

1

2

# <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> Sébastien 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

30 ABSTRACT: In modern High Energy Physics (HEP) experiments visualization of ex-  
31 perimental data has a key role in many activities and tasks across the whole data  
32 chain: from detector development to monitoring, from event generation to recon-  
33 struction of physics objects, from detector simulation to data analysis, and all the  
34 way to outreach and education. In this paper the definition, status, and evolution  
35 of data visualization for HEP experiments will be presented. Suggestions for the  
36 upgrade of data visualization tools and techniques in current experiments will be  
37 outlined, along with guidelines for future experiments. This paper expands on the  
38 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	10
46	2.2 Statistical data visualization	12
47	2.2.1 Desktop solutions	12
48	2.2.2 Web-based solutions	13
49	2.2.3 Issues	14
50	2.3 Non-spatial visualization	15
51	<b>3 Suggested guidelines and future development</b>	<b>17</b>
52	3.1 A common community-defined format	18
53	3.2 Serving the geometry and event data through services	19
54	3.3 Client-server architecture for geometry and event data visualization	20
55	3.4 Exploring modern technologies	20
56	3.4.1 Graphics and game engines	20
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	25
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	26
67	5.3 Five years: Towards HL-LHC	27
68	<b>6 Conclusions</b>	<b>27</b>
69	<b>7 Acknowledgements</b>	<b>27</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.

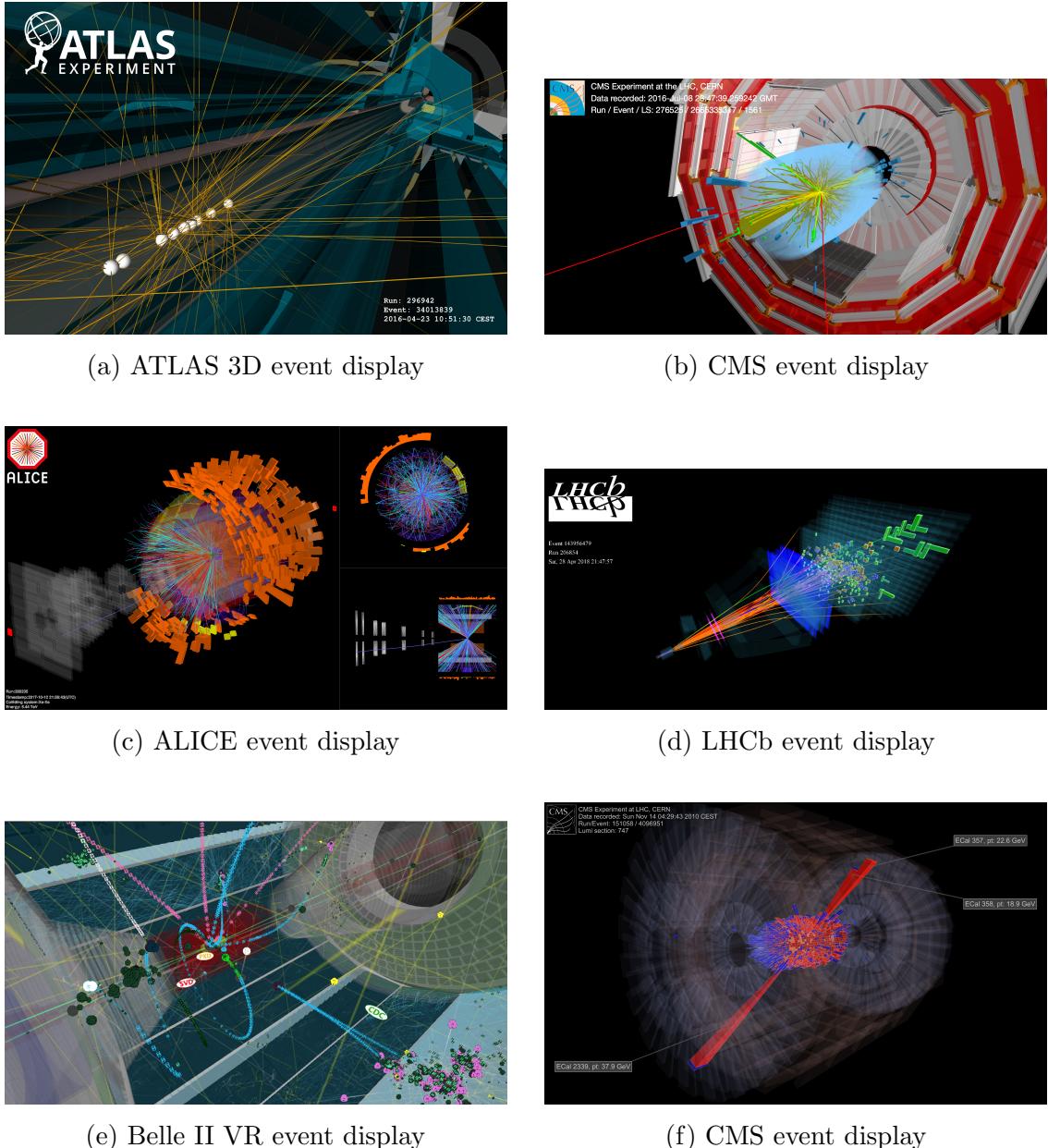
125 In the following subsections several important aspects of data access, application  
126 development and distribution, and geometry description and visualization, as they  
127 pertain to the current landscape of event displays, are discussed in more detail.  
128 Screenshots of several event displays 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) ATLAS 3D event display [4, 5]; b) CMS 3D event display [6]; c) ALICE 3D event display [7]; d) LHCb 3D event display [8]; e) 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.

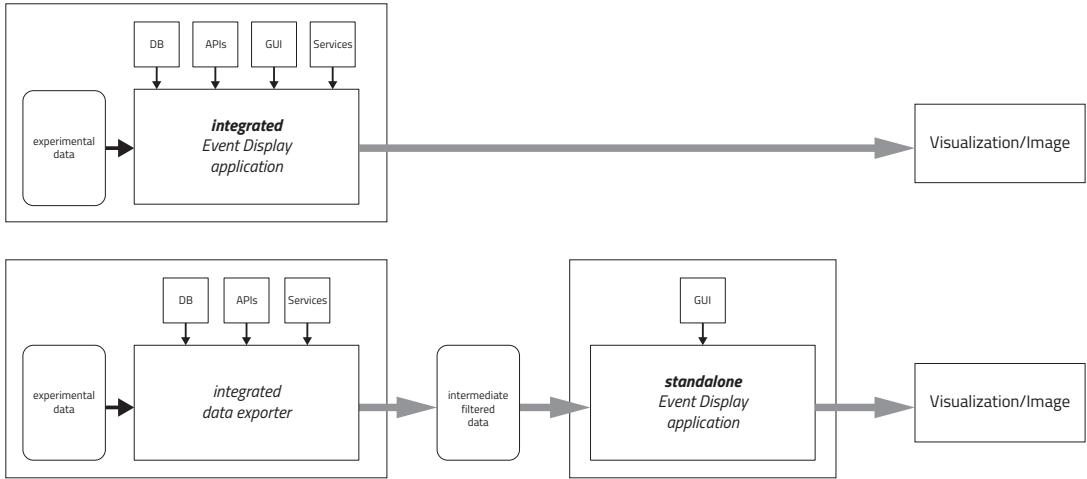
167 Regardless of the approach to data access one must consider the use-case: what is  
168 useful or necessary in one use-case may not be for another. A graphical representation  
169 of the two approaches can be seen in Figure 2.

