

<sup>1</sup><sup>2</sup>

# <sup>3</sup> **HEP Software Foundation Community White** <sup>4</sup> **Paper Working Group – Visualization**

---

<sup>5</sup> **HEP Software Foundation:** Matthew Bellis<sup>a,b</sup> Riccardo Maria Bianchi<sup>c,1</sup>  
<sup>6</sup> Sebastien Binet<sup>d</sup> Ciril Bohak<sup>e</sup> Benjamin Couturier<sup>f</sup> Hadrien Grasland<sup>g</sup> Oliver  
<sup>7</sup> Gutsche<sup>h</sup> Sergey Linev<sup>i</sup> Alex Martyniuk<sup>j</sup> Thomas McCauley<sup>k,1</sup> Edward Moyse<sup>l</sup>  
<sup>8</sup> Alja Mrak Tadel<sup>m</sup> Mark Neubauer<sup>n</sup> Jeremi Niedziela<sup>f</sup> Leo Piilonen<sup>p</sup> Jim  
<sup>9</sup> Pivarski<sup>q</sup> Martin Ritter<sup>r</sup> Tai Sakuma<sup>s</sup> Matevz Tadel<sup>m</sup> Barthélémy von Haller<sup>f</sup>  
<sup>10</sup> Ilija Vukotic<sup>t</sup> Ben Waugh<sup>j</sup>

<sup>11</sup> *<sup>a</sup>Siena College, Loudonville NY, USA*

<sup>12</sup> *<sup>b</sup>Cornell University, Ithaca NY, USA*

<sup>13</sup> *<sup>c</sup>University of Pittsburgh, Pittsburgh PA, USA*

<sup>14</sup> *<sup>d</sup>CNRS/IN2P3, Clermont-Ferrand, France*

<sup>15</sup> *<sup>e</sup>University of Ljubljana, Ljubljana, Slovenia*

<sup>16</sup> *<sup>f</sup>CERN, Geneva, Switzerland*

<sup>17</sup> *<sup>g</sup>LAL, Université Paris-Sud and CNRS/IN2P3, Orsay, France*

<sup>18</sup> *<sup>h</sup>FNAL, Batavia IL, USA*

<sup>19</sup> *<sup>i</sup>GSI Darmstadt, Germany*

<sup>20</sup> *<sup>j</sup>University College London, London, UK*

<sup>21</sup> *<sup>k</sup>University of Notre Dame, Notre Dame IN, USA*

<sup>22</sup> *<sup>l</sup>University of Massachusetts, Amherst MA, USA*

<sup>23</sup> *<sup>m</sup>University of California at San Diego, San Diego CA, USA*

<sup>24</sup> *<sup>n</sup>University of Illinois, IL, USA*

<sup>25</sup> *<sup>p</sup>Virginia Tech, VA, USA*

<sup>26</sup> *<sup>q</sup>Princeton University, Princeton PA, USA*

<sup>27</sup> *<sup>r</sup>LMU Munich, Munich, Germany*

<sup>28</sup> *<sup>s</sup>University of Bristol, Bristol, UK*

<sup>29</sup> *<sup>t</sup>University of Chicago, Chicago IL, USA*

---

<sup>1</sup>Paper Editors

<sup>30</sup> ABSTRACT: In modern High Energy Physics (HEP) experiments visualization of ex-  
<sup>31</sup> perimental data has a key role in many activities and tasks across the whole data  
<sup>32</sup> chain: from detector development to monitoring, from event generation to recon-  
<sup>33</sup> struction of physics objects, from detector simulation to data analysis, and all the  
<sup>34</sup> way to outreach and education. In this paper the definition, status, and evolution  
<sup>35</sup> of data visualization for HEP experiments will be presented. Suggestions for the  
<sup>36</sup> upgrade of data visualization tools and techniques in current experiments will be  
<sup>37</sup> outlined, along with guidelines for future experiments. This paper expands on the  
<sup>38</sup> summary content published in the HSF *Roadmap* Community White Paper [1].

---

39	<b>Contents</b>	
40	<b>1 Scope</b>	<b>3</b>
41	<b>2 Current landscape</b>	<b>4</b>
42	2.1 Event displays	4
43	2.1.1 Data access	4
44	2.1.2 Application development and distribution	6
45	2.1.3 Geometry description and visualization	11
46	2.2 Statistical data visualization	12
47	2.2.1 Desktop solutions	13
48	2.2.2 Web-based solutions	14
49	2.2.3 Issues	15
50	2.3 Non-spatial visualization	15
51	<b>3 Suggested guidelines and future development</b>	<b>16</b>
52	3.1 A common community-defined format	16
53	3.2 Serving the geometry and event data through services	20
54	3.3 Client-server architecture for geometry and event data visualization	20
55	3.4 Exploring modern technologies	21
56	3.4.1 Graphics and game engines	21
57	3.4.2 Web-based applications	22
58	3.4.3 Virtual and augmented reality	22
59	3.4.4 Mobile technologies	24
60	3.4.5 Multi-user applications	25
61	<b>4 Sharing knowledge and fostering collaboration</b>	<b>25</b>
62	4.1 Yearly workshop	26
63	4.2 Code repository	26
64	<b>5 Roadmap</b>	<b>26</b>
65	5.1 One year	26
66	5.2 Three years: ATLAS and CMS Computing TDRs	27
67	5.3 Five years: Towards HL-LHC	27
68	<b>6 Conclusions</b>	<b>27</b>
69	<b>7 Acknowledgements</b>	<b>28</b>



71 **1 Scope**

Visualization: Turning numbers  
into pixels

72

---

*Hadrien Grasland*  
*HSF Workshop 2017, Annecy*

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

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

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

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

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

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

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

## 108 2 Current landscape

### 109 2.1 Event displays

110 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  
111 the data collections belonging to a particular event. Data can be related to the actual  
112 physics events (*e.g.* a collection of reconstructed physics objects, like jets and tracks)  
113 or to the experimental conditions (*e.g.* different versions of the detector description  
114 and calibration data).

