiles of a lower resolution. For example, an 8K video would have 16 Full HD tiles while a 16K video would have 64 tiles; and, in such a situation, on a display of a Full HD resolution there would be 4 video tiles at most. In order to support the playback of tiled videos OVE relies on its synchronized clock. This introduces a considerable performance gain by resolving a limitation of web browsers performing poorly (and are often unable to retain the frame rates) when rendering very large video files at very high resolutions.

Results from **T2b** (Figure 3) show that playing an 8K video made up of 16 Full HD video tiles has comparable client-side RAM consumption to playing a 4K video. The CPU consumption would be similar to playing an 8K video that was not tiled, but the network bandwidth would be much less and roughly equal to playing 4 Full HD videos. On the server-side, OVE consumes a fixed amount of RAM but around 50% more CPU, due to the considerable amount of messaging involved to keep the playback in sync. The results from **T6** show how these overheads can be managed.

In summary, **T2a**, **T2b** and **T3** show that OVE is suitable for displaying ultra-high resolution content. SAGE2, unlike OVE does not have a concept of a synchronized clock and does not support the playback of tiled videos. It also had several bottlenecks preventing the playback of 4K and 8K videos and displaying ultra-high resolution DZIs across a large number of display tiles.

5.3. *Suitability for managing 1000 display tiles*

The third dimension of scalability and performance is the suitability for managing an extremely large display area. We used **T3**, **T4** and **T5** to show that OVE is capable of managing 1000 display tiles, each of which were Full HD providing a total resolution of 2.07 gigapixels. Using **T5** we were able to confirm that OVE consumes fixed CPU and RAM on both server-side and client-side when managing 10–1000 display tiles, if we did not deploy a single application (see supplementary material for additional details). Like OVE, SAGE2 is also capable of managing 1000 display tiles and shows similar performance characteristics. However, we see a major difference in terms of performance when we start deploying applications on these two frameworks.

In **T4** we displayed a PDF file[3] (sourced from the open

[3]`https://bitbucket.org/sage2/sage2/raw/8ac17e4a503c746f30dee1b428c29cd6e1673aad/public/uploads/pdfs/SAGE2_collaborate_com2014.pdf`

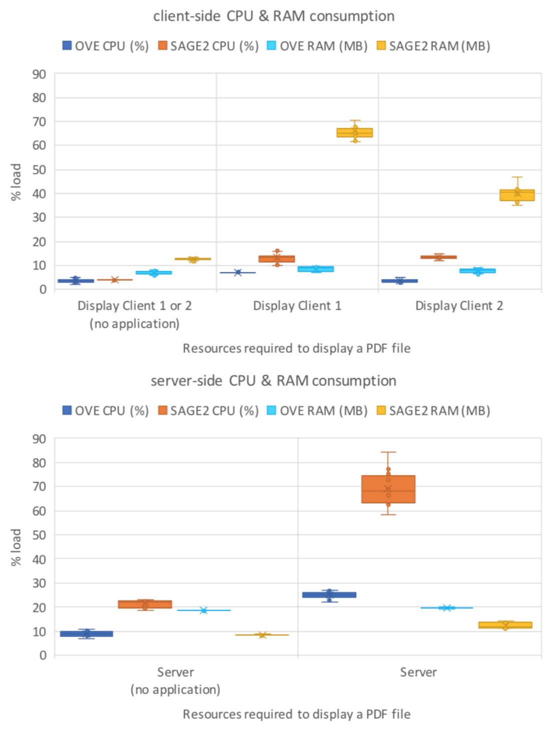

Figure 4: **Client-side and server-side resource consumption when displaying a PDF file on a display environment made up of 1000 display clients.** These graphs show results from **T4** and **T5**. Here we show the amounts of resources consumed when there are no applications deployed and then take two display clients: one on which the PDF application is deployed and visible and another where the application is not visible. Interesting differences in the results show us the benefits of visibility culling in OVE and the corresponding limitations in SAGE2

source repository of SAGE2) using an instance of the PDF application spanning `1920x1080` pixels on a display that had a total resolution of `480000x4320` pixels. On OVE, this resulted in 2–3% more CPU on the client-side and 15% more CPU on the server-side. The RAM consumption on both client-side and server-side had no noticeable changes. On SAGE2, the server-side RAM consumption remained unchanged, but everything else increased. On the client-side, SAGE2 consumed 4 times more RAM (an increase of around `500MB`), and the CPU consumption increased by around 8–10%. The two frameworks use the same library to display PDF files [46] and the difference in terms of performance was a questionable observation. Unlike OVE where each display tile was limited to a resolution of `1920x1080`, each display tile on SAGE2 spanned the entire `480000x4320` pixel resolution - as a result of **L3**. While this had no performance impact when no applications were deployed, the web browser consumed more CPU and RAM on the client-side even with a single application.

We also observed high server-side CPU (an increase from around 20% to 70%) on SAGE2 with **T4**. This was due to SAGE2 not supporting visibility culling. We found that SAGE2 created an instance of the PDF application on each and every display tile (1000 in total) even though it was only visible on a single display client, unlike OVE (which loaded the application on a single display tile). The communications overheads on SAGE2 was what contributed to the increase in server-side CPU consumption. These results (presented in Figure 4) point out major advantages of visibility culling in OVE. Despite these issues SAGE2 manages to display the PDF file at a resolution of `1920x1080` pixels, but from **T3** we find that it eventually fails when attempting to display an application spanning a sufficiently high resolution. Using the results from **T3** and **T4** we conclude that SAGE2 does not perform well when displaying content on SRDEs with an extremely large display area. And, based on these observations we find OVE to be more suitable than SAGE2 for a gigapixel display such as the RealityDeck [52].

