NUCL SCI TECH (2022) 33:142
https://doi.org/10.1007/s41365-022-01133-8

Method for detector description transformation to Unity and application in BESIII

Kai-Xuan Huang[1] · Zhi-Jun Li[1] · Zhen Qian[1] · Jiang Zhu[1] · Hao-Yuan Li[1] · Yu-Mei Zhang[2] · Sheng-Sen Sun[3,4] · Zheng-Yun You[1]

Received: 21 June 2022 / Revised: 3 September 2022 / Accepted: 20 September 2022 / Published online: 9 November 2022

Abstract Detector and event visualization are essential parts of the software used in high-energy physics (HEP) experiments. Modern visualization techniques and multimedia production platforms such as Unity provide impressive display effects and professional extensions for visualization in HEP experiments. In this study, a method for automatic detector description transformation is presented, which can convert the complicated HEP detector geometry from GDML in offline software to 3D modeling in Unity. The method was successfully applied in the BESIII experiment and can be further developed into applications such as event displays, data monitoring, or virtual reality. It has great potential in detector design, offline software development, physics analysis, and outreach for next-generation HEP experiments as well as applications in nuclear techniques for the industry.

Keywords Detector description · Visualization · Unity · GDML · BESIII

This work was supported by the National Natural Science Foundation of China (Nos. 11975021, 12175321, 11675275, U1832204, and U1932101), National Key Research and Development Program of China (Nos. 2020YFA0406300 and 2020YFA0406400), Guangdong Basic and Applied Basic Research Foundation (No. 2021A1515012039), State Key Laboratory of Nuclear Physics and Technology, Peking University (Nos. NPT2020KFY04 and NPT2020KFY05), Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA10010900), Chinese Academy of Sciences (CAS) Large-Scale Scientific Facility Program; Fundamental Research Funds for the Central Universities, Sun Yat-sen University, National College Students Science and Technology Innovation Project, and Undergraduate Base Scientific Research Project of Sun Yat-sen University.

✉ Zheng-Yun You
youzhy5@mail.sysu.edu.cn

Yu-Mei Zhang
zhangym26@mail.sysu.edu.cn

[1] School of Physics, Sun Yat-sen University, Guangzhou 510275, China

[2] Sino-French Institute of Nuclear Engineering and Technology, Sun Yat-sen University, Zhuhai 519082, China

[3] Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China

[4] University of Chinese Academy of Sciences, Beijing 100049, China

1 Introduction

Detector description and visualization play important roles in various aspects of the life cycle of a high-energy physics (HEP) experiment, including detector design, optimization, simulation, reconstruction, detector commissioning, monitoring, event display, physics analysis, outreach, and education. In the HEP Software Foundation (HSF) Community White Paper [1] and the Roadmap for HEP Software and Computing R&D for the 2020s [2], suggestions and guidelines for visualization tools and techniques in future experiments have been specifically discussed, focusing on detector geometry visualization, event display, and interactivity.

The detectors in HEP experiments are usually large-scale scientific apparatuses with complicated geometries, which are composed of millions of detector units, such as the ATLAS [3] and CMS [4] detectors at the Large Hadron Collider (LHC) [5]. The detector description is developed based on professional geometry toolkits, such as the commonly used Geometry Description Markup Language

(GDML) [6] and Detector Description Toolkit for High Energy Physics (DD4hep) [7]. Detector geometry information in GDML or DD4hep format can be imported into the offline software of an experiment to provide a consistent detector description for different applications [8, 9].

However, displaying complicated detectors in offline software is difficult because the HEP software community does not have a common visualization tool that meets the requirements of visualization effects, speed, efficiency, and portability. The ROOT software [10] and its EVE package [11] have been used in some experiments for detector visualization and event display, such as ALICE [12], BESIII [13], and JUNO [14, 15]. Although ROOT-based event display tools have the advantage of their development in offline software frameworks, their visualization effects are limited, and they are also difficult to implement on platforms other than Linux.

In recent years, popular video game engines from the industry, such as Unity [16], have been used in HEP experiments for detector visualization and event display. Unity is a cross-platform engine for creating games, architecture, videos, and animation. It supports more than 20 different platforms, including Windows, Linux, macOS, iOS, and Android, which makes Unity the most popular application development platform on Apple Store and Google Play. The applications of Unity in ATLAS [3], BelleII [17], and JUNO [18] have achieved good visualization effects, and thus Unity is a promising platform for visualization in future HEP experiments.

Despite the advantages of great display effects and cross-platform support, a critical problem in using Unity for detector visualization is that it does not support the commonly used toolkits for HEP detector description, GDML and DD4hep. Developers have to use the 3D modeling system in Unity to reconstruct the complicated detector from scratch again, which not only requires extra human work and creates maintenance problems, but could also make the detector description in Unity inconsistent with that in the offline software.

In this study, we present a method to automate the conversion of the GDML detector description into Unity. The full detector geometry and its architecture, as described with GDML or ROOT, can be imported into Unity for automatic 3D detector modeling. The correlation between detector elements and identifiers is also preserved, which makes it convenient for further developments in event displays or virtual reality (VR) applications. The method was tested and validated using the BESIII detector description [8, 9].

The remainder of this paper is organized as follows. In Sect. 2, we introduce the detector description and its visualization in HEP offline software and Unity. In Sect. 3, the method for transformation from GDML to Unity and the detector data flow is presented. Its application in BESIII is introduced in Sect. 4. The potential for further development of applications is discussed in Sect. 5.

2 Detector geometry and visualization

2.1 Detector description in HEP software

Detector description is an indispensable part of the HEP offline software as it provides the detector geometry and status information for different applications, including simulation, reconstruction, calibration, event display, and physics analysis. Commonly used HEP infrastructure software products, such as Geant4 [19] or ROOT [10], have their own specific detector construction systems. If software developers define the detector geometry in these types of software independently, potential inconsistencies between them could be created. To solve this problem, some frameworks provide software-independent detector descriptions, such as GDML and DD4hep, for detector-related applications in the HEP offline software.

GDML [6] is a detector description language based on extensible markup language (XML) [20]. It describes the detector information through a set of tags and attributes in plain text format to provide a persistent detector description for an experiment. Several HEP experiments, including BESIII [8, 9], PHENIX [21], LHCb [22–24], and JUNO [25], have used GDML to describe and optimize the detector geometry in conceptual design and offline software development [26–28].