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

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

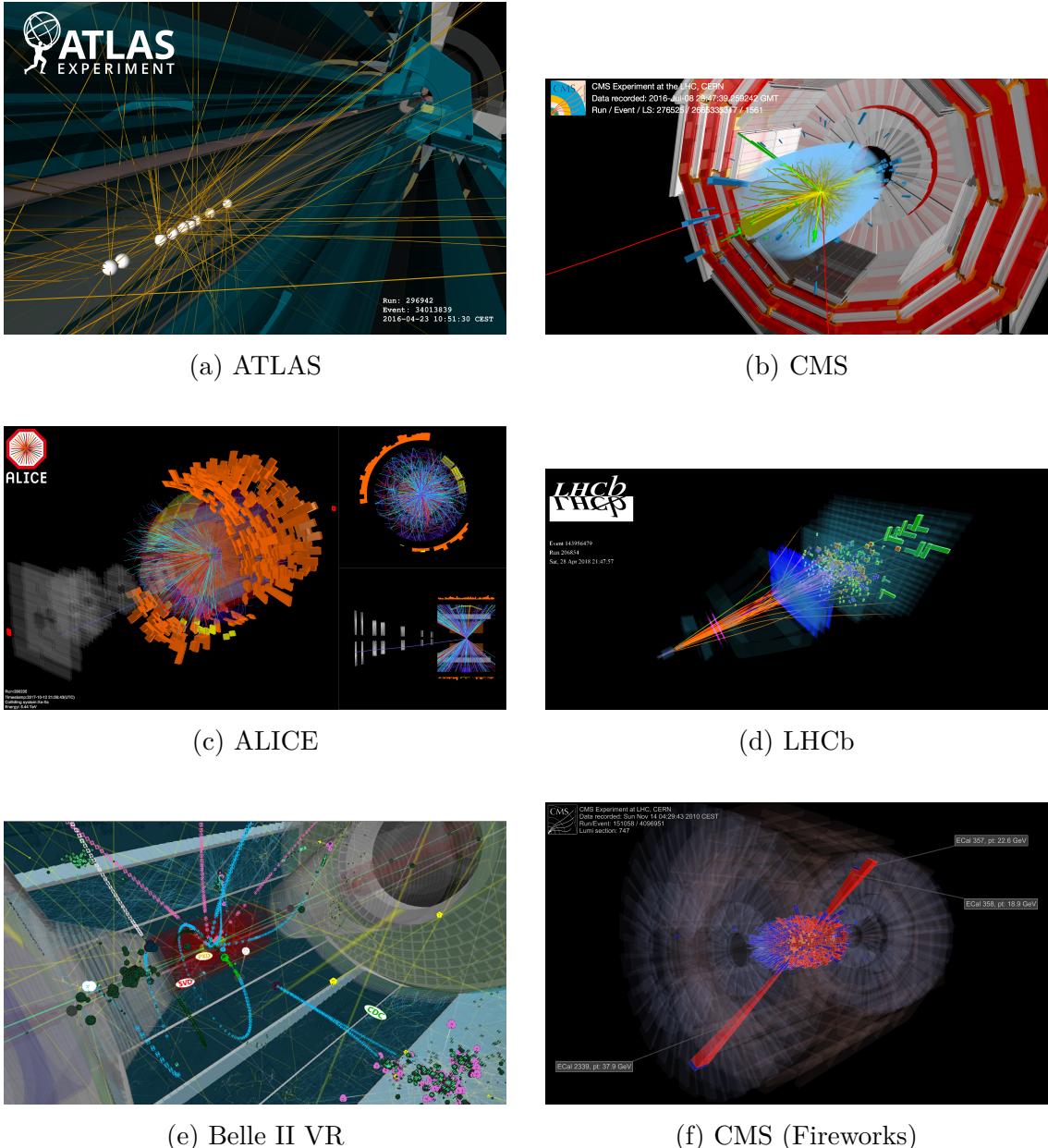
125 In the following subsections several important aspects of data access, application  
126 development and distribution, and geometry description and visualization, as they  
127 pertain to the current landscape of event displays, are discussed in more detail.  
128 Screenshots of several event displays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

#### 129 2.1.1 Data access

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

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

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



**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-

148 framework application is geared towards the expert.

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

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

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

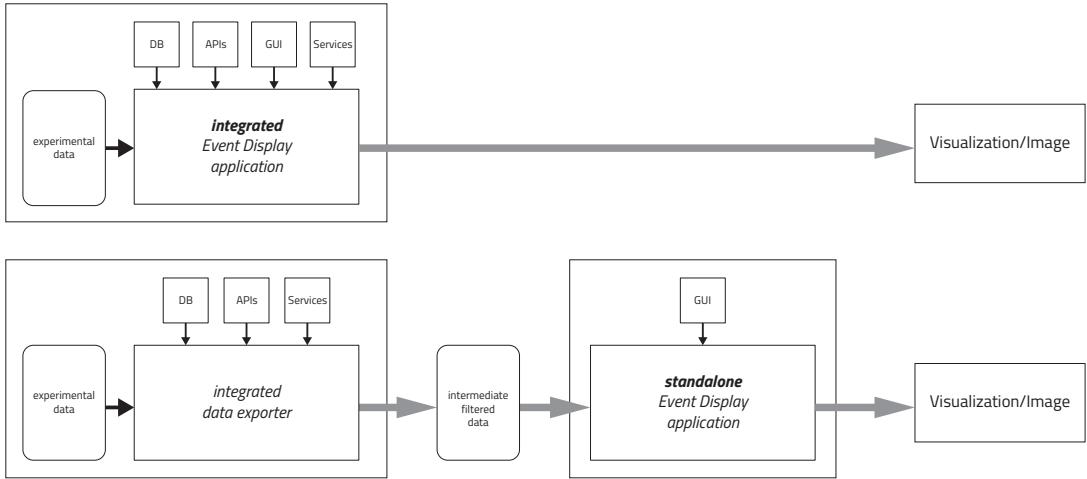
167 Regardless of the approach to data access one must consider the use-case: what is  
168 useful or necessary in one use-case may not be for another. A graphical representation  
169 of the two approaches can be seen in Figure 2.

### 170 2.1.2 Application development and distribution

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

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

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

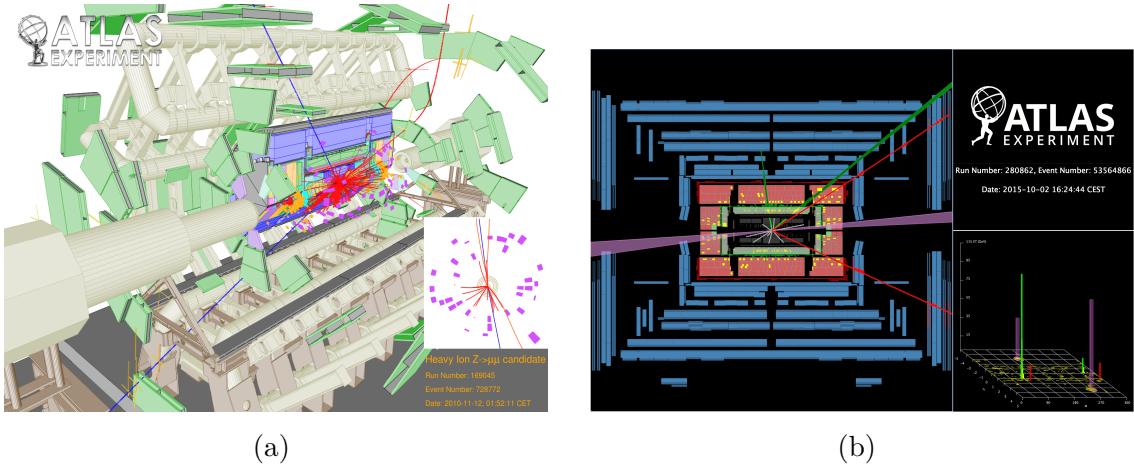


**Figure 2:** The two different types of event display applications. At the top, the framework-integrated event display application is able to access all experimental data, all services, APIs, and databases provided by the experiment’s framework; as a drawback, the application must be run on specific 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.

