HSF-CWP-2017-15
October 26, 2018

HEP Software Foundation Community White Paper Working Group – Visualization

HEP Software Foundation: Matthew Bellis[a,b] Riccardo Maria Bianchi[c,1] Sebastien Binet[d] Ciril Bohak[e] Benjamin Couturier[f] Hadrien Grasland[g] Oliver Gutsche[h] Sergey Linev[i] Alex Martyniuk[j] Thomas McCauley[k,1] Edward Moyse[l] Alja Mrak Tadel[m] Mark Neubauer[n] Jeremi Niedziela[f] Leo Piilonen[p] Jim Pivarski[q] Martin Ritter[r] Tai Sakuma[s] Matevz Tadel[m] Barthélémy von Haller[f] Ilija Vukotic[t] Ben Waugh[j]

[a] *Siena College, Loudonville NY, USA*
[b] *Cornell University, Ithaca NY, USA*
[c] *University of Pittsburgh, Pittsburgh PA, USA*
[d] *CNRS/IN2P3, Clermont-Ferrand, France*
[e] *University of Ljubljana, Ljubljana, Slovenia*
[f] *CERN, Geneva, Switzerland*
[g] *LAL, Université Paris-Sud and CNRS/IN2P3, Orsay, France*
[h] *FNAL, Batavia IL, USA*
[i] *GSI Darmstadt, Germany*
[j] *University College London, London, UK*
[k] *University of Notre Dame, Notre Dame IN, USA*
[l] *University of Massachusetts, Amherst MA, USA*
[m] *University of California at San Diego, San Diego CA, USA*
[n] *University of Illinois, IL, USA*
[p] *Virginia Tech, VA, USA*
[q] *Princeton University, Princeton PA, USA*
[r] *LMU Munich, Munich, Germany*
[s] *University of Bristol, Bristol, UK*
[t] *University of Chicago, Chicago IL, USA*

[1] Paper Editors

Abstract: In modern High Energy Physics (HEP) experiments visualization of experimental data has a key role in many activities and tasks across the whole data chain: from detector development to monitoring, from event generation to reconstruction of physics objects, from detector simulation to data analysis, and all the way to outreach and education. In this paper the definition, status, and evolution of data visualization for HEP experiments will be presented. Suggestions for the upgrade of data visualization tools and techniques in current experiments will be outlined, along with guidelines for future experiments. This paper expands on the summary content published in the HSF *Roadmap* Community White Paper [1].

Contents

1 Scope **3**

2 Current landscape **4**
2.1 Event displays 4
2.1.1 Data access 4
2.1.2 Application development and distribution 6
2.1.3 Geometry description and visualization 11
2.2 Statistical data visualization 12
2.2.1 Desktop solutions 13
2.2.2 Web-based solutions 14
2.2.3 Issues 15
2.3 Non-spatial visualization 15

3 Suggested guidelines and future development **16**
3.1 A common community-defined format 16
3.2 Serving the geometry and event data through services 20
3.3 Client-server architecture for geometry and event data visualization 20
3.4 Exploring modern technologies 21
3.4.1 Graphics and game engines 21
3.4.2 Web-based applications 22
3.4.3 Virtual and augmented reality 22
3.4.4 Mobile technologies 24
3.4.5 Multi-user applications 25

4 Sharing knowledge and fostering collaboration **25**
4.1 Yearly workshop 26
4.2 Code repository 26

5 Roadmap **26**
5.1 One year 26
5.2 Three years: ATLAS and CMS Computing TDRs 27
5.3 Five years: Towards HL-LHC 27

6 Conclusions **27**

7 Acknowledgements **28**

70 **References** **29**

1 Scope

Visualization: Turning numbers into pixels

Hadrien Grasland
HSF Workshop 2017, Annecy

This paper will describe three kinds of data visualization used in High-Energy Physics (HEP): interactive visualization of event data in applications known commonly as event displays, statistical data visualization such as histograms, and non-spatial data visualization such as networks and graphs.

Event displays are the main tool used to explore experimental data at the event level. There are two main types of displays. The first type are those that are integrated into an experiment's software frameworks, which are usually able to access and visualize all experimental data at the cost of greater application complexity and lesser portability. The second type of displays are those designed as cross-platform applications, lightweight and fast, often delivering a simplified version or a subset of the event data. All event displays show the detector geometry; the level of detail displayed depends on the application's use-case and targeted audience as well as on the application's capability to render geometries responsively.

Beyond event displays, HEP also uses statistical data visualizations such as histograms, which display the distributions of data variables in aggregate over multiple events. These visualizations are not strongly linked to a detector geometry. Data analysis tools and techniques used in HEP are described in the HSF *Data Analysis and Interpretation* Community White Paper [2].

The final types of visualization considered in this paper are those that visualize non-spatial data, such as the graphs used to visually describe the structure of the detector description, that is the representation of all geometrical volumes that compose the sub-detectors and the infrastructure of a HEP experiment. More details about the detector geometry can be found in the HSF *Detector Simulation* Community White Paper [3].

Other types of data visualization used in HEP experiments, such as visualization for slow control or dashboards for data analytics, are considered out of scope and are not discussed.

The content of this paper is the summary of the direct experience of the authors in designing, building, and deploying interactive data visualization applications for the experiments ALICE, ATLAS, Belle II, CMS, LHCb, and LSST, for scientific software frameworks such as ROOT, and in other cross-experiment projects. It is the outcome of a number of discussions and workshops organized under the auspices of the HEP Software Foundation, in which the authors worked to build a common,

shared view of the field, its current issues, and the potential ways it could and should evolve in the context of HEP.

2 Current landscape

2.1 Event displays

Three key features characterize HEP event displays. The first is an *event-based workflow*. Applications access experimental data on an event-by-event basis, visualizing the data collections belonging to a particular event. Data can be related to the actual physics events (*e.g.* a collection of reconstructed physics objects, like jets and tracks) or to the experimental conditions (*e.g.* different versions of the detector description and calibration data).

The second key feature is *geometry visualization*. The level of geometric detail displayed depends on the specific use-case, on the way the geometry information is stored and fetched (*e.g.* from a database as part of a software framework or from an external file), and on limitations of the application itself along with considerations about speed, efficiency, and portability.

The third key feature is *interactivity*. Applications offer different interfaces and tools for users to interact with the visualization, select event data, and to set cuts on objects' properties. In addition to the interactive usage, applications often store different settings to automate user actions.

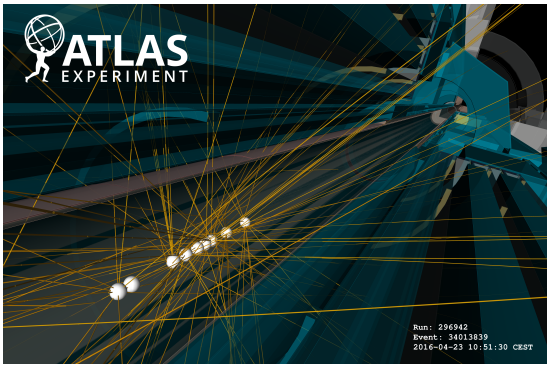
In the following subsections several important aspects of data access, application development and distribution, and geometry description and visualization, as they pertain to the current landscape of event displays, are discussed in more detail. Screenshots of several event displays can be seen in Figure 1.

2.1.1 Data access

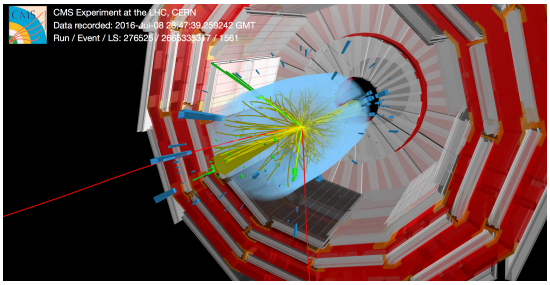
Access to event data comes either natively or via intermediate formats. In the former case direct access of native event formats is only possible for an application integrated with the experimental software framework.

There are several advantages to having access to the experiment's framework, such as full access to the experimental data in its native format and to software tools, services, and databases. Through them, event display applications can make use of the full detector simulation geometry, of conditions data, and of all the framework's application program interface (API).

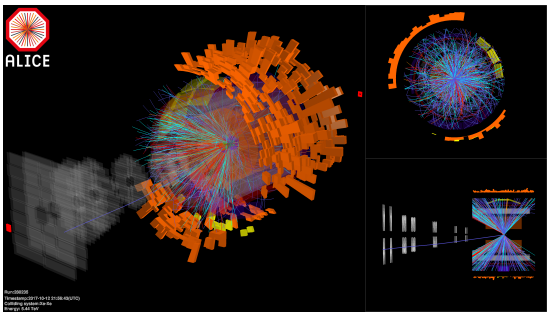
One disadvantage of this approach is that full support for the display application is often limited to those platforms on which the framework itself is supported, limiting cross-platform distribution and support. One way to mitigate this is to distribute a light version of the framework along with the application; CMS Fireworks [10] takes this approach. However, issues of platform support for the light framework