DD4hep [7] is a software framework that provides a complete solution for a full detector description, including geometry, materials, visualization, readout, alignment, and calibration, for the full experimental life cycle. It offers a consistent description through a single source of detector information for simulation, reconstruction, and analysis. DD4hep is aimed at applications in next-generation HEP experiments, such as CEPC [29], ILC [30], FCC [31], and STCF [32].

2.2 Visualization of detector and events

One important application of detector description is to visualize it so that users can have a distinct view to better understand HEP detectors. This is extremely important at the stages of detector conceptual design and commissioning. In combination with event visualization, event display provides a powerful tool to demonstrate the detector response to particles, which plays an essential role in offline software development and physics analysis.

However, as part of the traditional industrial design, computer-aided design (CAD) [33] has been dominant in several stages of HEP experiments, including detector design, construction, and commissioning. There have always been difficulties in sharing 3D modeling information between the CAD and HEP offline software because they belong to two different fields, industrial design and HEP experimentation, respectively.

In HEP experiments, physicists usually develop detector descriptions and event visualization tools in the HEP offline software. In offline software, developers can make full use of the detector geometry service and event data model to retrieve the corresponding information. Event display tools are typically based on commonly used HEP software, Geant4 or ROOT, whose visualization functions are friendly to HEP users and are naturally convenient for visualization software development. For example, the BESIII event display software is based on the infrastructure visualization function of ROOT. With the upgrade of ROOT and its EVE package [11], the development of event display tools has become more convenient. Several recent HEP experiments, such as ALICE [12], CMS [4], JUNO [25], and Mu2e [34], have developed event-display software based on ROOT EVE.

However, owing to the limited visualization support of ROOT, its display effects are unsatisfactory for meeting the diverse requirements of physicists. Most ROOT applications are limited to Linux platforms. For better visualization and interactivity, several event display tools have been developed based on an external visualization software. Atlantis [35, 36] and VP1 [37] are general-purpose event display tools in ATLAS. CMS has also developed several visualization systems such as Fireworks [38] and SketchUp [39].

2.3 3D modeling and visualization in Unity

Unity is a professional video and game production engine that supports more than 20 platforms. It is particularly popular for iOS and Android mobile app development and has recently been used in HEP for scientific research and outreach. The Cross-platform Atlas Multimedia Educational Lab for Interactive Analysis (CAMELIA) [40, 41] is an event display software based on Unity for ATLAS. Figure 1 shows the visualization of the ATLAS detector and a proton–proton collision event in CAMELIA. Another event display tool based on Unity, the Event Live Animation with Unity for Neutrino Analysis (ELAINA) [18] has also been developed in JUNO, as shown in Fig. 2.

The visualization software based on Unity has several advantages.

- *Impressive visualization effects.* The Unity engine, as a professional 3D software, provides a more detailed description of objects and striking visualization effects, which makes it much more powerful than the traditional event display based on ROOT.
- *Cross-platform support.* Thanks to the multi-platform support of Unity, after the development by building models and functions in Unity, a project can be directly exported and deployed in different operating systems, including Windows, Linux, macOS, iOS, Android, and network browsers. This feature makes the same visualization project available for all the users on different platforms. This not only reduces the workload in project development, but also facilitates the maintenance.
- *Extensibility.* Unity allows the project to be extended and further developed into VR [42] or augmented reality (AR) [43] projects, such as BelleII VR [44], thus providing a quite different way for detector design, offline software development, and physics analysis. It can also be very helpful for advertising scientific projects to the public, especially the physics behind the large-scale scientific apparatus. It could be a powerful

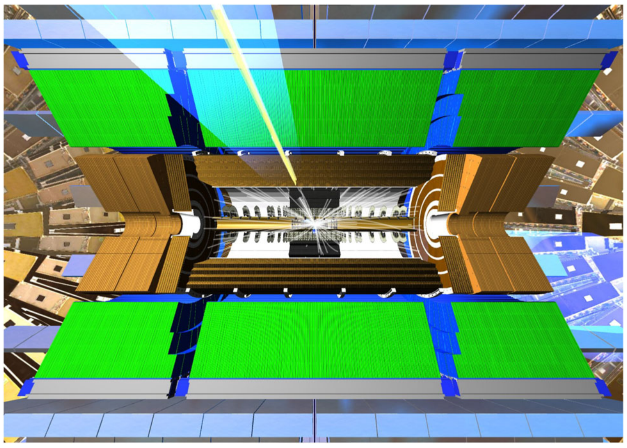
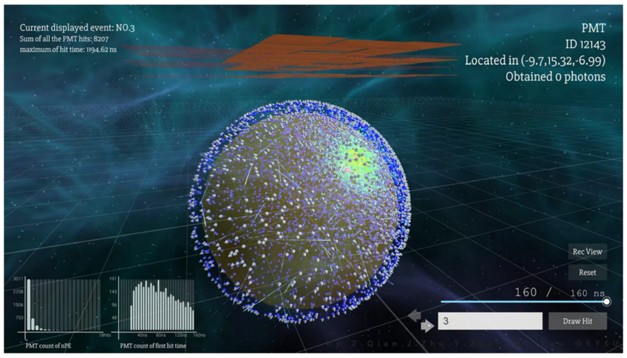


Fig. 1 (Color online) CAMELIA [40], the ATLAS event display tool based on Unity


Current displayed event: NO.3
Sum of all the PMT hits: 8207
maximum of hit time: 1194.62 ns
PMT
ID 12143
Located in (-9.7,15.32,-6.99)
Obtained 0 photons
Rec View
Reset
160 / 160 ns
3
Draw Hit
PMT count of nPE
PMT count of first hit time
Fig. 2 (Color online) ELAINA [18], the JUNO event display tool based on Unity

tool in education and outreach, such as the Total Event Visualiser (TEV) of the CERN Media Lab [45]. Support for various VR or AR devices has been integrated in Unity, and they will be continuously supported with Unity upgrades, so that the visualization project developers can focus on the software project itself, instead of the support for different hardware.

Because of its prominent features, Unity has huge potential and boosts promising prospects in the development of visualization software. Not only can it be used for scientific research in particle and nuclear physics, but it also has wide application potential in nuclear techniques for the industry.

3 Methodologies

Although Unity has great advantages for the development of visualization software, many endeavors have to be made for its application to large-scale scientific apparatuses. For example, HEP detectors are typically highly complicated, with millions of components, which makes it difficult to build a 3D model of the detector in Unity.

