VRIA: A Web-based Framework for Creating Immersive Analytics Experiences

Butcher, Peter; John, Nigel W.; Ritsos, Panagiotis D.

IEEE Transactions on visualization and computer graphics

DOI:
10.1109/TVCG.2020.2965109

Published: 01/07/2021

VRIA: A Web-based Framework for Creating Immersive Analytics Experiences

Peter W. S. Butcher, Nigel W. John, and Panagiotis D. Ritsos, *Member, IEEE*

Abstract—We present <VRIA>, a Web-based framework for creating Immersive Analytics (IA) experiences in Virtual Reality. <VRIA> is built upon WebVR, A-Frame, React and D3.js, and offers a visualization creation workflow which enables users, of different levels of expertise, to rapidly develop Immersive Analytics experiences for the Web. The use of these open-standards Web-based technologies allows us to implement VR experiences in a browser and offers strong synergies with popular visualization libraries, through the HTML Document Object Model (DOM). This makes <VRIA> ubiquitous and platform-independent. Moreover, by using WebVR's progressive enhancement, the experiences <VRIA> creates are accessible on a plethora of devices. We elaborate on our motivation for focusing on open-standards Web technologies, present the <VRIA> creation workflow and detail the underlying mechanics of our framework. We also report on techniques and optimizations necessary for implementing Immersive Analytics experiences on the Web, discuss scalability implications of our framework, and present a series of use case applications to demonstrate the various features of <VRIA>. Finally, we discuss current limitations of our framework, the lessons learned from its development, and outline further extensions.

Index Terms—Immersive Analytics, Virtual Reality, Web technologies.

1 Introduction

Immersive Analytics (IA) is an emerging research theme that builds on the recent evolution of computer interfaces, visualization and data science. IA seeks to investigate the use of novel display and interface technologies in analytical reasoning and decision making [1], with an ultimate goal to develop multi-sensory, collaborative, interactive systems that allow users to be immersed in their data. In this paper we explore the paradigm of Virtual Reality (VR) as a medium for creating IA experiences for the visualization of abstract data. Despite the fact that VR has been around for decades, the recent emergence of affordable, commercially available head-mounted-displays (HMDs), such as the HTC Vive [2] and Oculus Rift [3], has reinvigorated the interest in all things VR.

Following our early preliminary investigations [4], [5], [6], we were motivated to create a framework that would enable developers of any expertise to create consumable, interactive, immersive data visualizations using standards-based Web technologies. Addressing this challenge, we present our Web-based IA framework, <VRIA>, designed to facilitate the creation of IA experiences in VR (Figure 1), using standards-based Web technologies, and via a visualization creation workflow suitable for developers of all expertise levels. <VRIA>'s ultimate goal is to enable researchers to build and share IA experiences for the open Web, as consumable web components, that can be experienced through a plethora of VR capable devices, with a single code-base, in a similar manner to the contemporary, responsive Web.

Our motivation for creating <VRIA> as a Web-based tool, and with Web-based output, is based on the notion that the Web is the most ubiquitous, collaborative and platform-independent way to build and share information [7]. By creating <VRIA> as a native Web service, our intention is to democratize the development of IA solutions, building upon recent standards in the domain of the immersive Web. In the next section we elaborate on this motivation in more detail, and highlight the advantages of our approach.

The <VRIA> framework described in this paper significantly extends preliminary investigations that discussed early, now superseded, versions of our framework [4], [5], [6], [8]. The current incarnation of <VRIA> presented in this work is suitable for novice and expert users with varied programming and visualization experience, has an updated architecture and capabilities, a new visualization creation workflow and a Web-based building tool.

In this regard, we discuss <VRIA> into four main strands, corresponding to our main contributions from this paper:

i) A description of <VRIA>'s visualization creation workflow (Section 4), emphasizing usage patterns for novice, intermediate and expert users, along with the corresponding tools that they can utilize.
ii) A detailed account of <VRIA>'s underlying implementation and features, emphasizing architectural choices, interaction mechanisms, performance optimizations and scalability implications (Section 5).
iii) A comprehensive set of use cases demonstrating the various features and capabilities of <VRIA> (Section 6).
iv) A discussion on current limitations of <VRIA>, and similar Web-based systems, and how these can be addressed in future work (Section 7). We also discuss lessons learned from the design and development of our framework (Section 8).

We also elaborate on our motivation (Section 2) to use Web technologies and briefly discuss the current state-of-the-art in creating VR experiences for the Web. In addition, we present prior visualization works that have inspired the development of <VRIA> (Section 3), focusing in particular on a comparison with frameworks and toolkits that facilitate the creation of IA tools and demonstrations.

• *P.W.S. Butcher and N.W. John are with the Department of Computer Science, Chester University.*
E-mail: p.butcher, n.john@chester.ac.uk.
• *P.D. Ritsos is the School of Computer Science and Electronic Engineering, Bangor University.*
E-mail: p.ritsos@bangor.ac.uk.

2 Motivation and Background

The Web can be the most platform-independent way to build and share visualizations, thus achieving ubiquity and strong support for collaboration [7]. As Roher and Swing [9] highlight, *'the Web provides a flexible means for linking applications, data, information, and users'*. It is, therefore, no surprise that the most popular technologies for data visualization nowadays are from the Web technologies ecosystem and are essentially based on open-standards. Libraries and tools such as D3.js [10], Processing.js, Protovis [11] and Vega [12], [13] have been shaping the landscape of contemporary data visualization tools.

More importantly, these tools allow the production of interactive data visualizations as easily shareable, reusable and *consumable* components, that can be integrated with other Web-based tools, through the Document Object Model (DOM). The DOM provides a common abstraction layer, comprising a set of intrinsic object functions and their properties, such as a drawing space (e.g., canvas), data structures (e.g., JSON), formatting (CSS), or more exotic features such as real-time communications (WebRTC), geolocation (Geolocation API) or WebVR [14]. Alongside the DOM, the Web ecosystem provides an execution environment, JavaScript, that allows these features to interact with each other and users. As all Web-based applications are built on top of this abstraction layer, they share a common language, and can be consumed by each other, or other tools. Consumption in this case is not limited to merely re-displaying or extracting base information, but includes modification and interoperation across tools and services. Contrary to VR applications created with a game engine or even their WebGL export, an application built for the Web (and not on the Web) is truly consumable and can be loosely coupled with other systems. This approach is not new, as tools such as Webstrates [15] and Vistrates [16] work on the same principles. We explore the influence of those tools on <VRIA>, in Section 3.

The WebVR API [14], [17], and its successor WebXR [18], provide the link between VR and the DOM via Javascript. WebVR is an open specification API that has been gaining support with browser vendors and allows developers to build VR, and soon Mixed Reality (MR), applications that can be experienced in a Web browser, and make use of the HTML DOM, much like the aforementioned visualization libraries. Moreover, WebVR offers a level of platform independence that extends beyond the browser, as it can detect the available display hardware and corresponding handheld controllers, tailoring VR experiences to the user's current hardware set-up (a feature called *progressive enhancement* [14]). This way, with a single code-base, a VR scene can be experienced via a standards-compliant browser in most contemporary HMDs, conventional monitors and smartphones. [17] WebVR is currently under active development and will eventually be replaced by the WebXR Device API, which will also incorporate MR capabilities [18]. VR experiences built with WebVR, such as the ones produced with <VRIA>, can be reused as consumable components from a variety of systems and services in the Web-ecosystem. This can consequently result in the democratization of the development of IA systems and enable more in-depth and varied investigations of immersive data visualization.

Overall, <VRIA> has the following high level features, derived from prior-work influences, discussed in the next section:

i) <VRIA> is built entirely on open-standards Web technologies.
ii) Achieves platform independence through the use of Web technologies and WebVR's progressive enhancement features.
iii) Employs a declarative grammar that resembles Vega-lite and makes our toolkit more accessible and compatible with other visualization tools.
iv) Produces shareable, distributable and consumable immersive visualization outputs that can be easily integrated into other applications, and accessed via the HTML DOM.
v) Provides alternative development paths through: a) an optional, end-to-end visualization creation tool called the <VRIA> Builder that comes packaged with <VRIA> and is suitable for users with no prior programming experience, or b) via a programming API suitable for seasoned developers.

3 Related Work

Although the term 'Immersive Analytics' was only recently coined by Chandler et al. [19], many aspects of beyond-the-desktop visualizations [7] have been previously explored. For example, Lee et al. [20] discuss post-WIMP (Window-Icon-Mouse-Pointer) interactions; Elmqvist et al. [21] explore fluid, natural interactions for information visualization; Elmqvist and Pourang [22] describe the notion of ubiquitous analytics; Jansen and Dragicevic [23] propose an interaction model for beyond-the desktop; and Willet et al. [24] discuss situated and embedded data representations.

Amongst this multi-flavored research thrust, VR has a prominent place as one of the earliest paradigms for beyond-the-desktop visualization (e.g, [25], [26], [27]). In 2000, Van Dam et al. [25] presented a research agenda that included a call to action on how VR could be an effective medium for scientific visualization. Along with the ever-present challenges of display technologies, rendering performance, collaboration facilitation and interaction, Van Dam et al. highlighted a major obstacle being the lack of standardization that enables interoperability for the development of VR systems. Their observation, which we believe still applies to all genres of visualization, is the root of our aforementioned motivations, albeit in a manner that seeks to build specifically upon recent advancements of Web technologies [28].

3.1 Immersive Visualization in VR/MR

Aided by the recent availability of new and affordable immersive interfaces, a large number of efforts have emerged over the last decade, both in VR and MR. Examples include investigation on the use of CAVEs, for example in visualizing tensor-valued volumetric datasets [29], the comparison of CAVEs with HMD-based solutions in immersive network visualization [30], collaborative immersive worlds that can handle complex data [31] and the immersive display of 3D trajectories, such as for air traffic control [32]. Such use-cases demonstrate the potential of VR/MR in analytical settings, and serve as inspiration of what experiences <VRIA> can and will (see future work in Section 8) offer.