### 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 3 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 Visualisation  
 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.

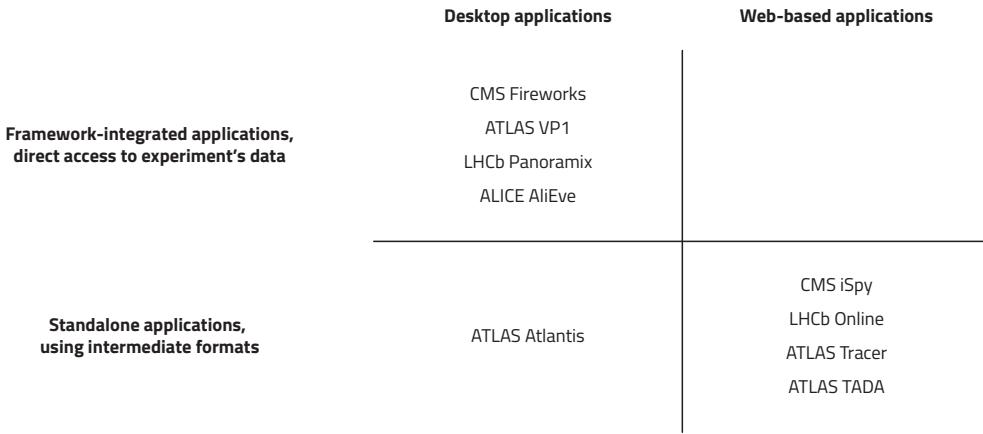
199 Other applications use higher-level interface libraries as graphics engines. This  
 200 has the advantage of delegating a large part of the lower-level development work to

201 external software packages, leaving the developers to concentrate on the application  
202 development itself. A popular graphics library used in HEP software has been Open  
203 Inventor [16], used by the defunct CMS Iguana [17, 18] application, or its clone im-  
204 plementation Coin (also known as Coin3D) [19], which has been used by applications  
205 such as ATLAS VP1 [5], LHCb Panoramix [20] and the defunct desktop version of  
206 the CMS iSpy application [21]. Coin / Open Inventor was chosen because of its  
207 integrability in C++ code, its performance, and its coding style. Moreover, the way  
208 Open Inventor handles graphical volumes could be easily matched with the way ge-  
209 ometry volumes are handled to describe the detectors in HEP experiments. Open  
210 Inventor organizes geometry volumes as a series of nodes in a tree-like structure in  
211 the same way as some HEP experiments do. ATLAS, for instance, developed their  
212 geometry library “GeoModel” [22] based on the same tree-like structure of nodes  
213 used by Open Inventor.

214 The drawback of this approach is the dependency on external software projects,  
215 which could end up with a loss of functionality if third-party library development  
216 and support are abandoned. Many scientific visualization applications, also in fields  
217 other than HEP, faced this when the support of the Coin library was dropped by the  
218 company that led its development [23]. The result is the aging of libraries which af-  
219 ter a while show incompatibilities with modern hardware, compilers, and platforms.  
220 The time spent by HEP developers to repair or to maintain those abandoned li-  
221 braries results is time not spent on actual development of the software applications  
222 themselves.

223 An additional approach to the development of event displays is to create and  
224 distribute an application using the Java programming language. The Atlantis [24]  
225 program and its derivative MINERVA [25], which is used as an educational tool, both  
226 developed for the ATLAS experiment, are based on the Java graphics libraries and  
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 [26],  
229 LHCb [27], and ATLAS [28, 29] have created web-based event displays using We-  
230 bGL (Web Graphics Library) [30]. 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 [31]. 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 [32], that provide straightforward



**Figure 3:** 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.

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 [33] which live-streams ATLAS events, processed and extracted through Atlantis [24], to a user’s phone and provided contextual information on ATLAS and the events being displayed. The CMS iSpy WebGL application [26] runs on mobile devices in the browser. The Camelia application [34], developed by the CERN Media Lab using the Unity game engine [35], 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 [36] and Belle II VR [37]. 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.

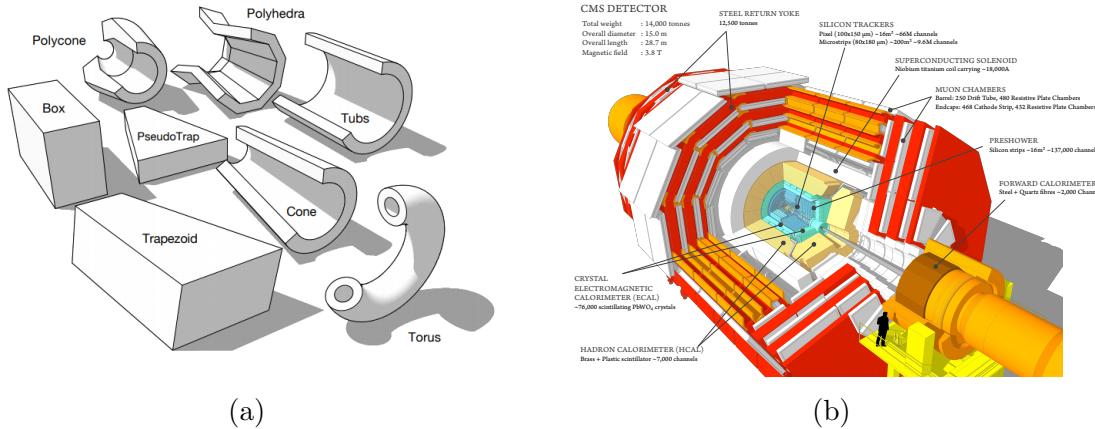
Augmented Reality (AR), instead, uses a camera to take a view of the real world around the user and screen where simulated objects are rendered in 3D and shown on top of that image dynamically, following the user’s interaction and motion. This lets the user move within an environment where real and simulated 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.

### 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.*, [22]), while others use the geometry tools provided by the ROOT framework [38]. More recently, some attempts



**Figure 4:** Use of a 3D software (SketchUp) to render the basic geometrical shapes imported from the CMS detector description and the final volumes [46].

have been made to build common formats and libraries for detector geometry, like DD4HEP [39], adopted in conceptual design studies for future high-energy colliders, including the CLICdp [40] and FCC [41] 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 [42], 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 [43], described in Ref. [44] (see Figure 4). The CMS detector description [45] 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.