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

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

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

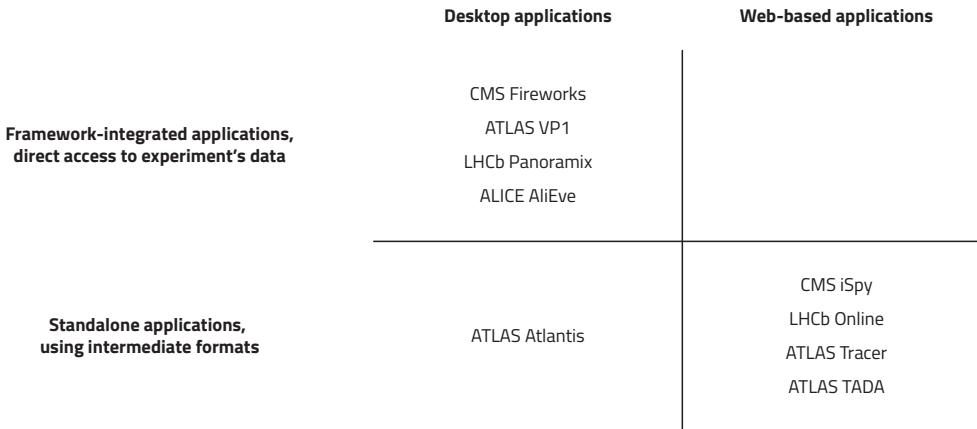


**Figure 3:** a) an ATLAS 3D event display made with Persint [13, 16]; b) an ATLAS 2D event display made with Atlantis [17, 18].

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 in 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.

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

228 **Web-based applications** Several experiments at the LHC, notably CMS [28],  
 229 LHCb [29], and ATLAS [30, 31] have created web-based event displays using We-  
 230 bGL (Web Graphics Library) [32]. WebGL is a JavaScript API that conforms to  
 231 OpenGL ES (a subset of the OpenGL API for embedded systems) conceived for  
 232 rendering interactive 3D and 2D graphics within any compatible web browser with-  
 233 out the need of external plug-ins. With WebGL in the browser one can combine  
 234 high-quality graphics with the functionality and accessibility of the browser. This  
 235 combination of graphics and user function was previously only available via bespoke  
 236 desktop applications based on OpenGL and graphical user interface toolkits such  
 237 as Qt [33]. Browser-based event displays have several distinct advantages: they are  
 238 easy to distribute to the user, they can be prototyped quickly, and the client is often  
 239 much lighter-weight, as the need for building, packaging and distributing external  
 240 libraries is greatly reduced. There are also several mature and actively developed  
 241 WebGL frameworks, such as the popular three.js [34], that provide straightforward  
 242 and simplified APIs for ease of development.

243 **Mobile applications** Mobile devices such as smart phones and tablets are more  
 244 and more ubiquitous and these devices are used more and more as substitutes for  
 245 desktop and laptop machines. However, mobile devices still do not have the comput-  
 246 ing power usually needed for HEP data analysis, where huge amount of experimental  
 247 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.

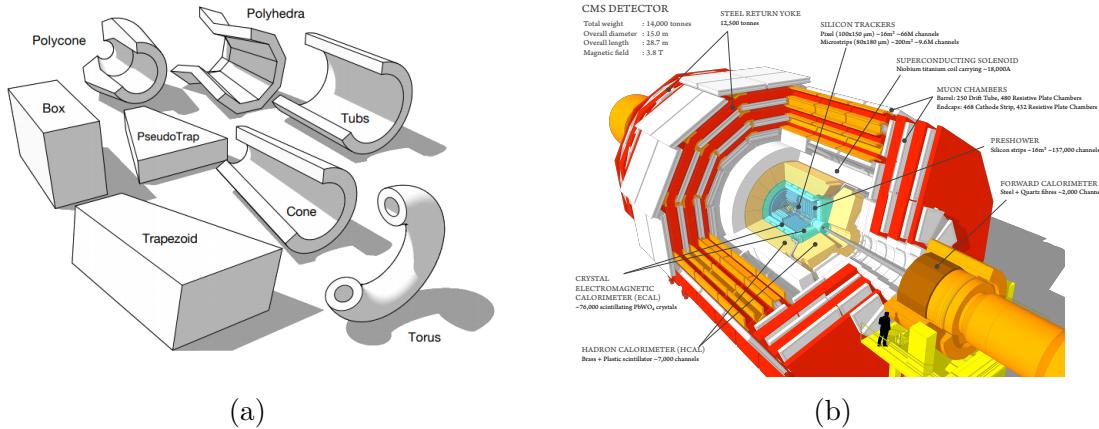
Several event display applications have been developed for, or at least can be run on, mobile devices. An example of a native application is LHSee [35] which live-streams ATLAS events, processed and extracted through Atlantis [18], to a user’s phone and provided contextual information on ATLAS and the 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.



**Figure 5:** Use of a 3D software (SketchUp) to render the basic geometrical shapes imported from the CMS detector description and the final volumes [48].

### 289 2.1.3 Geometry description and visualization

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

301 Currently, different geometry formats and libraries are used in HEP. Some ex-  
 302 periments use their own custom format (*e.g.*, [25]), while others use the geome-  
 303 try tools provided by the ROOT framework [40]. More recently, some attempts  
 304 have been made to build common formats and libraries for detector geometry, like  
 305 DD4HEP [41], adopted in conceptual design studies for future high-energy colliders,  
 306 including the CLICdp [42] and FCC [43] collaborations. In all cases detector volumes  
 307 are built from simpler geometrical entities: geometrical shapes like Tube, Cone, Box,  
 308 and more complex variations of these are combined in order to build the volumes of  
 309 the experiment’s geometry. Geometry libraries and formats are also described in the  
 310 HSF *Detector Simulation* Community White Paper [3].