The RealityDeck is made up of 416 display tiles, each of which has a resolution of `2560x1440`, providing a total resolution of `133120x11520`. At the Data Science Institute, we do not have an SRDE that can provide us with 416 screens for testing. Therefore, we replicated this environment by opening 13 web browsers per compute node on our SRDE with 64 screens. We then opened the DZI from **T3** to prove that OVE is capable of rendering content at such a resolution. We thereafter increased the resolution of each display client to `5120x2880` and repeated this experiment, proving that OVE can render content on a `6.13` gigapixel display. Figure 5 shows three photographs from these two experiments along with another taken while running **T3** showing OVE rendering the same DZI on a `2.07` gigapixel display made up of 1000 display clients. We have included the three videos corresponding to these three images as supplementary material. The RealityDeck still remains one of the largest SRDEs, and, due to the limitations in display resolutions, graphics adapters and display cables as well as the practical challenges in navigating such large environments, it is unlikely that there would be displays much larger than a few gigapixels, at least for a few more years. This makes OVE suitable for operating the largest SRDEs that can be found today and mostly likely for a few more years ahead.

5.4. Effect of deploying microservices on different virtual machines

One of the recurrent observations in Figures 2, 3, and 4 was that OVE consumed much more server-side RAM than SAGE2. We also found OVE to consume a higher amount of CPU when we increased the number of applications to 1000. We therefore designed **T6** to prove that the modular microservices architecture of OVE is capable of addressing this issue. In Figure 6 we show the server-side CPU and RAM footprints in four situations:

S1: The OVE core (inclusive of the MB) and the images application are deployed together on a single VM.

S2: The OVE core (inclusive of the MB) is deployed on a single VM and the images application is deployed on another VM — we monitor the performance of OVE core.

S3: The OVE core (inclusive of the MB) is deployed on a single VM and the images application is deployed on another VM — we monitor the performance of the images application.

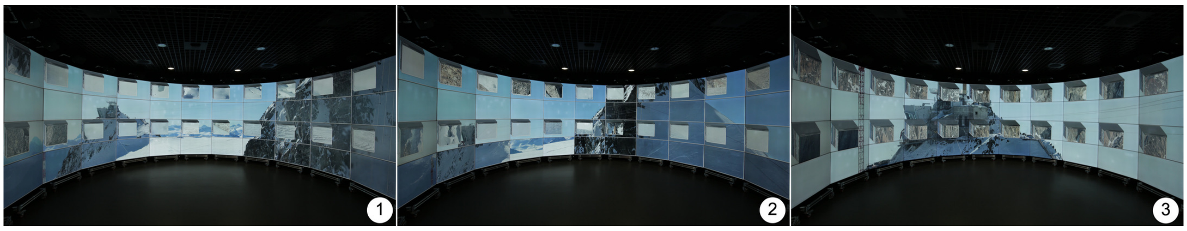

Figure 5: **Photographs showing OVE displaying a DZI across many display tiles.** These photographs were taken while running **T3** and show the DZI displayed across: ①: 416 display tiles with a total resolution of `1.53` gigapixels. ②: 416 display tiles with a total resolution of `6.13` gigapixels. ③: 1000 display tiles with a total resolution of `2.07` gigapixels. Readers may notice a discontinuity between image tiles in photographs ① and ② as the resolution of each tile exceeds that of each individual display making only a portion of each tile visible. This misalignment is more obvious in the videos corresponding to these photographs, which we have provided as supplementary material.

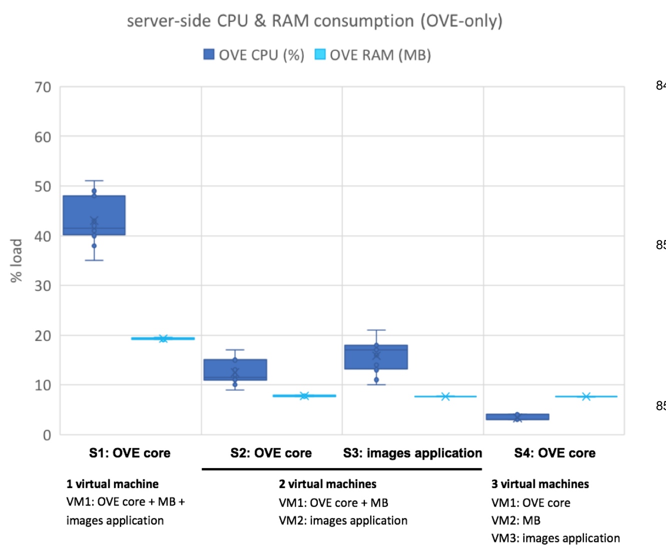

Figure 6: **Server-side CPU and RAM consumption in four situations S1–S4.** This graph shows results from **T6**. The four scenarios prove that isolating the images application from the OVE core halves the amount of CPU and RAM consumed on the server-side. Isolating the MB from the OVE core will reduce the CPU consumption to around 5%.

S4: The OVE core, the MB used by the images application and the images application are deployed on three separate VMs — we monitor the performance of OVE core.