To develop the event display tool for HEP experiments, the detector description in Unity should be fully consistent with that in the offline software, so that the displays of the tracks, hits, showers, and detector components match each other. Usually, for an HEP experiment, the detector description already exists in offline software. Therefore, we propose a method to convert the HEP detector description into 3D modeling in Unity automatically.

3.1 Automatic detector geometry transformation

In HEP offline software, to avoid inconsistency of the detector description in data processing, it is essential that the framework can provide a single source of detector description for all applications. This idea is typically realized by developing automatic geometry transformation interfaces between different software packages. For an existing detector description with GDML, the detector construction in Geant4 and ROOT for a specific experiment can be automatically completed with the GDML–Geant4 and GDML–ROOT interfaces [8, 9], respectively. In this way, detector geometry consistency between the Geant4-based simulation, ROOT-based reconstruction, event display, and data analysis is automatically guaranteed.

A similar idea can also be implemented in the 3D detector modeling of Unity. HEP experiments usually already have a detailed detector description in some formats, such as GDML. If a novel method for GDML–Unity conversion can be realized, and given that it is a universal technique, all the current and future HEP experiments can benefit, making it possible the easy development of applications for detector visualization, event display, and outreach.

3.2 Detector data conversion from GDML to Unity

In the industry market, there are tens of popular 3D file formats; however, none of them are commonly supported in both HEP software and Unity. Therefore, we must find a data flow path that starts from GDML and ends in Unity with minimum steps of conversion.

FreeCAD [46] is a general-purpose parametric 3D CAD and modeling software. It is free and open-source, and users can extend its functionality. Because FreeCAD supports the import of geometry in constructive solid geometry (CSG) format [47], an interface between GDML and FreeCAD was developed by Keith Sloan et al. [48] to import the GDML-format detector description. Meanwhile, FreeCAD allows to export the modeling in several types of widespread 3D formats, including STEP [49], BREP [50], and VRML [51].

Among these formats, STEP was adopted for further geometry transformation. Although STEP is still not a format that can be directly read by Unity, it can be transformed into the Filmbox (FBX) format [52] with Pixyz Studio software [53]. Pixyz Studio is a CAD data preparation and optimization software that supports more than 35 3D file formats. It provides tessellation algorithms, enabling the transformation of CAD data from industry-leading solutions into lightweight, optimized meshes such as FBX, which is a 3D mesh format supported by Unity. It is noteworthy that Pixyz joined Unity so that Unity can provide plugins to import and transform heavy and complex 3D CAD data, such as HEP detectors, into optimized meshes for real-time 3D engines.

The full chain of data flow is shown in Fig. 3. The original HEP detector description in the GDML format usually consists of the following parts: the material list, position and rotation list, shape list, detector component (physical node) list, and hierarchy of the whole detector tree. The GDML detector description is first imported into FreeCAD using the GDML–FreeCAD interface and then exported into the STEP format. Pixzy reads in the STEP format data and transforms it into the FBX format, which can be read directly by Unity retaining the detector unit association information.

Using the data flow chain described above, we provide a method to transform the GDML detector description into Unity with automated 3D detector construction. The method was realized using the GDML–FreeCAD interface, FreeCAD, and Pixyz software. The correctness of the

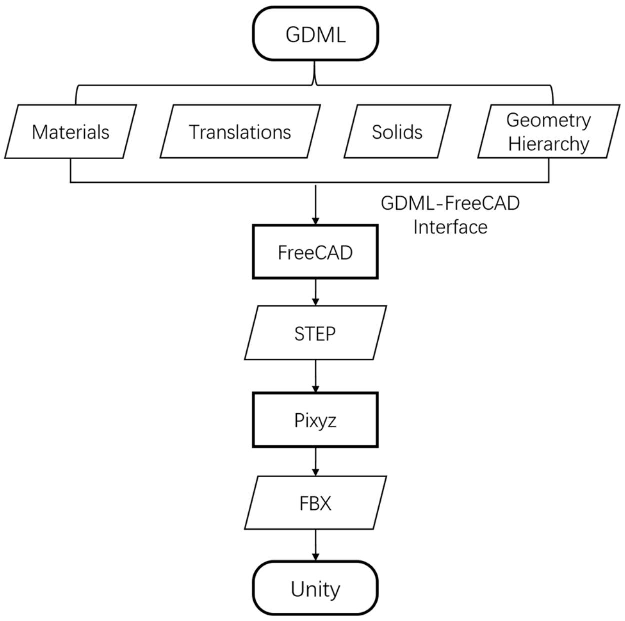


Fig. 3 Detector data flow from GDML to Unity

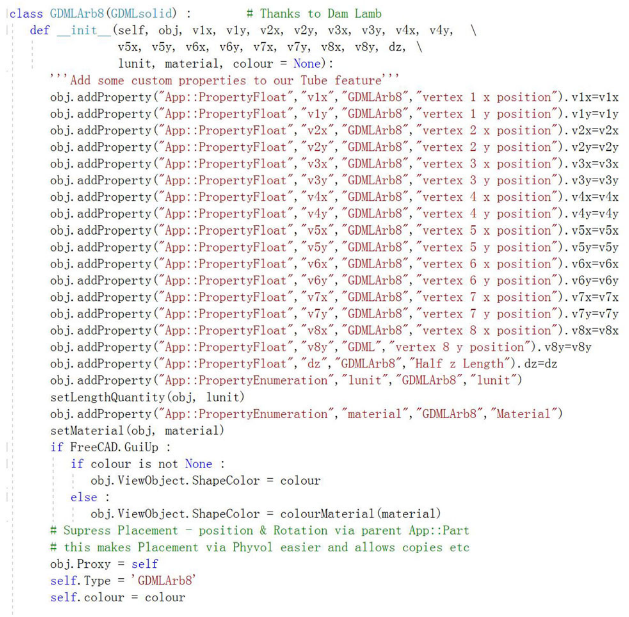
detector geometry conversion in each step was validated by visualization and comparison at the shape and detector levels. With the 3D detector modeling successfully constructed in Unity with automation, further application development based on Unity is achievable.

3.3 Integrity in Unity 3D modeling

For a specific HEP detector with a GDML description, to realize a complete 3D model of the detector in Unity, additional work is necessary in addition to the automatic geometry transformation method introduced above.