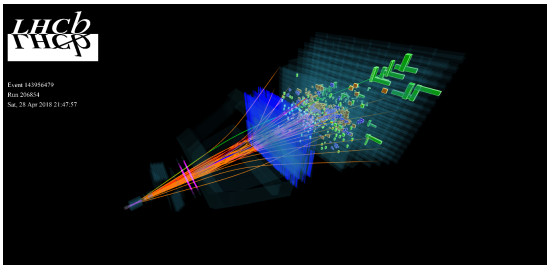
(a) ATLAS



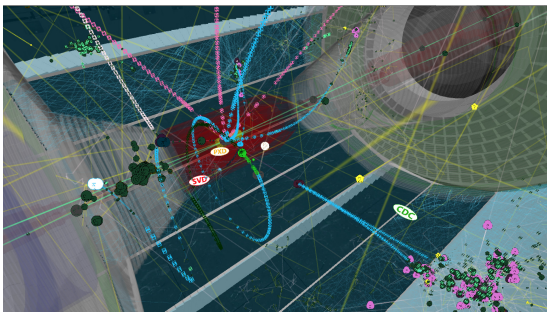
(b) CMS



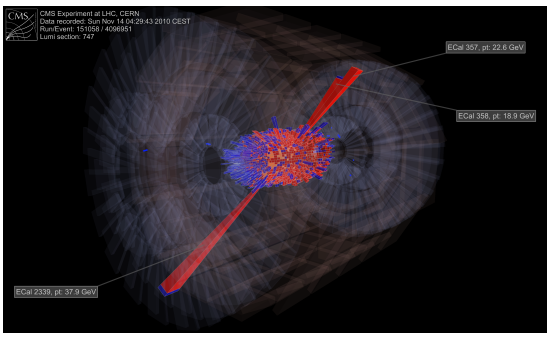
(c) ALICE



(d) LHCb



(e) Belle II VR



(f) CMS (Fireworks)

Figure 1: Different examples of HEP 3D event display applications: a) an ATLAS 3D event display made with VP1 [4, 5]; b) a CMS 3D event display made with iSpy [6]; c) an ALICE 3D event display [7]; d) an LHCb 3D event display [8]; e) a Belle II VR event display [9]; f) a CMS 3D event display made with Fireworks [10]

and for the visualization application can still exist. A further disadvantage to the full-framework (and even light-framework) approach is that one must also support various versions of the data format along with the underlying framework API. In addition, users have to have knowledge of the framework in order to interactively explore and visualize event-based data. Lastly often the user-interface to a full-

framework application is geared towards the expert.

The latter approach to data access is via an intermediate format. Usually the data needed for visualization is a subset of the full information found in the native experimental format. Therefore one can extract what is needed from the framework through the usage of dedicated exporting software tools and store in intermediate formats.

With the use of an intermediate data format (usually based on different flavors of the XML or JSON formats) the event display application is potentially separate from the experimental software framework and therefore its deployment is not limited to those platforms officially supported by the experiment. It is then possible to have both lightweight data and applications, which can be easily and broadly distributed.

The primary drawback to this approach is that, with no direct access to the experimental data, some information is necessarily not accessible. Moreover, every time there is the need to modify the content of the intermediate data files, one needs to run the data extraction tools on the native data again. In addition to that, only events which are identified as potentially interesting are extracted and their data reduced to be stored in the intermediate data file; so, if the end user wants to analyze and visualize other potentially interesting events, the extraction/reduction step must be performed again on the relevant data.

Regardless of the approach to data access one must consider the use-case: what is useful or necessary in one use-case may not be for another. A graphical representation of the two approaches can be seen in Figure 2.

2.1.2 Application development and distribution

Currently, the two most common ways of distributing event display applications are as a desktop application and as a web application running in the browser. Each approach has its advantages and disadvantages which are further described in this section. The current landscape is summarized in Figure 4 and is further described in this section.

Native mobile applications running on devices such as smartphones and virtual reality applications are less common in HEP. However, they are a growing feature in the current landscape and many experiments are exploring the possibilities of those emerging technologies. At the end of this section we describe briefly the mobile applications released so far. Further developments will be described in Section 3.4.

Desktop applications Many experiments have developed integrated event-display applications in C++, which is the main language used for developing HEP software frameworks, on top of the OpenGL [11] APIs. The choice of the OpenGL API, compared to other APIs like Direct3D [12], resides in its cross-platform nature as OpenGL is an open standard. The OpenGL consortium defines the API: the interface with which all the implementations have to comply. The actual implementation is provided by vendors, usually targeting a specific hardware. Many hardware and

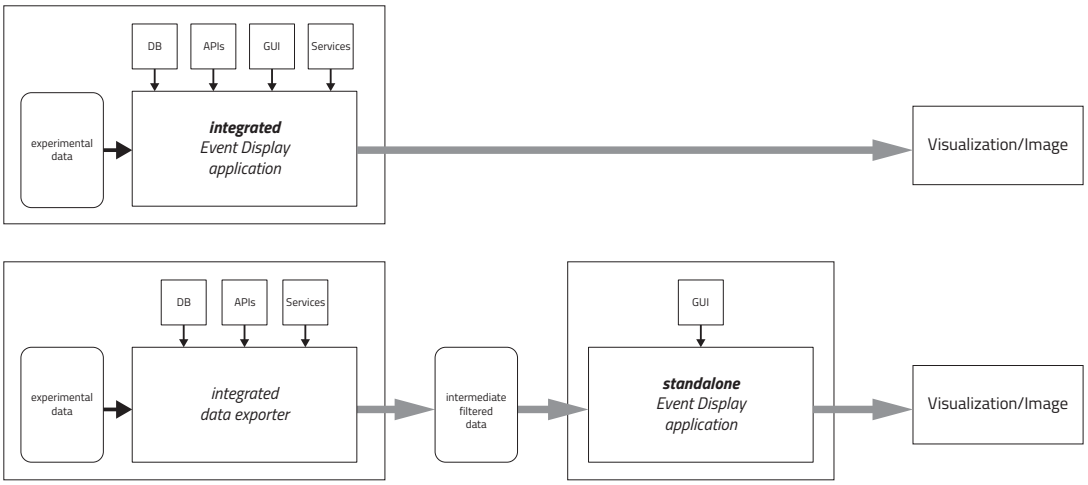
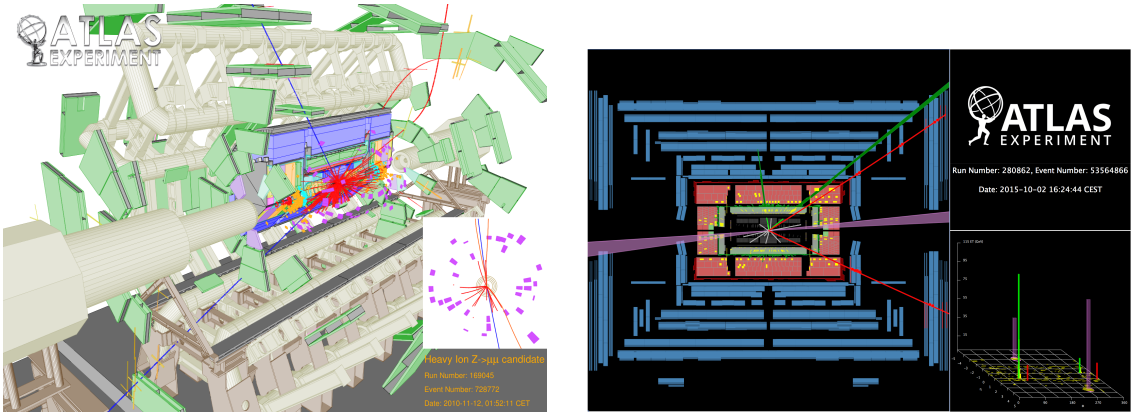


Figure 2: The two different types of event display applications. At the top, the framework-integrated event display application is able to access all experimental data, all services, APIs, and databases provided by the experiment's framework; as a drawback, the application must be run on specific platforms supported by the framework and it must use only graphics and GUI libraries compatible with them. At the bottom, the standalone approach, where experimental data are accessed, filtered, and extracted by using custom data exporters, which create intermediate data files containing only the interesting pieces of information; then, a standalone event display application reads those data in and it creates the required visualization; the advantage is having a cross-platform application which can use any graphics and GUI libraries, while the drawback is the lack of direct and full access to the experimental data and to the experiment's software tools, which prevents a detailed, full visualization.

software companies such as Intel and NVIDIA are part of the OpenGL consortium, which assure the support and the lifetime of the OpenGL API.

Some HEP visualization applications use OpenGL calls directly through custom graphics engines. This is the most robust approach as the developers can take full control over the OpenGL interface and the project can be independent of other software libraries. Two examples of HEP applications which follow this path are the ATLAS Persint application [13] and the ROOT EVE toolkit [14], which is used both by the CMS Fireworks application [10] and the ALICE Event Visualization Environment (AliEve) [15]. One disadvantage of this approach is that developer time has to be assured to maintain the graphics engine itself, on top of the effort needed for the development and maintenance of the event display application.

Other applications use higher-level interface libraries as graphics engines. This has the advantage of delegating a large part of the lower-level development work to



(a) (b)

Figure 3: a) an ATLAS 3D event display made with Persint [13, 16]; b) an ATLAS 2D event display made with Atlantis [17, 18].

external software packages, leaving the developers to concentrate on the application development itself. A popular graphics library used in HEP software has been Open Inventor [19], used by the defunct CMS Iguana [20, 21] application, or its clone implementation Coin (also known as Coin3D) [22], which has been used by applications such as ATLAS VP1 [5], LHCb Panoramix [23] and the defunct desktop version of the CMS iSpy application [24]. Coin / Open Inventor was chosen because of its integrability in C++ code, its performance, and its coding style. Moreover, the way Open Inventor handles graphical volumes could be easily matched with the way geometry volumes are handled to describe the detectors in HEP experiments. Open Inventor organizes geometry volumes as a series of nodes in a tree-like structure in the same way as some HEP experiments do. ATLAS, for instance, developed their geometry library "GeoModel" [25] based on the same tree-like structure of nodes used by Open Inventor.

The drawback of this approach is the dependency on external software projects, which could end up with a loss of functionality if third-party library development and support are abandoned. Many scientific visualization applications, also in fields other than HEP, faced this when the support of the Coin library was dropped by the company that led its development [26]. The result is the aging of libraries which after a while show incompatibilities with modern hardware, compilers, and platforms. The time spent by HEP developers to repair or to maintain those abandoned libraries results is time not spent on actual development of the software applications themselves.