326    **2.2 Statistical data visualization**

327

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

---

John Tukey [47]

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

331       In HEP, like most other scientific disciplines, visualization of the data and of the  
332    final results plays a key role in the analysis pipeline. A new projection of the data  
333    may provide new insight, results must be summarized in a clear and concise way,  
334    and multidimensional parameter spaces need to be visualized in an understandable  
335    fashion. The discussion of how to properly display data is not new [48], but the tools  
336    are constantly evolving. Since its introduction 20 years ago, ROOT [38] has become  
337    the most widely used package in HEP to make plots, graphics, and even to build  
338    event displays. It was developed at a time when there were few alternatives for the  
339    HEP community that did not have a significant financial cost, and it has performed  
340    admirably.

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

350    **2.2.1 Desktop solutions**

351       As it stands, ROOT is the most widely adopted plotting tool within the HEP and  
352    Nuclear Physics community. It has even made some inroads to the astrophysics  
353    community and some small pockets within the financial community, to where some  
354    physicists migrated. However, few other disciplines have adopted it. Still, it has  
355    many features beyond the standard 1D/2D/3D histogram/graphing tools, such as  
356    2D and 3D shapes, widgets for building a GUI, a JavaScript implementation for  
357    web-based analysis [49] and is available within a Jupyter notebook [50]. But to  
358    access the plotting features, an analyst must install the entire ROOT package which  
359    includes file I/O, scientific libraries, fitting routines etc., and often the installation  
360    process is non-trivial.

361 Many current HEP analysts make wide use of the Python programming language  
362 and the PyROOT libraries [51]. Python is also very popular outside the HEP com-  
363 munity and so it is worth looking at non-ROOT options available to Python users.  
364 A recent (as of 2017) summary of the field was presented by Jake VanderPlas at  
365 PyCon 2017 [52], a subset of which will be presented here. It is emphasized that this  
366 is just a sampling and that the number of options available is a function of time.

- 367 • The Python library *matplotlib* [53], released in 2003, is the most mature plot-  
368 ting tool for python and is the standard for most users. It can produce journal-  
369 quality graphics and there are some add-ons that can improve the default plot-  
370 ting options [54]. It does 1D, 2D, and 3D graphics with varying degrees of  
371 success, but does not integrate with OpenGL libraries and so it can slow down  
372 when the number of data points gets very large. It does produce most of the  
373 histograms found in HEP but some minimal, extra work must be done by the  
374 user to make histograms with error bars. Plots are reactive in the sense that  
375 you can zoom in on different regions of the graph, but you cannot do anything  
376 more significant with other mouseover commands (links, additional informa-  
377 tion, etc.).
- 378 • The R programming language has several widely used graphics tools, both  
379 built-in or provided by external modules. *ggplot2* [55] and *lattice* [56] are  
380 particularly useful to visualize data in multidimensional parameter spaces. *gg-*  
381 *plot2* is an implementation of the guidelines contained in the classic text *The*  
382 *Grammar of Graphics* [57], while *lattice* is an implementation of the so-called  
383 *Trellis Display* [58]. Both packages are very popular outside the HEP com-  
384 munity and a wide range of learning materials are available in books, online  
385 courses, and other media. Both packages are well developed and mature and  
386 offer mechanisms for users to extend them by adding new features. *lattice* has  
387 a longer history: in 2005, the year *ggplot2* first appeared, *lattice* was already  
388 popular. In fact, figures made with *lattice* were shown in the presentation in  
389 PHYSTAT05 [59] which introduced R to the particle physics community.

### 390 2.2.2 Web-based solutions

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

- D3 (Data Driven Documents) [60] 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 [61]. This is a plotting utility from Continuum [62], 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 [63]. 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

438 plot-making abilities from the data analysis framework. As a consequence, if a physi-  
439 cist's data can only be found on a particular server, the plot-generating code must  
440 also be located there and the outcome is sometimes hard to bring to the physicist's  
441 laptop screen. In the worst cases, graphics files (PNGs) must be copied from the  
442 server to the laptop for viewing. This causes a high interaction latency, discouraging  
443 exploration. That's why the development of new tools should go towards a sharper  
444 separation between the computation on data and the interactive data visualization  
445 routines, pushing the latter to the client side as much as possible.

446 **Separating the plotting functions and content from the plotting style:**

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

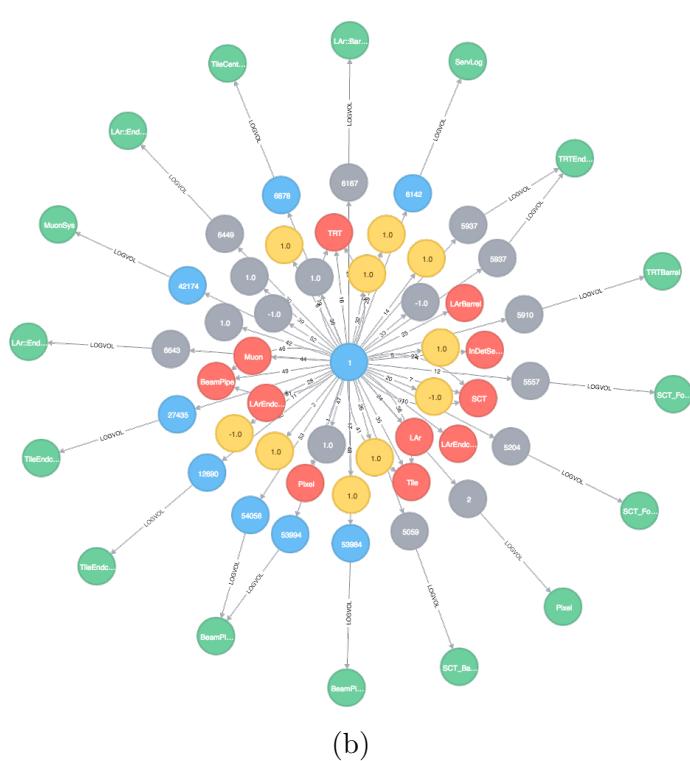
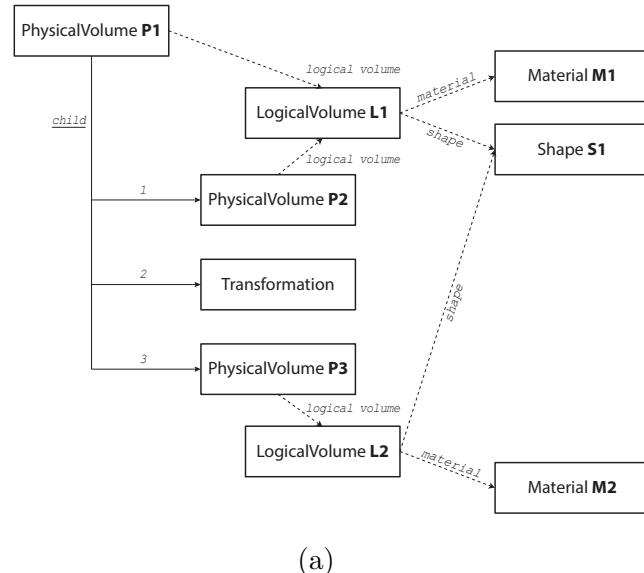
455 A looser coupling between style and content, as well as a looser coupling be-  
456 tween locality of computation and locality of rendering, would benefit the physics  
457 community.