From **S1** and **S2**, we show that deploying OVE core and OVE applications on separate VMs halves the consumption of RAM and CPU on the server-side. Isolating the MB from the OVE core reduces the amount of CPU consumed even further (as proven in **S4**). The VM on which we installed the images application also had 14 other OVE applications running, but, as they were not actively used, they made no noticeable contribution to amounts of CPU and RAM used in **S3**. Moreover, these applications can be separated and deployed on isolated VMs; individual applications can be replicated many times as required. The microservices architecture improves the scalability and reduces the overheads of running OVE on the server-side, however, we chose to deploy all these components on a single VM for **T1**–**T5** so that we can make a fair comparison with SAGE2.

5.5. Discussion

In summary, this analysis shows that OVE is suitable for: (a) simultaneously displaying hundreds of applications, (b) displaying ultra-high resolution content and (c) managing 1000 display tiles. Within our analysis, we also explained the advantages (in terms of server-side CPU and RAM consumption) of deploying OVE microservices on different virtual machines and pointed out some of the limitations of SAGE2 and explained how we have overcome those in OVE.

From this study, we find that the compositional microservices architecture, as well as the support for visibility and occlusion culling in OVE, provides better performance and scalability compared to SAGE2. The framework does not rely on server-side libraries for processing and rendering media formats, making it suitable for playing 4K and 8K videos spanning multiple display tiles. Together, these features significantly reduces the network bandwidth consumption as explained earlier in Section 5.1. Furthermore, OVE also adopts a non-invasive programming model for displaying existing content developed using JavaScript.

Despite these compelling features, OVE has a more limited scope than SAGE2. It lacks a collection of extensive applications and has not been designed to leverage input from various interaction devices. Instead, our framework can be used as an extension to the immensely popular SAGE2 middleware. Along with example use-cases and user feedback, we explain how OVE can be used as a supplement to SAGE2, in the next section. While it is possible to implement new types applications for OVE, further improve some of its features, and make it outperform SAGE2, our objective here was to build a framework that suited our requirements. Therefore, with such a restricted scenario, the focus of our tests was to prove that the innovative design of OVE (which we implemented to address **L1**–**L3** and **C1**–**C4**) does not introduce any performance and scalability limitations to the best of our knowledge.

6. Example Use-cases

So far, we have used OVE to develop over a dozen visualizations for projects at Imperial College London's Data Science Institute [6], and at three other institutions. The system was recurrently used by audiences from academia and industry for

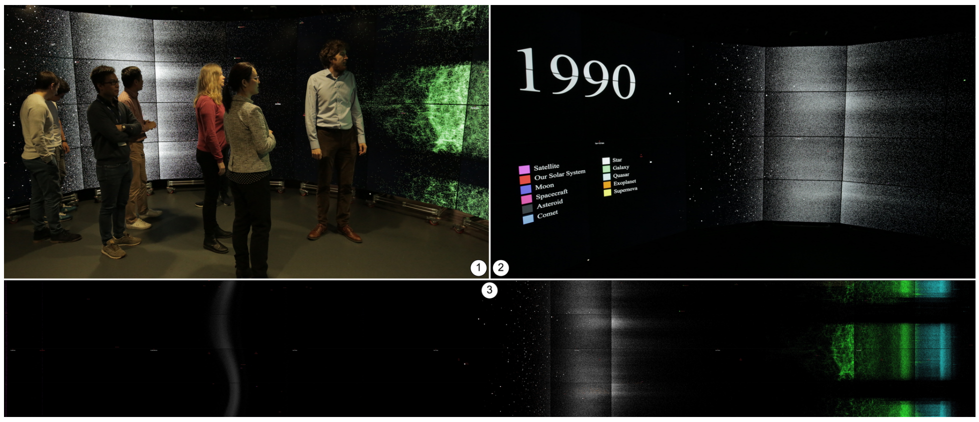

Figure 7: **The Map of the Universe.** ①: A group of people are discussing the distribution of galaxies. ②: A photograph of the animation displaying the evolution of our understanding of the universe from 1846 to 2019. The map in the year 1990, contains a majority of stars from the Milky Way but none of the exoplanets as they were not discovered by then. ③: The entire map as of December 2018, originally displayed at a resolution of `30720x4320`.

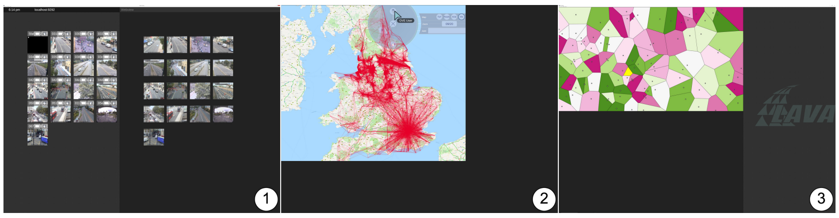

Figure 8: **Using OVE as a supplement to SAGE2.** In here we see three examples of the two frameworks working together. ① shows us how OVE can help resolve a limitation in SAGE2 and display 17 or more videos at a time. On the left we see SAGE2 failing to play the video on the top-left (which appears to be solid black). On the right we the same 17 videos loaded and playing fine on OVE running within a *Webview Container* of SAGE2 without being affected the same limitation. ② shows how the networks application of OVE has been used to overlay a network of cellular base stations above a Google Map rendered using SAGE2. Both the Google Map and the network of base stations are interactive visualizations; the figure shows us an example of using the SAGE2 controller. ③ shows an OVE instance embedded within SAGE2, displaying the *Voronoi Diagram* example from D3.js. In here, we provided OVE the URL of the original source (`https://strongriley.github.io/d3/ex/voronoi.html`) to simply display the SVG; without OVE, users require developing a SAGE2 application to display this. In ① and ③ OVE was embedded in SAGE2 and in ② SAGE2 was embedded in OVE, to show the two frameworks can complement each other.

collaborative data visualization exercises and interactive presentations. In this section we present two projects in which we used OVE: (i) The Map of the Universe and (ii) Polarization of public viewpoint on Brexit, based on tweets. Thereafter we present few examples on how OVE complements SAGE2.