An additional approach to the development of event displays is to create and distribute an application using the Java programming language. The Atlantis [18] program and its derivative MINERVA [27], which is used as an educational tool, both developed for the ATLAS experiment, are based on the Java graphics libraries and

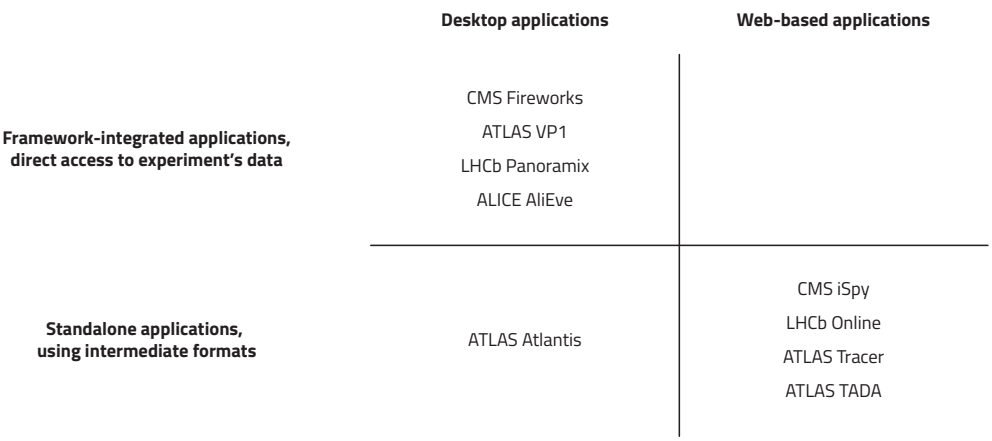


Figure 4: The image summarize the current landscape of event display applications in HEP. Many experiments developed full-framework desktop applications as well as light, web-based applications. As one can see from the plot, there are no examples, so far, of full-framework applications using web-based visualization graphics, due to the data access issues with today's experiments' frameworks. A new approach in that direction is what it is suggested in this paper, in Section 3.3.

can be run either online in a web browser or stand-alone on a desktop machine.

Web-based applications Several experiments at the LHC, notably CMS [28], LHCb [29], and ATLAS [30, 31] have created web-based event displays using WebGL (Web Graphics Library) [32]. WebGL is a JavaScript API that conforms to OpenGL ES (a subset of the OpenGL API for embedded systems) conceived for rendering interactive 3D and 2D graphics within any compatible web browser without the need of external plug-ins. With WebGL in the browser one can combine high-quality graphics with the functionality and accessibility of the browser. This combination of graphics and user function was previously only available via bespoke desktop applications based on OpenGL and graphical user interface toolkits such as Qt [33]. Browser-based event displays have several distinct advantages: they are easy to distribute to the user, they can be prototyped quickly, and the client is often much lighter-weight, as the need for building, packaging and distributing external libraries is greatly reduced. There are also several mature and actively developed WebGL frameworks, such as the popular three.js [34], that provide straightforward and simplified APIs for ease of development.

Mobile applications Mobile devices such as smart phones and tablets are more and more ubiquitous and these devices are used more and more as substitutes for desktop and laptop machines. However, mobile devices still do not have the computing power usually needed for HEP data analysis, where huge amount of experimental data are retrieved and processed. In addition they usually run dedicated operating

systems whose self-contained nature makes their integration within the HEP workflow difficult, particularly for the statistical-based visualization used in data analysis, described in Section 2.2. Despite the current limitations for these use-cases mobile devices can be found in the current landscape of event displays.

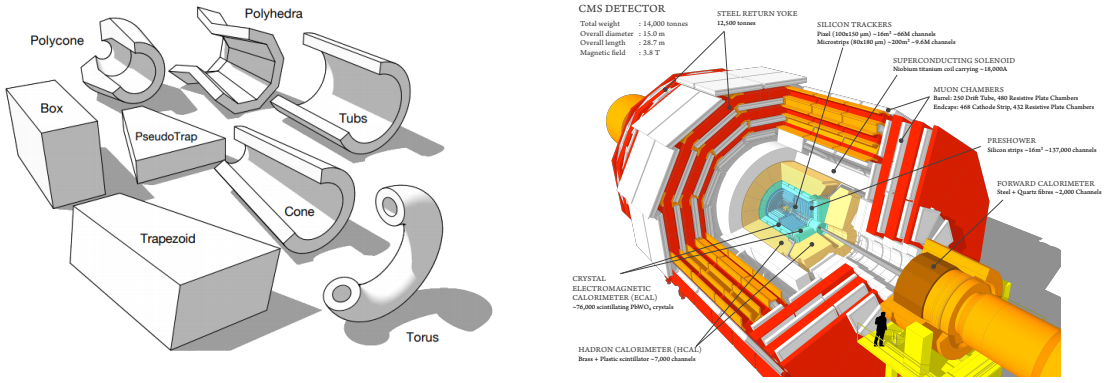
Several event display applications have been developed for, or at least can be run on, mobile devices. An example of a native application is LHSee [35] which live-streams ATLAS events, processed and extracted through Atlantis [18], to a user's phone and provided contextual information on ATLAS and the events being displayed. The CMS iSpy WebGL application [28] runs on mobile devices in the browser. The Camelia application [36], developed by the CERN Media Lab using the Unity game engine [37], can be built as a mobile application by using the tools provided by the game engine and run on mobile devices as well. More details on game engines can be found in Section 3.4.1.

Virtual and augmented reality applications Virtual Reality (VR) simulates the user's physical presence in a virtual environment, and the application is typically run on a head-mounted display that provides visual and aural experience of the simulated environment. Different degrees of realism and immersion are possible, depending on the targeted hardware.

VR technologies can be used to build immersive applications, to let the general public virtually visit HEP detectors and explore experimental sites. Many HEP experiments started developing VR applications, mostly as an educational tool, for outreach events. These include ATLASrift [38] and Belle II VR [39]. As many HEP experiments are not accessible during data taking or are classified as supervised areas due to security or safety issues, VR applications let the HEP community open their sites and experiments to the general public. Also, they let people look at simulated collisions in a simplified yet realistic environment, which help people acquire the basic concepts on which HEP experiments are built and run. Such applications are currently used in public events, in museums and science centers, and in meetings with the governments and the funding agencies.

Augmented Reality (AR), instead, uses a camera to take a view of the real world around the user and screen where simulated objects are rendered in 3D and shown on top of that image dynamically, following the user's interaction and motion. This lets the user move within an environment where real and simulated objects live together. AR can be used in HEP as an educational tool, for instance to dynamically show and describe a HEP detector to a group of people or a class. Some HEP experiments have been started exploring AR technologies for HEP, particularly ALICE, ATLAS, and CMS.

Both VR and AR applications are usually developed in specialized graphics engines, which prevent the interaction with the experiment's framework to access native data. More details on VR and AR applications for HEP are found in Section 3.4.3.



(a) (b)

Figure 5: Use of a 3D software (SketchUp) to render the basic geometrical shapes imported from the CMS detector description and the final volumes [40].

2.1.3 Geometry description and visualization

Geometry visualization provides important visual context for event displays and dedicated geometry displays are useful applications by themselves. There are typically three levels of detail found in applications. The most detailed geometry is typically called the *simulation geometry* and can include the sensitive elements of the detector as well as support structure. Detector experts are typical end-users of applications that display this level of geometrical detail. Less detailed is the so-called *reconstruction geometry*, which describes the sensitive elements of the detector such as calorimeter crystals and wire and strip chambers. It is this level of detail that is typically found in event displays. The least-detailed geometry descriptions are those that are simplified versions of the detector geometries and are used to provide visual context only.

Currently, different geometry formats and libraries are used in HEP. Some experiments use their own custom format (*e.g.*, [25]), while others use the geometry tools provided by the ROOT framework [41]. More recently, some attempts have been made to build common formats and libraries for detector geometry, like DD4HEP [42], adopted in conceptual design studies for future high-energy colliders, including the CLICdp [43] and FCC [44] collaborations. In all cases detector volumes are built from simpler geometrical entities: geometrical shapes like Tube, Cone, Box, and more complex variations of these are combined in order to build the volumes of the experiment's geometry. Geometry libraries and formats are also described in the HSF *Detector Simulation* Community White Paper [3].

The differences in geometry formats used by the different experiments, by detector simulation programs like Geant4 [45], and by data analysis frameworks like ROOT, typically require developers of visualization applications to write converters between the different formats. In addition it is often not easy to use a visualization

tool developed for one experiment with another one as current visualization tools are often tightly bound to the geometry library used by the experiment.

In framework-based applications the geometry information can come directly from the experiment's detector description and in many cases the hierarchical structure of the detector description is preserved and accessible. Standalone applications typically use a geometry file with information exported from the software framework. An example hybrid solution is the one developed for the CMS experiment using the SketchUp application [46], described in Ref. [47] (see Figure 5). The CMS detector description [48] as written in XML is parsed using Ruby scripts and 3D models are built using the SketchUp program via its Ruby API. SketchUp can then export to various standard 3D file formats. In this way detailed simulation geometry can be available in a standalone application.

2.2 Statistical data visualization

> The simple graph has brought more information to the data analyst's mind than any other device.
>
> *John Tukey [49]*

Data visualization also means visualizing quantities and properties taken from a series of events, in order to extract statistical meaning from them. Examples of statistical data visualizations are histograms and scatter plots.

In HEP, like most other scientific disciplines, visualization of the data and of the final results plays a key role in the analysis pipeline. A new projection of the data may provide new insight, results must be summarized in a clear and concise way, and multidimensional parameter spaces need to be visualized in an understandable fashion. The discussion of how to properly display data is not new [50], but the tools are constantly evolving. Since its introduction 20 years ago, ROOT [41] has become the most widely used package in HEP to make plots, graphics, and even to build event displays. It was developed at a time when there were few alternatives for the HEP community that did not have a significant financial cost, and it has performed admirably.

