Journal of Physics: Conference Series **396** (2012) 022052 doi:10.1088/1742-6596/396/2/022052

Multiple-view, Multiple-selection Visualization of Simulation Geometry in CMS

L A T Bauerdick[1], G Eulisse[1], C Jones[1], D Kovalskyi[3], T McCauley[1], A Mrak Tadel[2,4], I Osborne[1], M Tadel[2,4] and A Yagil[2]

[1] Fermilab, Batavia, IL 60510-5011, USA
[2] UC San Diego, La Jolla, CA 92093, USA
[3] UC Santa Barbara, Santa Barbara, CA 93106, USA

E-mail: amraktadel@ucsd.edu, mtadel@ucsd.edu

Abstract. Fireworks, the event-display program of CMS, was extended with an advanced geometry visualization package. ROOT's TGeo geometry is used as internal representation, shared among several geometry views. Each view is represented by a GUI list-tree widget, implemented as a flat vector to allow for fast searching, selection, and filtering by material type, node name, and shape type. Display of logical and physical volumes is supported. Color, transparency, and visibility flags can be modified for each node or for a selection of nodes. Further operations, like opening of a new view or changing of the root node, can be performed via a context menu. Node selection and graphical properties determined by the list-tree view can be visualized in any 3D graphics view of Fireworks. As each 3D view can display any number of geometry views, a user is free to combine different geometry-view selections within the same 3D view. Node-selection by proximity to a given point is possible. A visual clipping box can be set for each geometry view to limit geometry drawing into a specified region. Visualization of geometric overlaps, as detected by TGeo, is also supported. The geometry visualization package is used for detailed inspection and display of simulation geometry with or without the event data. It also serves as a tool for geometry debugging and inspection, facilitating development of geometries for CMS detector upgrades and for SLHC.

1. Introduction

Fireworks is a physics analysis-oriented event-display program of the CMS experiment [1]. Initial development of Fireworks in period from 2008 to early 2010 followed the transition of CMS to a new implementation of experiment software framework and was mostly driven by the requirements of early data taking: debugging of detector, understanding of trigger system, and facilitating the start of physics analyses on first data [2]. Through 2010 and part of 2011 Fireworks was integrated with the CMSSW framework and it became possible for users to run CMS software jobs in an interactive manner, modifying the parameters of algorithms, navigating through events, all while being able to view event-data collections after each event has been processed [3].

Only basic support for displaying of simulation geometry was possible until recently by using Event Visualization Environment (EVE) [4] of the ROOT framework [5]. Conversion from CMS XML detector description [6] into TGeo [7] format was written as a tool for Fireworks that allowed extraction of detector shapes and their positions used as detector outlines in Fireworks views and it was possible to visualize this geometry in a simple, stand-alone EVE application.

In this paper we present recent extensions of Fireworks that allow full visualization of TGeo simulation geometries, optionally superimposed together with the event-data. In section 2 we discuss the issues

[4] To whom any correspondence should be addressed.

related to visualization of simulation geometries. Section 3 describes the multi-view, multi-selection implementation of the geometry visualization package while section 4 focuses on filtering mechanisms allowing sub-selection of volumes to be presented to the user. Section 5 presents how TGeo volume-overlap checking functionality is integrated into a special view-type of the geometry browser.

2. Problems in visualization of HEP simulation geometries

Simulation geometries of current large HEP experiments consist of a couple million physical volumes that are replicated out of a couple thousand unique logical volumes in a hierarchical tree structure of daughter volumes enclosed within their mothers. This inherent self-similarity offers great opportunities for minimizing the memory footprint of geometries at the cost of making the iteration over nodes more complex. Also, as node representation is shared between all possible instances, individuality of each node is lost – both for assignment of visual properties and for unambiguous selection of a node from a 3D view. To uniquely identify a physical volume, full path of volume choices from the top-node down to the leaf node is required. Along the path, transformation matrices need to be multiplied in sequence to obtain the final transformation matrix for the node. The same issues arise when considering alignment of simulation geometry.