458 **2.3 Non-spatial visualization**

459 In HEP, there are data which are organized in a tree-like structure, and for which  
460 a graph or a network visualization is the best choice. The Detector Description  
461 is an example of a source of such data: it describes all the pieces which compose  
462 a HEP experiment detector. The different pieces of the Detector Description are  
463 interconnected through different relationships: geometrical volumes can be organized  
464 in a parent-child relationship, or a property node can be shared among many volumes.  
465 The visualization of those data in a network helps developers in the understanding  
466 and the debugging of the Detector Description, by visualizing the relationships among  
467 all the nodes and their properties. An example, from the ATLAS experiment [64],  
468 of a graph visualizing the inner structure of a HEP detector description can be seen  
469 in Figure 5.

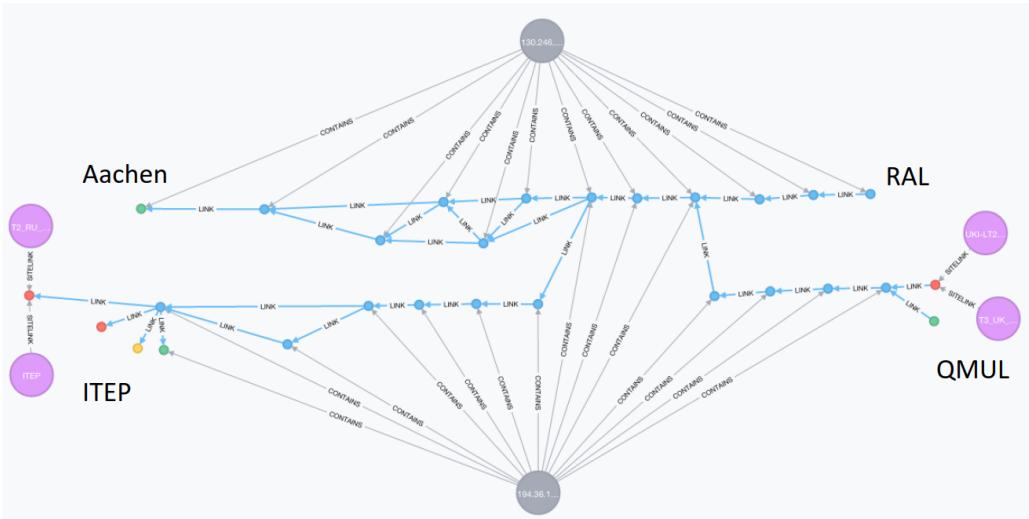
470 Networks and graphs are very effective ways of visualizing tree-like data, because  
471 they are able to show all the nodes, their relationships and their properties in a proper  
472 way. Some degree of interactivity can let the scientists applying different filters and  
473 layout, helping them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

475 Another example of HEP data that can benefit from a graph-based visualization  
476 is that one describing the execution chain of the jobs used to filter and reconstruct  
477 the experimental data. Very recently HEP experiments began to develop new paral-



**Figure 5:** 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 [64].

**478** lel frameworks to concurrently handle analysis or reconstruction jobs, to efficiently



**Figure 6:** 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 [66]].

479 exploit the parallelism offered by the modern hardware (more details can be found  
480 in the HSF *Event/Data Processing Frameworks* Community White Paper [65]). The  
481 jobs are handled by a scheduler, which organizes them according to their needed  
482 input and output data. The outcome of the scheduler is a directed acyclic graph  
483 (DAG). The visualization by means of a graph helps the developer understanding  
484 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 [66] (see Picture 6) or to store and query conditions data [67].

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

### 495 3 Suggested guidelines and future development

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

499 **3.1 A common community-defined format**

500 Visualization has a key role in the lifecycle of a HEP experiment, addressing input  
501 data from many sources and in many different formats. The input data are often  
502 quite tightly bound to a specific experiment’s software framework and because of  
503 that many visualization tools are integrated into the frameworks to some extent.  
504 However, the visualization is often the last step on the experiment data chain and  
505 the output of a visualization application is often not used by any other tool in the  
506 software framework. So while the direct interaction with experimental data formats  
507 is highly experiment specific, there is a real possibility of having the final stages of  
508 the visualization pipeline shared between several experiments.

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

517 The same reasoning made for the geometry can be done for the event data. Of  
518 course different experiments detect different objects and measure different quanti-  
519 ties, but there are many common entities, especially among experiments within the  
520 same research field. For example, all experiments working on hadron colliders use  
521 the notion of particle track, which is usually constructed translating the track mea-  
522 surements into space points or points and angles, and use the notion of particle jet,  
523 usually visualized as a cone whose length is related to its energy and whose radius  
524 is linked to the specific algorithm used for the jet reconstruction. Both of these ob-  
525 jects are currently often handled and visualized differently in different experiments,  
526 but could in principle be the target of a common definition within the community.  
527 If so, experiments could share best practices or snippets of code, if not complete  
528 basic tools, to build and handle their visualization. In this way, the know-how and  
529 the tools linked to visualization needs could be shared as well, and developed in a  
530 collaborative way.

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

534 For the moment, we observe that several event displays in experiments have  
535 exporters to translate geometry information to standard formats used in the com-  
536 munities outside HEP, mainly in computer graphics and engineering. For example  
537 the Belle II experiment has written exporters from Geant4 data to different formats,  
538 including VRML [68] and FBX [69], which are two of the most common formats used

539 to store and share 3D graphics data. The Unity game-development engine [35], in  
540 turn, can export the FBX geometry to the glTF format [70], an emerging royalty-free  
541 specification for 3D objects and scenes, for fast web distribution via the SketchFab  
542 community repository [71, 72]. Displays based on three.js have access to multiple  
543 importers and exporters of several geometry formats, including the ones mentioned  
544 above.

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

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

562 Some experiments in that direction have been performed within the community  
563 in the already. For example, the ALICE experiment made use of the mini “Visual-  
564 ization Summary Data” (VSD) set of classes, contained in the ROOT Event Visual-  
565 ization Environment (EVE), to make ALICE data visualization decoupled from the  
566 AliROOT experiment’s framework [73].

### 567 **3.2 Serving the geometry and event data through services**

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

579 experiment's geometry would be enough, and that a simpler way to retrieve it should  
580 be designed, to ease data access for visualization: for example, an online service could  
581 remotely serve the geometry data to the visualization applications in a standardized  
582 format.