First, in the GDML–FreeCAD interface [48], not all solid shapes that are currently defined in Geant4 or ROOT are fully supported. The current interface only supports approximately 10 types of shapes, which are mostly basic CSG shapes, such as boxes, cones, cylinders, ellipsoids, and tubes. More than 30 types of specific shapes are provided in Geant4 but are not commonly used. For a specific HEP detector description with the GDML format, if some of the special shapes used are not available in the interface, users may request to develop the corresponding shape transformation in the GDML–FreeCAD interface, so that the full detector transformation can be supported.

For example, Arb8 is an arbitrary trapezoid defined by eight vertices standing on two parallel planes perpendicular to the *Z*-axis. The Arb8 shape exists in GDML, ROOT, and FreeCAD. In the GDML–FreeCAD interface, the conversion interface is updated with Arb8 shape support, and the code example is shown in Fig. 4. The same Arb8 shape



```
class GDMLArb8(GDMLsolid) :        # Thanks to Dam Lamb
   def __init__(self, obj, v1x, v1y, v2x, v2y, v3x, v3y, v4x, v4y,  \
                v5x, v5y, v6x, v6y, v7x, v7y, v8x, v8y, dz, \
                lunit, material, colour = None):
      '''Add some custom properties to our Tube feature'''
      obj.addProperty("App::PropertyFloat","v1x","GDMLArb8","vertex 1 x position").v1x=v1x
      obj.addProperty("App::PropertyFloat","v1y","GDMLArb8","vertex 1 y position").v1y=v1y
      obj.addProperty("App::PropertyFloat","v2x","GDMLArb8","vertex 2 x position").v2x=v2x
      obj.addProperty("App::PropertyFloat","v2y","GDMLArb8","vertex 2 y position").v2y=v2y
      obj.addProperty("App::PropertyFloat","v3x","GDMLArb8","vertex 3 x position").v3x=v3x
      obj.addProperty("App::PropertyFloat","v3y","GDMLArb8","vertex 3 y position").v3y=v3y
      obj.addProperty("App::PropertyFloat","v4x","GDMLArb8","vertex 4 x position").v4x=v4x
      obj.addProperty("App::PropertyFloat","v4y","GDMLArb8","vertex 4 y position").v4y=v4y
      obj.addProperty("App::PropertyFloat","v5x","GDMLArb8","vertex 5 x position").v5x=v5x
      obj.addProperty("App::PropertyFloat","v5y","GDMLArb8","vertex 5 y position").v5y=v5y
      obj.addProperty("App::PropertyFloat","v6x","GDMLArb8","vertex 6 x position").v6x=v6x
      obj.addProperty("App::PropertyFloat","v6y","GDMLArb8","vertex 6 y position").v6y=v6y
      obj.addProperty("App::PropertyFloat","v7x","GDMLArb8","vertex 7 x position").v7x=v7x
      obj.addProperty("App::PropertyFloat","v7y","GDMLArb8","vertex 7 y position").v7y=v7y
      obj.addProperty("App::PropertyFloat","v8x","GDMLArb8","vertex 8 x position").v8x=v8x
      obj.addProperty("App::PropertyFloat","v8y","GDML","vertex 8 y position").v8y=v8y
      obj.addProperty("App::PropertyFloat","dz","GDMLArb8","Half z Length").dz=dz
      obj.addProperty("App::PropertyEnumeration","lunit","GDMLArb8","lunit")
      setLengthQuantity(obj, lunit)
      obj.addProperty("App::PropertyEnumeration","material","GDMLArb8","Material")
      setMaterial(obj, material)
      if FreeCAD.GuiUp :
         if colour is not None :
            obj.ViewObject.ShapeColor = colour
         else :
            obj.ViewObject.ShapeColor = colourMaterial(material)
      # Supress Placement - position & Rotation via parent App::Part
      # this makes Placement via Phyvol easier and allows copies etc
      obj.Proxy = self
      self.Type = 'GDMLArb8'
      self.colour = colour
```

Fig. 4 Code example for conversion of the Arb8 shape in the GDML–FreeCAD interface [48]

with the GDML description and its visualization in ROOT and FreeCAD are also compared in Fig. 5.

Second, in the description of an HEP detector, it is important to maintain the association between the detector unit and its unique identifier in the event data model. Such association information is essential for controlling the visualization properties of the detector unit in further application development in Unity, such as event displays. Usually, identifier information is stored in the name of each node in GDML. As long as the node name remains unchanged during the entire chain of conversion, the identifier can be extracted later in Unity from the unit's name to retrieve the mapping of the unit and its corresponding identifier in the offline software. This makes it possible to control the visual effects of a geometry unit with its identifier in event data. Hence, in the transformation of the detector geometry from GDML to Unity, it is

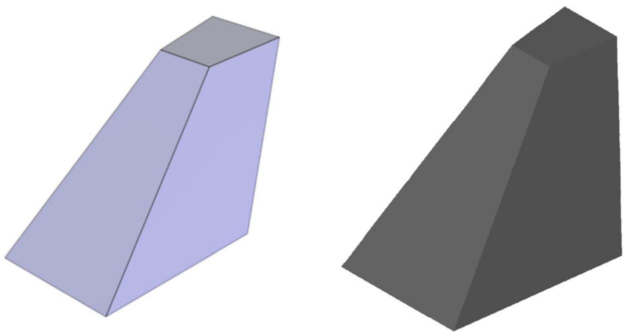


Fig. 5 (Color online) Visualization of the Arb8 shape in ROOT (left) and FreeCAD (right)

critical to retain the name of each detector unit to retain association information.

Third, although basic information such as density and makeup of materials can be transformed from GDML, Unity provides richer visualization properties such as material color, texture, transparency, and reflection, which allow users to have the freedom to render the detector geometry with more powerful visualization effects.

With the above supplementary information provided, 3D detector modeling has been successfully constructed in Unity and has been qualified for further development.

4 Application in BESIII

4.1 BESIII detector description and visualization

BESIII is a spectrometer operating at the Beijing Electron Positron Collider II (BEPCII). The BESIII detector records symmetric e^+e^- collisions provided by the BEPCII storage ring [54], which operates in the center-of-mass energy range of 2.0–4.7 GeV [55]. The cylindrical core of the BESIII detector covers 93% of the full solid angle and consists of a helium-based multilayer drift chamber (MDC), a plastic scintillator time-of-flight system (TOF), and a CsI(Tl) electromagnetic calorimeter (EMC), which are all enclosed in a superconducting solenoidal magnet providing a 1.0 T magnetic field. The solenoid is supported by an octagonal flux-return yoke with resistive plate counter muon identification modules interleaved with steel (MUC).