Of notable importance to our motivation is the work of Drouhard et al. [33] who combine VR and data analytics in materials science, discussing challenges as safety, cybersickness, interactivity, data computation, feedback elicitation and user behavior acquisition. Importantly, they highlight the requirement for rigorous research evaluations so that this incarnation of VR is not 'wasted' as previous ones have been. <VRIA>'s ultimate goal is to offer a platform upon which investigations on these aspects can be more accessible though the Web, through interoperability with tools assessing such issues. For example, with <VRIA> it is

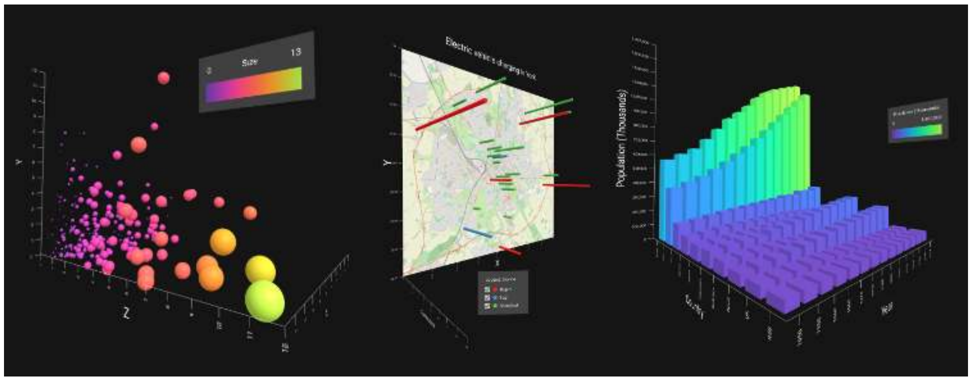


Fig. 1. <VRIA> enables the creation of interactive visualization experiences, ranging from simple bar charts and scatter plots to complex linked and composite designs. The left image shows a 3D bubble chart demonstrating how data can be mapped to several encoding channels, including x, y and z as well as size, color etc. The middle image shows a composite, situated visualization of electric vehicle charging points in York, UK in 2017 where the height of the cylinders, superimposed on a map, denote the number of charging points per location, and the color denotes charge speed. The right image is a 3D bar chart showing the populations of the top-ten most populated countries since 1950.

easy to build in-VR questionnaire-based and user-action recording evaluations using simple Web tools, as demonstrated previously [6].

Approaches that use MR, or its subset Augmented Reality (AR), [34], [35], [36], [37] are also of interest, as many of the underlying technologies of <VRIA> can work on the MR domain [38] as well. In fact, <VRIA> does offer support for basic investigations in MR, using Web-based MR libraries such as AR.js (see Section 7). The common theme of these aforementioned efforts is that they explore novel ways to interact with data, often seeking to overhaul traditional VR/AR [36] and visualization [34] approaches within the scope of data visualization. We believe that by creating <VRIA>, we provide a framework upon which researchers can build similar, multifaceted investigations on the Web, by combining different peripherals, libraries and systems over the unified abstraction layer that the HTML DOM offers.

3.2 Immersive Visualization Toolkits

In addition to the influences described in the previous section, <VRIA> draws inspiration from toolkits which focus specifically on creating data visualizations in VR and MR, developed in the last couple of years. We provide a comparison of <VRIA> to those tools, in Table 1, highlighting our framework's high-level feature parity, albeit using Web technologies natively and in its entirety. In that regard, reliance on game engines is traded for reliance on open standards, as envisioned by Van Dam et al. [25].

Cordeil et al. [39] introduce an immersive system built in Unity called ImAxes, which has evolved to IATK [40], for exploring multivariate data using virtual axes that can be arranged and combined in virtual space, and viewed on VR and AR displays. IATK utilizes an expressive grammar supporting 1D, 2D and 3D orthogonal plots. Visualizations are created via the Unity GUI and/or a low-level API allowing for custom visualizations and interactions. DXR [41] is another Unity-based visualization toolkit for rapidly prototyping data visualizations for VR/AR applications. It uses a declarative grammar, similar to that of Vega-Lite. DXR supports custom visual marks, rendering data points as Unity game objects. This results in lower performance compared to IATK where data points are rendered in a single mesh object, allowing computationally expensive tasks to be performed on the GPU instead of the CPU. For both toolkits, exports can be used on the Web via the WebAssembly (Wasm) WebGL export. <VRIA> has a similar philosophy to these toolkits, with regard to the usage features provided to the user, including a declarative grammar, linked-views support, and custom mark specification via A-Frame. However, due to its open-standards Web-based implementation, <VRIA> offers genuine platform independence and a vendor-agnostic development ecosystem, usage versatility as it can be used without licensing implications, and shareable, distributable and consumable output without Wasm.

Similar Web-based toolkits have also been developed. VR-Viz is a Web-based visualization library built with A-Frame and React which uses a grammar approach based on that of Vega-Lite [13]. However, it is relatively limited in versatility, as the visualizations are 'baked' into the renderer mechanisms and are effectively stand-alone depictions. Moreover, our work has significant differences to VR-Viz in terms of performance optimizations, usability and the visualization creation workflow (see subsection 5.2). Stardust [42], a GPU-based visualization library, harnesses the power of WebGL to perform visualization rendering, and offers an alternative to HTML5 Canvas, D3, and Vega for large numbers of graphical marks. Startdust shares similarities with our work, such as a declarative data model, web-technologies and WebVR support. However it leaves mechanisms such as visualization linking, VR interactions to other tools, and hence is not included in our comparison table. <VRIA>, on the other hand, provides a holistic framework for building consumable IA experiences on the Web, as VR spaces, with support for a variety of interface devices.

TABLE 1
Comparison: Immersive Analytics Toolkits

Feature	IATK	DXR	VR-Viz	<VRIA>
Platform	Unity	Unity	Web	Web
Displays	VR/AR	VR/AR	VR	VR/AR
Dimensionality	1/2/3D	1/2/3D	3D	1/2/3D
Interactions	✔	✔	✔	✔
Linked Views	✔	✔		✔
Custom Marks	✔	✔		✔
Code Playground	◇	◇		✔
GUI	✔	✔		✔
Declarative Config$^{(1)}$		✔	✔	✔
API	✔			✔

◇ other software is responsible for providing that feature

(1) the `vis config`, <VRIA>'s declarative configuration, is described in subsection 5.3

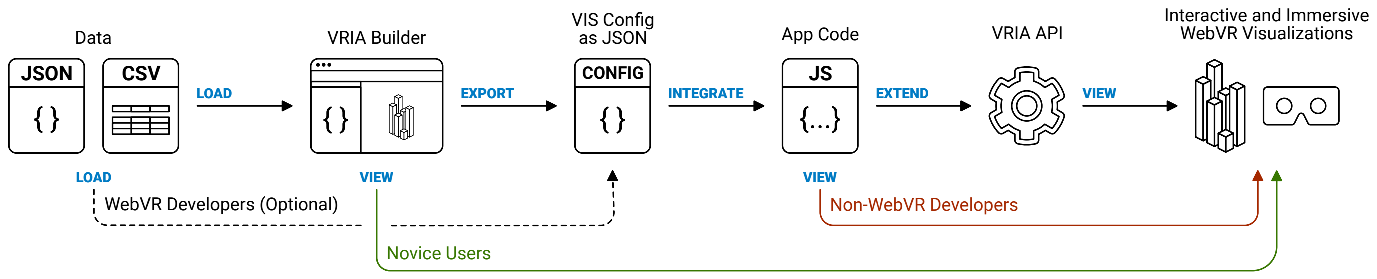


Fig. 2. Visualization creation workflow demonstrating how <VRIA> is suitable for users with different levels of programming experience. Novice users can upload datasets to the <VRIA> builder tool and quickly produce immersive visualizations without coding. Intermediate users can use the builder to create a visualization configuration file or write one from scratch to use in their application. Finally, advanced users can make use of <VRIA>'s API to develop additional features (see subsection 5.4).

3.3 Influences of Information Visualization Toolkits

In addition to the aforementioned influences, we have built <VRIA> to provide support for a number of established mechanisms and contemporary approaches that facilitate the creation of visualizations. The importance of this approach is twofold. First and foremost, <VRIA> seeks to integrate good practices from the visualization domain, and become a genuinely enabling technology for visualization researchers who are used to said techniques on non-immersive, Web-based tools and systems. Secondly, by doing so, <VRIA> is purposefully built for Immersive Analytics and can therefore produce more streamlined output, rather than being an add-on to a game engine that has a wider scope and purpose.

For example, we separate a visualization's specification from its implementation using a declarative grammar based on Vega-Lite [13], motivated by the latter's wide adoption, conciseness and expressiveness. We highlight commonalities and differences between our grammar and Vega-lite in subsection 5.2. We also provide building blocks for creating custom visualization designs, influenced by the DOM and tools such as Prefuse [43] and in a broader sense by the InfoVis toolkit [44]. This allows developers to rapidly prototype (Section 4) custom visualizations and interactions via simple configuration files, with A-Frame and React components, as well as by using <VRIA>'s declarative API (subsection 5.2). Finally, we integrate D3.js [10], which provides powerful data transformation and manipulation mechanisms, and is a library many visualization developers are familiar with.