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

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

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

### 599 **3.3 Client-server architecture for geometry and event data visualization**

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

604 The idea behind that is that if we can send commands from the client to the  
605 server, and get the answer back in the data stream, then we will be able to interact  
606 with the experiment's framework as well, in addition to using common visualization  
607 applications to visualize the common objects. In this way, we can develop a modular  
608 architecture where HEP experiments can share the design, the development and the  
609 maintenance of common visualization tools, while maintaining a certain degree of  
610 freedom to add custom content and objects and to interact with their own framework  
611 to retrieve specific content.

### 612 **3.4 Exploring modern technologies**

#### 613 **3.4.1 Graphics and game engines**

614 So far direct OpenGL, its derivatives (like WebGL), or old graphics libraries have  
615 been used in HEP visualization applications. Nowadays, another type of graphics  
616 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 [37] and the Total Event Visualiser (TEV) of the CERN Media Lab [74], which used the Unity game engine [35], and ATLAS which used the Unreal Engine [75] for its virtual reality application ATLASrift [36]. 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 [76]) 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 [77], the cross-platform 3D and computing API.

The Unity development platform [35] 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 [78]. 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 [79], 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 [80]. 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

658 SoQt [81] package used by ATLAS). By eliminating that, we could simplify the ar-  
 659 chitecture of our visualization tools and lower the maintenance workload. Qt3D is  
 660 still in development, but it shows an initial set of features which are worth a further  
 661 consideration of the new toolkit. We plan to take a look at its development in the  
 662 near future, to see if it can satisfy the HEP requirements. Also, being open source,  
 663 we could consider contributing to the Qt3D software project as a community, by  
 664 providing the pieces we need for our applications.

### 665 3.4.2 Web-based applications

666 Web-based graphics have traditionally been considered not powerful enough to handle  
 667 the thousands of volumes that can be shown in HEP event displays, for example,  
 668 when visualizing hits in a very busy event. But the technology has rapidly evolved  
 669 and web-based graphics can now visualize very complex scenes. For example, the  
 670 glTF [70] 3D model of the Belle II detector [82], with tens of thousands of elements,  
 671 can be loaded, viewed and manipulated in a web browser, even on a smartphone,  
 672 very effectively [71].

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 [49] could be used as an underlying layer for event data visualization as well as three.js [32], which have been used successfully by different experiments (*e.g.*, [26, 28, 29]) to visualize geometry and event data.

### 679 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 referred 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 [83] 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

697 orientation API in the browser. No hand controller is used in the Cardboard but for  
698 native applications a click event is available via a magnet attached to the Cardboard  
699 viewer.

700 The Google Daydream [84] is the next iteration of viewers developed by Google  
701 for their Google VR technology. Content is still delivered by a mobile phone (thus,  
702 the performance is limited by the computing power of the phone) but a Bluetooth-  
703 connected hand controller with rotational tracking is available. The Samsung Gear  
704 VR platform [85] is another headset powered by an inserted smartphone targeted for  
705 the Oculus platform.

706 Currently, the most immersive and interactive VR environments are provided by  
707 the Oculus Rift [86] and the HTC Vive [87], which combine the computing resources  
708 of a desktop machine with sensors, controllers, and high-quality HMDs. Those so-  
709 phisticated devices are relatively expensive (even if prices are lowering in the last  
710 months) and require a fairly powerful computer to run. The presence of a proper  
711 computer solves the performance issue of phone-based viewers, since the 3D graphics  
712 computations are handled by the computer, but that also increases the required ini-  
713 tial budget for people willing to test such technologies, and that limits the number  
714 of people getting access to such platforms, for example in public events organized by  
715 HEP institutes.

716 Meanwhile, a new type of device has been recently developed: a standalone  
717 viewer, equipped with an on-board CPU, able to run medium computation-intensive  
718 applications. Those devices lower the budget needed to develop and deploy VR  
719 applications significantly. Examples of those new devices are Oculus Go[88] and the  
720 stand-alone Lenovo headset for the Daydream environment [89].

721 Game engines, described in Section 3.4.1, provide powerful integrated devel-  
722 opment environments for creation of VR applications targeting multiple devices.  
723 Thanks to engines' abstraction of third party VR libraries, most HEP experiments  
724 should be able to develop VR applications which natively support both standard  
725 displays and all VR hardware. Currently, both ATLASrift [36] and Belle II VR [37]  
726 support Oculus, HTC Vive, and standard 2D displays. CMS, using the Unity game  
727 engine, is currently working on CMS.VR (to be released), targeting Oculus and HTC.

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

735 ATLAS and ALICE researchers have started to explore the possibilities offered  
736 by augmented reality for outreach and education, by using Unity [35] and the Vu-  
737 foria framework [90] (commercial, but free for development). The ATLAS-in-Your-

738 Pocket [91] application uses printed marks to place a rendered geometry of the AT-  
739 LAS detector on top of a view of the real world in front of the user. The More-Than-  
740 ALICE [92] application allows users to superimpose a description of the detector or  
741 event visualizations to the camera image of the actual ALICE detector (for example,  
742 on a screen or during public visits to the experimental site) or its paper model.

743 Development of web-based VR and AR applications for mobile browser can be  
744 done using a WebGL library such as three.js [32] and using the HTML device orien-  
745 tation control API. The viewport is split into two views, one for each eye, each with a  
746 dedicated camera view separated by an appropriate distance to create a stereoscopic  
747 effect (*e.g.* iSpy WebGL [26] has a stereo mode for Google Cardboard). The develop-  
748 ing WebVR specification [93] provides interfaces to VR hardware via the browser. A  
749 powerful framework for development of VR applications for various devices using the  
750 browser is A-Frame [94]. A-Frame has support for several device controllers as well,  
751 such as for the Daydream and Oculus. CMS is exploring the possibilites of A-Frame  
752 in the browser for both VR and AR (*e.g.* CMS A-Frame prototype [95]).

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

#### 759 **3.4.4 Mobile technologies**

760 Portability and simplicity of usage are the strong points of mobile devices. More  
761 than as “mobile” devices, smartphones, tablets and ultrabooks can be considered,  
762 as devices “close to people”. As such, the usage of such devices should be exploited  
763 more in the final steps of the visualization chain, where heavy batch data processing  
764 is not needed. For instance, their usage should be leveraged for the production  
765 and visualization of event displays. Ideally, a user should be able to easily retrieve  
766 interesting events from the experiment and interactively visualize them on all kinds  
767 of devices.

768 That is why we strongly promote the usage of the server-client architecture  
769 described and supported in this paper in Section 3.3 and the new data access patterns  
770 presented and supported in the the HSF *Data Access and Management* Community  
771 White Paper [96]. This would open up new possibilities for interactive visualization  
772 on mobile devices: it would let visualization clients running on mobile devices connect  
773 to server tools running in the experiment’s framework to easily and interactively  
774 retrieve the desired data.

