

¹²

³ HEP Software Foundation Community White ⁴ Paper Working Group – Visualization

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

¹¹ ^a*Siena College, Loudonville NY, USA*

¹² ^b*Cornell University, Ithaca NY, USA*

¹³ ^c*University of Pittsburgh, Pittsburgh PA, USA*

¹⁴ ^d*CNRS/IN2P3, Clermont-Ferrand, France*

¹⁵ ^e*University of Ljubljana, Ljubljana, Slovenia*

¹⁶ ^f*CERN, Geneva, Switzerland*

¹⁷ ^g*LAL, Université Paris-Sud and CNRS/IN2P3, Orsay, France*

¹⁸ ^h*FNAL, Batavia IL, USA*

¹⁹ ⁱ*GSI Darmstadt, Germany*

²⁰ ^j*University College London, London, UK*

²¹ ^k*University of Notre Dame, Notre Dame IN, USA*

²² ^l*University of Massachusetts, Amherst MA, USA*

²³ ^m*University of California at San Diego, San Diego CA, USA*

²⁴ ⁿ*University of Illinois, IL, USA*

²⁵ ^p*Virginia Tech, VA, USA*

²⁶ ^q*Princeton University, Princeton PA, USA*

²⁷ ^r*LMU Munich, Munich, Germany*

²⁸ ^s*University of Bristol, Bristol, UK*

²⁹ ^t*University of Chicago, Chicago IL, USA*

¹Paper 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	Contents	
40	1 Scope	3
41	2 Current landscape	4
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	3 Suggested guidelines and future development	17
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	4 Sharing knowledge and fostering collaboration	25
62	4.1 Yearly workshop	25
63	4.2 Code repository	26
64	5 Roadmap	26
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	6 Conclusions	27
69	7 Acknowledgements	27

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 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 in this chapter.

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

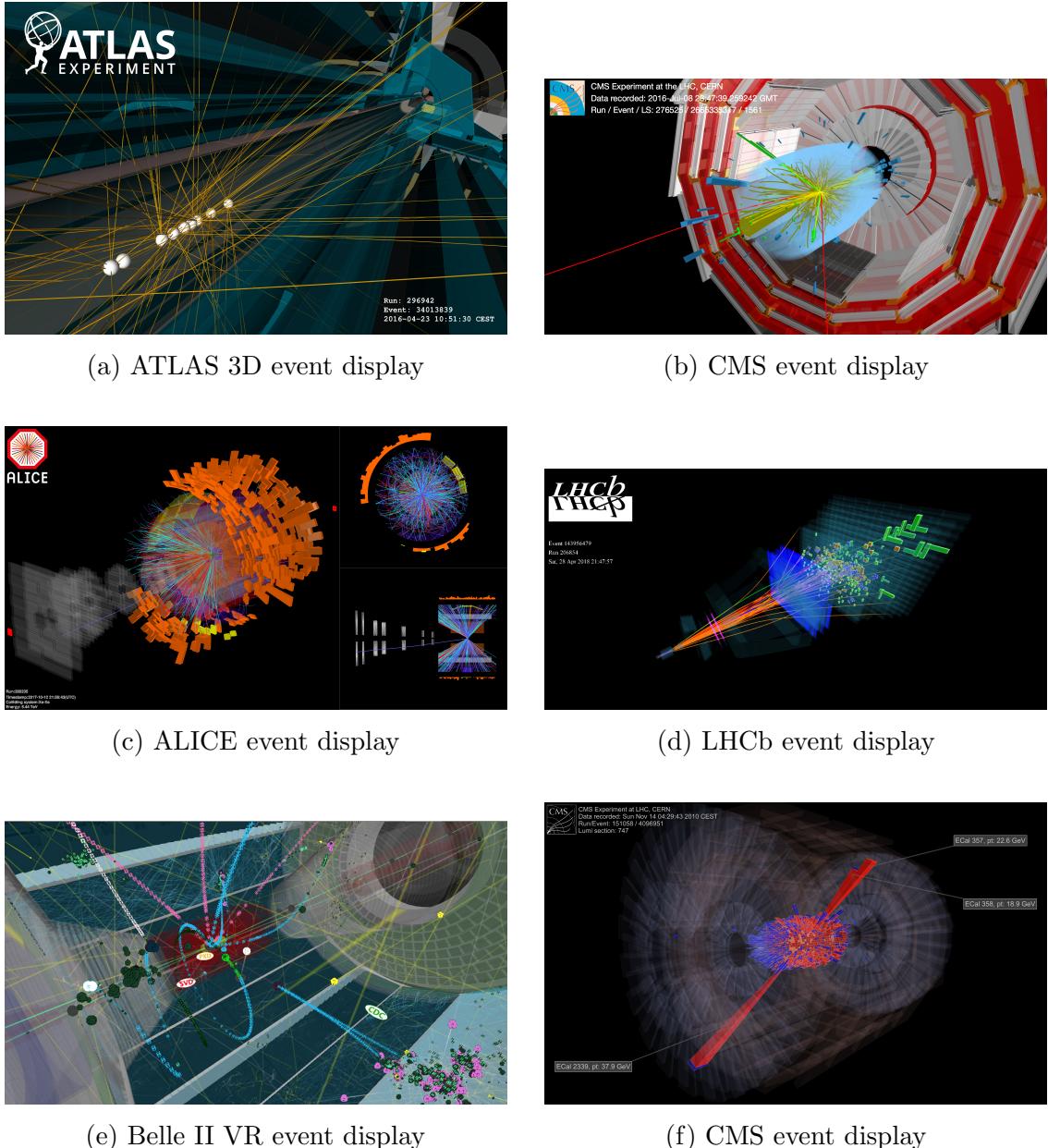


Figure 1: Different examples of HEP 3D event display applications: a) ATLAS 3D event display [4, 5]; b) CMS 3D event display [6]; c) ALICE 3D event display [7]; d) LHCb 3D event display [8]; e) Belle II VR event display [9]; f) A CMS 3D event display made with Fireworks [10]

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 some information is necessarily not accessible. Moreover,
161 every 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 6 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 [REF?], resides in its cross-platform
185 nature as OpenGL is an open standard. The OpenGL consortium defines the API:
186 the interface with which all the implementations have to comply. The actual im-
187 plementation is provided by vendors, usually targeting a specific hardware. Many