6.1. The Map of the Universe

The dataset for this project was sourced from catalogs of astronomical objects such as the Sloan Digital Sky Survey (SDSS) [53] and the NASA/IPAC Extragalactic Database (NED) [54]. In terms of volume, SDSS data release 15 contains 1.2 billion catalog objects, and the NED May 2019 release contains nearly 5 billion photometric data points. While we have access to all of these objects on our databases, the visualization that we developed renders 0.1% of the dataset (around 6 million data points) as a multiclass scatterplot on a `30720x4320` pixel space, as seen in Figure 7. While it is possible to zoom and pan to see more objects, we limit the number of points displayed to avoid any undesirable overlapping. The total number of data points displayed at a time is proportional to the total number of pixels in the display area, and therefore, with twice as many screens (providing twice the resolution), the system would display twice as many points.

For each data point we also store information such as the object type, computed distance from Earth, right ascension, declination, absolute magnitude, apparent magnitude, date of discovery, and the object identifier (or name) on catalog. Therefore, we are able to not only display the points but also color them based on their type or resize them according to their apparent magnitudes. Once the points are displayed we can filter them interactively; filtering on the date of discovery is shown in Figure 7. An animated version of the Figure 7.2 displays the evolution of the universe known to man from 1846 (when the planet Neptune was discovered) to 2019.

The project team who provided us with this dataset had two key challenges, which we solved relatively easily with OVE. Firstly, they wanted to emphasize a few key celestial objects, some of which were much smaller than neighbouring objects, such as an important exoplanet among a densely packed group of stars, or a well known moon of the planet Jupiter. By controlling the colours and sizes of the objects, as well as the order in which they were plotted, we were able to prevent larger insignificant objects obstructing the visibility of smaller significant objects. Secondly, they wanted to detect anomalies in the dataset such as duplicates and incomplete entries. This was easily achieved by constructing a series of simple filter expressions. In their feedback, the project team noted that the re-

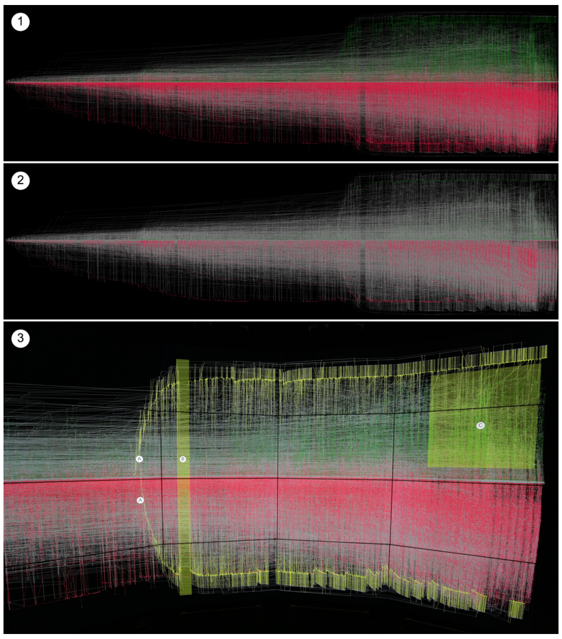

Figure 9: **Polarization of public viewpoint on Brexit.** The graphs ① and ② captured originally at a resolution of `15360x4320` show those who tweeted in support of UK leaving the EU in red, those who tweeted against the UK leaving the EU in green and those who did not express any support but simply retweeted in grey. These graphs exclude a wide majority of tweets from the same period that did not have anything to do with Brexit. The difference between graph ① and ② is that an edge connecting a tweet and a retweet is colored in the color corresponding to the tweet in graph ① but not in ②. This helps us visualize the dominance of original tweets in the leave-group compared to the remain-group. The photograph ③ shows the same graph from ① with some annotations: Ⓐ the presence of 2 bots (on either side) and their impact in changing as well as amplifying user opinion, Ⓑ a data quality issue where our provider failed to record tweets in a specific date range, and Ⓒ a trend where the remain-group had an overall drop of interest (excluding the bot) compared to the leave-group during the days just before the Brexit referendum.

sulting visualisation not only met their requirements but also exceeded expectations.

We do not find an equivalent of the networks application of OVE in other middleware. Even if we were to develop a new application on SAGE2, **L3** prevents us displaying 6 million data points as web browsers fail to cope with the data volume. Therefore, OVE was the best option for this visualization.

6.2. *Polarization of public viewpoint on Brexit, based on tweets*

In this experiment we analyzed close to 35 million tweets collected over a 6 month period from January 2016 to June 2016 to understand the polarization of the public viewpoint on the Brexit referendum. On average, around 100 billion tweets were made in this period and our dataset represents around 0.035% of the entire Twitter data volume.