775 It is worth noting that in other areas of science, for instance in astronomy,  
776 researchers have worked to facilitate data access and to migrate to more standard  
777 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 start exploring their usage. An example of such collaborative features in a 3D environment is integrated in the Med3D visualization framework [97].

## 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

815 share their solutions. In addition, external experts from industry were invited to  
816 present the latest advancements in the field and best practices [98].

817 It was the first topical workshop focused on HEP Visualization in many years.  
818 Many HEP experiments and communities showed their latest developments. Given  
819 the high number of presentations, a second mini-workshop [99] has been organized  
820 as a follow-up, to let the remaining communities to present their work.

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

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

## 829 **4.2 Code repository**

830 The HSF Visualization Working Group also agreed on the importance of fostering  
831 collaborative work. As a start a new dedicated project has been created within the  
832 HSF GitHub repository [100]. It is intended to be the space where members of the  
833 Working Group can share their work-in-progress studies and their solutions, and  
834 where community-driven projects will be stored.

# 835 **5 Roadmap**

## 836 **5.1 One year**

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

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

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

## 848 **5.2 Three years: ATLAS and CMS Computing TDRs**

849 In the second and third year the Visualization WG will work on designing and build-  
850 ing demonstrators to show the feasibility of the community-driven best practices and

851 tools. The goal will be to get a final design of those tools, to be included in the de-  
852 velopment plans of the different experiments. Moreover, the WG will work towards  
853 a more convenient access to geometry and event data. In collaboration with the HSF  
854 *Data Access and Management* working group, an API or a service to get streamed  
855 event data would be designed.

856 **5.3 Five years: Towards HL-LHC**

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

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

863 **6 Conclusions**

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

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

875 **7 Acknowledgements**

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

879 **References**

- 880 [1] Antonio Augusto Alves Jr et al. *A Roadmap for HEP Software and*  
881 *Computing R&D in the 2020s*. Tech. rep. HSF-CWP-2017-01. HEP Software  
882 Foundation, 2017. arXiv: [1712.06982](https://arxiv.org/abs/1712.06982) [[physics.comp-ph](#)].
- 883 [2] The HEP Software Foundation. *HEP Software Foundation Community*  
884 *White Paper Working Group – Data Analysis and Interpretation*. Tech. rep.  
885 HSF-CWP-2017-05. HEP Software Foundation, 2017. arXiv: [1804.03983](https://arxiv.org/abs/1804.03983)  
886 [[physics.comp-ph](#)].
- 887 [3] The HEP Software Foundation. *HEP Software Foundation Community*  
888 *White Paper Working Group – Detector Simulation*. Tech. rep.  
889 HSF-CWP-2017-07. HEP Software Foundation, 2017. arXiv: [1803.04165](https://arxiv.org/abs/1803.04165)  
890 [[physics.comp-ph](#)].
- 891 [4] *ATLAS Collaboration - 13 TeV Stable Beam Collisions*. URL:  
892 [https://twiki.cern.ch/twiki/pub/AtlasPublic/  
893 EventDisplayRun2Collisions/JiveXML\\_271298\\_403602858-RZ-  
894 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).
- 895 [5] T. Kittelmann et al. “The Virtual Point 1 event display for the ATLAS  
896 experiment”. In: *Journal of Physics: Conference Series* 219.3 (2010). URL:  
897 <https://atlas-vp1.web.cern.ch/atlas-vp1>.
- 898 [6] CMS Collaboration. *Higgs boson produced via vector boson fusion event*  
899 *recorded by CMS (Run 2, 13 TeV)*. URL:  
900 <http://cds.cern.ch/record/2210658/>.
- 901 [7] ALICE Collaboration. *One of the first Xenon-Xenon collisions at LHC*  
902 *registered by ALICE*. URL: <https://cds.cern.ch/record/2289018>.
- 903 [8] LHCb Collaboration. *Event collected at the beginning of 2018 data taking*.  
904 URL: <http://cds.cern.ch/record/2315673>.
- 905 [9] Leo Piilonen & Belle II Collaboration. *Belle II in Virtual Reality*. URL:  
906 <http://www1.phys.vt.edu/~piilonen/VR/>.
- 907 [10] L.A.T. Bauerdrick et al. “Event display for the visualization of CMS events”.  
908 In: *J.Phys.Conf.Ser.* 331.072039 (2011). URL: <https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookFireworks>.
- 910 [11] KHRONOS Group. *OpenGL*. URL: <https://www.opengl.org/about>.
- 911 [12] Microsoft. *Direct3D*. URL:  
912 <https://docs.microsoft.com/en-us/windows/desktop/direct3d>.
- 913 [13] L Chevalier et al. *PERSINT Event Display for ATLAS*. Tech. rep.  
914 ATL-SOFT-PUB-2012-001. ATLAS Experiment, CERN, 2012. URL:  
915 <https://twiki.cern.ch/twiki/bin/viewauth/Atlas/PersintWiki>.

- 916 [14] *ROOT-EVE*. URL: <https://root.cern.ch/eve>.
- 917 [15] Matevz Tadel et al. “ALICE Event Visualization Environment”. In: *CHEP*  
918 2006. 2006. URL:  
919 <https://indico.cern.ch/event/408139/contributions/979909/>.
- 920 [16] J. Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D*  
921 *Graphics with Open Inventor, Release 2*. Addison-Wesley, 1993.
- 922 [17] G. Alverson et al. “IGUANA: a high-performance 2D and 3D visualisation  
923 system”. In: *Nuclear Instruments and Methods in Physics Research Section*  
924 *A: Accelerators, Spectrometers, Detectors and Associated Equipment* 534.1  
925 (2004). Proceedings of the IXth International Workshop on Advanced  
926 Computing and Analysis Techniques in Physics Research, pp. 143–146. ISSN:  
927 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2004.07.036>. URL:  
928 <http://www.sciencedirect.com/science/article/pii/S0168900204014950>.
- 930 [18] CMS Collaboration. *Interactive Graphics for User Analysis*. URL:  
931 <http://iguana.web.cern.ch/iguana/>.
- 932 [19] *The Coin3D graphics engine*. URL:  
933 <https://bitbucket.org/Coin3D/coin/wiki/Home>.
- 934 [20] LHCb Collaboration. *The LHCb Panoramix visualization tool*. URL: <http://lhcb-comp.web.cern.ch/lhcb-comp/frameworks/Visualization/>.
- 936 [21] G. Alverson et al. “iSpy: A powerful and lightweight event display”. In:  
937 *J.Phys. Conf. Ser.* 396.022002 (2012).
- 938 [22] J. Boudreau and V. Tsulaia. “The GeoModel Toolkit for Detector  
939 Description”. In: *CHEP 2004*. 2004. URL:  
940 <https://cds.cern.ch/record/865601/>.
- 941 [23] *Coin3D "End of life letter"*. URL:  
942 <https://bitbucket.org/Coin3D/coin/wiki/EndOfLifeLetter>.
- 943 [24] ATLAS Collaboration. *The ATLAS Atlantis visualization tool*. URL:  
944 <http://www.cern.ch/atlantis>.
- 945 [25] ATLAS Collaboration. *The ATLAS Minerva visualization tool*. URL:  
946 <http://cern.ch/atlas-minerva>.
- 947 [26] T. McCauley. “A browser-based event display for the CMS experiment at  
948 the LHC using WebGL”. In: *J.Phys. Conf. Ser.* 898.072030 (2017). URL:  
949 <http://cern.ch/ispy-webgl>.
- 950 [27] *LHCb Online event display*. URL:  
951 <https://lhcb-public.web.cern.ch/lhcb-public/lbevent2/lbevent/>.