The detector description in the BESIII offline software is provided in GDML format. Each of the four subdetectors, MDC, TOF, EMC, and MUC, has a corresponding GDML file to describe it. A general GDML file provides a description of common materials and other passive detector components, such as the beam pipe and superconducting solenoid. All applications in BESIII offline software, including simulation, reconstruction, event display, calibration, and analysis, use GDML files as the single source of detector information.

BESIII Visualization (BesVis), an event display tool based on ROOT, was developed to visualize the detector and analyze physics events, as shown in Fig. 6. It has played an important role in BESIII's offline software development and physics analysis since 2005.

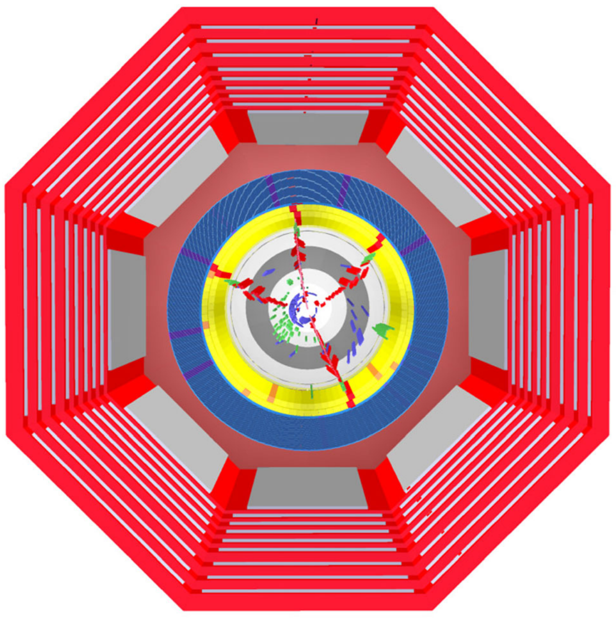


Fig. 6 (Color online) Visualization of the BESIII detector and a real data event in BesVis

4.2 Conversion of the BESIII geometry to Unity

Based on the method introduced by Sect. 3, it is possible to perform a format conversion of the BESIII detectors description from GDML to Unity. The detector data conversion process comprised the following steps:

First, the GDML description of each subdetector was imported into FreeCAD with the GDML–FreeCAD interface. Because the original interface only supports the basic CSG shapes, BESIII has used some complicated shapes, such as Twistedtubs, IrregBox, and Boolean shapes. We updated the interface to allow correct conversion of all the shapes being used in the GDML detector description of BESIII.

Second, after importing into FreeCAD, the BESIII detector data were exported in the STEP format. Then, the STEP files were converted to FBX files using the format conversion function provided by Pixyz Studio. As an example, the visualization of the BESIII TOF subdetector in FreeCAD and Pixyz is shown in Fig. 7.

Third, the FBX files were directly imported into Unity. For the BESIII detector, four sub-detector FBX files (MDC, TOF, EMC, and MUC) and one general FBX file describe the beam pipe and superconducting solenoid. The subdetectors were verified in Geant4 and ROOT to guarantee no space overlaps between each other, so the subdetectors could be directly combined to form the whole BESIII detector. Finally, 3D modeling of the BESIII detector was successfully performed in Unity after a set of automated steps.

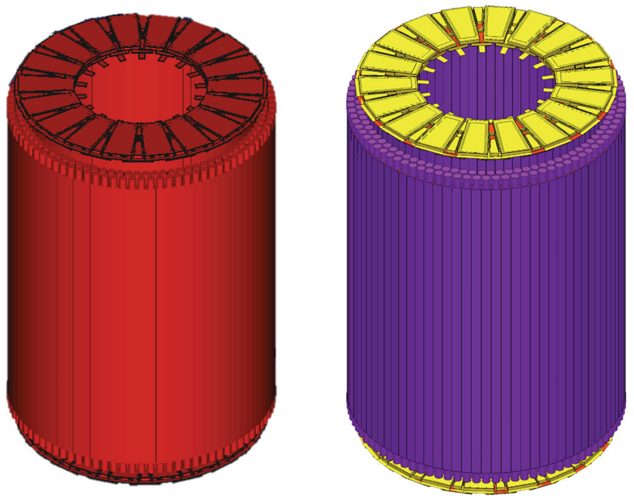


Fig. 7 (Color online) Visualization of the BESIII TOF sub-detector in FreeCAD (left) and Pixyz (right)

4.3 Display of the BESIII detector in Unity

Although at this stage the BESIII detector has been automatically built in Unity, it is still not well displayed in the scene. Because GDML does not save visualization attributes, all the detector units are still set with the default visualization attributes in Unity. The only object that users can see in the scene is a box, which is the top world volume in the definition of the GDML detector description.

A set of scripts must be developed to set the color, transparency, reflectivity, and texture of the material for the detector units. The top world volume and virtual mother volumes need to be set invisible to allow the inner detector units to appear. Another advantage of automatic conversion is that the name of each detector unit is maintained during the process of transformation so that the scripts can set the visualization attributes according to the name of each detector unit.

Figure 8 shows the display of the four BESIII subdetectors in Unity. The corresponding display parameters of the materials were updated to obtain a better display effect. The display attributes in Unity were set to be consistent with those in BesVis, but with more vivid and richer visualization effects than those supported by ROOT. The full BESIII detector in Unity is shown in Fig. 9.