311 The differences in geometry formats used by the different experiments, by de-  
 312 tector simulation programs like Geant4 [44], and by data analysis frameworks like  
 313 ROOT, typically require developers of visualization applications to write converters  
 314 between the different formats. In addition it is often not easy to use a visualization

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

317 In framework-based applications the geometry information can come directly  
318 from the experiment’s detector description and in many cases the hierarchical struc-  
319 ture of the detector description is preserved and accessible. Standalone applications  
320 typically use a geometry file with information exported from the software framework.  
321 An example hybrid solution is the one developed for the CMS experiment using the  
322 SketchUp application [45], described in Ref. [46] (see Figure 5). The CMS detector  
323 description [47] as written in XML is parsed using Ruby scripts and 3D models are  
324 built using the SketchUp program via its Ruby API. SketchUp can then export to  
325 various standard 3D file formats. In this way detailed simulation geometry can be  
326 available in a standalone application.

## 327 2.2 Statistical data visualization

328

The simple graph has brought more  
information to the data analyst’s  
mind than any other device.

---

John Tukey [49]

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

332 In HEP, like most other scientific disciplines, visualization of the data and of the  
333 final results plays a key role in the analysis pipeline. A new projection of the data  
334 may provide new insight, results must be summarized in a clear and concise way,  
335 and multidimensional parameter spaces need to be visualized in an understandable  
336 fashion. The discussion of how to properly display data is not new [50], but the tools  
337 are constantly evolving. Since its introduction 20 years ago, ROOT [40] has become  
338 the most widely used package in HEP to make plots, graphics, and even to build  
339 event displays. It was developed at a time when there were few alternatives for the  
340 HEP community that did not have a significant financial cost, and it has performed  
341 admirably.

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

351    **2.2.1 Desktop solutions**

352    As it stands, ROOT is the most widely adopted plotting tool within the HEP and  
353    Nuclear Physics community. It has even made some inroads to the astrophysics  
354    community and some small pockets within the financial community, to where some  
355    physicists migrated. However, few other disciplines have adopted it. Still, it has  
356    many features beyond the standard 1D/2D/3D histogram/graphing tools, such as  
357    2D and 3D shapes, widgets for building a GUI, a JavaScript implementation for  
358    web-based analysis [51] and is available within a Jupyter notebook [52]. But to  
359    access the plotting features, an analyst must install the entire ROOT package which  
360    includes file I/O, scientific libraries, fitting routines etc., and often the installation  
361    process is non-trivial.

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

- 368    • The Python library *matplotlib* [55], released in 2003, is the most mature plotting  
369    tool for python and is the standard for most users. It can produce journal-  
370    quality graphics and there are some add-ons that can improve the default  
371    plotting options [56]. It does 1D, 2D, and 3D graphics with varying degrees  
372    of success, but does not integrate with OpenGL libraries and so it can slow  
373    down when the number of data points gets very large. It does produce most  
374    of the histograms found in HEP but some minimal, extra work must be done  
375    by the user to make histograms with error bars. Plots are reactive in the sense  
376    that you can zoom in on different regions of the graph, but you cannot do  
377    anything more significant with other mouseover commands (links, additional  
378    information, etc.).
- 379    • The R programming language has several widely used graphics tools, both  
380    built-in or provided by external modules. *ggplot2* [57] and *lattice* [58] are par-  
381    ticularly useful to visualize data in multidimensional parameter spaces. *ggplot2*  
382    is an implementation of the guidelines contained in the classic text *The Gram-*  
383    *mar of Graphics* [59], while *lattice* is an implementation of the so-called *Trellis*  
384    *Display* [60]. Both packages are very popular outside the HEP community  
385    and a wide range of learning materials are available in books, online courses,  
386    and other media. Both packages are well developed and mature and offer  
387    mechanisms for users to extend them by adding new features. *lattice* has a  
388    longer history: in 2005, the year *ggplot2* first appeared, *lattice* was already

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

### 391 2.2.2 Web-based solutions

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

- 401 • D3 (Data Driven Documents) [62] is perhaps the first web-based visualization  
402 toolkit which has been widely adopted as the de-facto base solution for building  
403 interactive data visualization for the web. The strong point of D3 is the link  
404 of the data to the DOM entities and the possibility to work with SVG objects  
405 natively. D3 is also the foundation layer upon which many higher level toolkits  
406 are built.
- 407 • Bokeh [63]. This is a plotting utility from Continuum [64], the company behind  
408 the *Anaconda* Python distribution system and other Python modules. It is  
409 designed with the web in mind and builds in a high degree of interactivity into  
410 the plots, making it useful to share results publicly and for building dashboards.  
411 However, it works by writing HTML, which makes it difficult to work with  
412 unless you use specific IDEs like a Jupyter notebook. Exporting a figure for a  
413 journal article (*e.g.*, in the PNG format) is non-trivial as well, as that is not  
414 currently the primary use-case for Bokeh.
- 415 • Plotly [65]. This is another web-oriented solution, similar to Bokeh and based  
416 on the D3 library, where plots can be hosted in Plotly’s cloud service or viewed  
417 in a Jupyter notebook. The plots are similarly very interactive and there are  
418 ways to export figure images, but that is not the goal. Dashboards can be built  
419 with relative ease and plotly offers libraries in R and JavaScript, in addition  
420 to Python. They offer both a free and enterprise business model.

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

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

435 **2.2.3 Issues**

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

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

456 A looser coupling between style and content, as well as a looser coupling be-  
457 tween locality of computation and locality of rendering, would benefit the physics  
458 community.