Very large number of elements and significant sharing of data among individual nodes also has a direct influence on how the geometry can be represented and manipulated in both graphical user interfaces and in 3D views. It is obvious that displaying a couple million volumes to a user only serves to overwhelm him and has a fair chance of exhausting the system resources. There are two main ways to reduce this complexity, described in the next two sub-sections.

2.1. Explicit selection of displayed volumes

This is the standard display mode implemented by most GUI-based geometry browsers. The geometry is represented in a list-tree widget where a user can expand and collapse individual entries to navigate through the hierarchy. Switching between *physical* and *logical volume* view is usually supported. In physical volume mode all daughters of a given mother volume are always shown. In logical volume view only one instance of each volume type is shown for each mother and visibility and color operations performed on the representative volume get applied to all individual instances. There is some ambiguity as to what happens with instances in other branches that contain the same logical volume, depending on where the visual parameters are stored: if they are part of internal geometry representation then the changes get applied to all instances and if they are stored in the GUI structures, they are only applied on the selected branch.

2.2. Control by limiting traversal depth

In this approach, user defines the starting node and specifies to what level the drawing traversal should proceed. Several options are needed to fully specify how intermediate nodes should be displayed and how non-leaf nodes at the bottom of traversal should be treated. Visibility and color control is limited to representative volumes – all physical nodes derived from them behave the same. This makes implementation of GUI rather straightforward but results in a confusing experience for most users as changes of some parameters cause “inexplicable” changes elsewhere.

Such a drawing system is implemented in ROOT’s TGeo. Users mostly use this via the command-line interface but a basic implementation of geometry navigation is also available in the standard ROOT browser. EVE offers a somewhat more complete GUI over TGeo drawing functionality and adds the possibility to maintain and control several draw paths, each with its own top-node, depth of traversal and visualization options.

Neither of the two approaches is complete and either requires too much user action or, which is even more frustrating, does not provide enough control over the visualization parameters to allow users to satisfactorily complete their tasks. Next section describes how we attempted to combine and extend both approaches just described.

3. Implementation of multiple-view, multiple-selection geometry browser in Fireworks

3.1. Multiple-view, multiple-selection concept

Users are typically interested in a specific part of the geometry and often require a combination of very different elements, e.g. display of silicon detectors and cooling pipes surrounding them. Therefore, several different volume selections are necessary to satisfy all but the simplest tasks. However, already the GUI representation of a geometry view that is able to perform complex queries requires a significant effort, both for development as well as for efficient usage. Considering how this complexity could be reduced we arrived at the following solution:

i. Allow several concurrent geometry table views, each having a unique:
   - top-node, traversal depth, visibility & transparency controls;
   - selection of displayed volumes, based on their properties or position.

ii. Allow each 3D graphics view to display contents of any geometry table.

With this functionality, a user can open two geometry table views, search for volumes made of silicon in the first view (or by volume or shape name, if user knows the geometry very well) and for the volumes corresponding to cooling pipes in the second one. Then, all that remains to be done is to instruct both geometry views to draw themselves into a single 3D view. A similar composition of elements from two geometry tables into a single 3D view is shown in Figure 1.

3.2. Internal representation of a geometry view

As already mentioned, TGeo representation of geometry is used as the source of all information about volumes, shapes, materials, and material mixtures composing the detector. However, TGeo represents the geometry in the most economic format possible, with maximum reuse of volumes and their sub-hierarchies, and thus does not allow full, independent control over visibility and visual properties of individual physical volume. Further, to allow for multiple-view, multiple-selection concept, we need the ability to set those properties for each view independently so it is obvious that a structure overlaying the TGeo representation is required.

After considering various more complex options the simplicity of using a compact vector of simple structures (called NodeInfo) finally turned out to be the most versatile. The following information is contained in the NodeInfo structure:

- pointer to TGeoNode structure,
- vector index of parent so that each node can be fully identified (note that TGeoNode structure does not contain a pointer to the parent as every TGeoNode can be attached into the geometry at several points and can thus have several parents),
- visibility flags for this node and its children,
- color and transparency of this node (this could be easily extended to hold other visual effects like textures or shaders).

Size of NodeInfo structure is 18B (24B with alignment on 64-bit architectures). For CMS simulation geometry with 2 million nodes, this makes each full view use 48MB of memory. For comparison, TGeo geometry, fully voxelized and ready for tracking, uses 58MB.