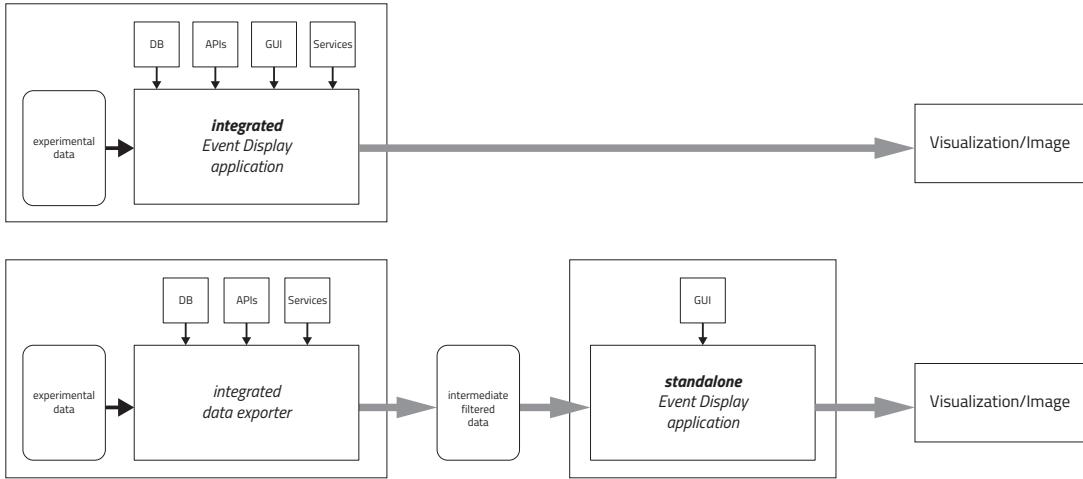


Figure 2: The two different types of event display applications. At the top, the framework-integrated event display application is able to access all experimental data, all services, APIs, and databases provided by the experiment's framework; as a drawback, the application must be run on specific platforms 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 hardware and software companies such as Intel and NVIDIA are part of the OpenGL
 189 consortium, 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 [12] and the ROOT EVE toolkit [13], which is used
 195 both by the CMS Fireworks application [10] and the ALICE Event Visualisation
 196 Environment (AliEve) [14]. One disadvantage of this approach is that personpower
 197 has to be assured to maintain the graphics engine itself, on top of the effort needed
 198 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 grpahics library used in HEP software has been Open
203 Inventor [15], used by the defunct CMS Iguana [16, 17] application, or its clone im-
204 plementation Coin (also known as Coin3D) [18], which has been used by applications
205 such the ATLAS VP1 [5], the LHCb Panoramix [19] and the defunct desktop ver-
206 sion of the CMS iSpy application [20]. Coin / Open Inventor was chosen because of
207 its integrability in C++ code, its performance, and its coding style. Moreover, the
208 way Open Inventor handles graphical volumes could be easily matched with the way
209 geometry 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” [21] 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 [22]. 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 [23]
225 program and its derivative MINERVA [24], 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 [25],
229 LHCb [26], and ATLAS [27, 28] have created web-based event displays using We-
230 bGL (Web Graphics Library) [29]. 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 [30]. 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 [31], that provide straightforward

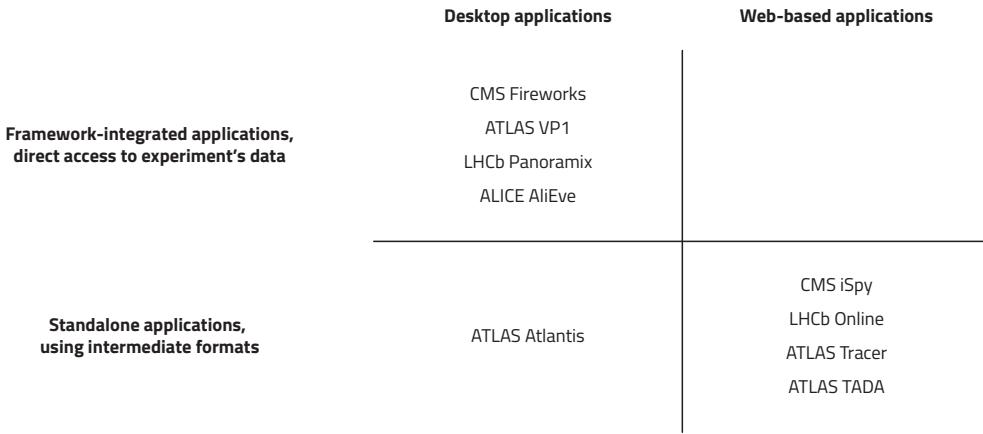


Figure 3: The image summarize the current landscape of event display applications in HEP. Many experiments developed full-framework desktop applications as well as light, web-based applications. As one can see from the plot, there are no examples, so far, of full-framework applications using web-based 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 display.

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 [32] which live-streamed ATLAS events, processed and extracted through Atlantis [23], to a user’s phone and provided contextual information on ATLAS and the events being displayed. The CMS iSpy WebGL application [25] runs on mobile devices in the browser. The Camelia application [33], developed by the CERN Media Lab using the Unity game engine [34], 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 [35] and Belle II VR [36]. 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. More information is found in Section 3.4.3.

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 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.*, [21]), while others use the geometry tools provided by the ROOT framework [37]. More recently, some attempts have

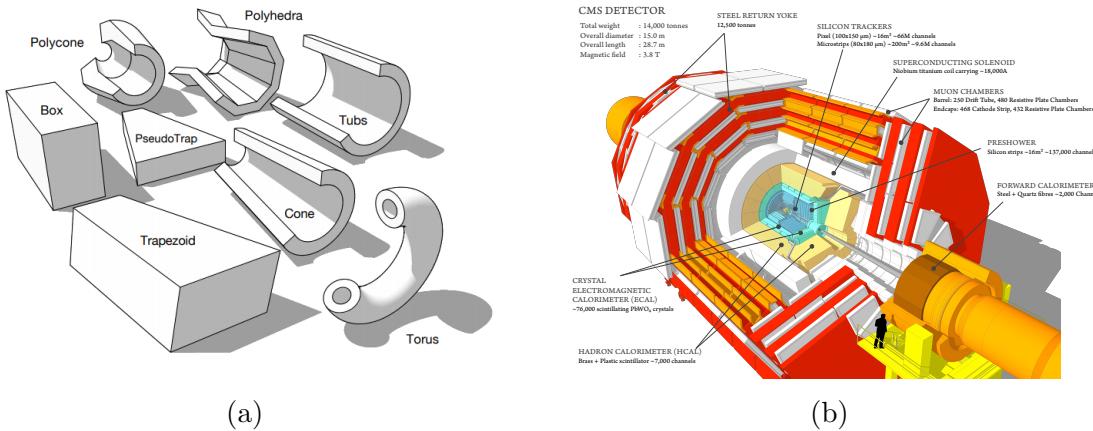


Figure 4: Use of a 3D software (SketchUp) to render the basic geometrical shapes imported from the CMS detector description and the final volumes [43].