More importantly, beyond the aforementioned operational similarities with visualization tools, <VRIA>'s philosophy has strong similarities to Webstrates. Webstrates is based on the notion of components, shareable among many users, distributable across many devices and maleable by users. <VRIA> has a similar character, in terms of the created outputs, which we deem as shareable, distributable and consumable. We prefer the term consumable as it emphasizes the reuse of <VRIA>'s exports in other contexts. However, <VRIA> does not have, or define the underlying infrastructure that these tools have. In that regard, the similarities are due to the use of Web technologies, and the DOM as the underlying and unifying mechanisms, rather than by dedicated architectural design. Nonetheless, being part of the Web ecosystem, we envision scenarios where <VRIA>'s components could be used along with, or consumed by, like-minded tools such as Vistrates [16]. In many ways, both approaches converge on Mackay's [45] vision of augmenting the environment through interactive, networked objects, which in many ways extends more traditional definitions of AR [46] by Milgram and Kishino [47], and Azuma [48]. We believe that by building VR-based interfaces with this ultimate vision in mind, the transition to MR-capable tools will be easier. This is also why with <VRIA> we have already explored the integration of basic AR capabilities, facilitated by the HTML DOM and popular AR-enabling JavaScript libraries (see Section 7).

4 Visualization Creation Workflow

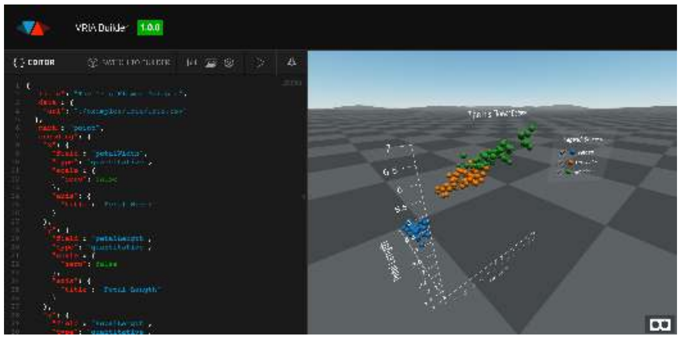
We have designed <VRIA> to be accessible to novices and experts alike by providing different, yet complementary, creation workflows. Figure 2 outlines how users of different levels of expertise and abilities in programming visualizations can utilize <VRIA>. The starting assumption is, naturally, that users have a dataset they want to visualize in an immersive manner. <VRIA> currently supports the visualization of tabular data in JSON or CSV format. From this stage of the <VRIA> workflow, users can follow different development paths to generate interactive and immersive WebVR visualizations, based on their expertise.

For example, users with some programming ability might choose to use the builder to generate a <VRIA> visualization configuration file which they can then use in their WebVR application code. Likewise, users with expertise with grammar-based visualization tools might choose to port an existing Vega-Lite or other similar configuration file into the <VRIA> grammar, test and customize it in the builder, and then integrate it in their application code. Finally, users with previous WebVR development experience can harness <VRIA>'s API to develop new visualizations and interactions.

4.1 The <VRIA> Builder (for beginners)

The <VRIA> Builder (see Figure 3), is aimed mainly at non-programmers and is a Web application available on the project's website. It also comes packaged with <VRIA>'s NodeJS module. It is essentially intended to be used as a tool for learning the grammar, as well as a demonstration of the visualizations that <VRIA> is capable of producing. Furthermore it allows users to conveniently prototype and test immersive experiences without leaving the browser. In this regard, it can also facilitate rapid prototyping and iterative development of visualization designs for power users as well. This frictionless ability to rapidly prototype visualization designs and get instant feedback makes <VRIA> accessible to a wider audience. Last but not least, the Builder is an example of <VRIA>'s consumable nature, as it is essentially a GUI-wrapper for customizing <VRIA>'s scenes.

The <VRIA> builder enables designers to build-up a visualization configuration, iteratively customize it and immediately



```
{
  [illegible]
  "data": {
    "url": [illegible]
  },
  "mark": "point",
  "encoding": {
    "x": {
      "field": "petalWidth",
      "type": "quantitative",
      "scale": {
        "zero": false
      },
      "axis": {
        "title": [illegible]
      }
    },
    "y": {
      "field": "petalLength",
      "type": "quantitative",
      "scale": {
        "zero": false
      },
      "axis": {
        "title": "Petal Length"
      }
    },
    "z": {
      "field": [illegible],
      "type": "quantitative",
```

Fig. 3. The <VRIA> Builder, an optional end-to-end visualization creation tool that comes packaged with our framework. It enables a user to browse an existing library of examples and allows them to rapidly prototype and test custom visualizations without leaving the browser. Here, a user is creating a 3D Scatter plot of the Iris Flower dataset from the JSON editor.

see the results. The resulting visualization can be viewed within the builder as a 3D interactive scene, or immersively via a VR headset. The builder's graphical user interface (GUI) allows a user to upload their own dataset or choose from a set of examples. These examples can then be configured via a set of drop-down menus, allowing the user to set various parameters including the type and shape of marks, and channel encodings. The advantage of presenting parameters in this way is that it requires no prior knowledge of the grammar, whereas users do not have to worry about having to fathom JSON, with which they may be unfamiliar. The builder works in a similar way to that of DXR's in-situ GUI, although it is not part of the VR scene and cannot yet be viewed from within the VR headset. We plan to offer this feature in a future version to let designers tailor their visualizations immersively. From within the builder, users can also browse a gallery of example plots which can act as templates. These plots can be loaded in, edited *in-situ* and immediately previewed which serves to speed up the learning and development process.

4.2 <VRIA> for non-WebVR developers

The <VRIA> builder also serves to accelerate the learning process and productivity of more experienced programmers, especially those that have little to no WebVR development experience. As such users are more likely to have used JSON before, they may be more inclined to edit any configuration files directly, as opposed to using the builder's GUI, which may be tedious if they're already familiar with the grammar.

For developers who are new to writing WebVR experiences, the simplest way to get started with <VRIA> is to create a new React project; however, with a few extra steps, <VRIA> can be integrated into an existing non-React project, with the only requirement being that the overarching application makes use of A-Frame scenes. As shown in Figure 2, intermediate users can write a configuration file themselves or produce one through the builder, which they can then export and integrate into a standalone application.

4.3 <VRIA> for WebVR developers

As the builder is responsible for parsing and generating visualization configuration files for standard <VRIA> features, any new bespoke features that are developed with the API will not be available in the builder. Thus, an advanced standalone project that uses <VRIA>'s API is beyond the scope of the builder's intended purpose. For users who have experience with WebVR and A-Frame programming, the builder is still useful for learning the grammar and prototyping the initial structure of their <VRIA> application. We elaborate on the <VRIA> API and how it can extend applications in subsection 5.2

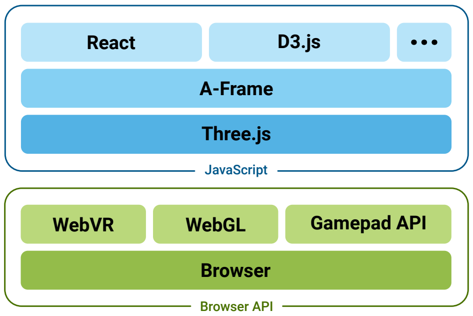


Fig. 4. <VRIA> is built upon the WebVR stack. At the lower level, the browser acts as a rendering engine. WebVR manages the interface with the HMD and its controllers (via the Gamepad Extensions API), or the necessary fallback to a non-immersive display (e.g., a desktop display in 2D). WebGL handles hardware acceleration for the 3D graphics. All 3D graphics are managed by Three.js and, at a higher level, A-Frame. At the top of the stack, <VRIA> utilizes other JavaScript libraries including React and D3.js (see Section 5).

5 The <VRIA> Framework

<VRIA> is a JavaScript (NodeJS) module written in React, a JavaScript library for building user interfaces, and A-Frame, a framework for building WebVR experiences. A-Frame is an entity component framework that provides declarative, extensible, and composable structure to Three.js [49], [50], an open-source Javascript library that enables programming of 3D scenes in a Web browser (see Figure 4). Three.js uses the HTML5 canvas element, SVG or WebGL as rendering engines and features a scene-graph, several cameras, navigation modes, shaders (including custom) and material support. <VRIA>, through A-Frame, provides support for all major HMDs and their handheld controllers, including the six degrees of freedom (6DOF) HTC Vive/Focus, Oculus Touch/Quest and Windows MR dual controllers, as well as 3DOF Daydream, Oculus Go and Gear VR single controllers.

More importantly, A-Frame exposes an HTML DOM scene graph that React can use to efficiently update only the parts of our scene that require re-rendering. While operations such as searching, modifying, adding and removing nodes from the HTML DOM are very fast, re-painting large parts of the DOM tree can be computationally expensive. React uses a virtual DOM to calculate the differences in the HTML DOM before and after a change, so that only the affected nodes get re-rendered. When re-rendering a DOM tree for a data visualization, with potentially thousands of data points, reducing unnecessary re-paints is essential.

5.1 How <VRIA> works

The internal process that <VRIA> undergoes to convert a dataset and visualization configuration file into an immersive and interactive WebVR visualization, is depicted in a high-level overview in Figure 5. First <VRIA> interprets a provided visualization configuration file (`vis config`), inferring any missing values, filling them with appropriate defaults. The complete configuration is then used to map data to graphical marks and their visual encoding channels. Once all of the mappings are completed, values are passed through to the visualization renderer which constructs the visualization, user interface controls, axes and legends etc., and adds them to the scene. Configuration interpretation and data mapping takes place at runtime and as a result any changes to the dataset or visualization configuration will permeate through a <VRIA> application to respond in real time. As well as producing details-on-demand by hovering over marks with an HMD controller or gaze cursor, interactions alter the visibility of marks via query filters and toggles, and can be triggered by built-in or custom controls. All interactions are real-time.