However, the landscape is changing and there are several existing tools driven by non-HEP communities available. This section will look at some of the current alternatives and comment on what options might be available in the future and what our needs are. The main focus is on data exploration and presentation tools. The former describes tools with which one prepares and builds visualizations for the purpose of exploring and attempting to understand one's dataset. The latter describes tools for the presentation of final plots in a convenient and accesible way. For more details on data analysis itself, one should refer to the HSF *Data Analysis and Interpretation* Community White Paper [2].

2.2.1 Desktop solutions

As it stands, ROOT is the most widely adopted plotting tool within the HEP and Nuclear Physics community. It has even made some inroads to the astrophysics community and some small pockets within the financial community, to where some physicists migrated. However, few other disciplines have adopted it. Still, it has many features beyond the standard 1D/2D/3D histogram/graphing tools, such as 2D and 3D shapes, widgets for building a GUI, a JavaScript implementation for web-based analysis [51] and is available within a Jupyter notebook [52]. But to access the plotting features, an analyst must install the entire ROOT package which includes file I/O, scientific libraries, fitting routines etc., and often the installation process is non-trivial.

Many current HEP analysts make wide use of the Python programming language and the PyROOT libraries [53]. Python is also very popular outside the HEP community and so it is worth looking at non-ROOT options available to Python users. A recent (as of 2017) summary of the field was presented by Jake VanderPlas at PyCon 2017 [54], a subset of which will be presented here. It is emphasized that this is just a sampling and that the number of options available is a function of time.

- The Python library *matplotlib* [55], released in 2003, is the most mature plotting tool for python and is the standard for most users. It can produce journal-quality graphics and there are some add-ons that can improve the default plotting options [56]. It does 1D, 2D, and 3D graphics with varying degrees of success, but does not integrate with OpenGL libraries and so it can slow down when the number of data points gets very large. It does produce most of the histograms found in HEP but some minimal, extra work must be done by the user to make histograms with error bars. Plots are reactive in the sense that you can zoom in on different regions of the graph, but you cannot do anything more significant with other mouseover commands (links, additional information, etc.).

- The R programming language has several widely used graphics tools, both built-in or provided by external modules. *ggplot2* [57] and *lattice* [58] are particularly useful to visualize data in multidimensional parameter spaces. *ggplot2* is an implementation of the guidelines contained in the classic text *The Grammar of Graphics* [59], while *lattice* is an implementation of the so-called *Trellis Display* [60]. Both packages are very popular outside the HEP community and a wide range of learning materials are available in books, online courses, and other media. Both packages are well developed and mature and offer mechanisms for users to extend them by adding new features. *lattice* has a longer history: in 2005, the year *ggplot2* first appeared, *lattice* was already

popular. In fact, figures made with *lattice* were shown in the presentation in PHYSTAT05 [61] which introduced R to the particle physics community.

2.2.2 Web-based solutions

Web-based data visualization is also being rapidly developed. Very sophisticated toolkits now provide tools to build web-based fully-responsive visualization of data on all types of devices. In addition, they also offer other features, specially useful for HEP, like full in-browser LaTeX rendering (with MathJAX) and real-time visualization of streamed data. Being JavaScript-based, those libraries integrate with the overall ecosystem of web-based technologies, letting them use all the tools offered by other web libraries. They are overall a good solution for data presentation, and can be combined with other tools such as Jupyter in order to be used for data exploration. Some of the most used toolkits are described below:

- D3 (Data Driven Documents) [62] is perhaps the first web-based visualization toolkit which has been widely adopted as the de-facto base solution for building interactive data visualization for the web. The strong point of D3 is the link of the data to the DOM entities and the possibility to work with SVG objects natively. D3 is also the foundation layer upon which many higher level toolkits are built.

- Bokeh [63]. This is a plotting utility from Continuum [64], the company behind the *Anaconda* Python distribution system and other Python modules. It is designed with the web in mind and builds in a high degree of interactivity into the plots, making it useful to share results publicly and for building dashboards. However, it works by writing HTML, which makes it difficult to work with unless you use specific IDEs like a Jupyter notebook. Exporting a figure for a journal article (*e.g.*, in the PNG format) is non-trivial as well, as that is not currently the primary use-case for Bokeh.

- Plotly [65]. This is another web-oriented solution, similar to Bokeh and based on the D3 library, where plots can be hosted in Plotly's cloud service or viewed in a Jupyter notebook. The plots are similarly very interactive and there are ways to export figure images, but that is not the goal. Dashboards can be built with relative ease and plotly offers libraries in R and JavaScript, in addition to Python. They offer both a free and enterprise business model.

The so-called notebooks are a rapidly evolving way of using web-based technology for both online and offline data analysis and visualization, with access to local resources as well. After having started from a Mathematica-like notebook user interface paradigm mixing server-side code snippet execution, structured text, and (mostly) static visualizations, the Jupyter community is now exploring more interactive user

interface paradigms, including in the area of visualization. The JupyterLab project is exploring a more MATLAB-like IDE user experience inside the web browser, with features such as multiple source editing tabs and interactive python consoles. Its ipywidgets sub-project tries to make Jupyter more interactive by moving more visualization work to client-side Javascript and introducing classic GUI widgets such as sliders and checkboxes for interacting with the live visualization. The Belle II experiment has started to use Jupyter notebooks to train new users in data analysis; the learning curve is much gentler than in traditional terminal-based tutorials and the time to useful visualization of results is much faster.

2.2.3 Issues

Separating data visualization from the data analysis: The suite of statistical plotting tools in ROOT, Matplotlib, etc. are adequate for analysis, and their development is very responsive to analysts' needs. However, it is often hard to separate plot-making abilities from the data analysis framework. As a consequence, if a physicist's data can only be found on a particular server, the plot-generating code must also be located there and the outcome is sometimes hard to bring to the physicist's laptop screen. In the worst cases, graphics files (PNGs) must be copied from the server to the laptop for viewing. This causes a high interaction latency, discouraging exploration. That's why the development of new tools should go towards a sharper separation between the computation on data and the interactive data visualization routines, pushing the latter to the client side as much as possible.

Separating the plotting functions and content from the plotting style: Another symptom of the tight coupling between data analysis infrastructure and plotting is that trivial changes to the plot— axis labels, colors, and such— are so deeply buried in the analysis script that persistifying changes to them often requires a full recalculation of the statistics. Changes to the final plot through the usage of on-display user interfaces, in fact, are overwritten and lost if a plot is updated for other reasons (new version of data upstream, for example). Here, one could use inspiration from the increasing separation of logic and presentation that is occurring in GUI toolkits (see *e.g.* use of CSS stylesheets in the GTK/GNOME environment).

A looser coupling between style and content, as well as a looser coupling between locality of computation and locality of rendering, would benefit the physics community.

2.3 Non-spatial visualization

In HEP, there are data which are organized in a tree-like structure, and for which a graph or a network visualization is the best choice. The Detector Description is an example of a source of such data: it describes all the pieces which compose a HEP experiment detector. The different pieces of the Detector Description are interconnected through different relationships: geometrical volumes can be organized

in a parent-child relationship, or a property node can be shared among many volumes. The visualization of those data in a network helps developers in the understanding and the debugging of the Detector Description, by visualizing the relationships among all the nodes and their properties. An example, from the ATLAS experiment [66], of a graph visualizing the inner structure of a HEP detector description can be seen in Figure 6.

Networks and graphs are very effective ways of visualizing tree-like data, because they are able to show all the nodes, their relationships and their properties in a proper way. Some degree of interactivity can let the scientists applying different filters and layout, helping them to get rid of the clutter, to better understand and analyze the data.

Another example of HEP data that can benefit from a graph-based visualization is that one describing the execution chain of the jobs used to filter and reconstruct the experimental data. Very recently HEP experiments began to develop new parallel frameworks to concurrently handle analysis or reconstruction jobs, to efficiently exploit the parallelism offered by the modern hardware (more details can be found in the HSF *Event/Data Processing Frameworks* Community White Paper [67]). The jobs are handled by a scheduler, which organizes them according to their needed input and output data. The outcome of the scheduler is a directed acyclic graph (DAG). The visualization by means of a graph helps the developer understanding and debugging the reconstruction and framework code.

Other HEP experiments use graph-based tools to analyze and visualize other types of data, like geographical distribution and load of computer networks used to transfer data between GRID sites [68] (see Picture 7) or to store and query conditions data [69].

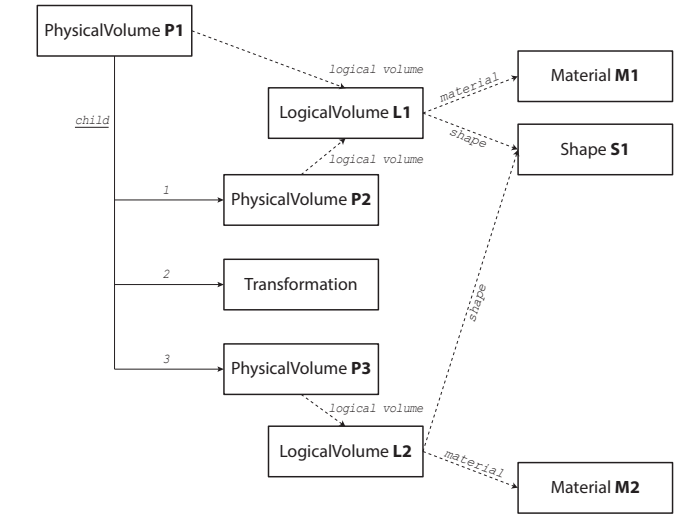
All those data are not space- nor time-dependent, and they are better visualized through a graph or a network. Graph-based visualization, as well as graph-databases, are fairly new in the HEP landscape but they can be very powerful tools to effectively visualize non-spatial data which are by their nature organized with a network layout. We suggest the community further explores those tools, to better understand the possibility offered by graph-based solutions for HEP needs.