OVE is only used at the latter stages of the visualization. The rest of the system rates these tweets on a scale from -1 to +1 and subsequently runs an offline layout algorithm (using Gephi [55]) based on this rating (polarization score) and the date on which the tweet was made. A score of -1 means that a tweet was strongly supporting the UK leaving the EU, while a score of +1 means that a tweet was strongly supporting the UK remaining in the EU. The scores were calculated using a Support Vector Machine classifier based on hashtags such as `#brexit`, `#voteremain`, `#strongerin`, `#leaveeu` and `#voteout`.

We excluded all tweets that mentioned nothing about Brexit. We also found that a few highly active users were producing a significant number of polarized tweets, skewing our results. Therefore, we randomly eliminated 95% of the tweets made by the top 1000 active users. Subsequently, OVE was used to render the resulting graph (see Figure 9). Each node corresponds to a tweet while each edge represents a relationship, which is either a retweet or a mention. Based on this experiment we were able to conclude that the public opinion on the UK leaving the EU became increasingly polarized during the referendum campaign (at least within the *Twittersphere*).

The annotation capabilities of OVE are superior to existing frameworks such as SAGE2. In this visualization, we used semi-transparent overlay annotations (a feature unique to OVE) to point out (a) the presence of bots as well as their impact on changing and amplifying user opinion, (b) a data quality issue where our provider failed to record tweets in a specific date range, as well as (c) an evolutionary change in trend, as seen in Figure 9.3.

In their qualitative feedback, the users highlighted one key advantage of using OVE. They were able to rapidly change the visibility and the colours of the nodes based on various filter criteria with no additional development effort, making it easy to isolate various clusters of Twitter users as the study progressed.

6.3. *Using OVE as a supplement to SAGE2*

OVE and SAGE2 are both web-based frameworks, and one key advantage (compared to non-web-based alternatives) is that they can embed (or be embedded in) other web-based frameworks. This gives users the opportunity to use OVE as a supplement to SAGE2 and resolve **L1**–**L3**. Embedding OVE within SAGE2 (or vice versa) does not require developing a special application. The combinations of these two frameworks enable many use-cases; we present three examples (Figure 8):

1. OVE provides an immediate resolution to feature gaps in SAGE2. For example, OVE can be used (within a *Webview Container*) to display 17 or more videos on SAGE2 or to play videos of `4K` and `8K` resolutions.

2. OVE can be used to create transparent overlays above existing SAGE2 applications, for example to plot a network of cellular base stations on a Google Map. We displayed the Google Map using an embedded SAGE2 environment (using the HTML application), and thereafter used the networks application of OVE to draw the overlay.

3. With OVE, users need not redevelop existing content based on popular JavaScript frameworks. Instead of developing new applications, an embedded OVE can be used display existing content (such as interactive/animated SVGs developed using D3.js) in a SAGE2 environment

Similarly, users can use a wide range of applications that are already available with SAGE2 (such as ParaSAGE [56]), and use an embedded instance of OVE to display high resolution videos, networks and D3.js based content.

7. Conclusion

This paper introduces OVE, a web framework for scalable rendering of data visualizations — an open-source[4] middleware that is hardware, platform and web browser independent. Our framework offers two key contributions; firstly, it is a highly performing, scalable and modular middleware framework that provides an ecosystem for data visualization in SRDEs. Secondly, it can be used as a plug-and-play supplement to the popular SAGE2 middleware for common data visualization use-cases; the combination of the two frameworks addresses key limitations in SAGE2.

We reviewed several frameworks for our requirements on data visualization on SRDEs. Frameworks based on OpenGL technology were technically superior but were not web-based and did not meet the expectations of our users. Among the web-based alternatives, many were (a) restricted in terms of their functionality or (b) not designed as general purpose frameworks or (c) purpose-built commercial software that did not suit the specifications of our SRDEs. Therefore, as explained in Section 2, SAGE2 was the best among the options that were available to us. This was not a particularly bad choice, as it has been deployed nearly a 100 times worldwide. But, SAGE2 has several limitations, and three of them (**L1**–**L3**) had a significant impact on our projects. Having read through the excellent survey by Ni et al [40], we found other unresolved (or partly-resolved) research challenges (**C1**–**C4**) that were relevant for middleware designed to visualize data in SRDEs. We developed OVE as a solution to these limitations and challenges.

The design of OVE was greatly influenced by that of SAGE2 and also one of our previous publications on a compositional microservices architecture for visual analytics [41]. It includes features for (a) visibility and occlusion culling, (b) clock synchronization, (c) asset management and (d) compositing visualizations. In Section 4 we explained the architecture of our framework, outlining its key components; namely, OVE core, applications, specialized services and UIs.

The highlight of this paper is our performance and scalability analysis. Using six tests (**T1**–**T6**) we proved that OVE is suitable for (a) simultaneously displaying hundreds of applications, (b) displaying ultra-high resolution content and (c) managing 1000 display tiles. Our analysis included a rigorous comparison of OVE and SAGE2 on both client-side and server-side. While pointing out limitations of SAGE2, we showed that OVE not only addresses those limitations but also has a highly performing and scalable microservices architecture.