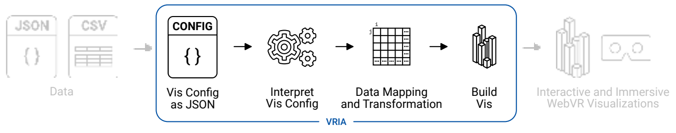

JSON
CSV
Data
CONFIG
Vis Config as JSON
Interpret Vis Config
Data Mapping and Transformation
Build Vis
VRIA
Interactive and Immersive WebVR Visualizations

Fig. 5. High-level overview of <VRIA>'s process for converting data and a visualization configuration into an immersive visualization. <VRIA> interprets the configuration file (formatted in JSON) and its associated dataset. Missing values in the `vis config` are inferred to build a complete configuration. The data is then mapped to graphical marks and their visual encoding channels. <VRIA> then constructs and renders the visualization, and corresponding UI controls.

5.2 Architectural Overview

<VRIA> is packaged as a NodeJS module which exposes a React component of the same name plus an API for extended functionality. The exact structure and implementation of the high-level DOM architecture of an application that makes use of <VRIA> is up to the user, and there is no requirement for the whole application to be written in React. <VRIA> can be integrated into existing applications, as long as it makes use of A-Frame scenes to present a virtual environment in which the visualizations can be placed. Figure 6 shows a simplified component tree of an example <VRIA> application and the entry point for the `vis config`.

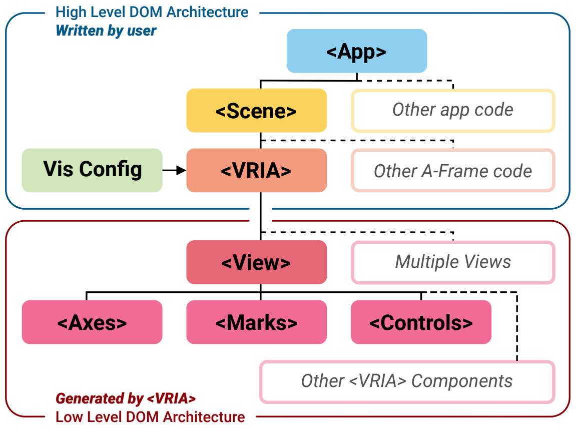
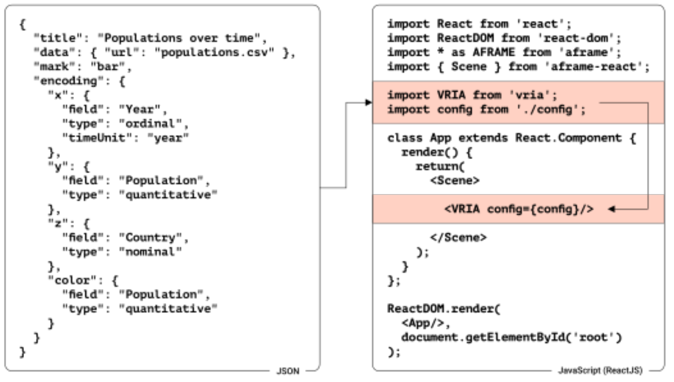


Fig. 6. The high-level DOM architecture is written by the user. <VRIA> can be integrated into existing 2D or immersive visualization applications. The low level DOM architecture is generated by <VRIA> based on the visualization configuration (simplified for brevity).

The top level node depicted in Figure 6 is a React component (depicted as 🟦) that contains an A-Frame scene component (🟨). The latter can contain any other A-Frame components that a user desires for their scene. For example they may wish to create a virtual office or a sports pitch that they can then overlay with visualizations. The <VRIA> NodeJS module provides a React component called <VRIA> (🟧) which must be placed within the A-Frame scene element and should be passed a visualization configuration file (🟩). Every aspect of the visualizations that this component generates is mandated by the configuration file that is passed to it. The basic structure of what is generated is shown in Figure 6. <VRIA> will generate one or more view components (🟥), with each containing a single visualization and its corresponding interactions. Each view component maps data to React components (🟥) that depict the visualization, and all user interfaces with A-Frame and ThreeJS. They also respond to user interactions which then trigger any required updates to the visualization. Views can be linked together in the `vis config`, and interactions that are made on any set of views that are linked together will update the other views in that set, with the appropriate filtered result. An example of a boilerplate implementation of the high level DOM architecture can be seen in Figure 7.



```
{
  "title": "Populations over time",
  "data": { "url": "populations.csv" },
  "mark": "bar",
  "encoding": {
    "x": {
      "field": "Year",
      "type": "ordinal",
      "timeUnit": "year"
    },
    "y": {
      "field": "Population",
      "type": "quantitative"
    },
    "z": {
      "field": "Country",
      "type": "nominal"
    },
    "color": {
      "field": "Population",
      "type": "quantitative"
    }
  }
}
```

JSON

```
import React from 'react';
import ReactDOM from 'react-dom';
import * as AFRAME from 'aframe';
import { Scene } from 'aframe-react';

import VRIA from 'vria';
import config from './config';

class App extends React.Component {
  render() {
    return(
      <Scene>

        <VRIA config={config}/>

      </Scene>
    );
  }
};

ReactDOM.render(
  <App/>,
  document.getElementById('root')
);
```

JavaScript (ReactJS)

(a) <VRIA> Example Vis Config File

(b) <VRIA> React boilerplate example

Fig. 7. (a) Basic visualization configuration as JSON for a 3D bar chart visualization (shown in Figure 13). (b) Example boilerplate code for a <VRIA> application written in React. The `vis config` is imported into the application and passed to the <VRIA> React component.

5.3 Visualization Configuration

<VRIA> currently supports tabular data formatted as JSON or CSV. Data sets may contain any number of fields and records. Supported visualization types currently include Cartesian plots such as scatter plots and bar graphs with more visualization types planned.

<VRIA> provides a simple method of converting a dataset into WebVR-ready 2D and 3D visualizations, coupled with a set of appropriate user interface controls (interactions), through a declarative format described with JSON. This forms a visualization configuration that our framework can interpret (see Figure 5). Our grammar is based on Vega-Lite [13] and allows users to create different visualizations through setting parameters, properties and constraints tailored to each visualization type. It does not yet support Vega-Lite's advanced transform functionality, but existing Vega-Lite visualization specifications should need little to no adjustments to work in <VRIA>.

In its most basic form, a visualization configuration file consists of a dataset, the type of graphical mark to use and a set of visual encoding channels which map to individual data points (eg., *x*, *y*, *z*, color, opacity, size, width, height, depth, *x*-rotation, *y*-rotation and *z*-rotation) written up in a JSON file or as a Javascript module (see Figure 7). Omitted fields and settings are inferred during interpretation and are tailored for optimal presentation and comfortable viewing. Mandating that users must specify values for every parameter of the `vis config` could be cumbersome. Instead, by inferring missing values, the resultant configuration is concise. <VRIA> requires only a `vis config` for basic functionality. If extra functionality is required, such as a custom set of interactions, graphical marks or channel encodings, then these can be created with A-Frame and <VRIA>'s API.

5.4 <VRIA> API

<VRIA> offers sufficient built-in flexibility to design basic charts for many different applications, yet designers may wish to alter certain characteristics of their visualizations and customize them further. For example, they may wish to alter the visual style of a visualization to blend it into the theme of an existing application, where custom marks or custom interaction controls are more appropriate than what is offered as standard. <VRIA> offers the ability to alter the visual properties of visualization features (e.g., axes, titles, legends, UI controls etc.) through the `vis config`. However, for adding custom marks, encoding channels and interaction UI controls, designers will need to utilize <VRIA>'s API. With the API, developers can map inputs and UI controls to data transformations, which are then reflected in the visualization. The exact nature of the interactions and UI controls are left to the user; the framework then connects everything together. Primitive shapes, models, animations and any physics-based UI controls (e.g., pushable buttons, sliders etc.) can be created with A-Frame and then integrated into a visualization via the API and `vis config`.

The API reports the current state of each visualization, including information such as all of the currently visible data points. This information can then be consumed by other parts of an application, outside of the context of <VRIA>. This is particularly useful in order to integrate with other tools, which may require information on the current state of a <VRIA> visualization. Data transformations for new visualizations can be implemented from this data with existing libraries such as D3.js. Examples of usage of our API can be seen in Section 6.

5.5 Interaction Mechanisms

The ability to provide appropriate interaction mechanisms based on the platform and hardware available to users is critical for Web-based IA applications. The Web's ubiquitousness is very much based on the notion of adaptability and responsiveness. <VRIA> utilizes A-Frame's capability to provide suitable interaction mechanisms regardless of available hardware, employing progressive enhancement [17]. An example of this might be offering a drag interaction in the form of two separate gaze clicks at the extremities of the desired selection. Most of the applications built with <VRIA> for our use cases (see Section 6) are not intended for use on smartphones. However, when viewed on a smartphone the user is still able to fully interact with every UI control available, through touch-based interaction when not in VR mode, and gaze interactions when using Google Cardboard. UI controls (e.g., sliders, buttons, toggles, marks etc.) are bound to the depicted data and update the visibility of marks according to the type of interaction which is being made. Since <VRIA> employs progressive enhancement strategies, the UI controls which enable the various types of interactions that are available depend on the platform and hardware that are currently available to the user. Custom UI controls can be created with the <VRIA> API.

Selection, Filtering and **Details on Demand** are facilitated in <VRIA> through a set of interaction types which can be selectively enabled or disabled via the `vis config`. <VRIA> supports gaze and controller-based selection for filtering tasks and details on demand. Multi-dimensional queries are also possible through constraint combinations. See subsection 6.2 for further description and usage examples of these interaction mechanisms.

Brushing and linking capabilities are also enabled through the `vis config`. <VRIA> provides a mechanism to constrain the range of variables per axis resulting in filtered selections which can then be mapped across linked views. <VRIA>'s support for the linking of visualizations with multiple coordinated views enables the developer to compose combinations of visualizations and integrate them with existing scenes, or to create dashboards. Each view can be filtered and the results from such operations will update all other linked views in the scene.