3 Suggested guidelines and future development

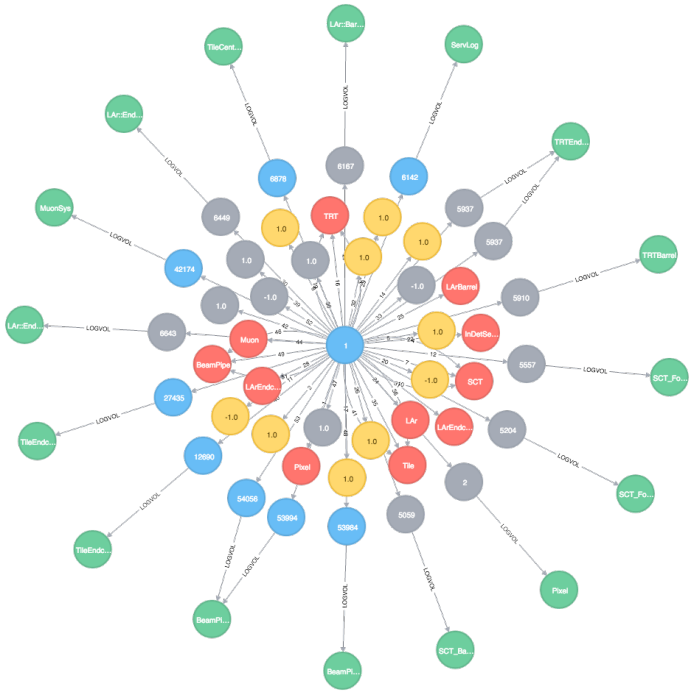
As a community what we want to suggest here is the design and the usage of common base visualization guidelines, to be able to share knowledge and best practices among the different groups, and to foster collaboration among the HEP experiments.

3.1 A common community-defined format

Visualization has a key role in the lifecycle of a HEP experiment, addressing input data from many sources and in many different formats. The input data are often



(a)



(b)

Figure 6: a) A schematic drawing depicting the tree-structure of the data describing the geometry of the ATLAS detector. Such data structures are better visualized using graphs and networks. b) A graph visualizing the first layer of the nodes of the ATLAS Detector Description. Different colors indicate different types of nodes; also, the labels along the lines state the different types of relationship between two data nodes [66].

quite tightly bound to a specific experiment's software framework and because of

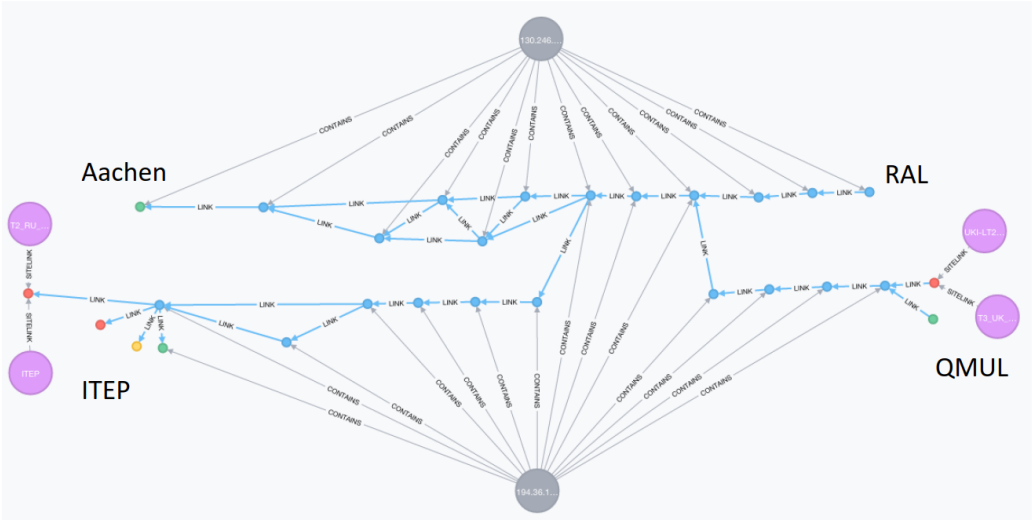


Figure 7: The image shows a graph used to visualize particular snapshot of the network topology between four WLCG sites, which was used to perform network path analysis. [Image provided by the authors of [68]].

that many visualization tools are integrated into the frameworks to some extent. However, the visualization is often the last step on the experiment data chain and the output of a visualization application is often not used by any other tool in the software framework. So while the direct interaction with experimental data formats is highly experiment specific, there is a real possibility of having the final stages of the visualization pipeline shared between several experiments.

Let us take the example of the detector geometry. There are very many different geometry libraries and formats in use among the HEP experiments. However, geometry libraries are all different ways to describe and handle basic geometrical entities and combinations of them. From a visualization point of view, the output of all geometry libraries are mere descriptions of 3D shapes, which could be abstracted from the underlying actual implementation. A shape like a box, or a cone, or a tube, or some boolean combination of them, could be interpreted and handled the same way by a visualization tool in all experiments.

The same reasoning made for the geometry can be done for the event data. Of course different experiments detect different objects and measure different quantities, but there are many common entities, especially among experiments within the same research field. For example, all experiments working on hadron colliders use the notion of particle track, which is usually constructed translating the track measurements into space points or points and angles, and use the notion of particle jet, usually visualized as a cone whose length is related to its energy and whose radius is linked to the specific algorithm used for the jet reconstruction. Both of these objects are currently often handled and visualized differently in different experiments,

but could in principle be the target of a common definition within the community. If so, experiments could share best practices or snippets of code, if not complete basic tools, to build and handle their visualization. In this way, the know-how and the tools linked to visualization needs could be shared as well, and developed in a collaborative way.

We as a community still need to propose and design such common definitions and guidelines. This will be addressed in the second phase of this Community action, following the completion of the Community White Paper.

For the moment, we observe that several event displays in experiments have exporters to translate geometry information to standard formats used in the communities outside HEP, mainly in computer graphics and engineering. For example the Belle II experiment has written exporters from Geant4 data to different formats, including VRML [70] and FBX [71], which are two of the most common formats used to store and share 3D graphics data. The Unity game-development engine [37], in turn, can export the FBX geometry to the glTF format [72], an emerging royalty-free specification for 3D objects and scenes, for fast web distribution via the SketchFab community repository [73, 74]. Displays based on three.js have access to multiple importers and exporters of several geometry formats, including the ones mentioned above.

In order to start sharing knowledge and to start working on demonstrators to show and share best practices, we propose to start defining a common format to exchange data among the experiments. We should start by finding and listing common shared objects from geometry and event data. After that, we should start converging on a shared definition of those objects, to build a common design toward a data model to handle and serve them. The idea, in fact, is to enable usage of this common format to visualize data from the different experiments with the same shared best practices, if not the same foundation software tools.

We think that community-developed common formats and tools should also be extendable, to let the experiments add their own custom content and objects. As an example, calorimeter cells can be of very different shapes, and an experiment might need to add its own custom shapes to the common format to visualize them properly. Thus, in addition to the part handling the common objects, there should be a part of the format targeted at storing extended custom content, specific to a given experiment. For such custom content, experiments will have to develop custom visualization tools as well; but they could build them upon the foundation of the community-driven part.

Some experiments in that direction have been performed within the community in the already. For example, the ALICE experiment made use of the mini "Visualization Summary Data" (VSD) set of classes, contained in the ROOT Event Visualization Environment (EVE), to make ALICE data visualization decoupled from the AliROOT experiment's framework [75].

3.2 Serving the geometry and event data through services

Once a common format for shared objects is defined, we believe that the design and the development of online services to query and serve the geometry data would be a very useful addition to the landscape. The main driving force is the realization that detector description should be much more accessible than it is today. For many experiments, accessing the detector description means starting and running at least parts of the experiment's framework. The need of accessing a specific geometry version, in fact, is critical for reconstruction and simulation. However, the geometry data needed for event visualization can often be simpler: even when showing the actual geometry of the experiment, accessing the latest alignment constants is not crucial for visualization purposes, because small differences in the geometry are rarely visible in an event display. So, we think that serving a "frozen" version of the experiment's geometry would be enough, and that a simpler way to retrieve it should be designed, to ease data access for visualization: for example, an online service could remotely serve the geometry data to the visualization applications in a standardized format.

It would be desirable for the new mechanism to have a search/filter functionality too, to let client applications query for a specific subset of information and for a way to select the level of details, to set the desired accuracy and complexity of the retrieved geometry.

The same could be envisaged for event data, even though that is a more complicated task, involving many different layers and services: very often event data are stored on the Grid and very often they need to be processed in order to be usable for event displays. However, experimental data should be made more accessible for visualization. Thus, in collaboration with the HSF *Data Access and Management* working group, an API or a service to get streamed event data will be designed. In addition, simulation data description for visualization could be handled in the same way, by implementing converters from generators and simulation applications.

After a first phase of development and stand-alone testing, the streamed data could be used by the current visualization tools as well, as the first step of their modernization and towards the usage of common community-developed techniques and best practices.

3.3 Client-server architecture for geometry and event data visualization

After common data formats and mechanisms to serve them are designed, together with a set of exporters required to translate the experiments' data to the common format, we are proposing to build a client-server architecture, upon which next generation visualization applications can be built.

The idea behind that is that if we can send commands from the client to the server, and get the answer back in the data stream, then we will be able to interact

with the experiment's framework as well, in addition to using common visualization applications to visualize the common objects. In this way, we can develop a modular architecture where HEP experiments can share the design, the development and the maintenance of common visualization tools, while maintaining a certain degree of freedom to add custom content and objects and to interact with their own framework to retrieve specific content.