5 Further development of applications

Once the detector geometry and its visualization attributes are constructed in Unity, various applications can be further developed. The characteristics of Unity, including fantastic visual effects, multi-platform support, and VR [42] or AR [43] device integration, make it convenient

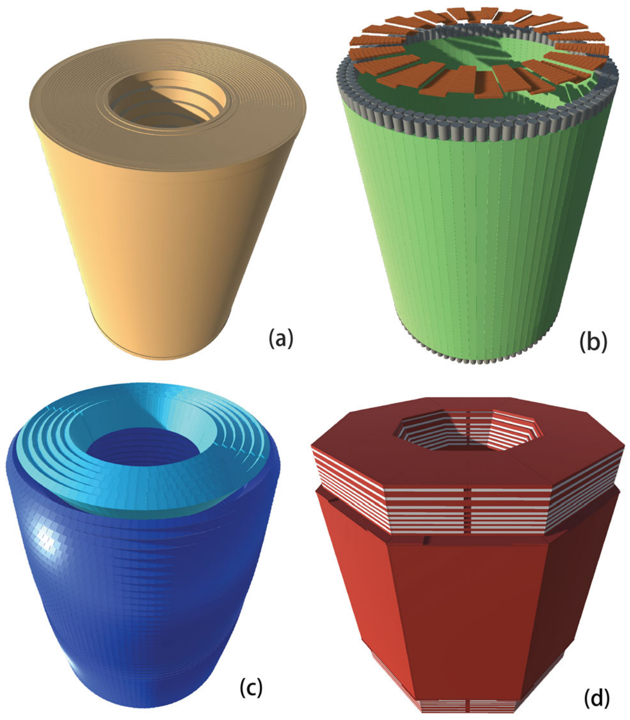


Fig. 8 (Color online) Visualization of the BESIII sub-detectors in Unity, **a** MDC, **b** TOF, **c** EMC, and **d** MUC

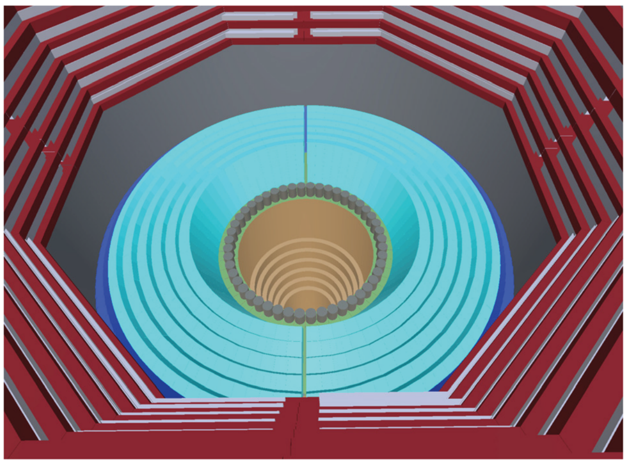


Fig. 9 (Color online) Display of the full BESIII detector in Unity. From inside to outside: MDC, TOF, EMC and MUC

for developing event display tools, detector status monitoring software, and VR/AR applications for scientific research and education.

5.1 Event display tool

Event displays are convenient tools for offline software tuning and physical analysis in HEP experiments. In Sect. 2.3, the application of Unity for event displays in ATLAS and JUNO was introduced. However, for both programs, the detectors were manually constructed independently in Unity, which required a large amount of developers' work.

Using the automatic geometry transformation method, the BESIII detector was constructed in Unity. Compared with the geometry construction in the BESIII offline software and its ROOT-based event display, which is composed of more than 5000 lines of code, this method avoids the repetitive coding work required in a different software. To develop an event display tool based on Unity, the event data information is obtained and the fired detector units are associated with their identifiers, which can be decoded from the name of each detector unit.

Once the association relationship is constructed, a set of scripts can be developed in Unity to control the different visual effects of the fired and unfired detector units. The basic functions of the event display tool can then be realized. Figure 10 shows a prototype of the BESII event display tool under development in Unity, and its rendering effects.

5.2 Detector and data monitoring

With the detector constructed in Unity, not only the offline event display but also the online monitoring software can be developed to help monitor the operation status of the experiment and the quality of the real-time data collected by the detector.

Detector units with abnormal operational statuses, such as dead or hot channels, can be distinctly displayed, which will help shift operators diagnose potential detector problems.

Owing to the multi-platform support of Unity, a monitoring project can be easily deployed on different platforms and devices. In addition to the traditional Windows and Linux operating systems, the monitoring project can also be built into apps on mobile platforms, such as Android and iOS, so that users can conveniently monitor the status of an experiment remotely from their mobile phones or pads.

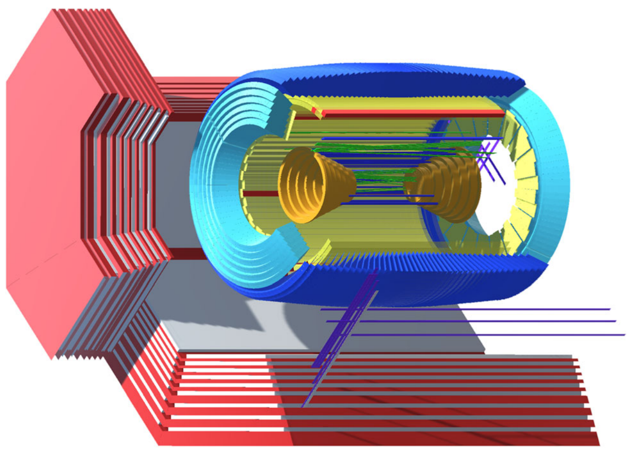


Fig. 10 (Color online) Prototype of a new BESIII event display tool in Unity and its rendering effects

5.3 Virtual reality applications

In the roadmap for HEP software and computing R&D for the 2020s [2], an application based on VR is an important development direction [1]. VR simulates the physical presence of a user in a virtual environment. By presenting the emitting particles and their interactions with the detector, it can provide a new method with immersive experience for physicists to tune the offline simulation and reconstruction software. They can also perform physics analysis for rare events, as if the users were personally in the scene of the running detector. Some creative productions have started in the HEP community, such as ATLASrift [56] and BelleII VR [44].

With the method for automatic detector construction in Unity and the extensible support of VR devices from Unity, VR applications for nuclear and HEP experiments can also be conveniently developed. Because most of the nuclear or HEP experiments are not accessible during data collection because of safety or security requirements, our method has another advantage, allowing the general public to explore the detector with an immersive experience and watch nuclear or HEP collisions in a realistic environment. It will be of great benefit to let the general public have a basic idea and understanding of the type of scientific research that the nuclear physics and HEP communities are doing.

5.4 Interdisciplinary applications

In addition to its applications in nuclear physics or HEP experiments, this method could also have wide applications in applied nuclear science, techniques, and industry.

For example, in nuclear power plant (NPP) monitoring, this method can be used to monitor the emitting neutrons or neutrinos and then reconstruct the distribution of the inner active fission area to monitor the operation status of the NPP.

In the field of nuclear medical imaging, the human body is more or less like a nuclear particle source or detector, but with dynamic geometry. Because our detector modeling method is automatic, it can rapidly construct and visualize the human body, allowing observation of its status after interaction with nuclear particles in real time or semi-real time. Other potential interdisciplinary applications include muon tomography, X-ray inspection, and fusion diagnosis [57–60].