Lastly, in Section 6 we presented two projects that used OVE and three examples of how OVE can supplement SAGE2. As noted previously, the objective of OVE is not to outperform SAGE2, but to provide an extension or an alternative for users who are interested in visualizing data on SRDEs. OVE is an open-source[4] middleware that is available under a permissive MIT license. It is available as a portable multi-container docker application that can be deployed on many operating systems. Installing OVE is straightforward and the documentation includes tutorials to guide users on how to use it for popular data visualization use-cases. We invite readers to download and install OVE on their SRDEs and use it as a platform to compose new data visualizations, display content that already exist, or enhance projects that have been developed using SAGE2.

[4] `https://ove.readthedocs.io`

Acknowledgments

The authors wish to thank Roberto Trotta and Chris Snowden for their contributions to the experiments. M. Molina-Solana was funded by the EU's H2020 programme (MSCA g.a. No 743623 and No 754446).

References

[1] J. Heer, M. Agrawala, Design considerations for collaborative visual analytics, Information visualization 7 (1) (2008) 49–62. `doi:10.1057/palgrave.ivs.9500167`.

[2] T. Dwyer, K. Marriott, T. Isenberg, K. Klein, N. Riche, F. Schreiber, W. Stuerzlinger, B. H. Thomas, Immersive analytics: An introduction, in: Immersive Analytics, Springer, Cham, Switzerland, 2018, pp. 1–23. `doi:10.1007/978-3-030-01388-2_1`.

[3] A. Barredo-Arrieta, N. Díaz-Rodríguez, J. D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI, Information Fusion 58 (2020) 82–115. `doi:10.1016/j.inffus.2019.12.012`.

[4] A. Holzinger, P. Kieseberg, E. Weippl, A. M. Tjoa, Current advances, trends and challenges of machine learning and knowledge extraction: From machine learning to explainable AI, in: A. Holzinger, P. Kieseberg, A. M. Tjoa, E. Weippl (Eds.), Machine Learning and Knowledge Extraction, Springer, Cham, Switzerland, 2018, pp. 1–8. `doi:10.1007/978-3-319-99740-7_1`.

[5] M. Molina-Solana, D. Birch, Y. Guo, Improving data exploration in graphs with fuzzy logic and large-scale visualisation, Applied Soft Computing 53 (2017) 227–235. `doi:10.1016/j.asoc.2016.12.044`.

[6] S. Fernando, J. Amador, O. Şerban, J. Gómez-Romero, M. Molina-Solana, Y. Guo, Towards a large-scale Twitter observatory for political events, Future Generation Computer Systems`doi:10.1016/j.future.2019.10.013`.
URL `http://www.sciencedirect.com/science/article/pii/S0167739X19309720`

[7] L. Renambot, T. Marrinan, J. Aurisano, A. Nishimoto, V. Mateevitsi, K. Bharadwaj, L. Long, A. Johnson, M. Brown, J. Leigh, SAGE2: A collaboration portal for scalable resolution displays, Future Generation Computer Systems 54 (2016) 296–305. `doi:10.1016/j.future.2015.05.014`.

[8] T. Marrinan, J. Leigh, L. Renambot, A. Forbes, S. Jones, A. E. Johnson, Mixed presence collaboration using scalable visualizations in heterogeneous display spaces, in: Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing, ACM, 2017, pp. 2236–2245. `doi:10.1145/2998181.2998346`.

[9] T. Marrinan, J. Aurisano, A. Nishimoto, K. Bharadwaj, V. Mateevitsi, L. Renambot, L. Long, A. Johnson, J. Leigh, SAGE2: A new approach for data intensive collaboration using Scalable Resolution Shared Displays, in: Proceedings of the 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, IEEE, Miami, FL, USA, 2014, pp. 177–186. `doi:10.4108/icst.collaboratecom.2014.257337`.

[10] M. Bostock, V. Ogievetsky, J. Heer, D^3 Data-Driven Documents, IEEE Transactions on visualization and computer graphics 17 (12) (2011) 2301–2309. `doi:10.1109/TVCG.2011.185`.

[11] R. Cabello, et al., Three.js, https://github.com/mrdoob/three.js, [Online; accessed 17-September-2019].

[12] K. Reda, A. Febretti, A. Knoll, J. Aurisano, J. Leigh, A. Johnson, M. E. Papka, M. Hereld, Visualizing large, heterogeneous data in hybrid-reality environments, IEEE Computer Graphics and Applications 33 (4) (2013) 38–48. doi:10.1109/MCG.2013.37.

[13] C. Andrews, A. Endert, B. Yost, C. North, Information visualization on large, high-resolution displays: Issues, challenges, and opportunities, Information Visualization 10 (4) (2011) 341–355. doi:10.1177/1473871611415997.

[14] C. Andrews, A. Endert, C. North, Space to think: Large high-resolution displays for sensemaking, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '10, ACM, New York, NY, USA, 2010, pp. 55–64. doi:10.1145/1753326.1753336.

[15] X. Bi, R. Balakrishnan, Comparing usage of a large high-resolution display to single or dual desktop displays for daily work, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '09, ACM, New York, NY, USA, 2009, pp. 1005–1014. doi:10.1145/1518701.1518855.

[16] B. Yost, Y. Haciahmetoglu, C. North, C. North, Beyond visual acuity: The perceptual scalability of information visualizations for large displays, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07, ACM, New York, NY, USA, 2007, pp. 101–110. doi:10.1145/1240624.1240639.