3.4 Exploring modern technologies

3.4.1 Graphics and game engines

So far direct OpenGL, its derivatives (like WebGL), or old graphics libraries have been used in HEP visualization applications. Nowadays, another type of graphics library is rapidly evolving, those embedded in so-called game engines. Game engines are software frameworks targeted at the gaming industry, and they feature very efficient, optimized, and modern 3D graphics. Integration with existing code is not easy, however, because they are usually meant to be used as development environments, and not as embedded libraries like those commonly used in HEP. So they would probably require some major changes in the usual software architecture used in HEP. But they offer very optimized graphics and modern features, like tools for Virtual Reality, which could be exploited in our applications.

Some HEP experiments have recently started to successfully use them to build visualization applications and event displays, like Belle II [39] and the Total Event Visualiser (TEV) of the CERN Media Lab [76], which used the Unity game engine [37], and ATLAS which used the Unreal Engine [77] for its virtual reality application ATLASrift [38]. These two game engines are the most popular ones on the market, and they are free for educational and non-commercial projects.

Unreal Engine is fully open source and it supports two modes of development (C++ and the so-called Blueprints [78]) that can be used interchangeably even in the same project. It produces extremely performant executables for basically all platforms (Windows, OSX, Linux, iOS, Android, Web, all VR platforms). All parts of development cycle are fast even for a novice, thanks to powerful tools implemented as plugins. They have large developer communities and are very fast in supporting the latest technologies; for example it already supports Vulkan [79], the cross-platform 3D and computing API.

The Unity development platform [37] is very intuitive for novices as well as experts and provides rapid turnaround during the development cycle: the project can be executed immediately without having to compile and link an executable. All of the platforms supported by Unreal are supported as targets by Unity as well. Presently, user code is written in either C# or an adaptation of Javascript.

Another game engine that has gained attention in recent years is the open-source engine Godot [80]. While it is still not quite at the same level as Unity or Unreal

Engine, it allows the deployment to similar platforms as Unity and Unreal Engine but is very lightweight. It offers support for 2D and 3D graphics and for multiple programming languages *e.g.* GDScript (a Python-like scripting language), C# 7.0 (by using Mono), and C++. It also offers visual scripting using blocks and connections and support for additional languages with community-provided support for Python, Nim [81], D and other languages.

As a community, we would like to explore further the features those modern game engines can offer. Also, we would like to take a look at possible usage patterns in the context and within the workflow of HEP visualization.

Finally, another new entry in the 3D graphics engines landscape is Qt3D [82]. The key feature of Qt3D is that it is natively integrated with the Qt framework, which eliminates a layer which was needed until now, *i.e.*, a glue package to connect the Qt GUI with the window showing the 3D content (like, for example, the SoQt [83] package used by ATLAS). By eliminating that, we could simplify the architecture of our visualization tools and lower the maintenance workload. Qt3D is still in development, but it shows an initial set of features which are worth a further consideration of the new toolkit. We plan to take a look at its development in the near future, to see if it can satisfy the HEP requirements. Also, being open source, we could consider contributing to the Qt3D software project as a community, by providing the pieces we need for our applications.

3.4.2 Web-based applications

Web-based graphics have traditionally been considered not powerful enough to handle the thousands of volumes that can be shown in HEP event displays, for example, when visualizing hits in a very busy event. But the technology has rapidly evolved and web-based graphics can now visualize very complex scenes. For example, the glTF [72] 3D model of the Belle II detector [84], with tens of thousands of elements, can be loaded, viewed and manipulated in a web browser, even on a smartphone, very effectively [73].

The advantages of visualization in the browser have been mentioned previously and several application have already been developed. There is therefore already a strong interest in the community in supporting the usage and the development of web-based tools. In particular, JSROOT [51] could be used as an underlying layer for event data visualization as well as three.js [34], which have been used successfully by different experiments (*e.g.*, [28, 30, 31]) to visualize geometry and event data.

3.4.3 Virtual and augmented reality

As briefly described in Section 2.1.2, Virtual Reality (VR) describes the simulation of the user's physical presence in a virtual environment. This simulation is typically delivered via a Head Mounted Display (HMD) that provides visual and aural experience of the simulated world. Rotational and positional tracking of the user's

head and hand motion (when available) allow for interaction and navigation in the virtual environment. This lets the user live an immersive experience of the virtual world offering many degrees of freedom. Purely rotational motion is usually refered to as 3 degree-of-freedom motion; when combined with positional motion support, an application is said to support 6 degrees-of-freedom.

There are several ways to deliver VR to the user with varying levels of functionality, accessibility, and cost. They range from applications running on a mobile phone viewed through simple headsets to the most realistic and immersive VR experiences provided by the combination of advanced HMDs and desktop computers.

The simplest and most inexpensive way to deliver VR is via the web browser on a mobile phone and viewed through a Google Cardboard [85] headset (which can itself be literally made from cardboard). Rotational tracking is achieved through device orientation controls, either in a native application or using the HTML5 device orientation API in the browser. No hand controller is used in the Cardboard but for native applications a click event is available via a magnet attached to the Cardboard viewer.

The Google Daydream [86] is the next iteration of viewers developed by Google for their Google VR technology. Content is still delivered by a mobile phone (thus, the performance is limited by the computing power of the phone) but a Bluetooth-connected hand controller with rotational tracking is available. The Samsung Gear VR platform [87] is another headset powered by an inserted smartphone targeted for the Oculus platform.

Currently, the most immersive and interactive VR environments are provided by the Oculus Rift [88] and the HTC Vive [89], which combine the computing resources of a desktop machine with sensors, controllers, and high-quality HMDs. Those sophisticated devices are relatively expensive (even if prices are lowering in the last months) and require a fairly powerful computer to run. The presence of a proper computer solves the performance issue of phone-based viewers, since the 3D graphics computations are handled by the computer, but that also increases the required initial budget for people willing to test such technologies, and that limits the number of people getting access to such platforms, for example in public events organized by HEP institutes.

Meanwhile, a new type of device has been recently developed: a standalone viewer, equipped with an on-board CPU, able to run medium computation-intensive applications. Those devices lower the budget needed to develop and deploy VR applications significantly. Examples of those new devices are Oculus Go[90] and the stand-alone Lenovo headset for the Daydream environment [91].

Game engines, described in Section 3.4.1, provide powerful integrated development environments for creation of VR applications targeting multiple devices. Thanks to engines' abstraction of third party VR libraries, most HEP experiments should be able to develop VR applications which natively support both standard

displays and all VR hardware. Currently, both ATLASrift [38] and Belle II VR [39] support Oculus, HTC Vive, and standard 2D displays. CMS, using the Unity game engine, is currently working on CMS.VR (to be released), targeting Oculus and HTC.

Augmented Reality (AR) applications use the device's camera to looks at the world around the user, to use that as an underlying layer, over which, based on the user's interaction and motion, they dynamically render simulated virtual entities. The user can navigate in the real world, while looking at and interacting with the virtual objects. AR technology can be used in HEP as an educational tool, for instance to dynamically show and describe a HEP detector to a group of people or a class.

ATLAS and ALICE researchers have started to explore the possibilities offered by augmented reality for outreach and education, by using Unity [37] and the Vuforia framework [92] (commercial, but free for development). The ATLAS-in-Your-Pocket [93] application uses printed marks to place a rendered geometry of the ATLAS detector on top of a view of the real world in front of the user. The More-Than-ALICE [94] application allows users to superimpose a description of the detector or event visualizations to the camera image of the actual ALICE detector (for example, on a screen or during public visits to the experimental site) or its paper model.

Development of web-based VR and AR applications for mobile browser can be done using a WebGL library such as three.js [34] and using the HTML device orientation control API. The viewport is split into two views, one for each eye, each with a dedicated camera view separated by an appropriate distance to create a stereoscopic effect (*e.g.* iSpy WebGL [28] has a stereo mode for Google Cardboard). The developing WebVR specification [95] provides interfaces to VR hardware via the browser. A powerful framework for development of VR applications for various devices using the browser is A-Frame [96]. A-Frame has support for several device controllers as well, such as for the Daydream and Oculus. CMS is exploring the possibilites of A-Frame in the browser for both VR and AR (*e.g.* CMS A-Frame prototype [97]).

VR and AR applications could be used, in principle, to build event displays and data visualization tools for research work as well. However, the current data access model prevents a straightforward use of those technologies together with the experiments' software frameworks. A future client-server approach to data access could fill the current gap and let developers build new VR and AR tools targeting HEP physicists as end users.

3.4.4 Mobile technologies

Portability and simplicity of usage are the strong points of mobile devices. More than as "mobile" devices, smartphones, tablets and ultrabooks can be considered, as devices "close to people". As such, the usage of such devices should be exploited more in the final steps of the visualization chain, where heavy batch data processing is not needed. For instance, their usage should be leveraged for the production

and visualization of event displays. Ideally, a user should be able to easily retrieve interesting events from the experiment and interactively visualize them on all kinds of devices.

That is why we strongly promote the usage of the server-client architecture described and supported in this paper in Section 3.3 and the new data access patterns presented and supported in the the HSF *Data Access and Management* Community White Paper [98]. This would open up new possibilities for interactive visualization on mobile devices: it would let visualization clients running on mobile devices connect to server tools running in the experiment's framework to easily and interactively retrieve the desired data.

It is worth noting that in other areas of science, for instance in astronomy, researchers have worked to facilitate data access and to migrate to more standard data formats. This allowed for the possibility of having data visualization tools on mobile devices, in addition to desktop and laptop machines. Moreover, this not only helped the researchers in accessing and visualizing their data, but it also helped in making science accessible by the public, having eased the development of programs used in Outreach and Education activities and events. It is true that HEP data are usually much more complex than astronomy data, and so it will be harder to achieve, but we think that an effort in simplifying the access to experimental data would be worthwhile.

Therefore, the leverage of the usage of mobile devices in HEP adds a strong point to the development and the support of common client-server tools and data exchange formats among HEP experiments in the near future.