6 Summary

Detector description and visualization are essential techniques in software development for next-generation HEP experiments. Unity is a popular and versatile multimedia creation platform that provides impressive visual effects, multiple platform support, and extensibility to VR and AR applications. For large-scale scientific projects, building a complicated detector with up to millions of components in Unity while keeping them consistent with the geometry in the offline data processing software is very difficult and requires significant work for software developers.

A method for automatic detector data transformation from GDML to Unity is presented, which can construct 3D modeling of detectors in Unity for further applications. The method has been realized in the BESIII detector with hundreds of thousands of components and can be used in future experiments such as CEPC. In addition to HEP experiments, the method also has great potential for multiple interdisciplinary applications, education, and outreach.

Author Contributions All authors contributed to the study conception and design. Material preparation, data collection and analysis were performed by Kai-Xuan Huang, Zhi-Jun Li, Zhen Qian, Jiang Zhu and Hao-Yuan Li. The first draft of the manuscript was written by Kai-Xuan Huang, Yu-Mei Zhang, Sheng-Sen Sun and Zheng-Yun You, and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

References

1. M. Bellis, R.M. Bianchi, S. Binet et al., Hep software foundation community white paper working group—visualization. (2018). https://doi.org/10.48550/ARXIV.1811.10309
2. J. Albrecht, A.A. Alves Jr., G. Amadio et al., A roadmap for HEP software and computing R&D for the 2020s. Comput. Softw. Big Sci. **3**, 7 (2019). https://doi.org/10.1007/s41781-018-0018-8
3. T.A. Collaboration, The ATLAS experiment at the CERN large hadron collider. JINST **3**, S08003 (2008). https://doi.org/10.1088/1748-0221/3/08/s08003
4. T.C. Collaboration, The CMS experiment at the CERN LHC. JINST **3**, S08004 (2008). https://doi.org/10.1088/1748-0221/3/08/s08004
5. L. Evans, The large hadron collider. New J. Phys. **9**, 335 (2007). https://doi.org/10.1088/1367-2630/9/9/335
6. R. Chytracek, J. Mccormick, W. Pokorski et al., Geometry description markup language for physics simulation and analysis applications. IEEE Trans. Nucl. Sci. **53**, 2892–2896 (2006). https://doi.org/10.1109/TNS.2006.881062
7. M. Frank, F. Gaede, C. Grefe et al., DD4hep: a detector description toolkit for high energy physics experiments. J. Phys: Conf. Ser. **513**, 022010 (2014). https://doi.org/10.1088/1742-6596/513/2/022010
8. Y.T. Liang, B. Zhu, Z.Y. You et al., A uniform geometry description for simulation, reconstruction and visualization in the BESIII experiment. Nucl. Instrum. Meth. A **603**, 325–327 (2009). https://doi.org/10.1016/j.nima.2009.02.036
9. Z.Y. You, Y.T. Liang, Y.J. Mao, A method for detector description exchange among ROOT GEANT4 and GEANT3. Chin. Phys. C **32**, 572–575 (2008). https://doi.org/10.1088/1674-1137/32/7/012
10. R. Brun, A. Gheata, M. Gheata, The ROOT geometry package. Nucl. Instrum. Meth. A **502**, 676–680 (2003). https://doi.org/10.1016/S0168-9002(03)00541-2
11. M. Tadel, Overview of EVE - the event visualization environment of ROOT. J. Phys: Conf. Ser. **219**, 042055 (2010). https://doi.org/10.1088/1742-6596/219/4/042055
12. T.A. Collaboration, ALICE: physics performance report, volume I. J. Phys. G: Nucl. Part. Phys. **30**, 1517–1763 (2004). https://doi.org/10.1088/0954-3899/30/11/001
13. M. Ablikim, Z. An, J. Bai et al., Design and construction of the BESIII detector. Nucl. Instrum. Meth. A **614**, 345–399 (2010). https://doi.org/10.1016/j.nima.2009.12.050
14. Z. You, K. Li, Y. Zhang et al., A ROOT based event display software for JUNO. J. Instrum. **13**, T02002 (2018). https://doi.org/10.1088/1748-0221/13/02/t02002
15. S. Zhang, J.S. Li, Y.J. Su et al., A method for sharing dynamic geometry information in studies on liquid-based detectors. Nucl. Sci. Tech. **32**, 21 (2021). https://doi.org/10.1007/s41365-021-00852-8
16. W. Goldstone, *Unity game development essentials*, (Packt Publishing Ltd, 2009)
17. T. Abe, I. Adachi, K. Adamczyk et al., BELLE II technical design report. (2010). https://doi.org/10.48550/ARXIV.1011.0352
18. J. Zhu, Z. You, Y. Zhang et al., A method of detector and event visualization with unity in JUNO. J. Instrum. **14**, T01007 (2019). https://doi.org/10.1088/1748-0221/14/01/t01007
19. S. Agostinelli, J. Allison, K. Amako et al., Geant4- a simulation toolkit. Nucl. Instrum. Meth. A **506**, 250–303 (2003). https://doi.org/10.1016/S0168-9002(03)01368-8
20. T. Bray, J. Paoli, C. Sperberg-McQueen, Extensible markup language (xml) 1.0., http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm
21. K. Adcox, S. Adler, M. Aizama et al., PHENIX detector overview. Nucl. Instrum. Meth. A **499**, 469–479 (2003). https://doi.org/10.1016/S0168-9002(02)01950-2
22. T.L. Collaboration, The LHCb detector at the LHC. J. Instrum. **3**, S08005 (2008). https://doi.org/10.1088/1748-0221/3/08/s08005
23. A. Trisovic, LHCb event display. https://inspirehep.net/literature/1920683
24. L. Pescatore, Status of outreach activities at LHCb. PoS 2018, 277 (2019). https://doi.org/10.22323/1.340.0277
25. G. Ranucci et al., Status and prospects of the JUNO experiment. J. Phys: Conf. Ser. **888**, 012022 (2017). https://doi.org/10.1088/1742-6596/888/1/012022
26. Z. Qian, V. Belavin, V. Bokov et al., Vertex and energy reconstruction in JUNO with machine learning methods. Nucl. Instrum. Meth. A **1010**, 165527 (2021). https://doi.org/10.1016/j.nima.2021.165527
27. Z.Y. Li, Y.M. Zhang, G.F. Cao et al., Event vertex and time reconstruction in large-volume liquid scintillator detectors. Nucl. Sci. Tech. **32**, 49 (2021). https://doi.org/10.1007/s41365-021-00885-z
28. Z.Y. Li, Z. Qian, J.H. He et al., Improving the machine learning based vertex reconstruction for large liquid scintillator detectors with multiple types of PMTs. Nucl. Sci. Tech. **33**, 93 (2022). https://doi.org/10.1007/s41365-022-01078-y
29. T.C.S. Group, CEPC conceptual design report: Volume 2 - Physics and Detector. (2018). https://doi.org/10.48550/ARXIV.1811.10545arXiv:1811.10545