[17] R. Ball, C. North, Analysis of user behavior on high-resolution tiled displays, in: M. F. Costabile, F. Paternò (Eds.), Human-Computer Interaction - INTERACT 2005, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 350–363. doi:10.1007/11555261_30.

[18] C. Rooney, R. A. Ruddle, HiReD: a high-resolution multi-window visualisation environment for cluster-driven displays, in: Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, ACM, Duisburg, Germany, 2015, pp. 2–11. doi:10.1145/2774225.2774850.

[19] S. Su, V. Perry, N. Cantner, D. Kobayashi, J. Leigh, High-resolution interactive and collaborative data visualization framework for large-scale data analysis, in: Proceedings of the 2016 International Conference on Collaboration Technologies and Systems (CTS), IEEE, Orlando, FL, USA, 2016, pp. 275–280. doi:10.1109/CTS.2016.0059.

[20] S. Eilemann, M. Makhinya, R. Pajarola, Equalizer: A scalable parallel rendering framework, IEEE Transactions on Visualization and Computer Graphics 15 (3) (2009) 436–452. doi:10.1109/TVCG.2008.104.

[21] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, J. T. Klosowski, Chromium: a stream-processing framework for interactive rendering on clusters, ACM Transactions on Graphics (TOG) 21 (3) (2002) 693–702. doi:10.1145/566654.566639.

[22] G. P. Johnson, G. D. Abram, B. Westing, P. Navrátil, K. Gaither, DisplayCluster: An interactive visualization environment for tiled displays, in: Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER), IEEE, 2012, pp. 239–247. doi:10.1109/CLUSTER.2012.78.

[23] K. Doerr, F. Kuester, CGLX: a scalable, high-performance visualization framework for networked display environments, IEEE Transactions on visualization and computer graphics 17 (3) (2011) 320–332. doi:10.1109/TVCG.2010.59.

[24] M. Rittenbruch, CubIT: large-scale multi-user presentation and collaboration, in: Proceedings of the 2013 ACM international conference on Interactive tabletops and surfaces, ACM, New York, NY, USA, 2013, pp. 441–444. doi:10.1145/2512349.2514923.

[25] M. Klapperstuck, T. Czauderna, C. Goncu, J. Glowacki, T. Dwyer, F. Schreiber, K. Marriott, ContextuWall: peer collaboration using (large) displays, in: 2016 Big Data Visual Analytics (BDVA), IEEE, Piscataway, NJ, USA, 2016, pp. 1–8. doi:10.1109/BDVA.2016.7787047.

[26] MultiTaction, Canvus collaboration software from MultiTaction - see the big picture!, https://www.multitaction.com/product/canvus-collaboration-software/, [Online; accessed 17-September-2019] (2019).

[27] L. Renambot, A. Rao, R. Singh, B. Jeong, N. Krishnaprasad, V. Vishwanath, V. Chandrasekhar, N. Schwarz, A. Spale, C. Zhang, et al., SAGE: the Scalable Adaptive Graphics Environment, in: Proceedings of the 4th Workshop on Advanced Collaborative Environments (WACE), Vol. 9, 2004, pp. 2004–2009.

[28] M. Bostock, J. Heer, Protovis: A graphical toolkit for visualization, IEEE transactions on visualization and computer graphics 15 (6) (2009) 1121–1128. doi:10.1109/TVCG.2009.174.

[29] A. S. Perez, OpenLayers cookbook, Packt Publishing Ltd, Birmingham, UK, 2012.

[30] P. Crickard III, Leaflet.js essentials, Packt Publishing Ltd, Birmingham, UK, 2014.

[31] A. Jacomy, G. Plique, Sigma.js, http://sigmajs.org/, [Online; accessed 17-September-2019].

[32] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, J. Heer, Vega-Lite: A grammar of interactive graphics, IEEE transactions on visualization and computer graphics 23 (1) (2016) 341–350. doi:10.1109/TVCG.2016.2599030.

[33] A. Febretti, A. Nishimoto, T. Thigpen, J. Talandis, L. Long, J. Pirtle, T. Peterka, A. Verlo, M. Brown, D. Plepys, et al., CAVE2: a hybrid reality environment for immersive simulation and information analysis, in: IS&T/SPIE Electronic Imaging, SPIE, Bellingham, WA, USA, 2013, pp. 864903–864903. doi:10.1117/12.2005484.

[34] N. Fabian, K. Moreland, D. Thompson, A. C. Bauer, P. Marion, B. Gevecik, M. Rasquin, K. E. Jansen, The ParaView coprocessing library: A scalable, general purpose in situ visualization library, in: 2011 IEEE symposium on large data analysis and visualization, IEEE, Providence, RI, USA, 2011, pp. 89–96. doi:10.1109/LDAV.2011.6092322.

[35] N. Harshfield, D.-j. Chang, et al., A Unity 3D framework for algorithm animation, in: Proceedings of the 20th International Conference on Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES), IEEE, Louisville, KY, USA, 2015, pp. 50–56. doi:10.1109/CGames.2015.7272955.

[36] S. Marks, J. E. Estevez, A. M. Connor, Towards the Holodeck: Fully immersive virtual reality visualisation of scientific and engineering data, in: Proceedings of the 29th International Conference on Image and Vision Computing New Zealand, IVCNZ '14, ACM, New York, NY, USA, 2014, pp. 42–47. doi:10.1145/2683405.2683424.