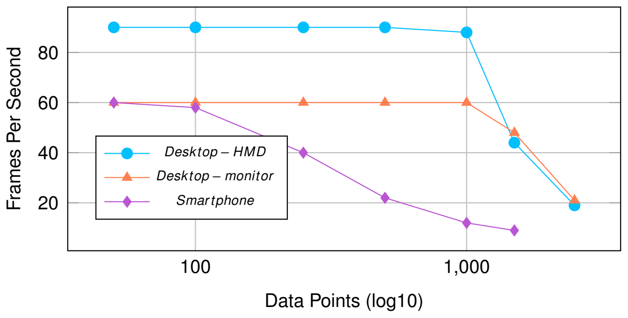


Fig. 8. Performance profiles of a scatter-plot implemented in <VRIA> and rendered by the builder in the Mozilla Firefox (v68) browser on Windows 10 with an Nvidia GTX 1070 GPU and AMD Ryzen 5 1600X, viewed with an Oculus Rift CV1 (2160×1200 @90Hz), and monitor (1920*1080 @60Hz). The same scatter-plot is rendered on a OnePlus 6T Android smartphone (1080×2280 @60Hz), in Mozilla Firefox (v68).

5.6 Performance and Scalability

Maintaining adequate frame rates, while rendering high number of data points, is one of the most challenging aspects of building a VR experience. A web browser's resultant frame budget is typically 16.6ms for a common 60Hz display [51]. For a VR scene using a HMD such as the Oculus Rift or HTC Vive, the frame budget will be 11ms as these devices have a 90Hz refresh rate. Desktop displays usually only require a scene to be rendered in mono, whereas a VR headset requires a stereo image, leaving just 5.5ms to create an image for each eye. Consequently, performing the necessary computations for rendering each frame in time can be challenging. If the frame budget is exceeded, the browser is unable to render a new frame, which results in (at least) a lost frame, leading to uncomfortable VR experiences and increased risk of cybersickness. Therefore, certain performance optimizations are necessary.

Figure 8 depicts a performance profile of <VRIA>, rendering a scatter-plot of different numbers of data points encoded as spheres. Three platform set-ups are presented: a smartphone, a desktop-HMD set up, and a desktop-monitor set-up. The results demonstrate the challenges inherent in Web-based VR with frame rates dropping significantly over 1000 points for non-mobile set-ups, and at a slightly slower rate after 100 points for mobile. Nevertheless, the observed performance is very similar to the HoloLens condition reported for IATK [40]. It is also noteworthy that the choice of text rendering can also have a detrimental effect on performance This consequently has implications on labeling strategies required for 3D plots. <VRIA> utilizes Signed Distance Field (SDF) fonts which provide good clarity (crisp edges) and relatively good performance.

In <VRIA>, some performance optimizations incorporated are at the React level, and are intended to provide fine-grained control over when a component should update (e.g., using the aptly named `shouldComponentUpdate` lifecycle method), preventing unnecessary re-renders. There are also performance optimizations at the ThreeJS level, such as the re-use and merging of object geometries to save on memory and the number of draw calls per frame. Built-in and user-created visualizations and UI controls make use of these optimizations.

Inevitably, if the rendered scene is very complex users will notice stuttering or choppiness (”jank” in JavaScript lingo) when there is motion on the display. One solution is to ”debounce” interaction operations, a technique for limiting the rate at which a function can be triggered. In <VRIA> we reduce the jank associated with stacking multiple expensive computations together in quick succession, by waiting for a user’s interaction to be completed before attempting a re-render. Furthermore, we batch computations over multiple frames and only render the result once all computations have been made. We achieve this by utilizing a callback function to perform computations on the main event loop, during periods in which the CPU would normally be idle.

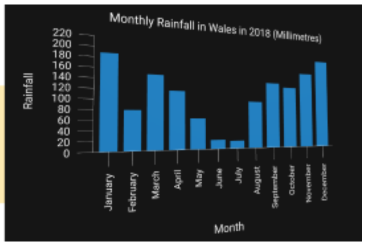
6 Use Cases

To demonstrate the features and versatility of <VRIA> we have devised a set of use cases, which we discuss in this section. The use cases can automatically be viewed and interacted with in VR.

6.1 Cartesian Plots

<VRIA> can create simple Cartesian visualizations using configuration files similar to those of Vega-Lite to create one, two (Figure 9 top) and three (Figure 9 bottom) axis scatter plots and bar charts with multiple mark shapes. One and two dimensional charts are supported as they can be useful in linked-view visualizations. These plots can be produced by mapping data fields and their corresponding data type to the *x*, *y* or *z* and other encoding channels in the `vis config` (highlighted in 🟨).

```
{
  "title": "Monthly Rainfall...",
  "data": { "url": "cy18rain.csv" },
  "mark": "bar",
  "encoding": {
    "x": {
      "field": "Month",
      "type": "nominal"
    },
    "y": {
      "field": "Rainfall",
      "type": "quantitative"
    }
  }
}
```



```
{
  "title": "The Iris Flower Dataset",
  "data": { "url": "iris.csv" },
  "mark": "point",
  "encoding": {
    "x": {
      "field": "petalWidth",
      "type": "quantitative"
    },
    "y": {
      "field": "petalLength",
      "type": "quantitative"
    },
    "z": {
      "field": "sepalLength",
      "type": "quantitative"
    },
    "color": {
      "field": "species",
      "type": "nominal"
    }
  }
}
```

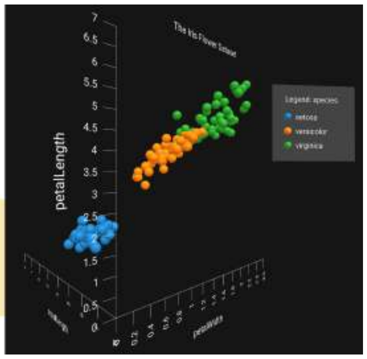


Fig. 9. Top: A 2D bar chart visualizing precipitation in Wales, UK in 2018 and its corresponding `vis config` file. Bottom: A 3D bar chart visualizing the Iris Flower Dataset and its corresponding `vis config` file.

6.2 Interaction

As aforementioned, interactions can be selectively enabled or disabled via the `vis config` file, which also allows for the creation of basic selection and filtering controls. New, bespoke interaction controls can be added to <VRIA> via the API. These two approaches allow for both constraints-based interactions and interactions that allow for external manipulation. Figure 10 shows a demonstration of three types of interaction mechanisms in <VRIA> and how these are implemented via the `vis config`.

```
{ ...
  "mark": { "type": "point",
    "tooltip": { "content": "data" }
  },
  "encoding": {
    "x": {
      "field": "petalWidth",
      "type": "quantitative",
      "scale": {
        "zero": false
      },
      "axis": {
        "filter": true
      }
    },
    ...
    "color": {
      "field": "species",
      "type": "nominal",
      "legend": {
        "title: "Species",
        "filter": true
      } ...
```

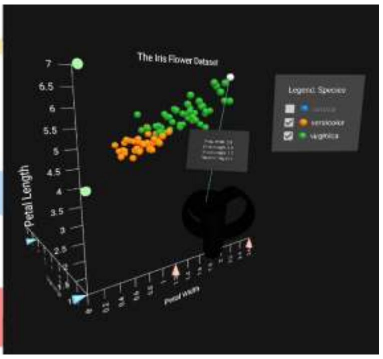


Fig. 10. Demonstration of three types of interaction mechanisms in <VRIA>, selection and filtering, and how these are implemented via the `vis config`.

The user has filtered the data via three axis filters which denote the threshold of values to show for the particular axis. These can be moved up or down the axes to fine tune the selection. They are enabled in the `vis config` by adding the filter option to the axis for a particular encoding channel (🟦). The user has also filtered the data via the legend, which allows them to selectively toggle the visibility of data in a certain category. This can be enabled in the `vis config` by adding the filter option to the legend (🟥). By hovering over a data point, a user can fetch details-on-demand which are attached to the controller, if one is present, or displayed to the side of the chart otherwise. This interaction is enabled by specifying the tooltip property of the mark (🟨). Clicking a data point will freeze the selection and let the user bring the controller closer to further inspect the output. On displays which do not support controller input, say a smartphone, the user is presented with a gaze cursor instead. This interaction is supported on non-immersive displays via a mouse or touch interaction.

6.3 Embedded Visualizations

IA scenes can also include A-Frame assets, upon which we can embed data to give them additional meaning, or enhance their context. Such assets can either be A-Frame primitives, models or combinations of those, and are added in the A-Frame scene alongside <VRIA>. Figure 11 demonstrates a linked and filtered

```
{ ...
  "views": [
    "mark": "point",
    "encoding": {
      "x": {
        "field": "X",
        "type": "quantitative",
        "axis": { "filter": true }
      },
      "y": {
        "field": "Y",
        "type": "quantitative",
        "axis": { "filter": true }
      },
      "color": {
        "field": "Type",
        "type": "nominal",
        "legend": { "filter": true }
      }
    }
  ], [ ... ]
}
```