3.4.5 Multi-user applications

Nowadays multi-user technology is used in many applications: for example in Google-Docs, where many users can simultaneously interact with the same document. What we would like to provide is multi-user support for visualization to let several users explore and interact with events at the same time. Beside being a useful feature for expert users (for example, when asking for advice on the visualization of a piece of detector to another person at a distant institute), it could be important for outreach and education activities, where students and other people could interact together with an event display, or for virtual guided tours. Game engines offer multi-user support natively, thus we could starting explore their usage. An example of such collaborative features in a 3D environment is integrated in the Med3D visualization framework [99].

4 Sharing knowledge and fostering collaboration

During the kick-starter meetings and the different workshops organized to start and develop the present Community White Paper, the whole HSF Visualization Working

Group has agreed on the importance of sharing knowledge among the whole HEP community, as well as best practices and know-how. Too often, in fact, solutions and tools developed for one HEP experiment are not sufficiently advertised to the rest of the HEP Visualization community, with the result that the community base knowledge is fragmented and not efficiently exploited.

The focus of this Working Group in fact is not limited to the preparation of this white paper. Instead, a longer term collaboration among the experiments is foreseen, in order to collaborate on common visualization projects. To foster collaboration and sharing, we agree on following courses-of-action which are described below.

4.1 Yearly workshop

On March 2017 the first HSF Visualization Workshop was organized at CERN to let all the experts from the different experiments and projects show their work and share their solutions. In addition, external experts from industry were invited to present the latest advancements in the field and best practices [100].

It was the first topical workshop focused on HEP Visualization in many years. Many HEP experiments and communities showed their latest developments. Given the high number of presentations, a second mini-workshop [101] has been organized as a follow-up, to let the remaining communities to present their work.

The Working Group agreed on the importance of meeting to share findings, knowledge and solutions; and it was decided to try to organize a topical workshop on HEP visualization once per year.

An important point was raised while organizing the first Workshop: other scientific fields have visualization and graphics needs similar to HEP, for example Geophysics. In future workshops we will try to have presentations from other communities as well, in order to try to foster friendly and fruitful collaborations which could benefit the whole scientific community.

4.2 Code repository

The HSF Visualization Working Group also agreed on the importance of fostering collaborative work. As a start a new dedicated project has been created within the HSF GitHub repository [102]. It is intended to be the space where members of the Working Group can share their work-in-progress studies and their solutions, and where community-driven projects will be stored.

5 Roadmap

5.1 One year

In the first year the Visualization Working Group will work on defining R&D projects, based on the key points and ideas discussed in this community white paper.

The main goal will be developing techniques and tools which let visualization applications and event displays be less dependent on specific experiments' software frameworks, leveraging the usage of common packages and common data formats.

In a first phase, the community will identify common objects and will agree on common definitions, as described in section 3.1. Then, a common data exchange format, either based on custom data formats or, if possible, on open standards, will be designed by the community. After that, exporters and interface packages would be designed as bridges from the experiments' frameworks, which are needed to access data at a high level of detail, to the common packages.

5.2 Three years: ATLAS and CMS Computing TDRs

In the second and third year the Visualization WG will work on designing and building demonstrators to show the feasibility of the community-driven best practices and tools. The goal will be to get a final design of those tools, to be included in the development plans of the different experiments. Moreover, the WG will work towards a more convenient access to geometry and event data. In collaboration with the HSF *Data Access and Management* working group, an API or a service to get streamed event data would be designed.

5.3 Five years: Towards HL-LHC

In the fourth and fifth year, the focus will be on developing the actual community-driven tools, to be used by the experiments for their visualization needs in production.

The goal will be the usage of the community-developed tools within the experiments' visualization applications; and perhaps the usage of a simplified data access, but that depends on the actual feasibility, which will be established after an initial study.

6 Conclusions

Modern and modular visualization tools, which will feature simplified data access and retrieval as well, would leverage the accessibility, letting end users exploit all the possibilities offered by modern visualization solutions, without the need of running them on specific platforms, running them within the experiments' software frameworks, or being bound to specific solutions. And a better experience will reflect to a better usage, which will positively affect the usage of such visualization tools for detector development and simulation of new experiments, as well as the data analysis and the upgrade studies of the current ones.

In the end, common community-driven tools will let users of all experiments make use of the latest and best tools, while sharing the development, the maintenance and the workload among all the experiments.

7 Acknowledgements

The authors would like to thank Guy Barrand *(Geant4, LSST, LAL)* for the fruitful discussions and input about what is done in astronomy in terms of data access and visualization on mobile platforms.

References

[1] Antonio Augusto Alves Jr et al. *A Roadmap for HEP Software and Computing R&D in the 2020s*. Tech. rep. HSF-CWP-2017-01. HEP Software Foundation, 2017. arXiv: 1712.06982 [physics.comp-ph].

[2] The HEP Software Foundation. *HEP Software Foundation Community White Paper Working Group – Data Analysis and Interpretation*. Tech. rep. HSF-CWP-2017-05. HEP Software Foundation, 2017. arXiv: 1804.03983 [physics.comp-ph].

[3] The HEP Software Foundation. *HEP Software Foundation Community White Paper Working Group – Detector Simulation*. Tech. rep. HSF-CWP-2017-07. HEP Software Foundation, 2017. arXiv: 1803.04165 [physics.comp-ph].

[4] *ATLAS Collaboration - 13 TeV Stable Beam Collisions*. URL: https://twiki.cern.ch/twiki/pub/AtlasPublic/EventDisplayRun2Collisions/JiveXML_271298_403602858-RZ-LegoPlot-EventInfo-RZ-YX-2015-08-06-15-01-42.png.

[5] T. Kittelmann et al. "The Virtual Point 1 event display for the ATLAS experiment". In: *Journal of Physics: Conference Series* 219.3 (2010). URL: https://atlas-vp1.web.cern.ch/atlas-vp1.

[6] CMS Collaboration. *Higgs boson produced via vector boson fusion event recorded by CMS (Run 2, 13 TeV)*. URL: http://cds.cern.ch/record/2210658/.

[7] ALICE Collaboration. *One of the first Xenon-Xenon collisions at LHC registered by ALICE*. URL: https://cds.cern.ch/record/2289018.

[8] LHCb Collaboration. *Event collected at the beginning of 2018 data taking*. URL: http://cds.cern.ch/record/2315673.

[9] Leo Piilonen & Belle II Collaboration. *Belle II in Virtual Reality*. URL: http://www1.phys.vt.edu/~piilonen/VR/.

[10] L.A.T. Bauerdick et al. "Event display for the visualization of CMS events". In: *J.Phys.Conf.Ser.* 331.072039 (2011). URL: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookFireworks.

[11] KHRONOS Group. *OpenGL*. URL: https://www.opengl.org/about.

[12] Microsoft. *Direct3D*. URL: https://docs.microsoft.com/en-us/windows/desktop/direct3d.

[13] L Chevalier et al. *PERSINT Event Display for ATLAS*. Tech. rep. ATL-SOFT-PUB-2012-001. ATLAS Experiment, CERN, 2012. URL: https://twiki.cern.ch/twiki/bin/viewauth/Atlas/PersintWiki.

[14] *ROOT-EVE*. URL: https://root.cern.ch/eve.

[15] Matevz Tadel et al. "ALICE Event Visualization Environment". In: *CHEP 2006*. 2006. URL: https://indico.cern.ch/event/408139/contributions/979909/.

[16] ATLAS Collaboration. *A heavy ion collision with a candidate $Z \to \mu^+\mu^-$ decay*. URL: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/EventDisplayHeavyIonCollisions#Heavy_Ion_Collisions_with_a_Z_Ca.

[17] ATLAS Collaboration. *Display of a candidate Higgs boson event from proton-proton collisions recorded by ATLAS with LHC stable beams at a collision energy of 13 TeV*. URL: https://twiki.cern.ch/twiki/bin/view/AtlasPublic/EventDisplayRun2Physics#H_2e2_candidate_event_record_AN4.

[18] ATLAS Collaboration. *The ATLAS Atlantis visualization tool*. URL: http://www.cern.ch/atlantis.

[19] J. Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2*. Addison-Wesley, 1993.

[20] G. Alverson et al. "IGUANA: a high-performance 2D and 3D visualisation system". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 534.1 (2004). Proceedings of the IXth International Workshop on Advanced Computing and Analysis Techniques in Physics Research, pp. 143–146. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2004.07.036. URL: http://www.sciencedirect.com/science/article/pii/S0168900204014950.

[21] CMS Collaboration. *Interactive Graphics for User Analysis*. URL: http://iguana.web.cern.ch/iguana/.

[22] *The Coin3D graphics engine*. URL: https://bitbucket.org/Coin3D/coin/wiki/Home.

[23] LHCb Collaboration. *The LHCb Panoramix visualization tool*. URL: http://lhcb-comp.web.cern.ch/lhcb-comp/frameworks/Visualization/.

[24] G. Alverson et al. "iSpy: A powerful and lightweight event display". In: *J.Phys.Conf.Ser.* 396.022002 (2012).

[25] J. Boudreau and V. Tsulaia. "The GeoModel Toolkit for Detector Description". In: *CHEP 2004*. 2004. URL: https://cds.cern.ch/record/865601/.

[26] *Coin3D "End of life letter"*. URL: https://bitbucket.org/Coin3D/coin/wiki/EndOfLifeLetter.

[27] ATLAS Collaboration. *The ATLAS Minerva visualization tool*. URL: http://cern.ch/atlas-minerva.

[28] T. McCauley. "A browser-based event display for the CMS experiment at the LHC using WebGL". In: *J.Phys.Conf.Ser.* 898.072030 (2017). URL: http://cern.ch/ispy-webgl.

[29] *LHCb Online event display*. URL: https://lhcb-public.web.cern.ch/lhcb-public/lbevent2/lbevent/.

[30] G. Sabato et al. "ATLAS fast physics monitoring: TADA". In: *J.Phys.Conf.Ser.* 898.092015 (2017). DOI: https://doi.org/10.1088/1742-6596/898/9/092015.