been done in order to build common formats and libraries for detector geometry, like DD4HEP [38], adopted in conceptual design studies for future high-energy colliders, including the CLICdp [39] and FCC [40] 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 [41], 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 because 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 that of CMS and SketchUp [42] (see Figure 4). The CMS detector description 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.

325 **2.2 Statistical data visualization**

326

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

John Tukey [44]

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

330 In HEP, like most other scientific disciplines, visualization of the data and of the
331 final results plays a key role in the analysis pipeline. A new projection of the data
332 may provide new insight, results must be summarized in a clear and concise way,
333 and multidimensional parameter spaces need to be visualized in an understandable
334 fashion. The discussion of how to properly display data is not new [45], but the tools
335 are constantly evolving. Since its introduction 20 years ago, ROOT [37] has become
336 the most widely used package in HEP to make plots, graphics, and even to build
337 event displays. It was developed at a time when there were few alternatives for the
338 HEP community that did not have a significant financial cost and it has performed
339 admirably.

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

349 **2.2.1 Desktop solutions**

350 As it stands, ROOT is the most widely adopted plotting tool within the HEP and
351 Nuclear Physics community. It has even made some inroads to the astrophysics
352 community and some small pockets within the financial community, to where some
353 physicists migrated. However, few other disciplines have adopted it. Still, it has
354 many features beyond the standard 1D/2D/3D histogram/graphing tools such as 2D
355 and 3D shapes, widgets for building a GUI, a JavaScript implementation for web-
356 based analysis [46] and is available within a Jupyter notebook [47]. But to access the
357 plotting features, an analyst must install the entire ROOT package which includes
358 file I/O, scientific libraries, fitting routines, etc. and often the installation process is
359 non-trivial.

360 Many current HEP analysts make wide use of the Python programming language
361 and the PyROOT libraries [48]. Python is also very popular outside the HEP com-
362 munity and so it is worth looking at non-ROOT options available to Python users.
363 A recent (as of 2017) summary of the field was presented by Jake VanderPlas at
364 PyCon 2017 [49], a subset of which will be presented here. It is emphasized that this
365 is just a sampling and that the number of options available is a function of time.

- 366 • The Python library *matplotlib* [50], released in 2003, is the most mature plot-
367 ting tool for python and is the standard for most users. It can produce journal-
368 quality graphics and there are some add-ons that can improve the default plot-
369 ting options [51]. It does 1D, 2D, and 3D graphics with varying degrees of
370 success, but does not integrate with OpenGL libraries and so it can slow down
371 when the number of data points gets very large. It does produce most of the
372 histograms found in HEP but some minimal, extra work must be done by the
373 user to make histograms with error bars. Plots are reactive in the sense that
374 you can zoom in on different regions of the graph, but you cannot do anything
375 more significant with other mouseover commands (links, additional informa-
376 tion, etc.).
- 377 • The R programming language has several widely used graphics tools, both
378 built-in or provided by external modules. *ggplot2* [52] and *lattice* [53] are
379 particularly useful to visualize data in multidimensional parameter spaces. *gg-*
380 *plot2* is an implementation of the guidelines contained in the classic text *The*
381 *Grammar of Graphics* [54], while *lattice* is an implementation of the so-called
382 *Trellis Display* [55]. Both packages are very popular outside the HEP com-
383 munity and a wide range of learning materials are available in books, online
384 courses, and other media. Both packages are well developed and matured and
385 offer mechanism for users to extend them by adding new features. *lattice* has
386 longer history: in 2005, the year *ggplot2* first appeared, *lattice* was already
387 popular. In fact, figures made with *lattice* were shown in the presentation in
388 PHYSTAT05 [56] which introduced R to the particle physics community.

389 2.2.2 Web-based solutions

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

- D3 (Data Driven Documents) [57] 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 [58]. This is a plotting utility from Continuum [59], 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 [60]. 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 of 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

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

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

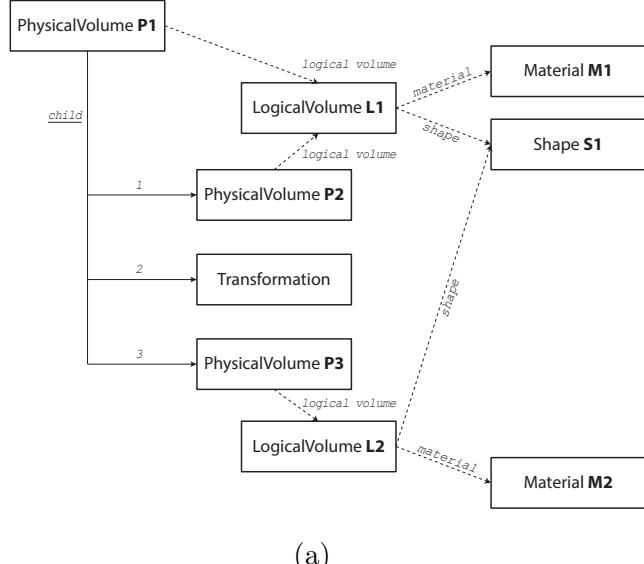
454 A looser coupling between style and content, as well as a looser coupling be-
455 tween locality of computation and locality of rendering, would benefit the physics
456 community.