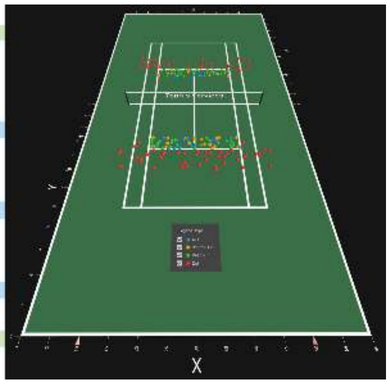
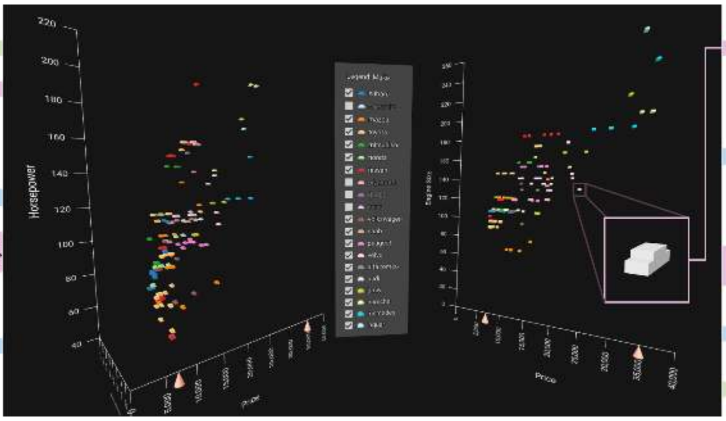


Fig. 11. Assets can add meaning to visualizations. Here we have created a tennis court asset inside A-Frame and used it to mark out the area over which we place a linked and filtered visualization showing the service game of two tennis players.

```
{ ...
  "data": { "url": "cars.csv" },
  "views": [
    {
      "mark": {
        "type": "point", "shape": "car"
      },
      "encoding": {
        "x": {
          "field": "price",
          "type": "quantitative",
          "axis": {
            "title": "Price",
            "filter": true
          }
        }, "y": { ... }, "z": { ... },
        "width": { "field": "width", ... },
        "height": { "field": "height", ... },
        "depth": { "field": "length", ... },
        "color": {
          "field": "make",
          "type": "nominal",
          "legend": {
            "title": "Make",
            "filter": true
          }
        }
      }, ...
    }, ...
```

continued on the right...



```
... {
      "mark": {
        "type": "point", "shape": "car"
      },
      "encoding": {
        "x": {
          "field": "price",
          "type": "quantitative",
          "axis": {
            "title": "Price",
            "filter": true
          }
        },
        "y": {
          "field": "engine-size",
          "type": "quantitative", ...
        },
        "width": { "field": "width", ... },
        "height": { "field": "height", ... },
        "depth": { "field": "length", ... },
        "color": {
          "field": "make",
          "type": "nominal",
          "legend": false
        }
      }, ...
    }
  ]
}
```

Fig. 12. An example of multiple linked views in <VRIA> and their corresponding `vis config` file (shortened for brevity). Any number of plots can be linked and changes to one of the filters or the legend will update the other views. This example also demonstrates the use of a custom mark. Custom marks can also be 3D models (e.g., gLTF), which are nonetheless expensive to render, hence why we opt for a simpler mark for this example.

visualization of the service of two tennis players which has been placed over a scaled-down tennis court model, created with A-Frame. The tennis court asset puts the data points into context and provides a better understanding of the data, than if the points were presented as regular side-by-side scatter plots, for example. It is possible to remove all chart elements and leave just the marks, so that plots can be used to symbolize data without explicitly inferring its magnitude.

6.4 Linked Views and Custom Marks

Linking allows for the output of a selection in one visualization to be mapped to the output of other visualizations and vice versa. Any number of visualizations can be linked together in <VRIA>. Figure 12 shows an example setup using a chunk of the data from the UCI Machine Learning Repository's Automobile Data Set and demonstrates the linking of multiple views. A 3D and 2D scatter plot have been linked together in a single `vis config` via the views array (🟩). The views array contains the mark type and encoding channels associated with each individual plot. These plots use a custom car mark (🟪) (loaded via the API), each with an individual width, height and depth corresponding to the size of the car they depict. Any changes to the legend, or filters (🟦) in either plot will update the other. The legend in the second plot has been disabled to avoid duplication.

6.5 Multiple Users

<VRIA> supports the creation of collaborative environments, allowing multiple users to interact with visualizations in real time, and can be integrated with existing JavaScript networking libraries to offer this functionality. Figure 13 demonstrates an application that uses the Networked A-Frame component (an abstraction layer over WebRTC and WebSockets), and its default avatars, and shows two users interacting with a visualization. Users in such a collaborative VR space have access to all available interactions provided by <VRIA>. In addition, <VRIA>'s API can instead be used with lower-level networking libraries such as Socket.IO and the aforementioned WebSockets and WebRTC. A description of any updates, resulting from any interactions that have been made is passed between <VRIA> instances. These messages synchronize the current view between multiple clients.

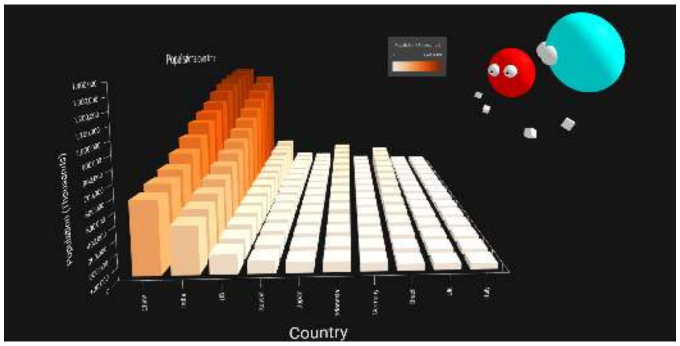


Fig. 13. An example of a multi-user environment enabled by the open-source Networked A-Frame component.

7 Limitations and Future Work

We have so far presented our motivations for developing <VRIA> using standards-based Web technologies, along with our framework's visualization creation workflow, implementation architecture, and features demonstrated through use cases. In this section, we focus on limitations, and future work.

Visualization Types. <VRIA> currently supports Cartesian plots where data points are represented in a bar graph or scatter plot structure. The range of possible encoding channels that these plots support, beyond *x*, *y* and *z*, allows for the visualization of high dimensional datasets. The ability to add custom marks and encoding channels, as well as being able to alter a chart's appearance make these two underlying chart structures remarkably versatile and suitable for a wide variety of use-cases. However, we plan to add support for other coordinate systems (eg. geographical/spherical) to facilitate a wider range of data visualization use-cases.

Data model. <VRIA> currently supports tabular data formatted as JSON or CSV. We plan to support other data types such as hierarchical, network and relational data models, to create visualizations such as node-link diagrams. We also plan to extend current mechanisms to support GeoJSON and TopoJSON.

User Experience (UX)**.** <VRIA> supports the development of IA experiences by developers of different expertise levels. Regardless of the effort to make <VRIA> as user-friendly as we can perceive, our evidence on the UX associated with using our framework is limited. We have previously evaluated IA experiences, produced with previous versions of <VRIA> in terms of the resulting UX [6]. Students have also used the current <VRIA>

version in undergraduate Computer Science final-year projects successfully, for prototyping a variety of IA demonstrations, similar to our use cases. Finally, we make <VRIA> publicly available for the visualization community to use and wish to conduct a longitudinal assessment of our framework's UX, from the points of view of developers and end-users of the experiences.

Visual vocabulary for Immersive Analytics. A challenge that transcends all flavors and underlying technologies of IA, is devising a suitable visual vocabulary for VR/MR worlds, that facilitates the visualization pipeline. Bearing in mind the usage patterns of IA systems in VR/MR can be very different to traditional approaches, there is a need to determine and evaluate mechanisms for displaying data in immersive ways. <VRIA> facilitates this investigation due to the aforementioned synergy with visualization libraries, as well as due to the plethora of interface devices it supports from the outset, and with a common codebase.

Future Development. In addition to the aforementioned enhancements in supported visualization types and coordinate systems, we also plan to offer other features in the future. As discussed in subsection 5.3, we aim to extend the features of our `vis config` to support advanced data transforms akin to those supported by Vega-Lite. This addition would enhance <VRIA>'s linked view capabilities especially, by enabling transforms such as aggregation, binning and other more advanced filtering operations. The <VRIA> builder currently offers a more streamlined approach to visualization creation by enabling the creation of visualizations without leaving the browser. We would like to extend the builder's functionality to offer users the possibility of prototyping visualizations completely inside VR, building up a `vis config` without the need to remove the headset between iterations, similar to what DXR offers with their *in-situ* GUI.

One significant extension that we have already started to explore is to expand <VRIA>'s capabilities into XR. The use of the term 'XR' in our context is akin to that of the W3C, denoting alternative immersive technologies (i.e., VR/MR/AR) [18]. Our prior preliminary investigations have utilized marker-based registration using Vuforia, within the Argon4 browser [52], and AR.js [53]. At this stage of <VRIA>'s evolution we have integrated the latter which works with mobile and desktop browsers, demonstrating <VRIA>'s consumability (besides the aforementioned Builder), as the output of our framework is directly included in a typical AR.js set-up. We plan to extend <VRIA>'s XR capabilities with more registration mechanisms in the future, while maintaining our conformance to standards-based Web technologies. The WebVR community is already exploring different registration solutions, which can be easily integrated with <VRIA> via the DOM. <VRIA> is also supported in the Microsoft Edge browser on the Hololens without any reliance on third-party AR software.

8 Lessons Learned

<VRIA> evolved over the period of three years, growing from a simple, single, stand-alone A-frame visualization application of a 3D bar chart [5], to the current framework. During the creation, usage and evaluation of early <VRIA> versions [4], [6], [8], we encountered several technical challenges that directed the evolution of our framework. In this section we discuss the lessons learned from addressing these challenges, and provide guidance for the development of suchlike, Web-based frameworks.