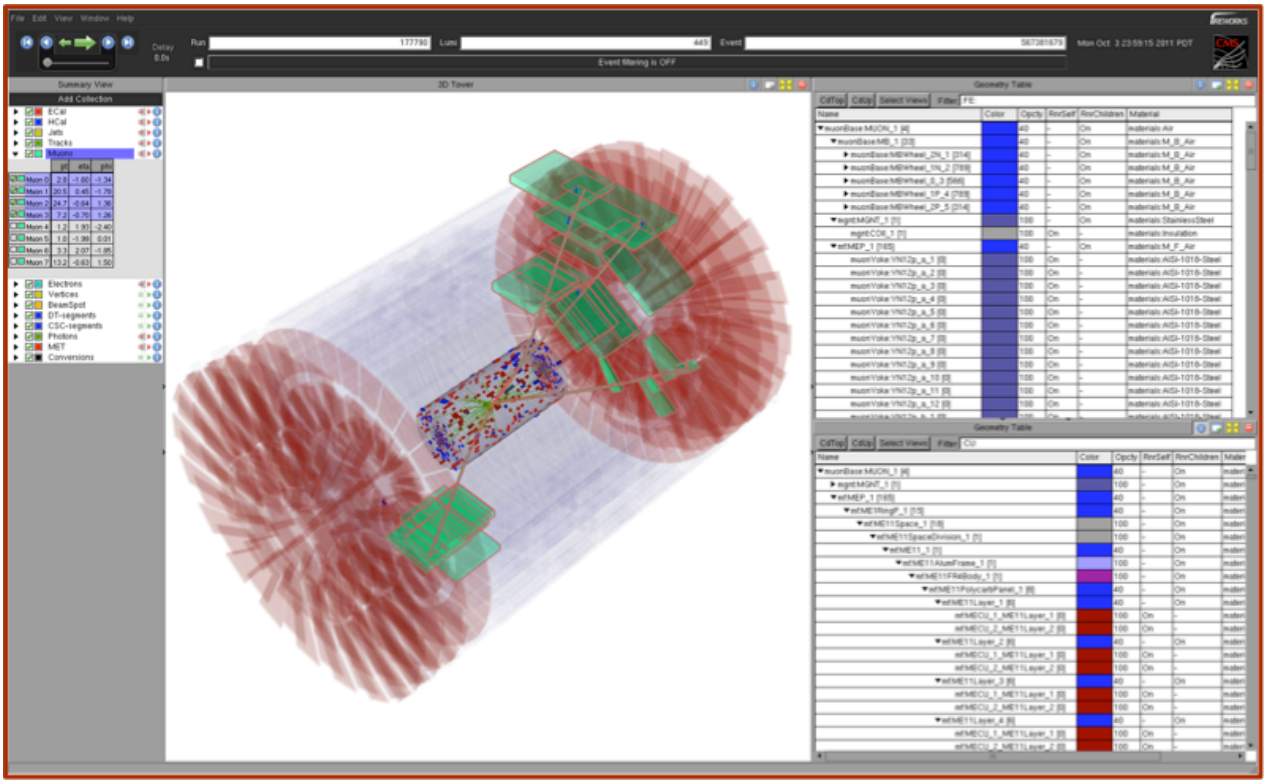

Figure 1. Volumes with more than 50% Cu (red) or Fe (blue) in the muon detector, overlaid over a four-muon event. Two geometry views are shown on the right, each selecting detector elements made of one type of material.

It is worth mentioning that we had quite a mental barrier to cross before choosing to "sacrifice" memory to gain flexibility and speed. The following list of capabilities that suddenly became easy to implement finally tipped the scale:

i. Full flexibility in setting visual parameters of each physical node;
ii. Efficient implementation of GUI table – no additional memory is required;
iii. Efficient traversal for rendering with OpenGL;
iv. Vector index is used to uniquely identify a physical node:
    - selecting a node in 3D view can reveal its full path to the top
    - several useful options are available from context menus
    - correlated selection highlighting between all views is possible
    - mouse hoover in 3D view shows a tooltip and outlines the volume

Further, one should consider that browsing of geometry isn't an every day task and that it is mostly used by specialists for performing of complex tasks – in these cases flexibility of the tool is far more important than system requirements for the visualization machine.