457 2.3 Non-spatial visualization

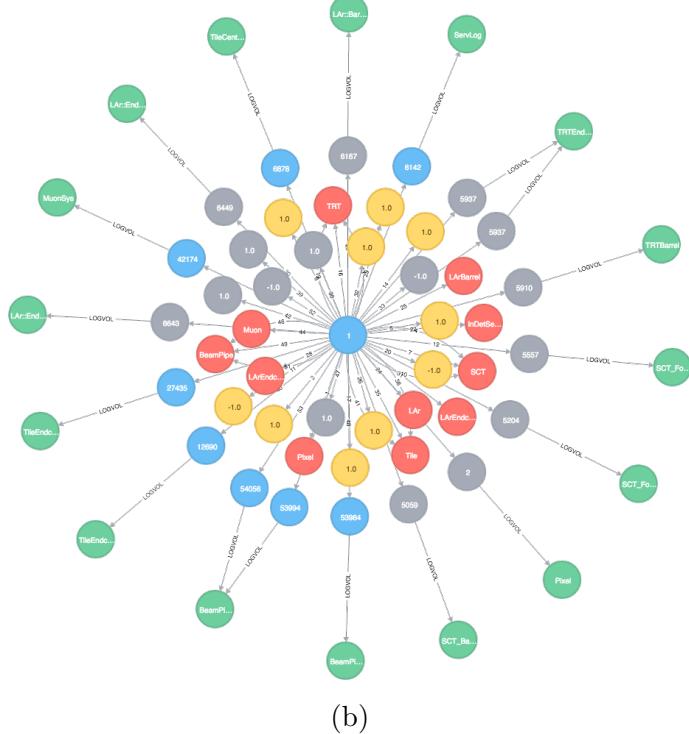
458 In HEP, there are data which are organized in a tree-like structure, and for which
459 a graph or a network visualization is the best choice. The Detector Description
460 is an example of a source of such data: it describes all the pieces which compose
461 a HEP experiment detector. The different pieces of the Detector Description are
462 interconnected through different relationships: geometrical volumes can be organized
463 in a parent-child relationship, or a property node can be shared among many volumes.
464 The visualization of those data in a network helps developers in the understanding
465 and the debugging of the Detector Description, by visualizing the relationships among
466 all the nodes and their properties. An example, from the ATLAS experiment [61],
467 of a graph visualizing the inner structure of a HEP detector description can be seen
468 in Figure 5.

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

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



(a)



(b)

Figure 5: a) a schematic drawing depicting the tree-structure of the data describing the geometry of the ATLAS detector. Data structure in such a way 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. [61].

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

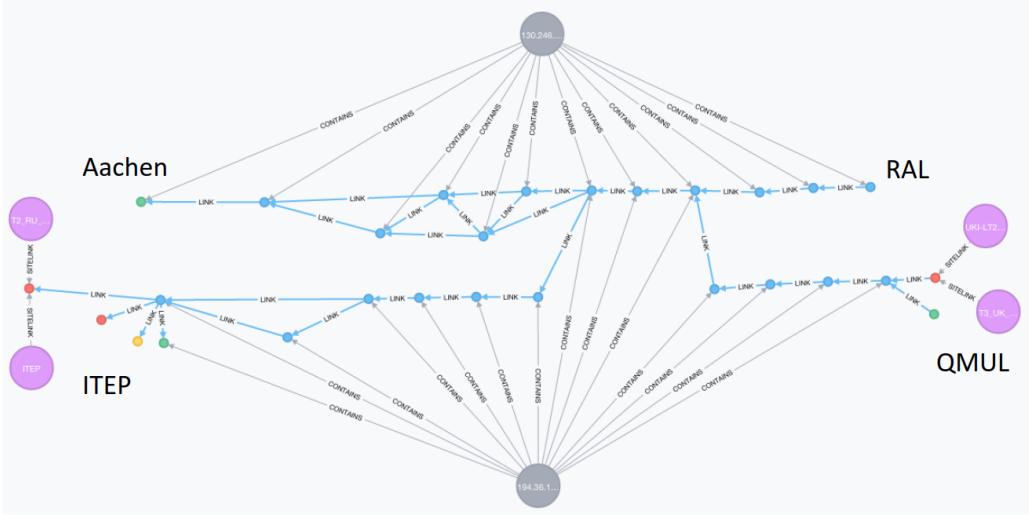


Figure 6: The image shows a graph used to visualize the links between Grid nodes and sites. [63].

478 exploit the parallelism offered by the modern hardware (more details can be found
 479 in the HSF *Event/Data Processing Frameworks* Community White Paper [62]). The
 480 jobs are handled by a scheduler, which organize them according to their need input
 481 and output data. The outcome of the scheduler is a directed acyclic graph (DAG).
 482 The visualization of that by mean of a graph helps the developer understanding and
 483 debugging the reconstruction code and the experiment's framework itself.

484 Other HEP experiments use graph-based tools to analyze and visualize other
 485 types of data, like geographical distribution and load of computer networks used to
 486 transfer data between GRID sites [63] or to store and query conditions data [64].

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

493 3 Suggested guidelines and future development

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

497 **3.1 A common community-defined format**

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

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

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

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

532 For the moment, we observe that several event displays in experiments have
533 exporters to translate geometry information to standard formats used in the com-
534 munities outside HEP, mainly in computer graphics and engineering. For example
535 the Belle II experiment has written exporters from Geant4 data to different formats,
536 including VRML [65] and FBX [66], which are two of the most common formats used

537 to store and share 3D graphics data. The Unity game-development engine [34], in
538 turn, can export the FBX geometry to the glTF format [67], an emerging royalty-free
539 specification for 3D objects and scenes, for fast web distribution via the SketchFab
540 community repository [68, 69]. Displays based on three.js have access to multiple
541 importers and exporters of several geometry formats, including the ones mentioned
542 above.

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

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

560 Some experiments in that direction have been performed within the community
561 in the past years, already. For example, the ALICE experiment made use of the mini
562 “Visualization Summary Data” (VSD) set of classes, contained in the ROOT Event
563 Visualization Environment (EVE), to make ALICE data visualization decoupled from
564 the AliROOT experiment’s framework [70].

565 3.2 Serving the geometry and event data through services

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

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

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

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

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

597 **3.3 Client-server architecture for geometry and event data visualization**

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

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

610 **3.4 Exploring modern technologies**

611 **3.4.1 Graphics and game engines**

612 So far direct OpenGL, its derivatives (like WebGL), or old graphics libraries have
613 been used in HEP visualization applications. Nowadays, another type of graphics
614 library is rapidly evolving, those embedded in the so-called game engines. Game

engines software frameworks targeted at the gaming industry, and they feature very efficient, optimized, and modern 3D graphics. The 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 [36] and the Total Event Visualiser (TEV) of the CERN Media Lab [71], which used the Unity game engine [34], and ATLAS which used the Unreal Engine [72] for its virtual reality application ATLASrift [35]. 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 [73]) 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 [74], the cross-platform 3D and computing API.