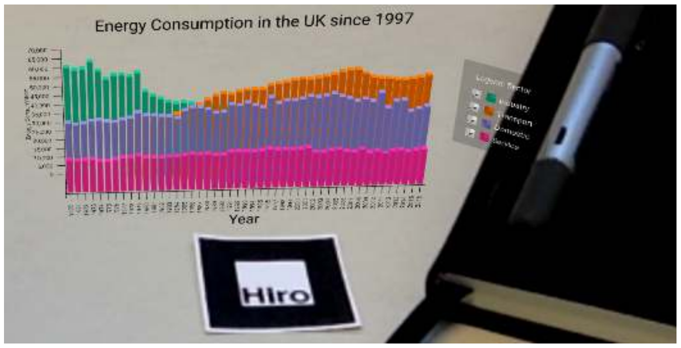


Fig. 14. <VRIA> can be easily integrated with AR.js to offer data visualizations in augmented reality on desktop and mobile browsers. Here we have used a Hiro marker to provided a reference for AR.js to render a 3D bar chart visualization of the BEIS UK energy consumption data set. It shows energy consumption across 4 sectors from 1970-2016.

8.1 VRIA synergies for Visualization

As we have experienced during the development of all of <VRIA>'s versions, the synergistic use of established visualization libraries, such as D3.js, with <VRIA> can be an enabler for the visualization community, allowing more opportunities for investigations and application development. Likewise, the same applies when we consider libraries that introduce XR capability on Web-based solutions, such as AR.js. For example, we have been using D3.js seamlessly with <VRIA>, such as for our scatter-plot component displayed in Figure 12. In that regard, our experience from developing and using [6] <VRIA> strengthens our belief that this synthesis of standards-based tools, over the HTML DOM, is a strong foundation for immersive data visualizations. Moreover, <VRIA> and its outputs can be interoperable, or even consumed by other popular tools, such as say Vistrates [16] via the DOM, resulting in hybrid systems that push the boundaries of immersive data visualization [7].

8.2 Accessibility vs Performance

As discussed in subsection 5.6, <VRIA> and similar Web-based tools face important trade-offs between pervasiveness, ease of use and performance. The use of libraries, such as Three.js, A-Frame, and React, simplifies development and usage, especially for novice users. However, as these libraries are incrementally added on top of WebGL, and due to the thin-client model and single-threaded nature of JavaScript, they introduce some performance barriers. This challenge required us to implement, over time, performance optimizations to ensure <VRIA> applications could maintain adequate frame rates when rendering large datasets with real-time interactions, including those on less powerful devices such as smartphones. For instance, as our component trees grew more complex, and the need to pass data down to deeply nested components increased, we integrated Redux, one of the many state management solutions available for use in JavaScript applications. Since its inclusion, Redux has simplified the flow of data through <VRIA> applications, resulting in a cleaner code base. Overall, as our intention with <VRIA> is to facilitate quick prototyping of IA experiences, we opted for openness and ease of use, while ensuring adequate performance.

Inevitably, although <VRIA> fulfills its purpose as Web-based tool for building varied, shareable, distributable and consumable IA experiences, when it comes to rendering high number of data points it can not compete, in terms of performance, with game-engine based systems due to the aforementioned limitations. For <VRIA>, and similar systems, to achieve higher performances, they needs to operate closer to the WebGL level, and thus without abstraction layers such as the one offered by A-Frame, or general purpose state management mechanisms such as Redux. In this regard, in future versions of <VRIA> we endeavor to work at the Three.js level, as well as replace Redux with a bespoke state-management system. In addition, optimizing rendering mechanisms, such as via mesh and geometry merging to reduce draw calls, akin to the approach taken by Stardust [42] and IATK [40] would allow us to render more points with improved rendering performance.

8.3 Web Standards and maturity

Lastly, it is important to emphasize that standards-based Web technologies for VR and MR are inevitably not as mature as those found in the game engine ecosystem. WebVR (soon to be WebXR) is an experimental technology whose standards are constantly adjusting as it develops and matures. Browser support for WebVR is forever changing and software that relies on browser APIs to display VR/MR content requires frequent updates as a result. This problem is not limited to the Web, as native desktop applications often need to be updated to support new versions, as well as their dependencies. However, as VR/MR enabling technology on the Web is changing so rapidly, and the ecosystem is currently very volatile, new software and existing libraries need to constantly adapt. This is, and will continue to be, an important consideration that developers need to take into account, before they choose to build their software using experimental, bleeding-edge technologies and emerging standards such as WebVR/XR.

We therefore acknowledge that Web-based standards-based immersive technologies are just past their infancy, and it may take some time until they mature, become established and are easily deployable. Limitations such as providing low-level access to capabilities such as positional sensing, computer vision and context awareness may not be evident in VR, but emerge in other XR flavors [28]. Nevertheless, we believe that their potential for VR/MR-based IA is significant and therefore worth the effort to explore them and utilize them in analytical scenarios.

9 Conclusion

We present <VRIA>, a framework for building Immersive Analytics solutions in VR, using standards-based Web-technologies. <VRIA> is built using WebVR, A-Frame and React. The resulting VR solutions can be experienced through a WebVR-compliant browser on a variety of devices, ranging from smartphones to HMD-equipped desktop computers. <VRIA> uses a declarative format for specifying visualization types through simple configuration files, simplifying visualization prototyping, data binding and interaction configuration. We elaborate on <VRIA>'s visualization creation workflow that provides different development paths for novice, intermediate and expert developers. The workflow makes optional use of a dedicated visualization builder, a Web-based interface that enables developers to easily prototype immersive analytics experiences and export their visualization configurations. These configurations can be further customized via the <VRIA> API to create new immersive depictions. We also present a series of use cases that demonstrate the functionality and versatility of <VRIA>. Finally, we discuss current limitations and expand on future work.

As the interest in immersive data visualization increases, there is a clear need to investigate and devise appropriate visualization techniques, tailor-made for immersive 3D graphical environments. Indeed, the question of what works effectively in 3D immersive visualization built with modern technologies, in terms of data visualizations and the corresponding interaction techniques, remains outstanding. We believe that <VRIA> is in a position to facilitate such investigations, as it allows researchers to deploy their experimental set-ups though a Web-browser in a variety of devices, and consequently increase participation and data-collection opportunities. <VRIA> is free and open source, and available at https://github.com/vriajs.

References

[1] T. Dwyer, K. Marriott, T. Isenberg, K. Klein, N. Riche, F. Schreiber, W. Stuerzlinger, and B. H. Thomas, *Immersive Analytics: An Introduction*. Cham: Springer International Publishing, 2018, pp. 1–23.

[2] HTC Corporation. (2017) *HTC Vive*. [Online]. Available http://www.htcvive.com/uk/. Accessed: 12/17/19.

[3] Oculus VR, LLC. (2017) *Oculus Rift S*. [Online]. Available https://www.oculus.com/rift-s/. Accessed: 12/17/19.

[4] P. W. Butcher, J. C. Roberts, and P. D. Ritsos, "Immersive Analytics with WebVR and Google Cardboard," in *Posters of the IEEE Conference on Visualization (IEEE VIS 2016), Baltimore, MD, USA*, 2016.

[5] P. W. Butcher and P. D. Ritsos, "Building Immersive Data Visualizations for the Web," in *Procs. of International Conference on Cyberworlds (CW'17)*, 2017, pp. 142–145.

[6] P. W. S. Butcher, N. W. John, and P. D. Ritsos, "VRIA - A Framework for Immersive Analytics on the Web," in *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '19. New York, NY, USA: ACM, 2019, pp. LBW2615:1–LBW2615:6.

[7] J. C. Roberts, P. D. Ritsos, S. K. Badam, D. Brodbeck, J. Kennedy, and N. Elmqvist, "Visualization Beyond the Desktop - the next big thing," *IEEE Comput. Graph. Appl.*, vol. 34, no. 6, pp. 26–34, Nov. 2014.

[8] P. W. Butcher, N. W. John, and P. D. Ritsos, "Towards a Framework for Immersive Analytics on the Web," in *Posters of the IEEE Conference on Visualization (IEEE VIS 2018), Berlin, Germany*, 2018.

[9] R. M. Rohrer and E. Swing, "Web-based information visualization," *IEEE Comput. Graph. Appl.*, vol. 17, no. 4, pp. 52–59, Jul. 1997.

[10] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 Data-Driven Documents," *IEEE Trans. Vis. Comput. Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.

[11] M. Bostock and J. Heer, "Protovis: A Graphical Toolkit for Visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 15, no. 6, pp. 1121–1128, Nov. 2009.

[12] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer, "Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 22, no. 1, pp. 659–668, Jan. 2016.

[13] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-Lite: A Grammar of Interactive Graphics," *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 1, pp. 341–350, Jan. 2017.

[14] V. Vukicevic and B. Jones and K. Gilbert and C. Van Wiemeersch. (2016) *WebXR Device API Specifications*. [Online]. Available https://immersive-web.github.io/webvr/spec/1.1/. Accessed: 17/12/19.

[15] C. N. Klokmose, J. R. Eagan, S. Baader, W. Mackay, and M. Beaudouin-Lafon, "Webstrates: Shareable dynamic media," in *Procs. of the 28th Annual ACM Symposium on User Interface Software & Technology*, ser. UIST '15. New York, NY, USA: ACM, 2015, pp. 280–290.

[16] S. K. Badam, A. Mathisen, R. Rädle, C. N. Klokmose, and N. Elmqvist, "Vistrates: A component model for ubiquitous analytics," *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 1, pp. 586–596, Jan. 2019.

[17] W3C. (2017) *WebVR*. [Online]. Available https://webvr.info/. Accessed: 17/12/19.

[18] B. Jones and N. Waliczek. (2019) *WebXR Device API*. [Online]. Available https://www.w3.org/TR/webxr/. Accessed: 17/12/19.

[19] T. Chandler, M. Cordeil, T. Czauderna, T. Dwyer, J. Glowacki, C. Goncu, M. Klapperstueck, K. Klein, K. Marriott, F. Schreiber, and E. Wilson, "Immersive Analytics," in *Procs. of Big Data Visual Analytics*, ser. (BDVA), 2015, pp. 1–8.