[37] V. Kalivarapu, A. MacAllister, M. Hoover, S. Sridhar, J. Schlueter, A. Civitate, P. Thompkins, J. Smith, J. Hoyle, J. Oliver, E. Winer, G. Chernoff, Game-day football visualization experience on dissimilar virtual reality platforms, in: M. Dolinsky, I. E. McDowall (Eds.), The Engineering Reality of Virtual Reality 2015, Vol. 9392, International Society for Optics and Photonics, SPIE, 2015, pp. 1 – 14. doi:10.1117/12.2083250.

[38] R. Sicat, J. Li, J. Choi, M. Cordeil, W. Jeong, B. Bach, H. Pfister, DXR: A toolkit for building immersive data visualizations, IEEE transactions on visualization and computer graphics 25 (1) (2018) 715–725. doi:10.1109/TVCG.2018.2865152.

[39] M. Cordeil, A. Cunningham, B. Bach, C. Hurter, B. H. Thomas, K. Marriott, T. Dwyer, IATK: An immersive analytics toolkit, in: Proceedings of the 26th IEEE Conference on Virtual Reality and 3D User Interfaces (VR), IEEE, Osaka, Japan, 2019, pp. 200–209. doi:10.1109/VR.2019.8797978.

[40] T. Ni, G. S. Schmidt, O. G. Staadt, M. A. Livingston, R. Ball, R. May, A survey of large high-resolution display technologies, techniques, and applications, in: Proceedings of the IEEE Virtual Reality Conference (VR 2006), IEEE, Piscataway, NJ, USA, 2006, pp. 223–236. doi:10.1109/VR.2006.20.

[41] S. Fernando, D. Birch, M. Molina-Solana, D. Mcilwraith, Y. Guo, Compositional microservices for immersive social visual analytics, in: Proceedings of the 23rd International Conference Information Visualisation (IV), IEEE, Paris, France, 2019, pp. 216–223. doi:10.1109/IV.2019.00044.

[42] R. Gusella, S. Zatti, The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD, IEEE transactions on Software Engineering 15 (7) (1989) 847–853. doi:10.1109/32.29484.

[43] V. Martínez, S. Fernando, M. Molina-Solana, Y. Guo, Tuoris, https://github.com/fvictor/tuoris, [Online; accessed 17-September-2019].

[44] OpenSeadragon contributors, OpenSeadragon, https://openseadragon.github.io/, [Online; accessed 17-September-2019].

[45] J. Simpson, et al., howler.js, https://howlerjs.com/, [Online; accessed 17-September-2019].

[46] Mozilla, et al., PDF.js, https://mozilla.github.io/pdf.js/, [Online; accessed 17-September-2019].

[47] Google, YouTube Player API Reference for iFrame Embeds, https://

developers.google.com/youtube/iframe_api_reference, [Online; accessed 17-September-2019].

[48] J. Fletcher, Marketing for the QRious: the beginner's guide to using QR codes for library promotions and resources, Multimedia Information and Technology 36 (3) (2010) 26–27.
URL http://irep.ntu.ac.uk/id/eprint/8617/1/198471_MmITLandscapeLayoutAug10-jonQRcodes.pdf

[49] B. García, F. Gortázar, M. Gallego, E. Jiménez, User impersonation as a service in end-to-end testing, in: Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development - Volume 1: AMARETTO, INSTICC, SciTePress, 2018, pp. 707–714. doi:10.5220/0006752207070714.

[50] D. Gonzalez, Developing Microservices with Node.js, Packt Publishing Ltd, Birmingham, UK, 2016.

[51] D. McGinn, D. Birch, D. Akroyd, M. Molina-Solana, Y. Guo, W. Knottenbelt, Visualizing dynamic Bitcoin transaction patterns, Big Data 4 (2) (2016) 109–119. doi:10.1089/big.2015.0056.

[52] C. Papadopoulos, K. Petkov, A. E. Kaufman, K. Mueller, The Reality Deck–an immersive gigapixel display, IEEE computer graphics and applications 35 (1) (2014) 33–45. doi:10.1109/MCG.2014.80.

[53] D. Sánchez Aguado, R. Ahumada, A. Almeida, S. F. Anderson, B. H. Andrews, B. Anguiano, E. A. Ortiz, A. Aragón-Salamanca, M. Argudo-Fernández, M. Aubert, et al., The fifteenth data release of the Sloan Digital Sky Surveys: first release of MaNGA-derived quantities, data visualization tools, and stellar library, The Astrophysical Journal Supplement Series 240 (2) (2019) 23. doi:10.3847/1538-4365/aaf651.

[54] J. M. Mazzarella, N. Team, et al., Evolution of the NASA/IPAC Extragalactic Database (NED) into a data mining discovery engine, Proceedings of the International Astronomical Union 12 (S325) (2016) 379–384. doi:10.1017/S1743921316013132.

[55] M. Bastian, S. Heymann, M. Jacomy, Gephi: An open source software for exploring and manipulating networks, in: Proceedings of the third international AAAI conference on weblogs and social media, AAAI, Menlo Park, CA, USA, 2009, pp. 361–362. doi:10.1136/qshc.2004.010033.

[56] S. Su, V. Perry, N. Cantner, D. Kobayashi, J. Leigh, High-resolution interactive and collaborative data visualization framework for large-scale data analysis, in: 2016 International Conference on Collaboration Technologies and Systems (CTS), IEEE, Orlando, FL, USA, 2016, pp. 275–280. doi:10.1109/CTS.2016.0059.