The Unity development platform [34] 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 [75]. 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 the 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 [76], 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 [77]. The key feature of Qt3D is that it is natively integrated with the Qt framework, which eliminates a layer which was needed until now, *i.e.*, a glue package to connect the Qt GUI with the window showing the 3D content (like, for example, the SoQt [78]

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

663 3.4.2 Web-based applications

664 Web-based graphics have traditionally been considered not powerful enough to handle
665 the thousands of volumes that can be shown in a HEP event displays, for example,
666 when visualizing hits in a very busy event. But the technology has rapidly evolved
667 and Web-based graphics can now visualize very complex scenes. For example, the
668 glTF [67] 3D model of the Belle II detector [79], with tens of thousands of elements,
669 can be loaded, viewed and manipulated in a web browser, even on a smartphone,
670 very effectively [68].

671 The advantages of visualization in the browser have been mentioned previously
672 and several application have already been developed. There is therefore already a
673 strong interest in the community in supporting the usage and the development of
674 web-based tools. In particular, JSROOT [46] could be used as underlying layer for
675 event data visualization as well as three.js [31] and WebGL [29], which have been
676 used successfully by different experiments (*e.g.*, [25, 27, 28]) to visualize geometry
677 and event data.

678 3.4.3 Virtual and augmented reality

679 As briefly described in Section 2.1.2, Virtual reality (VR) describes the simulation
680 of the user’s physical presence in a virtual environment. This simulation is typi-
681 cally delivered via a Head Mounted Display (HMD) that provides visual and aural
682 experience of the simulated world. Rotational and positional tracking of the user’s
683 head and hand motion (when available) allow for interaction and navigation in the
684 virtual environment. This lets the user live an immersive experience of the virtual
685 world offering many degrees of freedom. Purely rotational motion is usually referred
686 to as 3 degree-of-freedom motion; when combined with positional motion support,
687 an application is said to support 6 degrees-of-freedom.

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

692 The simplest and most inexpensive way to deliver VR is via the web browser
693 on a mobile phone and viewed through a Google Cardboard [80] headset (which can
694 itself be literally made from cardboard). Rotational tracking is achieved through

device orientation controls, either in a native application or using the HTML5 device orientation API in the browser. No hand controller is used in the Cardboard but for native applications a click event is available via a magnet attached to the Cardboard viewer.

The Google Daydream [81] is the next iteration of viewers developed by Google for their Google VR technology. Content is still delivered by a mobile phone (thus, the performance is limited by the computing power of the phone) but a Bluetooth-connected hand controller with rotational tracking is available. The Samsung Gear VR platform [82] is another headset powered by an inserted smartphone targeted for the Oculus platform.

Currently, the most immersive and interactive VR environments are provided by the Oculus Rift [83] and the HTC Vive [84], which combine the computing resources of a desktop machine with sensors, controllers, and high-quality HMDs. Those sophisticated devices are quite expensive (even if prices are lowering in the last months) and require a quite powerful computer to run. The presence of a proper computer lets solve the performance issue of phone-based viewers, since the 3D graphics computations are handled by the computer, but that also rise the initial budget for people willing to test such technologies, and that limits the number of people getting access to such platforms, as like as their usage in public events organized by HEP institutes.

In between, a new type of device has been recently developed: a standalone viewer, equipped with an on-board CPU, able to run medium computation-intensive applications. Those devices lower the budget needed to develop and deploy VR applications significantly. One example of those new devices is Oculus Go[85] and the stand-alone Lenovo headset for the Daydream environment [86].

Game engines, described in Section 3.4.1, provide powerful integrated development environments for creation of VR applications targeting multiple devices. Thanks to engines' abstraction of third party VR libraries, most HEP experiments should be able to develop VR applications which natively support both standard displays and all VR hardware. Currently, both ATLASrift [35] and Belle II VR [36] support Oculus, HTC Vive, and standard 2D displays. CMS, using the Unity game engine, is currently working on CMS.VR (to be released), targeting Oculus and HTC.

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

ATLAS and ALICE researchers have started to explore the possibilities offered by augmented reality for outreach and education, by using Unity [34] and the Vuforia framework [87] (commercial, but free for development). The ATLAS-in-Your-

736 Pocket [88] application uses printed marks to place a rendered geometry of the AT-
737 LAS detector on top of a view of the real world in front of the user. The More-Than-
738 ALICE [89] application allows users to superimpose a description of the detector or
739 event visualizations to the camera image of the actual ALICE detector (for example,
740 on a screen or during public visits to the experimental site) or its paper model.

741 Development of web-based VR and AR applications for mobile browser can be
742 done using a WebGL library such as three.js [31] and using the HTML device ori-
743 entation control API. The viewport is split into two views for each eye, each with
744 dedicated camera view separated by an appropriate distance to create a stereoscopic
745 effect (*e.g.* iSpy WebGL [25] has a stereo mode for Google Cardboard). The develop-
746 ing WebVR specification [90] provides interfaces to VR hardware via the browser. A
747 powerful framework for development of VR applications for various devices using the
748 browser is A-Frame [91]. A-Frame has support for several device controllers as well,
749 such as for the Daydream and Oculus. CMS is exploring the possibilites of A-Frame
750 in the browser for both VR and AR (*e.g.* CMS A-Frame prototype [92]).

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

757 3.4.4 Mobile technologies

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

766 That is why we strongly promote the usage of the server-client architecture
767 described and supported in this paper in Section 3.3 and the new data access patterns
768 presented and supported in the the HSF *Data Access and Management* Community
769 White Paper [93]. This would open up new possibilities for interactive visualization
770 on mobile devices: it would let visualization clients running on mobile devices connect
771 to server tools running in the experiment's framework to easily and interactively
772 retrieve the desired data.

773 It is worth noting that in other areas of science, for instance in astronomy,
774 researchers have worked to facilitate data access and to migrate to more standard
775 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 paid out 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 worth anyway.