[20] B. Lee, P. Isenberg, N. H. Riche, and S. Carpendale, "Beyond Mouse and Keyboard: Expanding Design Considerations for Information Visualization Interactions," *IEEE Trans. Vis. Comput. Graphics*, vol. 18, no. 12, pp. 2689–2698, Dec. 2012.

[21] N. Elmqvist, A. Vande Moere, H. C. Jetter, D. l. Cernea, H. Reiterer, and T. Jankun-Kelly, “Fluid interaction for information visualization,” *Information Visualization*, vol. 10, no. 4, pp. 327–340, Oct. 2011.

[22] N. Elmqvist and P. Irani, “Ubiquitous Analytics: Interacting with Big Data Anywhere, Anytime,” *Computer*, vol. 46, no. 4, pp. 86–89, Apr. 2013.

[23] Y. Jansen and P. Dragicevic, “An Interaction Model for Visualizations Beyond The Desktop,” *IEEE Trans. Vis. Comput. Graphics*, vol. 19, no. 12, pp. 2396–2405, Dec. 2013.

[24] W. Willett, Y. Jansen, and P. Dragicevic, “Embedded Data Representations,” *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 1, pp. 461–470, Jan. 2017.

[25] A. van Dam, A. S. Forsberg, D. H. Laidlaw, J. J. LaViola, and R. M. Simpson, “Immersive VR for scientific visualization: a progress report,” *IEEE Comput. Graph. Appl.*, vol. 20, no. 6, pp. 26–52, Nov. 2000.

[26] G. de Haan, M. Koutek, and F. H. Post, “Towards Intuitive Exploration Tools for Data Visualization in VR,” in *Procs. of the ACM Symposium on Virtual Reality Software and Technology*, ser. (VRST). New York, NY, USA: ACM, 2002, pp. 105–112.

[27] D. Germans, H. J. W. Spoelder, L. Renambot, and H. E. Bal, “VIRPI: A High-level Toolkit for Interactive Scientific Visualization in Virtual Reality,” in *Procs. of the Eurographics Conference on Immersive Projection Technology and Virtual Environments*, ser. (EGVE). Eurographics Association, 2001, pp. 109–120.

[28] B. Maclntyre and T. F. Smith, “Thoughts on the future of WebXR and the immersive web,” in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, Oct. 2018, pp. 338–342.

[29] S. Zhang, C. Demiralp, D. F. Keefe, M. DaSilva, D. H. Laidlaw, B. D. Greenberg, P. J. Basser, C. Pierpaoli, E. A. Chiocca, and T. S. Deisboeck, “An immersive virtual environment for DT-MRI volume visualization applications: a case study,” in *Procs. of Visualization*, Oct. 2001, pp. 437–584.

[30] M. Cordeil, T. Dwyer, K. Klein, B. Laha, K. Marriott, and B. H. Thomas, “Immersive Collaborative Analysis of Network Connectivity: CAVE-style or Head-Mounted Display?” *IEEE Trans. Vis. Comput. Graphics*, vol. 23, no. 1, pp. 441–450, Jan. 2017.

[31] C. Donalek, S. G. Djorgovski, A. Cioc, A. Wang, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. Graham, A. Drake *et al.*, “Immersive and collaborative data visualization using virtual reality platforms,” in *Procs. of IEEE International Conference on Big Data*, Oct. 2014, pp. 609–614.

[32] C. Hurter, N. H. Riche, S. M. Drucker, M. Cordeil, R. Alligier, and R. Vuillemot, “Fiberclay: Sculpting three dimensional trajectories to reveal structural insights,” *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 1, pp. 704–714, Jan. 2019.

[33] M. Drouhard, C. A. Steed, S. Hahn, T. Proffen, J. Daniel, and M. Matheson, “Immersive visualization for materials science data analysis using the Oculus Rift,” in *Procs. of the IEEE International Conference on Big Data (Big Data)*, Oct. 2015, pp. 2453–2461.

[34] N. A. ElSayed, B. H. Thomas, K. Marriott, J. Piantadosi, and R. T. Smith, “Situated Analytics: Demonstrating immersive analytical tools with Augmented Reality,” *Journal of Visual Languages & Computing*, vol. 36, pp. 13 – 23, 2016.

[35] B. Bach, R. Sicat, J. Beyer, M. Cordeil, and H. Pfister, “The Hologram in My Hand: How Effective is Interactive Exploration of 3D Visualizations in Immersive Tangible Augmented Reality?” *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 1, pp. 457–467, Jan. 2018.

[36] M. Luboschik, P. Berger, and O. Staadt, “On Spatial Perception Issues In Augmented Reality Based Immersive Analytics,” in *Procs. of ACM International Conference on Interactive Surfaces and Spaces*, ser. ISS. New York, NY, USA: ACM, 2016, pp. 47–53.

[37] S. Butscher, S. Hubenschmid, J. Müller, J. Fuchs, and H. Reiterer, “Clusters, Trends, and Outliers: How Immersive Technologies Can Facilitate the Collaborative Analysis of Multidimensional Data,” in *Procs. of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’18. New York, NY, USA: ACM, 2018, pp. 90:1–90:12.

[38] A. Lu, J. Huang, S. Zhang, C. Wang, and W. Wang, “Towards Mobile Immersive Analysis: A Study of Applications,” in *Procs. of Immersive Analytics Workshop, IEEE VR*, J. Chen, E. G. Marai, K. Mariott, F. Schreiber, and B. H. Thomas, Eds., Mar. 2016.

[39] M. Cordeil, A. Cunningham, T. Dwyer, B. H. Thomas, and K. Marriott, “ImAxes: Immersive Axes As Embodied Affordances for Interactive Multivariate Data Visualisation,” in *Procs. of the ACM Symposium on User Interface Software and Technology*, ser. (UIST). New York, NY, USA: ACM, 2017, pp. 71–83.

[40] M. Cordeil, A. Cunningham, B. Bach, C. Hurter, B. Thomas, K. Marriott, and T. Dwyer, “IATK: An Immersive Analytics Toolkit,” in *Procs of the IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, Mar. 2019.

[41] R. Sicat, J. Li, J. Choi, M. Cordeil, W. Jeong, B. Bach, and H. Pfister, “DXR: A Toolkit for Building Immersive Data Visualizations,” *IEEE Trans. Vis. Comput. Graphics*, vol. 25, no. 1, pp. 715–725, Jan. 2019.

[42] D. Ren, B. Lee, and T. Höllerer, “Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering,” *Computer Graphics Forum*, vol. 36, no. 3, pp. 179–188, 2017.

[43] J. Heer, S. K. Card, and J. A. Landay, “Prefuse: A Toolkit for Interactive Information Visualization,” in *Procs. of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’05. New York, NY, USA: ACM, 2005, pp. 421–430.

[44] J. D. Fekete, “The InfoVis Toolkit,” in *Procs. of the IEEE Symposium on Information Visualization*, ser. INFOVIS ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 167–174.

[45] W. E. Mackay, “Augmented Reality: Linking real and virtual worlds: A new paradigm for interacting with computers,” in *Procs. of the Working Conference on Advanced Visual Interfaces*, ser. AVI ’98. New York, NY, USA: ACM, 1998, pp. 13–21.

[46] E. Barba, B. MacIntyre, and E. Mynatt, “Here We Are! Where Are We? Locating Mixed Reality in The Age of the Smartphone,” *Procs. of the IEEE*, vol. 100, no. 4, pp. 929 –936, Apr. 2012.

[47] P. Milgram and F. Kishino, “A Taxonomy of Mixed Reality Visual Displays,” *IEICE Trans. Inf. Sys.*, vol. E77-D, no. 12, 1994, pp. 1321–1329.

[48] R. Azuma, “A Survey of Augmented Reality,” *Presence*, vol. 6, no. 4, pp. 355–385, 1997.

[49] R. Cabello. (2017) *Three.js*. [Online]. Available https://github.com/mrdoob/three.js. Accessed: 12/17/19.

[50] J. Dirksen, *Learning Three.js: the JavaScript 3D library for WebGL*. Packt Publishing Ltd, 2013.

[51] World Wide Web Consortium. (2016) *Frame Timing*. [Online]. Available https://www.w3.org/TR/frame-timing/. Accessed: 17/12/19.

[52] P. D. Ritsos, J. Jackson, and J. C. Roberts, “Web-based Immersive Analytics in Handheld Augmented Reality,” in *Posters presented at the IEEE Conference on Visualization (IEEE VIS 2017), Phoenix, Arizona, USA*, 2017.

[53] P. D. Ritsos, J. Mearman, J. R. Jackson, and J. C. Roberts, “Synthetic Visualizations in Web-based Mixed Reality,” in *Immersive Analytics: Exploring Future Visualization and Interaction Technologies for Data Analytics Workshop, IEEE Conference on Visualization (VIS), Phoenix, Arizona, USA*, B. Bach, M. Cordeil, T. Dwyer, B. Lee, B. Saket, A. Endert, C. Collins, and S. Carpendale, Eds., Oct. 2017.

Peter W. S. Butcher completed his M.Sc. in Computer Science at Bangor University, UK in 2015. He is currently studying for a PhD in Immersive Analytics at the Department of Computer Science, University of Chester, UK. His research interests include Virtual Reality, Web Technologies and Information Visualization.

Nigel W. John received his PhD degree from the University of Bath, UK, in 1989. He is currently a Senior Research Professor within the Department of Computer Science, University of Chester, UK. His research interests include Virtual Reality, Computer Graphics and Medical Visualization. In 2006 he was awarded the 12th annual Satava Award to acknowledge his accomplishments in the field of computer graphics and medical visualization.

Panagiotis D. Ritsos received the PhD degree from the University of Essex, UK in 2006. He is currently a Lecturer in Visualization, in the School of Computer Science and Electronic Engineering, Bangor University, UK. His research interests include Human-Computer Interaction, Mixed and Virtual Reality, and Information Visualization. He is a member of the IEEE.