- 952 [28] G. Sabato et al. “ATLAS fast physics monitoring: TADA”. In:  
953 *J.Phys.Conf.Ser.* 898.092015 (2017). DOI:  
954 <https://doi.org/10.1088/1742-6596/898/9/092015>.
- 955 [29] ATLAS Collaboration. *The ATLAS Tracer visualization tool*. URL:  
956 <https://atlas-tracer.web.cern.ch/>.
- 957 [30] KHRONOS Group. *WebGL - OpenGL ES for the Web*. URL:  
958 <https://www.khronos.org/webgl/>.
- 959 [31] *The Qt software development framework*. URL: <https://www.qt.io/>.
- 960 [32] *The ThreeJS 3D graphics framework*. URL: <https://threejs.org/>.
- 961 [33] *LHSee*. URL:  
962 <https://www2.physics.ox.ac.uk/about-us/outreach/public/lhsee>.
- 963 [34] *CERN Camelia visualization tool*. URL:  
964 <http://medialab.web.cern.ch/content/camelia>.
- 965 [35] *Unity Technologies - The Unity3D game engine*. URL:  
966 <https://unity3d.com>.
- 967 [36] Ilija Vukotic & ATLAS Collaboration. *ATLASRift, the ATLAS VR*  
968 *experience*. URL: <https://atlasrift.web.cern.ch/>.
- 969 [37] Leo Piilonen. *Belle II in Virtual Reality*. URL:  
970 <http://www1.phys.vt.edu/~piilonen/VR/>.
- 971 [38] R. Brun and F. Rademakers. “ROOT — An object oriented data analysis  
972 framework”. In: *Nuclear Instruments and Methods A* 389 (1997). DOI:  
973 [http://dx.doi.org/10.1016/S0168-9002\(97\)00048-X](http://dx.doi.org/10.1016/S0168-9002(97)00048-X).
- 974 [39] *DD4HEP*. URL: <https://dd4hep.web.cern.ch/dd4hep/>.
- 975 [40] The CLICdp Collaboration. *CLIC detector and physics study*. URL:  
976 <http://clicdp.web.cern.ch/>.
- 977 [41] The Future Circular Collider Study Collaboration. *Future Circular Collider*  
978 *Study*. URL: <https://fcc.web.cern.ch/Pages/default.aspx>.
- 979 [42] J Allison et al. “Recent developments in Geant4”. In: *Nuclear Instruments*  
980 *and Methods in Physics Research A* 835 (2016). URL:  
981 <http://geant4.cern.ch/>.
- 982 [43] Trimble Inc. “SketchUp”. In: (). URL: <https://www.sketchup.com/>.
- 983 [44] T. Sakuma and T. McCauley. “Detector and Event Visualization with  
984 SketchUp at the CMS Experiment”. In: *J.Phys.Conf.Ser.* 513.022032 (2014).  
985 DOI: [10.1088/1742-6596/513/2/022032](https://doi.org/10.1088/1742-6596/513/2/022032).

- 986 [45] M. Case et al. “CMS Detector Description : New Developments”. In:  
987 *Computing in High Energy Physics and Nuclear Physics 2004*. 2004. DOI:  
988 [10.5170/CERN-2005-002.498](https://doi.org/10.5170/CERN-2005-002.498).
- 989 [46] Tai Sakuma & CMS Collaboration. *3D SketchUp images of the CMS*  
990 *detector*. URL: <https://cds.cern.ch/record/2628527>.
- 991 [47] J. W. Tukey. “The Future of Data Analysis”. In: *The Annals of*  
992 *Mathematical Statistics* 33.1 (1962), pp. 1–67. URL:  
993 <http://www.jstor.org/stable/2237638>.
- 994 [48] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics  
995 Press, 1986.
- 996 [49] *ROOTJS*. URL: <https://root.cern.ch/js/>.
- 997 [50] Project Jupyter. *The Jupyter Notebook*. URL: <http://jupyter.org/>.
- 998 [51] CERN. *PyROOT*. URL: <https://root.cern.ch/pyroot>.
- 999 [52] Jake VanderPlas. *Python’s Visualization Landscape (PyCon 2017)*. URL:  
1000 <https://speakerdeck.com/jakevdp/pythons-visualization-landscape-pycon-2017>.
- 1002 [53] J.D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In*  
1003 *Science & Engineering* 9.3 (2007). DOI:  
1004 <http://dx.doi.org/10.1109/MCSE.2007.55>.
- 1005 [54] *Seaborn*. URL: <https://seaborn.pydata.org>.
- 1006 [55] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. New York:  
1007 Springer-Verlag, 2009. DOI: [10.1007/978-0-387-98141-3](https://doi.org/10.1007/978-0-387-98141-3).
- 1008 [56] D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag,  
1009 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).
- 1010 [57] L. Wilkinson. *The Grammar of Graphics*. Springer, 2005. ISBN: ISBN  
1011 978-0387245447. DOI: [10.1007/0-387-28695-0](https://doi.org/10.1007/0-387-28695-0).
- 1012 [58] R. A. Becker and W. S. Cleveland. *S-PLUS Trellis Graphics User’s Manual*.  
1013 MathSoft, 1996. URL: <http://ect.bell-labs.com/sl/project/trellis/software.writing.html>.
- 1015 [59] Marc Paterno. “Easy Data Analysis Using R”. In: *PhyStat05*. Oxford, 2005.  
1016 URL: <http://www.physics.ox.ac.uk/phystat05/Talks/PhyStat05-paterno.pdf>.
- 1018 [60] M. Bostock et al. “D3: Data-Driven Documents”. In: *IEEE Trans.*  
1019 *Visualization & Comp. Graphics (Proc. InfoVis)*. 2011. URL:  
1020 <http://vis.stanford.edu/papers/d3>.
- 1021 [61] *Bokeh*. URL: <http://www.bokeh.pydata.org>.