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-users 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 a 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 an advise 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 people or students could interact together with an event display, or for virtual guided tours. Game engines offer multi-users 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 [94].

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 the knowledge among the whole HEP community, as well as the best practices and the 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

813 share their solutions. In addition, external experts from industry were invited to
814 present the latest advancements in the field and best practices [95].

815 It was the first topical workshop focused on HEP Visualization in many years.
816 Many HEP experiments and communities showed their latest developments. Given
817 the high number of presentations, a second mini-workshop [96] has been organized
818 has a follow-up, to let the remaining communities to present their work.

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

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

827 **4.2 Code repository**

828 The HSF Visualization Working Group also agreed on the importance of fostering
829 collaborative work. As a start a new dedicated project has been created within the
830 HSF GitHub repository [97]. It is intended to be the space where members of the
831 Working Group can share their work-in-progress studies and their solutions, and
832 where community-driven projects will be stored.

833 **5 Roadmap**

834 **5.1 One year**

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

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

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

846 **5.2 Three years: ATLAS and CMS Computing TDRs**

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

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

854 **5.3 Five years: Towards HL-LHC**

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

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

861 **6 Conclusions**

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

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

873 **7 Acknowledgements**

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

877 **References**

- 878 [1] Antonio Augusto Alves Jr et al. *A Roadmap for HEP Software and*
879 *Computing R&D in the 2020s*. Tech. rep. HSF-CWP-2017-01. HEP Software
880 Foundation, 2017. arXiv: [1712.06982 \[physics.comp-ph\]](https://arxiv.org/abs/1712.06982).
- 881 [2] The HEP Software Foundation. *HEP Software Foundation Community White*
882 *Paper Working Group – Data Analysis and Interpretation*. Tech. rep.
883 HSF-CWP-2017-05. HEP Software Foundation, 2017. arXiv: [1804.03983](https://arxiv.org/abs/1804.03983)
884 [[physics.comp-ph](https://arxiv.org/abs/1804.03983)].
- 885 [3] The HEP Software Foundation. *HEP Software Foundation Community White*
886 *Paper Working Group – Detector Simulation*. Tech. rep. HSF-CWP-2017-07.
887 HEP Software Foundation, 2017. arXiv: [1803.04165 \[physics.comp-ph\]](https://arxiv.org/abs/1803.04165).
- 888 [4] *ATLAS Collaboration - 13 TeV Stable Beam Collisions*. URL:
889 [https://twiki.cern.ch/twiki/pub/AtlasPublic/
890 EventDisplayRun2Collisions/JiveXML_271298_403602858-RZ-LegoPlot-
891 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).
- 892 [5] T. Kittelmann et al. “The Virtual Point 1 event display for the ATLAS
893 experiment”. In: *Journal of Physics: Conference Series* 219.3 (2010). URL:
894 <https://atlas-vp1.web.cern.ch/atlas-vp1>.
- 895 [6] CMS Collaboration. *Higgs boson produced via vector boson fusion event*
896 *recorded by CMS (Run 2, 13 TeV)*. URL:
897 <http://cds.cern.ch/record/2210658/>.
- 898 [7] ALICE Collaboration. *One of the first Xenon-Xenon collisions at LHC*
899 *registered by ALICE*. URL: <https://cds.cern.ch/record/2289018>.
- 900 [8] LHCb Collaboration. *Event collected at the beginning of 2018 data taking*.
901 URL: <http://cds.cern.ch/record/2315673>.
- 902 [9] Leo Piilonen & Belle II Collaboration. *Belle II in Virtual Reality*. URL:
903 <http://www1.phys.vt.edu/~piilonen/VR/>.
- 904 [10] L.A.T. Bauerdick et al. “Event display for the visualization of CMS events”.
905 In: *J.Phys.Conf.Ser.* 331.072039 (2011). URL:
906 <https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookFireworks>.
- 907 [11] KHRONOS Group. *OpenGL*. URL: <https://www.opengl.org/about>.
- 908 [12] L Chevalier et al. *PERSINT Event Display for ATLAS*. Tech. rep.
909 ATL-SOFT-PUB-2012-001. ATLAS Experiment, CERN, 2012. URL:
910 <https://twiki.cern.ch/twiki/bin/viewauth/Atlas/PersintWiki>.
- 911 [13] *ROOT-EVE*. URL: <https://root.cern.ch/eve>.

- 912 [14] Matevz Tadel et al. “ALICE Event Visualization Environment”. In: *CHEP*
913 *2006*. 2006. URL:
914 <https://indico.cern.ch/event/408139/contributions/979909/>.
- 915 [15] J. Wernecke. *The Inventor Mentor: Programming Object-Oriented 3D*
916 *Graphics with Open Inventor, Release 2*. Addison-Wesley, 1993.
- 917 [16] G. Alverson et al. “IGUANA: a high-performance 2D and 3D visualisation
918 system”. In: *Nuclear Instruments and Methods in Physics Research Section*
919 *A: Accelerators, Spectrometers, Detectors and Associated Equipment* 534.1
920 (2004). Proceedings of the IXth International Workshop on Advanced
921 Computing and Analysis Techniques in Physics Research, pp. 143–146. ISSN:
922 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2004.07.036>. URL:
923 <http://www.sciencedirect.com/science/article/pii/S0168900204014950>.
- 925 [17] CMS Collaboration. *Interactive Graphics for User Analysis*. URL:
926 <http://iguana.web.cern.ch/iguana/>.
- 927 [18] *The Coin3D graphics engine*. URL:
928 <https://bitbucket.org/Coin3D/coin/wiki/Home>.
- 929 [19] LHCb Collaboration. *The LHCb Panoramix visualization tool*. URL:
930 <http://lhcb-comp.web.cern.ch/lhcb-comp/frameworks/Visualization/>.
- 931 [20] G. Alverson et al. “iSpy: A powerful and lightweight event display”. In:
932 *J.Phys.Conf.Ser.* 396.022002 (2012).
- 933 [21] J. Boudreau and V. Tsulaia. “The GeoModel Toolkit for Detector
934 Description”. In: *CHEP 2004*. 2004. URL:
935 <https://cds.cern.ch/record/865601>.
- 936 [22] *Coin3D "End of life letter"*. URL:
937 <https://bitbucket.org/Coin3D/coin/wiki/EndOfLifeLetter>.
- 938 [23] ATLAS Collaboration. *The ATLAS Atlantis visualization tool*. URL:
939 <http://www.cern.ch/atlantis>.
- 940 [24] ATLAS Collaboration. *The ATLAS Minerva visualization tool*. URL:
941 <http://cern.ch/atlas-minerva>.
- 942 [25] T. McCauley. “A browser-based event display for the CMS experiment at the
943 LHC using WebGL”. In: *J.Phys.Conf.Ser.* 898.072030 (2017). URL:
944 <http://cern.ch/ispy-webgl>.
- 945 [26] *LHCb Online event display*. URL:
946 <https://lhcb-public.web.cern.ch/lhcb-public/lbevent2/lbevent/>.