[31] ATLAS Collaboration. *The ATLAS Tracer visualization tool*. URL: https://atlas-tracer.web.cern.ch/.

[32] KHRONOS Group. *WebGL - OpenGL ES for the Web*. URL: https://www.khronos.org/webgl/.

[33] *The Qt software development framework*. URL: https://www.qt.io/.

[34] *The ThreeJS 3D graphics framework*. URL: https://threejs.org/.

[35] *LHSee*. URL: https://www2.physics.ox.ac.uk/about-us/outreach/public/lhsee.

[36] *CERN Camelia visualization tool*. URL: http://medialab.web.cern.ch/content/camelia.

[37] *Unity Technologies - The Unity3D game engine*. URL: https://unity3d.com.

[38] Ilija Vukotic & ATLAS Collaboration. *ATLASRift, the ATLAS VR experience*. URL: https://atlasrift.web.cern.ch/.

[39] Leo Piilonen. *Belle II in Virtual Reality*. URL: http://www1.phys.vt.edu/~piilonen/VR/.

[40] Tai Sakuma & CMS Collaboration. *3D SketchUp images of the CMS detector*. URL: https://cds.cern.ch/record/2628527.

[41] R. Brun and F. Rademakers. "ROOT — An object oriented data analysis framework". In: *Nuclear Instruments and Methods A* 389 (1997). DOI: http://dx.doi.org/10.1016/S0168-9002(97)00048-X.

[42] *DD4HEP*. URL: https://dd4hep.web.cern.ch/dd4hep/.

[43] The CLICdp Collaboration. *CLIC detector and physics study*. URL: http://clicdp.web.cern.ch/.

[44] The Future Circular Collider Study Collaboration. *Future Circular Collider Study*. URL: https://fcc.web.cern.ch/Pages/default.aspx.

[45] J Allison et al. "Recent developments in Geant4". In: *Nuclear Instruments and Methods in Physics Research A* 835 (2016). URL: http://geant4.cern.ch/.

[46] Trimble Inc. "SketchUp". In: (). URL: https://www.sketchup.com/.

[47] T. Sakuma and T. McCauley. "Detector and Event Visualization with SketchUp at the CMS Experiment". In: *J.Phys.Conf.Ser.* 513.022032 (2014). DOI: 10.1088/1742-6596/513/2/022032.

[48] M. Case et al. "CMS Detector Description : New Developments". In: *Computing in High Energy Physics and Nuclear Physics 2004*. 2004. DOI: 10.5170/CERN-2005-002.498.

[49] J. W. Tukey. "The Future of Data Analysis". In: *The Annals of Mathematical Statistics* 33.1 (1962), pp. 1–67. URL: http://www.jstor.org/stable/2237638.

[50] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1986.

[51] *ROOTJS*. URL: https://root.cern.ch/js/.

[52] Project Jupyter. *The Jupyter Notebook*. URL: http://jupyter.org/.

[53] CERN. *PyROOT*. URL: https://root.cern.ch/pyroot.

[54] Jake VanderPlas. *Python's Visualization Landscape (PyCon 2017)*. URL: https://speakerdeck.com/jakevdp/pythons-visualization-landscape-pycon-2017.

[55] J.D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007). DOI: http://dx.doi.org/10.1109/MCSE.2007.55.

[56] *Seaborn*. URL: https://seaborn.pydata.org.

[57] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. New York: Springer-Verlag, 2009. DOI: 10.1007/978-0-387-98141-3.

[58] D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag, 2008. ISBN: ISBN 978-0-387-75968-5. DOI: 10.1007/978-0-387-75969-2.

[59] L. Wilkinson. *The Grammar of Graphics*. Springer, 2005. ISBN: ISBN 978-0387245447. DOI: 10.1007/0-387-28695-0.

[60] R. A. Becker and W. S. Cleveland. *S-PLUS Trellis Graphics User's Manual*. MathSoft, 1996. URL: http://ect.bell-labs.com/sl/project/trellis/software.writing.html.

[61] Marc Paterno. "Easy Data Analysis Using R". In: *PhyStat05*. Oxford, 2005. URL: http://www.physics.ox.ac.uk/phystat05/Talks/PhyStat05-paterno.pdf.

[62] M. Bostock et al. "D3: Data-Driven Documents". In: *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*. 2011. URL: http://vis.stanford.edu/papers/d3.

[63] *Bokeh*. URL: http://www.bokeh.pydata.org.

[64] *Continuum*. URL: https://www.continuum.io/.

[65] *Plotly*. URL: https://plot.ly.

[66] R.M. Bianchi, J. Boudreau, and I. Vukotic. "A new experiment-independent mechanism to persistify and serve the detector geometry of ATLAS". In: *J. Phys.: Conf. Ser.* 898.072015 (2017). DOI: https://doi.org/10.1088/1742-6596/898/7/072015.

[67] The HEP Software Foundation. *HEP Software Foundation Community White Paper Working Group – Event/Data Processing Frameworks*. Tech. rep. HSF-CWP-2017-08. HEP Software Foundation, 2017.

[68] S. McKee et al. "Integrating network and transfer metrics to optimize transfer efficiency and experiment workflows". In: *J. Phys. Conf. Ser.* 664 (2015), p. 052003. DOI: 10.1088/1742-6596/664/5/052003.

[69] M. Clemencic et al. "LHCb conditions database operation assistance systems". In: *J. Phys. Conf. Ser.* 396 (2012), p. 052022. DOI: 10.1088/1742-6596/396/5/052022.

[70] *VRML (Virtual Reality Modeling Language)*. URL: https://en.wikipedia.org/wiki/VRML.

[71] Belle II Collaboration. *The Belle II Geometry Exporters*. URL: https://github.com/HSF/Visualization/tree/master/demonstrators/GeometryWriters.

[72] *KHRONOS Group — glTF - The GL Transmission Format*. URL: https://www.khronos.org/gltf/.

[73] *BelleII VR models on Sketchfab*. URL: https://sketchfab.com/models/6c7aa0c71dc64079b08a0dddfa756ef3.

[74] *Sketchfab*. URL: https://sketchfab.com.

[75] Matevz Tadel. "ALICE Event Visualization Environment". In: *CHEP 2006*. Proceedings of the Computing in High Energy and Nuclear Physics conference (Tata Institute of Fundamental Research, Feb. 13–17, 2006). Mumbai, India, 2006. URL: https://indico.cern.ch/event/408139/contributions/979909/.

[76] *CERN TEV visualization framework*. URL: https://gitlab.cern.ch/CERNMediaLab/.

[77] *Epic Games - The "Unreal Engine" game engine*. URL: https://www.unrealengine.com.

[78] Epic Games, Inc. *Blueprints Visual Scripting*. URL: https://docs.unrealengine.com/en-us/Engine/Blueprints.

[79] The Khronos Group Inc. *The Vulkan API*. URL: https://www.khronos.org/vulkan/.

[80] Juan Linietsky, Ariel Manzur, and contributors. *The Godot game engine*. URL: https://godotengine.org/.

[81] Andreas Rumpf. *The Vulkan API*. URL: https://nim-lang.org/.

[82] *The Qt 3D framework*. URL: https://doc.qt.io/qt-5.11/qt3d-index.html.

[83] *The SoQt package*. URL: https://bitbucket.org/Coin3D/soqt.

[84] *The Belle II Experiment*. URL: http://belle2.jp.

[85] Google. *Google Cardboard*. URL: https://vr.google.com/cardboard/.

[86] Google. *Google Daydream*. URL: https://vr.google.com/daydream/.

[87] Samsung Electronics. *Samsung Gear VR*. URL: https://www.samsung.com/global/galaxy/gear-vr.

[88] LLC. Oculus VR. *Oculus Rift*. URL: https://www.oculus.com/rift/.

[89] HTC Corporation. *HTC Vive*. URL: https://www.vive.com/eu/.

[90] LLC. Oculus VR. *Oculus Go*. URL: https://www.oculus.com/go/.

[91] Lenovo Group Ltd. Google. *Lenovo Mirage Solo*. URL: https://vr.google.com/daydream/standalonevr/.

[92] PTC Inc. *Vuforia AR framework*. URL: https://www.vuforia.com/engine.html.

[93] Steffen Henkelmann (University of British Columbia, CA) & ATLAS Collaboration. *ATLAS in your pocket — An Augmented Reality application*. URL: https://atlasinyourpocket.web.cern.ch/.

[94] Jeff Ouellette. "More Than ALICE: Development of an augmented reality mobile application for the ALICE detector". In: (Aug. 2016). URL: https://cds.cern.ch/record/2207593.

[95] *WebVR API*. URL: https://immersive-web.github.io/webvr/.

[96] *AFrame - A web framework for building virtual reality experiences*. URL: https://aframe.io/.

[97] *CMS AFrame*. URL: https://github.com/HEPPanoramic/cms-aframe.

[98] The HEP Software Foundation. *HEP Software Foundation Community White Paper Working Group – Data Organisation, Management and Access*. Tech. rep. HSF-CWP-2017-04. HEP Software Foundation, 2017.

[99] P. Lavrič, C. Bohak, and M. Marolt. "Collaborative view-aligned annotations in web-based 3D medical data visualization". In: *MIPRO 2017: 40th Jubilee International Convention*. Opatija, Croatia, May 2017, pp. 276–280. URL: http://lgm.fri.uni-lj.si/wp-content/uploads/2017/10/1537435587.pdf.

[100] HSF. "HSF Visualization Workshop". In: CERN, 2017. URL: https://indico.cern.ch/event/617054/.

[101] HSF. "HSF Mini-workshop on GUI, graphic libraries, interactive visualization". In: CERN, 2017. URL: https://indico.cern.ch/event/628675/.

[102] HSF Visualization Working Group GitHub repository. URL: https://github.com/HEP-SF/Visualization.