459 **2.3 Non-spatial visualization**

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

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

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

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

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

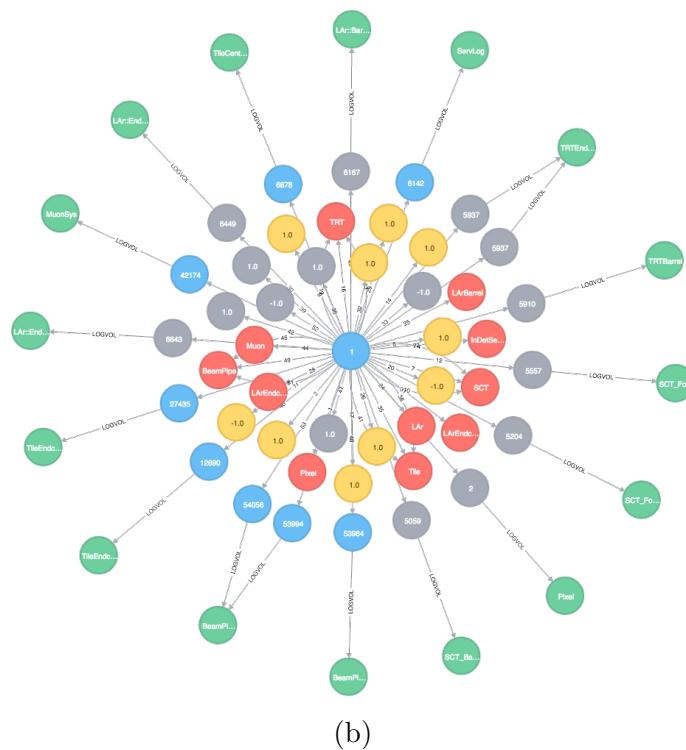
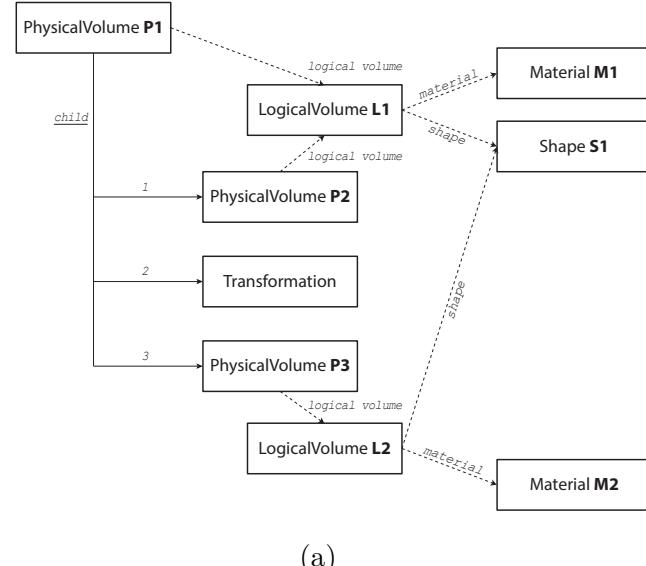
490 All those data are not space- nor time-dependent, and they are better visualized  
491 through a graph or a network. Graph-based visualization, as well as graph-databases,  
492 are fairly new in the HEP landscape but they can be very powerful tools to effectively  
493 visualize non-spatial data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

### 496 3 Suggested guidelines and future development

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

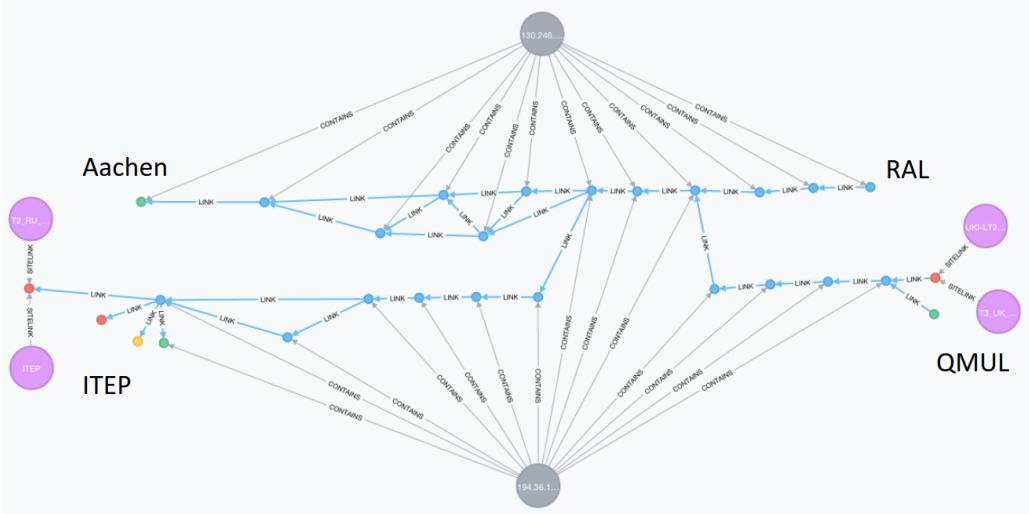
#### 500 3.1 A common community-defined format

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



**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].

503 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

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

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

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

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

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

563 Some experiments in that direction have been performed within the community  
564 in the already. For example, the ALICE experiment made use of the mini “Visual-  
565 ization Summary Data” (VSD) set of classes, contained in the ROOT Event Visual-  
566 ization Environment (EVE), to make ALICE data visualization decoupled from the  
567 AliROOT experiment’s framework [75].

568 **3.2 Serving the geometry and event data through services**

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

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

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

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

600 **3.3 Client-server architecture for geometry and event data visualization**

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

605 The idea behind that is that if we can send commands from the client to the  
606 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

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

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

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

#### 666 **3.4.2 Web-based applications**

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

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

#### 680 **3.4.3 Virtual and augmented reality**

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

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

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

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

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

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

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

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

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

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

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

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

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

#### 760 3.4.4 Mobile technologies

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

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

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

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

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

#### 789 **3.4.5 Multi-user applications**

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

## 801 **4 Sharing knowledge and fostering collaboration**

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

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

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

#### 813 **4.1 Yearly workshop**

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

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

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

825 An important point was raised while organizing the first Workshop: other sci-  
826 entific fields have visualization and graphics needs similar to HEP, for example Geo-  
827 physics. In future workshops we will try to have presentations from other communi-  
828 ties as well, in order to try to foster friendly and fruitful collaborations which could  
829 benefit the whole scientific community.

#### 830 **4.2 Code repository**

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

### 836 **5 Roadmap**

#### 837 **5.1 One year**

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

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

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

## 849        **5.2 Three years: ATLAS and CMS Computing TDRs**

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

## 857        **5.3 Five years: Towards HL-LHC**

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

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

## 864        **6 Conclusions**

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

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

<sup>876</sup> **7 Acknowledgements**

<sup>877</sup> The authors would like to thank Guy Barrand (*Geant4, LSST, LAL*) for the fruitful  
<sup>878</sup> discussions and input about what is done in astronomy in terms of data access and  
<sup>879</sup> visualization on mobile platforms.