- 947 [27] G. Sabato et al. “ATLAS fast physics monitoring: TADA”. In:
948 *J.Phys.Conf.Ser.* 898.092015 (2017). DOI:
949 <https://doi.org/10.1088/1742-6596/898/9/092015>.
- 950 [28] ATLAS Collaboration. *The ATLAS Tracer visualization tool*. URL:
951 <https://atlas-tracer.web.cern.ch/>.
- 952 [29] KHRONOS Group. *WebGL - OpenGL ES for the Web*. URL:
953 <https://www.khronos.org/webgl/>.
- 954 [30] *The Qt software development framework*. URL: <https://www.qt.io/>.
- 955 [31] *The ThreeJS 3D graphics framework*. URL: <https://threejs.org/>.
- 956 [32] *LHSee*. URL:
957 <https://www2.physics.ox.ac.uk/about-us/outreach/public/lhsee>.
- 958 [33] *CERN Camelia visualization tool*. URL:
959 <http://medialab.web.cern.ch/content/camelia>.
- 960 [34] *Unity Technologies - The Unity3D game engine*. URL: <https://unity3d.com>.
- 961 [35] Ilija Vukotic & ATLAS Collaboration. *ATLASRift, the ATLAS VR*
962 *experience*. URL: <https://atlasrift.web.cern.ch/>.
- 963 [36] Leo Piilonen. *Belle II in Virtual Reality*. URL:
964 <http://www1.phys.vt.edu/~piilonen/VR/>.
- 965 [37] R. Brun and F. Rademakers. “ROOT — An object oriented data analysis
966 framework”. In: *Nuclear Instruments and Methods A* 389 (1997). DOI:
967 [http://dx.doi.org/10.1016/S0168-9002\(97\)00048-X](http://dx.doi.org/10.1016/S0168-9002(97)00048-X).
- 968 [38] *DD4HEP*. URL: <https://dd4hep.web.cern.ch/dd4hep/>.
- 969 [39] The CLICdp Collaboration. *CLIC detector and physics study*. URL:
970 <http://clicdp.web.cern.ch/>.
- 971 [40] The Future Circular Collider Study Collaboration. *Future Circular Collider*
972 *Study*. URL: <https://fcc.web.cern.ch/Pages/default.aspx>.
- 973 [41] J Allison et al. “Recent developments in Geant4”. In: *Nuclear Instruments*
974 *and Methods in Physics Research A* 835 (2016). URL:
975 <http://geant4.cern.ch/>.
- 976 [42] T. Sakuma and T. McCauley. “Detector and Event Visualization with
977 SketchUp at the CMS Experiment”. In: *J.Phys.Conf.Ser.* 513.022032 (2014).
- 978 [43] Tai Sakuma & CMS Collaboration. *3D SketchUp images of the CMS detector*.
979 URL: <https://cds.cern.ch/record/2628527>.
- 980 [44] J. W. Tukey. “The Future of Data Analysis”. In: *The Annals of Mathematical*
981 *Statistics* 33.1 (1962), pp. 1–67. URL:
982 <http://www.jstor.org/stable/2237638>.

- 983 [45] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press,
 984 1986.
- 985 [46] *ROOTJS*. URL: <https://root.cern.ch/js/>.
- 986 [47] Project Jupyter. *The Jupyter Notebook*. URL: <http://jupyter.org/>.
- 987 [48] CERN. *PyROOT*. URL: <https://root.cern.ch/pyroot>.
- 988 [49] Jake VanderPlas. *Python’s Visualization Landscape (PyCon 2017)*. URL:
 989 <https://speakerdeck.com/jakevdp/pythons-visualization-landscape-pycon-2017>.
- 990 [50] J.D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing In
 991 Science & Engineering* 9.3 (2007). DOI:
 992 <http://dx.doi.org/10.1109/MCSE.2007.55>.
- 993 [51] *Seaborn*. URL: <https://seaborn.pydata.org>.
- 994 [52] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. New York:
 995 Springer-Verlag, 2009.
- 996 [53] D. Sarkar. *Lattice: Multivariate Data Visualization with R*. Springer-Verlag,
 997 2008. ISBN: ISBN 978-0-387-75968-5.
- 998 [54] L. Wilkinson. *The Grammar of Graphics*. Springer, 2005. ISBN: ISBN
 999 978-0387245447.
- 1000 [55] R. A. Becker and W. S. Cleveland. *S-PLUS Trellis Graphics User’s Manual*.
 1001 MathSoft, 1996. URL: <http://ect.bell-labs.com/sl/project/trellis/software.writing.html>.
- 1002 [56] Marc Paterno. “Easy Data Analysis Using R”. In: *PhyStat05*. Oxford, 2005.
 1003 URL: <http://www.physics.ox.ac.uk/phystat05/Talks/PhyStat05-paterno.pdf>.
- 1004 [57] M. Bostock et al. “D3: Data-Driven Documents”. In: *IEEE Trans.
 1005 Visualization & Comp. Graphics (Proc. InfoVis)*. 2011. URL:
 1006 <http://vis.stanford.edu/papers/d3>.
- 1007 [58] *Bokeh*. URL: <http://www.bokeh.pydata.org>.
- 1008 [59] *Continuum*. URL: <https://www.continuum.io/>.
- 1009 [60] *Plotly*. URL: <https://plot.ly>.
- 1010 [61] R.M. Bianchi, J. Boudreau, and I. Vukotic. “A new experiment-independent
 1011 mechanism to persistify and serve the detector geometry of ATLAS”. In: *J.
 1012 Phys.: Conf. Ser.* 898.072015 (2017). DOI:
 1013 <https://doi.org/10.1088/1742-6596/898/7/072015>.