- 1022 [62] *Continuum*. URL: <https://www.continuum.io/>.
- 1023 [63] *Plotly*. URL: <https://plot.ly>.
- 1024 [64] R.M. Bianchi, J. Boudreau, and I. Vukotic. “A new experiment-independent  
1025 mechanism to persistify and serve the detector geometry of ATLAS”. In: *J.  
1026 Phys.: Conf. Ser.* 898.072015 (2017). DOI:  
1027 <https://doi.org/10.1088/1742-6596/898/7/072015>.
- 1028 [65] The HEP Software Foundation. *HEP Software Foundation Community  
1029 White Paper Working Group – Event/Data Processing Frameworks*.  
1030 Tech. rep. HSF-CWP-2017-08. HEP Software Foundation, 2017.
- 1031 [66] S. McKee et al. “Integrating network and transfer metrics to optimize  
1032 transfer efficiency and experiment workflows”. In: *J. Phys. Conf. Ser.* 664  
1033 (2015), p. 052003. DOI: [10.1088/1742-6596/664/5/052003](https://doi.org/10.1088/1742-6596/664/5/052003).
- 1034 [67] M. Clemencic et al. “LHCb conditions database operation assistance  
1035 systems”. In: *J. Phys. Conf. Ser.* 396 (2012), p. 052022. DOI:  
1036 [10.1088/1742-6596/396/5/052022](https://doi.org/10.1088/1742-6596/396/5/052022).
- 1037 [68] *VRML (Virtual Reality Modeling Language)*. URL:  
1038 <https://en.wikipedia.org/wiki/VRML>.
- 1039 [69] Belle II Collaboration. *The Belle II Geometry Exporters*. URL:  
1040 [https://github.com/HSF/Visualization/tree/master/demonstrators/  
1041 GeometryWriters](https://github.com/HSF/Visualization/tree/master/demonstrators/GeometryWriters).
- 1042 [70] *KHRONOS Group — glTF - The GL Transmission Format*. URL:  
1043 <https://www.khronos.org/gltf/>.
- 1044 [71] *BelleII VR models on Sketchfab*. URL:  
1045 <https://sketchfab.com/models/6c7aa0c71dc64079b08a0dddfa756ef3>.
- 1046 [72] *Sketchfab*. URL: <https://sketchfab.com>.
- 1047 [73] Matevz Tadel. “ALICE Event Visualization Environment”. In: *CHEP 2006*.  
1048 Proceedings of the Computing in High Energy and Nuclear Physics  
1049 conference (Tata Institute of Fundamental Research, Feb. 13–17, 2006).  
1050 Mumbai, India, 2006. URL:  
1051 <https://indico.cern.ch/event/408139/contributions/979909/>.
- 1052 [74] *CERN TEV visualization framework*. URL:  
1053 <https://gitlab.cern.ch/CERNMediaLab/>.
- 1054 [75] *Epic Games - The “Unreal Engine” game engine*. URL:  
1055 <https://www.unrealengine.com>.
- 1056 [76] Epic Games, Inc. *Blueprints Visual Scripting*. URL:  
1057 <https://docs.unrealengine.com/en-us/Engine/Blueprints>.

- 1058 [77] The Khronos Group Inc. *The Vulkan API*. URL:  
 1059 <https://www.khronos.org/vulkan/>.
- 1060 [78] Juan Linietsky, Ariel Manzur, and contributors. *The Godot game engine*.  
 1061 URL: <https://godotengine.org/>.
- 1062 [79] Andreas Rumpf. *The Vulkan API*. URL: <https://nim-lang.org/>.
- 1063 [80] *The Qt 3D framework*. URL:  
 1064 <https://doc.qt.io/qt-5.11/qt3d-index.html>.
- 1065 [81] *The SoQt package*. URL: <https://bitbucket.org/Coin3D/soqt>.
- 1066 [82] *The Belle II Experiment*. URL: <http://belle2.jp>.
- 1067 [83] Google. *Google Cardboard*. URL: <https://vr.google.com/cardboard/>.
- 1068 [84] Google. *Google Daydream*. URL: <https://vr.google.com/daydream/>.
- 1069 [85] Samsung Electronics. *Samsung Gear VR*. URL:  
 1070 <https://www.samsung.com/global/galaxy/gear-vr>.
- 1071 [86] LLC. Oculus VR. *Oculus Rift*. URL: <https://www.oculus.com/rift/>.
- 1072 [87] HTC Corporation. *HTC Vive*. URL: <https://www.vive.com/eu/>.
- 1073 [88] LLC. Oculus VR. *Oculus Go*. URL: <https://www.oculus.com/go/>.
- 1074 [89] Lenovo Group Ltd. Google. *Lenovo Mirage Solo*. URL:  
 1075 <https://vr.google.com/daydream/standalonevr>.
- 1076 [90] PTC Inc. *Vuforia AR framework*. URL:  
 1077 <https://www.vuforia.com/engine.html>.
- 1078 [91] Steffen Henkelmann (University of British Columbia, CA) & ATLAS  
 1079 Collaboration. *ATLAS in your pocket — An Augmented Reality application*.  
 1080 URL: <https://atlasinyourpocket.web.cern.ch/>.
- 1081 [92] Jeff Ouellette. “More Than ALICE: Development of an augmented reality  
 1082 mobile application for the ALICE detector”. In: (Aug. 2016). URL:  
 1083 <https://cds.cern.ch/record/2207593>.
- 1084 [93] *WebVR API*. URL: <https://immersive-web.github.io/webvr/>.
- 1085 [94] *AFrame - A web framework for building virtual reality experiences*. URL:  
 1086 <https://aframe.io/>.
- 1087 [95] *CMS AFrame*. URL: <https://github.com/HEPPanoramic/cms-aframe>.
- 1088 [96] The HEP Software Foundation. *HEP Software Foundation Community  
 1089 White Paper Working Group – Data Organisation, Management and Access*.  
 1090 Tech. rep. HSF-CWP-2017-04. HEP Software Foundation, 2017.

- 1091 [97] P. Lavrič, C. Bohak, and M. Marolt. “Collaborative view-aligned  
1092 annotations in web-based 3D medical data visualization”. In: *MIPRO 2017:*  
1093 *40th Jubilee International Convention*. Opatija, Croatia, May 2017,  
1094 pp. 276–280. URL: [http://lgm.fri.uni-lj.si/wp-](http://lgm.fri.uni-lj.si/wp-content/uploads/2017/10/1537435587.pdf)  
1095 [content/uploads/2017/10/1537435587.pdf](http://lgm.fri.uni-lj.si/wp-content/uploads/2017/10/1537435587.pdf).
- 1096 [98] HSF. “HSF Visualization Workshop”. In: CERN, 2017. URL:  
1097 <https://indico.cern.ch/event/617054/>.
- 1098 [99] HSF. “HSF Mini-workshop on GUI, graphic libraries, interactive  
1099 visualization”. In: CERN, 2017. URL:  
1100 <https://indico.cern.ch/event/628675/>.
- 1101 [100] HSF Visualization Working Group GitHub repository. URL:  
1102 <https://github.com/HEP-SF/Visualization>.