880 **References**

- 881 [1] Antonio Augusto Alves Jr et al. *A Roadmap for HEP Software and*  
882 *Computing R&D in the 2020s*. Tech. rep. HSF-CWP-2017-01. HEP Software  
883 Foundation, 2017. arXiv: [1712.06982](https://arxiv.org/abs/1712.06982) [[physics.comp-ph](#)].
- 884 [2] The HEP Software Foundation. *HEP Software Foundation Community*  
885 *White Paper Working Group – Data Analysis and Interpretation*. Tech. rep.  
886 HSF-CWP-2017-05. HEP Software Foundation, 2017. arXiv: [1804.03983](https://arxiv.org/abs/1804.03983)  
887 [[physics.comp-ph](#)].
- 888 [3] The HEP Software Foundation. *HEP Software Foundation Community*  
889 *White Paper Working Group – Detector Simulation*. Tech. rep.  
890 HSF-CWP-2017-07. HEP Software Foundation, 2017. arXiv: [1803.04165](https://arxiv.org/abs/1803.04165)  
891 [[physics.comp-ph](#)].
- 892 [4] *ATLAS Collaboration - 13 TeV Stable Beam Collisions*. URL:  
893 [https://twiki.cern.ch/twiki/pub/AtlasPublic/  
894 EventDisplayRun2Collisions/JiveXML\\_271298\\_403602858-RZ-  
895 LegoPlot-EventInfo-RZ-YX-2015-08-06-15-01-42.png](https://twiki.cern.ch/twiki/pub/AtlasPublic/EventDisplayRun2Collisions/JiveXML_271298_403602858-RZ-LegoPlot-EventInfo-RZ-YX-2015-08-06-15-01-42.png).
- 896 [5] T. Kittelmann et al. “The Virtual Point 1 event display for the ATLAS  
897 experiment”. In: *Journal of Physics: Conference Series* 219.3 (2010). URL:  
898 <https://atlas-vp1.web.cern.ch/atlas-vp1>.
- 899 [6] CMS Collaboration. *Higgs boson produced via vector boson fusion event*  
900 *recorded by CMS (Run 2, 13 TeV)*. URL:  
901 <http://cds.cern.ch/record/2210658/>.
- 902 [7] ALICE Collaboration. *One of the first Xenon-Xenon collisions at LHC*  
903 *registered by ALICE*. URL: <https://cds.cern.ch/record/2289018>.
- 904 [8] LHCb Collaboration. *Event collected at the beginning of 2018 data taking*.  
905 URL: <http://cds.cern.ch/record/2315673>.
- 906 [9] Leo Piilonen & Belle II Collaboration. *Belle II in Virtual Reality*. URL:  
907 <http://www1.phys.vt.edu/~piilonen/VR/>.
- 908 [10] L.A.T. Bauerdtick et al. “Event display for the visualization of CMS events”  
909 In: *J.Phys.Conf.Ser.* 331.072039 (2011). URL: <https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookFireworks>.
- 911 [11] KHRONOS Group. *OpenGL*. URL: <https://www.opengl.org/about>.
- 912 [12] Microsoft. *Direct3D*. URL:  
913 <https://docs.microsoft.com/en-us/windows/desktop/direct3d>.
- 914 [13] L Chevalier et al. *PERSINT Event Display for ATLAS*. Tech. rep.  
915 ATL-SOFT-PUB-2012-001. ATLAS Experiment, CERN, 2012. URL:  
916 <https://twiki.cern.ch/twiki/bin/viewauth/Atlas/PersintWiki>.

- 917 [14] *ROOT-EVE*. URL: <https://root.cern.ch/eve>.
- 918 [15] Matevz Tadel et al. “ALICE Event Visualization Environment”. In: *CHEP*  
919 2006. 2006. URL:  
920 <https://indico.cern.ch/event/408139/contributions/979909/>.
- 921 [16] ATLAS Collaboration. *A heavy ion collision with a candidate  $Z \rightarrow \mu^+ \mu^-$*   
922 *decay*. URL: [https://twiki.cern.ch/twiki/bin/view/AtlasPublic/  
923 EventDisplayHeavyIonCollisions#Heavy\\_Ion\\_Collisions\\_with\\_a\\_Z\\_Ca](https://twiki.cern.ch/twiki/bin/view/AtlasPublic/EventDisplayHeavyIonCollisions#Heavy_Ion_Collisions_with_a_Z_Ca).
- 924 [17] ATLAS Collaboration. *Display of a candidate Higgs boson event from*  
925 *proton-proton collisions recorded by ATLAS with LHC stable beams at a*  
926 *collision energy of 13 TeV*. URL:  
927 [https://twiki.cern.ch/twiki/bin/view/AtlasPublic/  
928 EventDisplayRun2Physics#H\\_2e2\\_candidate\\_event\\_record\\_AN4](https://twiki.cern.ch/twiki/bin/view/AtlasPublic/EventDisplayRun2Physics#H_2e2_candidate_event_record_AN4).
- 929 [18] ATLAS Collaboration. *The ATLAS Atlantis visualization tool*. URL:  
930 <http://www.cern.ch/atlantis>.
- 931 [19] J. Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D*  
932 *Graphics with Open Inventor, Release 2*. Addison-Wesley, 1993.
- 933 [20] G. Alverson et al. “IGUANA: a high-performance 2D and 3D visualisation  
934 system”. In: *Nuclear Instruments and Methods in Physics Research Section*  
935 *A: Accelerators, Spectrometers, Detectors and Associated Equipment* 534.1  
936 (2004). Proceedings of the IXth International Workshop on Advanced  
937 Computing and Analysis Techniques in Physics Research, pp. 143–146. ISSN:  
938 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2004.07.036>. URL:  
939 <http://www.sciencedirect.com/science/article/pii/S0168900204014950>.
- 941 [21] CMS Collaboration. *Interactive Graphics for User Analysis*. URL:  
942 <http://iguana.web.cern.ch/iguana/>.
- 943 [22] *The Coin3D graphics engine*. URL:  
944 <https://bitbucket.org/Coin3D/coin/wiki/Home>.
- 945 [23] LHCb Collaboration. *The LHCb Panoramix visualization tool*. URL: <http://lhcb-comp.web.cern.ch/lhcb-comp/frameworks/Visualization/>.
- 947 [24] G. Alverson et al. “iSpy: A powerful and lightweight event display”. In:  
948 *J.Phys.Conf.Ser.* 396.022002 (2012).
- 949 [25] J. Boudreau and V. Tsulaia. “The GeoModel Toolkit for Detector  
950 Description”. In: *CHEP 2004*. 2004. URL:  
951 <https://cds.cern.ch/record/865601/>.
- 952 [26] *Coin3D "End of life letter"*. URL:  
953 <https://bitbucket.org/Coin3D/coin/wiki/EndOfLifeLetter>.