30. T. Behnke, J.E. Bran, B. Foster et al., The international linear collider technical design report - volume 1: executive summary. (2013). https://doi.org/10.48550/ARXIV.1306.6327
31. T.F. Collaboration, FCC physics opportunities: future circular collider conceptual design report volume 1. Eur. Phys. J. C 79, 474 (2019) https://doi.org/10.1140/epjc/s10052-019-6904-3
32. Q. Luo, D. Xu, Progress on preliminary conceptual study of hiepa, a super tau-charm factory in china. In: *Proceedings 9th international particle accelerator conference (IPAC'18), Vancouver, BC, Canada, April 29-May 4, 2018*, pp. 422–424. https://doi.org/10.18429/JACoW-IPAC2018-MOPML013
33. R.W. Kennard, L.A. Stone, Computer aided design of experiments. Technometrics **11**, 137–148 (1969). https://doi.org/10.1080/00401706.1969.10490666
34. L. Bartoszek, E. Barnes, J. Miller et al., Mu2e technical design report. (2015). https://doi.org/10.48550/ARXIV.1501.05241
35. G. Taylor, Visualizing the ATLAS inner detector with Atlantis. Nucl. Instrum. Meth. A **549**, 183–187 (2005). https://doi.org/10.1016/j.nima.2005.04.049
36. N. Konstantinidis, Z. Maxa, P. Klok et al., The Atlantis event visualisation program for the ATLAS experiment. https://cds.cern.ch/record/865603/files/p361.pdf
37. T. Kittelmann, V. Tsulaia, J. Boudreau et al., The virtual point 1 event display for the ATLAS experiment. J. Phys: Conf. Ser. **219**, 032012 (2010). https://doi.org/10.1088/1742-6596/219/3/032012
38. D. Kovalskyi, M. Tadel, A. Mrak-Tadel et al., Fireworks: a physics event display for CMS. J. Phys: Conf. Ser. **219**, 032014 (2010). https://doi.org/10.1088/1742-6596/219/3/032014
39. T. Sakuma, T. McCauley, Detector and event visualization with SketchUp at the CMS experiment. J. Phys: Conf. Ser. **513**, 022032 (2014). https://doi.org/10.1088/1742-6596/513/2/022032
40. J. Pequenao, US-DOE, ATLAS multimedia educational lab for interactive analysis. (2008). https://doi.org/10.11578/dc.20210416.15
41. Camelia webpage. https://pdgusers.lbl.gov/~pequenao/camelia
42. J. Zheng, K. Chan, I. Gibson, Virtual reality. IEEE Potent. **17**, 20–23 (1998). https://doi.org/10.1109/45.666641
43. J. Carmigniani, B. Furht, *Augmented Reality: An Overview* (Springer, New York, 2011), pp.3–46
44. Z. Duer, L. Piilonen, G. Glasson, Belle2VR: a virtual-reality visualization of subatomic particle physics in the BELLE II experiment. IEEE Comput. Graph. Appl. **38**, 33–43 (2018). https://doi.org/10.1109/MCG.2018.032421652
45. CERN TEV visualization framework webpage., https://gitlab.cern.ch/CERNMediaLab/
46. FreeCAD webpage., https://www.freecadweb.org
47. D.H. Laidlaw, W.B. Trumbore, J.F. Hughes, Constructive solid geometry for polyhedral objects. SIGGRAPH Comput. Graph. **20**, 161–170 (1986). https://doi.org/10.1145/15886.15904
48. KeithSloan, et al., FreeCAD GDML workbench., https://github.com/KeithSloan/GDML (2022)
49. S. Kemmerer, STEP: The grand experience. (1999). https://doi.org/10.6028/NIST.SP.939
50. I. Stroud, *Boundary Representation Modelling Techniques* (Springer, Cham, 2006)
51. R. Carey, The virtual reality modeling language explained. IEEE Multimed. **5**, 84–93 (1998). https://doi.org/10.1109/93.713310
52. FBX webpage. https://www.autodesk.com/products/fbx/overview
53. Pixyz studio software., https://www.pixyz-software.com/studio
54. C. Yu, Y. Zhang, Q. Qin et al., BEPCII performance and beam dynamics studies on luminosity. In: *Proceedings of International Particle Accelerator Conference (IPAC'16), Busan, Korea, May 8-13, 2016*, pp. 1014–1018. https://doi.org/10.18429/JACoW-IPAC2016-TUYA01
55. M. Ablikim, M. Achasov, P. Adlarson et al., Future physics programme of BESIII. Chin. Phys. C **44**, 040001 (2020). https://doi.org/10.1088/1674-1137/44/4/040001
56. I. Vukotic, E. Moyse, R.M. Bianchi, Atlasrift - a virtual reality application. (2015). arXiv:1511.00047
57. D. Carbone, D. Gibert, J. Marteau et al., An experiment of muon radiography at Mt Etna (Italy). Geophys. J. Int. **196**, 633–643 (2013). https://doi.org/10.1093/gji/ggt403
58. K. Morishima, M. Kuno, N. Akira et al., Discovery of a big void in Khufu's Pyramid by observation of cosmic-ray muons. Nature **552**, 386–390 (2017). https://doi.org/10.1038/nature24647
59. Y.P. Chen, R. Han, Z.W. Li et al., Imaging internal density structure of the Laoheishan volcanic cone with cosmic ray muon radiography. Nucl. Sci. Tech. **33**, 88 (2022). https://doi.org/10.1007/s41365-022-01072-4
60. C.F. Yang, Y.B. Huang, J.L. Xu et al., Reconstruction of a muon bundle in the JUNO central detector. Nucl. Sci. Tech. **33**, 59 (2022). https://doi.org/10.1007/s41365-022-01049-3