- 1017 [62] The HEP Software Foundation. *HEP Software Foundation Community White*
 1018 *Paper Working Group – Event/Data Processing Frameworks*. Tech. rep.
 1019 HSF-CWP-2017-08. HEP Software Foundation, 2017.
- 1020 [63] Marian Babik and Shawn McKee and Saul Youssef. *Proximity service*. URL:
 1021 [https://indico.cern.ch/event/382624/attachments/762785/1046477/
 1022 proximity.pdf](https://indico.cern.ch/event/382624/attachments/762785/1046477/proximity.pdf).
- 1023 [64] M. Clemencic et al. “LHCb conditions database operation assistance
 1024 systems”. In: *J. Phys. Conf. Ser.* 396 (2012), p. 052022. DOI:
 1025 [10.1088/1742-6596/396/5/052022](https://doi.org/10.1088/1742-6596/396/5/052022).
- 1026 [65] *VRML (Virtual Reality Modeling Language)*. URL:
 1027 <https://en.wikipedia.org/wiki/VRML>.
- 1028 [66] Belle II Collaboration. *The Belle II Geometry Exporters*. URL:
 1029 <https://github.com/HSF/Visualization/tree/master/demonstrators/>
 1030 *GeometryWriters*.
- 1031 [67] *KHRONOS Group — glTF - The GL Transmission Format*. URL:
 1032 <https://www.khronos.org/gltf/>.
- 1033 [68] *BelleII VR models on Sketchfab*. URL:
 1034 <https://sketchfab.com/models/6c7aa0c71dc64079b08a0dddfa756ef3>.
- 1035 [69] *Sketchfab*. URL: <https://sketchfab.com>.
- 1036 [70] Matevz Tadel. “ALICE Event Visualization Environment”. In: *CHEP 2006*.
 1037 Proceedings of the Computing in High Energy and Nuclear Physics
 1038 conference (Tata Institute of Fundamental Research, Feb. 13–17, 2006).
 1039 Mumbai, India, 2006. URL:
 1040 <https://indico.cern.ch/event/408139/contributions/979909/>.
- 1041 [71] *CERN TEV visualization framework*. URL:
 1042 <https://gitlab.cern.ch/CERNMediaLab/>.
- 1043 [72] *Epic Games - The “Unreal Engine” game engine*. URL:
 1044 <https://www.unrealengine.com>.
- 1045 [73] Epic Games, Inc. *Blueprints Visual Scripting*. URL:
 1046 <https://docs.unrealengine.com/en-us/Engine/Blueprints>.
- 1047 [74] The Khronos Group Inc. *The Vulkan API*. URL:
 1048 <https://www.khronos.org/vulkan/>.
- 1049 [75] Juan Linietsky, Ariel Manzur, and contributors. *The Godot game engine*.
 1050 URL: <https://godotengine.org/>.
- 1051 [76] Andreas Rumpf. *The Vulkan API*. URL: <https://nim-lang.org/>.

- 1052 [77] *The Qt 3D framework*. URL:
<https://doc.qt.io/qt-5.11/qt3d-index.html>.
- 1053 [78] *The SoQt package*. URL: <https://bitbucket.org/Coin3D/soqt>.
- 1054 [79] *The Belle II Experiment*. URL: <http://belle2.jp>.
- 1055 [80] Google. *Google Cardboard*. URL: <https://vr.google.com/cardboard/>.
- 1056 [81] Google. *Google Daydream*. URL: <https://vr.google.com/daydream/>.
- 1057 [82] Samsung Electronics. *Samsung Gear VR*. URL:
<https://www.samsung.com/global/galaxy/gear-vr>.
- 1058 [83] LLC. Oculus VR. *Oculus Rift*. URL: <https://www.oculus.com/rift/>.
- 1059 [84] HTC Corporation. *HTC Vive*. URL: <https://www.vive.com/eu/>.
- 1060 [85] LLC. Oculus VR. *Oculus Go*. URL: <https://www.oculus.com/go/>.
- 1061 [86] Lenovo Group Ltd. Google. *Lenovo Mirage Solo*. URL:
<https://vr.google.com/daydream/standalonevr/>.
- 1062 [87] PTC Inc. *Vuforia AR framework*. URL:
<https://www.vuforia.com/engine.html>.
- 1063 [88] Steffen Henkelmann (University of British Columbia, CA) & ATLAS
1064 Collaboration. *ATLAS in your pocket — An Augmented Reality application*.
1065 URL: <https://atlasinyourpocket.web.cern.ch/>.
- 1066 [89] Jeff Ouellette. “More Than ALICE: Development of an augmented reality
1067 mobile application for the ALICE detector”. In: (Aug. 2016). URL:
1068 <https://cds.cern.ch/record/2207593>.
- 1069 [90] *WebVR API*. URL: <https://immersive-web.github.io/webvr/>.
- 1070 [91] *AFrame - A web framework for building virtual reality experiences*. URL:
<https://aframe.io/>.
- 1071 [92] *CMS AFrame*. URL: <https://github.com/HEPPanoramic/cms-aframe>.
- 1072 [93] The HEP Software Foundation. *HEP Software Foundation Community White
1073 Paper Working Group – Data Organisation, Management and Access*.
1074 Tech. rep. HSF-CWP-2017-04. HEP Software Foundation, 2017.
- 1075 [94] P. Lavrič, C. Bohak, and M. Marolt. “Collaborative view-aligned annotations
1076 in web-based 3D medical data visualization”. In: *MIPRO 2017: 40th Jubilee
1077 International Convention*. Opatija, Croatia, May 2017, pp. 276–280. URL:
1078 [http://lgm.fri.uni-lj.si/wp-
1079 content/uploads/2017/10/1537435587.pdf](http://lgm.fri.uni-lj.si/wp-content/uploads/2017/10/1537435587.pdf).
- 1080 [95] HSF. “HSF Visualization Workshop”. In: CERN, 2017. URL:
<https://indico.cern.ch/event/617054/>.

- 1087 [96] HSF. “HSF Mini-workshop on GUI, graphic libraries, interactive
1088 visualization”. In: CERN, 2017. URL:
1089 <https://indico.cern.ch/event/628675/>.
- 1090 [97] HSF Visualization Working Group GitHub repository. URL:
1091 <https://github.com/HEP-SF/Visualization>.