- 954 [27] ATLAS Collaboration. *The ATLAS Minerva visualization tool*. URL:  
955 <http://cern.ch/atlas-minerva>.
- 956 [28] T. McCauley. “A browser-based event display for the CMS experiment at  
957 the LHC using WebGL”. In: *J.Phys.Conf.Ser.* 898.072030 (2017). URL:  
958 <http://cern.ch/ispy-webgl>.
- 959 [29] *LHCb Online event display*. URL:  
960 <https://lhcb-public.web.cern.ch/lhcb-public/lbevent2/lbevent/>.
- 961 [30] G. Sabato et al. “ATLAS fast physics monitoring: TADA”. In:  
962 *J.Phys.Conf.Ser.* 898.092015 (2017). DOI:  
963 <https://doi.org/10.1088/1742-6596/898/9/092015>.
- 964 [31] ATLAS Collaboration. *The ATLAS Tracer visualization tool*. URL:  
965 <https://atlas-tracer.web.cern.ch/>.
- 966 [32] KHRONOS Group. *WebGL - OpenGL ES for the Web*. URL:  
967 <https://www.khronos.org/webgl/>.
- 968 [33] *The Qt software development framework*. URL: <https://www.qt.io/>.
- 969 [34] *The ThreeJS 3D graphics framework*. URL: <https://threejs.org/>.
- 970 [35] *LHSee*. URL:  
971 <https://www2.physics.ox.ac.uk/about-us/outreach/public/lhsee>.
- 972 [36] *CERN Camelia visualization tool*. URL:  
973 <http://medialab.web.cern.ch/content/camelia>.
- 974 [37] *Unity Technologies - The Unity3D game engine*. URL:  
975 <https://unity3d.com>.
- 976 [38] Ilija Vukotic & ATLAS Collaboration. *ATLASRift, the ATLAS VR*  
977 *experience*. URL: <https://atlasrift.web.cern.ch/>.
- 978 [39] Leo Piilonen. *Belle II in Virtual Reality*. URL:  
979 <http://www1.phys.vt.edu/~piilonen/VR/>.
- 980 [40] R. Brun and F. Rademakers. “ROOT — An object oriented data analysis  
981 framework”. In: *Nuclear Instruments and Methods A* 389 (1997). DOI:  
982 [http://dx.doi.org/10.1016/S0168-9002\(97\)00048-X](http://dx.doi.org/10.1016/S0168-9002(97)00048-X).
- 983 [41] *DD4HEP*. URL: <https://dd4hep.web.cern.ch/dd4hep/>.
- 984 [42] The CLICdp Collaboration. *CLIC detector and physics study*. URL:  
985 <http://clicdp.web.cern.ch/>.
- 986 [43] The Future Circular Collider Study Collaboration. *Future Circular Collider*  
987 *Study*. URL: <https://fcc.web.cern.ch/Pages/default.aspx>.

- 988 [44] J Allison et al. “Recent developments in Geant4”. In: *Nuclear Instruments*  
 989 and *Methods in Physics Research A* 835 (2016). URL:  
 990 <http://geant4.cern.ch/>.
- 991 [45] Trimble Inc. “SketchUp”. In: (). URL: <https://www.sketchup.com/>.
- 992 [46] T. Sakuma and T. McCauley. “Detector and Event Visualization with  
 993 SketchUp at the CMS Experiment”. In: *J.Phys.Conf.Ser.* 513.022032 (2014).  
 994 DOI: [10.1088/1742-6596/513/2/022032](https://doi.org/10.1088/1742-6596/513/2/022032).
- 995 [47] M. Case et al. “CMS Detector Description : New Developments”. In:  
 996 *Computing in High Energy Physics and Nuclear Physics 2004*. 2004. DOI:  
 997 [10.5170/CERN-2005-002.498](https://doi.org/10.5170/CERN-2005-002.498).
- 998 [48] Tai Sakuma & CMS Collaboration. *3D SketchUp images of the CMS*  
 999 *detector*. URL: <https://cds.cern.ch/record/2628527>.
- 1000 [49] J. W. Tukey. “The Future of Data Analysis”. In: *The Annals of*  
 1001 *Mathematical Statistics* 33.1 (1962), pp. 1–67. URL:  
 1002 <http://www.jstor.org/stable/2237638>.
- 1003 [50] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics  
 1004 Press, 1986.
- 1005 [51] *ROOTJS*. URL: <https://root.cern.ch/js/>.
- 1006 [52] Project Jupyter. *The Jupyter Notebook*. URL: <http://jupyter.org/>.
- 1007 [53] CERN. *PyROOT*. URL: <https://root.cern.ch/pyroot>.
- 1008 [54] Jake VanderPlas. *Python’s Visualization Landscape (PyCon 2017)*. URL:  
 1009 <https://speakerdeck.com/jakevdp/pythons-visualization-landscape-pycon-2017>.
- 1010 [55] J.D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In*  
 1011 *Science & Engineering* 9.3 (2007). DOI:  
 1012 [http://dx.doi.org/10.1109/MCSE.2007.55](https://dx.doi.org/10.1109/MCSE.2007.55).
- 1013 [56] *Seaborn*. URL: <https://seaborn.pydata.org>.
- 1014 [57] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. New York:  
 1015 Springer-Verlag, 2009. DOI: [10.1007/978-0-387-98141-3](https://doi.org/10.1007/978-0-387-98141-3).
- 1016 [58] D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag,  
 1017 2008. ISBN: ISBN 978-0-387-75968-5. DOI: [10.1007/978-0-387-75969-2](https://doi.org/10.1007/978-0-387-75969-2).
- 1018 [59] L. Wilkinson. *The Grammar of Graphics*. Springer, 2005. ISBN: ISBN  
 1019 978-0387245447. DOI: [10.1007/0-387-28695-0](https://doi.org/10.1007/0-387-28695-0).
- 1020 [60] R. A. Becker and W. S. Cleveland. *S-PLUS Trellis Graphics User’s Manual*.  
 1021 MathSoft, 1996. URL: <http://ect.bell-labs.com/sl/project/trellis/software.writing.html>.
- 1022
- 1023

- 1024 [61] Marc Paterno. “Easy Data Analysis Using R”. In: *PhyStat05*. Oxford, 2005.  
 1025 URL: <http://www.physics.ox.ac.uk/phystat05/Talks/PhyStat05-paterno.pdf>.
- 1026
- 1027 [62] M. Bostock et al. “D3: Data-Driven Documents”. In: *IEEE Trans.*  
 1028 *Visualization & Comp. Graphics (Proc. InfoVis)*. 2011. URL:  
 1029 <http://vis.stanford.edu/papers/d3>.
- 1030 [63] *Bokeh*. URL: <http://www.bokeh.pydata.org>.
- 1031 [64] *Continuum*. URL: <https://www.continuum.io/>.
- 1032 [65] *Plotly*. URL: <https://plot.ly>.
- 1033 [66] R.M. Bianchi, J. Boudreau, and I. Vukotic. “A new experiment-independent  
 1034 mechanism to persistify and serve the detector geometry of ATLAS”. In: *J.*  
 1035 *Phys.: Conf. Ser.* 898.072015 (2017). DOI:  
 1036 <https://doi.org/10.1088/1742-6596/898/7/072015>.
- 1037 [67] The HEP Software Foundation. *HEP Software Foundation Community*  
 1038 *White Paper Working Group – Event/Data Processing Frameworks*.  
 1039 Tech. rep. HSF-CWP-2017-08. HEP Software Foundation, 2017.
- 1040 [68] S. McKee et al. “Integrating network and transfer metrics to optimize  
 1041 transfer efficiency and experiment workflows”. In: *J. Phys. Conf. Ser.* 664  
 1042 (2015), p. 052003. DOI: <10.1088/1742-6596/664/5/052003>.
- 1043 [69] M. Clemencic et al. “LHCb conditions database operation assistance  
 1044 systems”. In: *J. Phys. Conf. Ser.* 396 (2012), p. 052022. DOI:  
 1045 <10.1088/1742-6596/396/5/052022>.
- 1046 [70] *VRML (Virtual Reality Modeling Language)*. URL:  
 1047 <https://en.wikipedia.org/wiki/VRML>.
- 1048 [71] Belle II Collaboration. *The Belle II Geometry Exporters*. URL:  
 1049 <https://github.com/HSF/Visualization/tree/master/demonstrators/GeometryWriters>.
- 1050
- 1051 [72] *KHRONOS Group — glTF - The GL Transmission Format*. URL:  
 1052 <https://www.khronos.org/gltf/>.
- 1053 [73] *BelleII VR models on Sketchfab*. URL:  
 1054 <https://sketchfab.com/models/6c7aa0c71dc64079b08a0dddfa756ef3>.
- 1055 [74] *Sketchfab*. URL: <https://sketchfab.com>.
- 1056 [75] Matevz Tadel. “ALICE Event Visualization Environment”. In: *CHEP 2006*.  
 1057 Proceedings of the Computing in High Energy and Nuclear Physics  
 1058 conference (Tata Institute of Fundamental Research, Feb. 13–17, 2006).  
 1059 Mumbai, India, 2006. URL:  
 1060 <https://indico.cern.ch/event/408139/contributions/979909/>.

- 1061 [76] *CERN TEV visualization framework*. URL:  
1062 <https://gitlab.cern.ch/CERNMediaLab/>.
- 1063 [77] *Epic Games - The “Unreal Engine” game engine*. URL:  
1064 <https://www.unrealengine.com>.
- 1065 [78] Epic Games, Inc. *Blueprints Visual Scripting*. URL:  
1066 <https://docs.unrealengine.com/en-us/Engine/Blueprints>.
- 1067 [79] The Khronos Group Inc. *The Vulkan API*. URL:  
1068 <https://www.khronos.org/vulkan/>.
- 1069 [80] Juan Linietsky, Ariel Manzur, and contributors. *The Godot game engine*.  
1070 URL: <https://godotengine.org/>.
- 1071 [81] Andreas Rumpf. *The Vulkan API*. URL: <https://nim-lang.org/>.
- 1072 [82] *The Qt 3D framework*. URL:  
1073 <https://doc.qt.io/qt-5.11/qt3d-index.html>.
- 1074 [83] *The SoQt package*. URL: <https://bitbucket.org/Coin3D/soqt>.
- 1075 [84] *The Belle II Experiment*. URL: <http://belle2.jp>.
- 1076 [85] Google. *Google Cardboard*. URL: <https://vr.google.com/cardboard/>.
- 1077 [86] Google. *Google Daydream*. URL: <https://vr.google.com/daydream/>.
- 1078 [87] Samsung Electronics. *Samsung Gear VR*. URL:  
1079 <https://www.samsung.com/global/galaxy/gear-vr>.
- 1080 [88] LLC. Oculus VR. *Oculus Rift*. URL: <https://www.oculus.com/rift/>.
- 1081 [89] HTC Corporation. *HTC Vive*. URL: <https://www.vive.com/eu/>.
- 1082 [90] LLC. Oculus VR. *Oculus Go*. URL: <https://www.oculus.com/go/>.
- 1083 [91] Lenovo Group Ltd. Google. *Lenovo Mirage Solo*. URL:  
1084 <https://vr.google.com/daydream/standalonevr>.
- 1085 [92] PTC Inc. *Vuforia AR framework*. URL:  
1086 <https://www.vuforia.com/engine.html>.
- 1087 [93] Steffen Henkelmann (University of British Columbia, CA) & ATLAS  
1088 Collaboration. *ATLAS in your pocket — An Augmented Reality application*.  
1089 URL: <https://atlasinyourpocket.web.cern.ch/>.
- 1090 [94] Jeff Ouellette. “More Than ALICE: Development of an augmented reality  
1091 mobile application for the ALICE detector”. In: (Aug. 2016). URL:  
1092 <https://cds.cern.ch/record/2207593>.
- 1093 [95] *WebVR API*. URL: <https://immersive-web.github.io/webvr/>.
- 1094 [96] *AFrame - A web framework for building virtual reality experiences*. URL:  
1095 <https://aframe.io/>.

- 1096 [97] *CMS AFrame*. URL: <https://github.com/HEPPanoramic/cms-aframe>.
- 1097 [98] The HEP Software Foundation. *HEP Software Foundation Community*  
1098 *White Paper Working Group – Data Organisation, Management and Access*.  
1099 Tech. rep. HSF-CWP-2017-04. HEP Software Foundation, 2017.
- 1100 [99] P. Lavrič, C. Bohak, and M. Marolt. “Collaborative view-aligned  
1101 annotations in web-based 3D medical data visualization”. In: *MIPRO 2017:*  
1102 *40th Jubilee International Convention*. Opatija, Croatia, May 2017,  
1103 pp. 276–280. URL: [http://lgm.fri.uni-lj.si/wp-](http://lgm.fri.uni-lj.si/wp-content/uploads/2017/10/1537435587.pdf)  
1104 [content/uploads/2017/10/1537435587.pdf](http://lgm.fri.uni-lj.si/wp-content/uploads/2017/10/1537435587.pdf).
- 1105 [100] HSF. “HSF Visualization Workshop”. In: CERN, 2017. URL:  
1106 <https://indico.cern.ch/event/617054/>.
- 1107 [101] HSF. “HSF Mini-workshop on GUI, graphic libraries, interactive  
1108 visualization”. In: CERN, 2017. URL:  
1109 <https://indico.cern.ch/event/628675/>.
- 1110 [102] HSF Visualization Working Group GitHub repository. URL:  
1111 <https://github.com/HEP-SF/Visualization>.