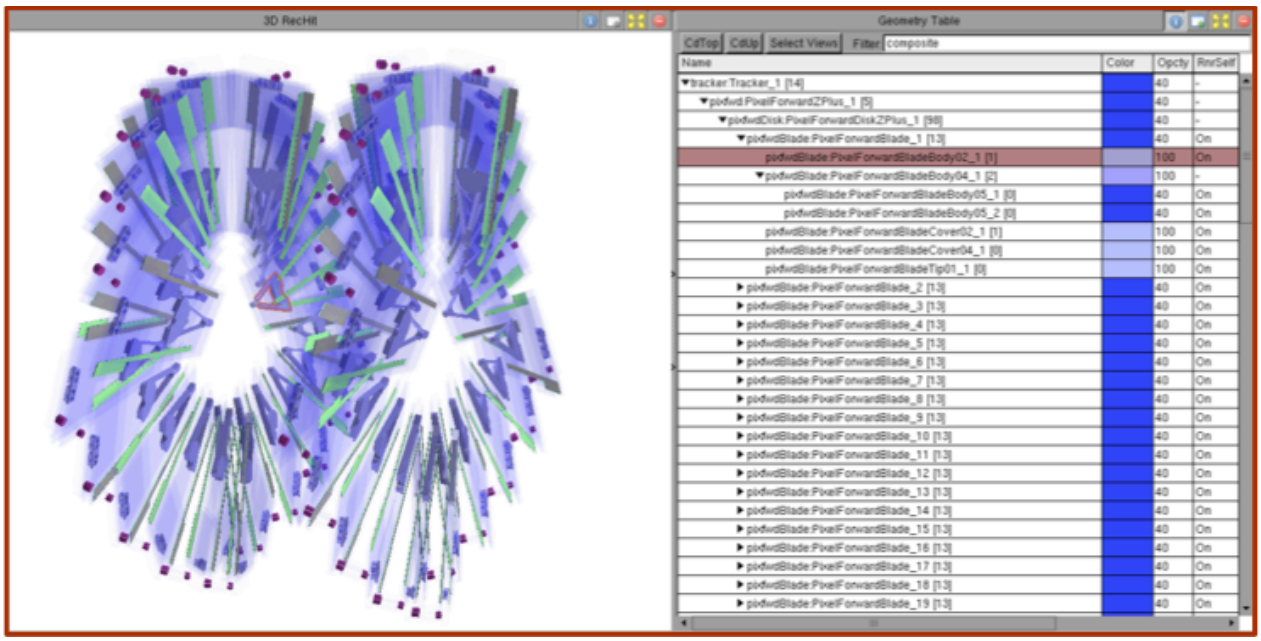

Figure 2. All volumes using boolean / composite shapes in pixel-forward detector.

3.3. Integration with Fireworks event display & availability outside CMS software

Code for geometry browser is part of standard CMSSW releases and is also distributed with the light-weight, stand-alone version of Fireworks based on FWLite. Simulation geometry can be saved to a ROOT file in TGeo format, taking about 890kB.

The following points present how geometry viewing is presented to Fireworks users:

- Geometry table view is added as any Fireworks view.
- Geometry data is taken from memory or from a ROOT file specified on the command-line.
- Standard 3D views are used to display the geometry.

Geometry state is saved and restored as part of configuration.

As geometry view is just another Fireworks view, all previous Fireworks functionality for displaying of event-data and reconstruction geometry remains intact. Usage of standard 3D views for displaying of the simulation geometry makes it trivial to either combine event-data and geometry within a single view or separate it into several independent views. Figure 1 shows an example of such combined view.

Implementation of geometry table and 3D rendering is practically independent of CMS software and it should be relatively straightforward to port the implementation to become available in ROOT Event Visualization Environment (EVE). The main problem is that the table widget written for Fireworks is used for drawing of and interaction with the geometry table view so this component would have to be extracted as well.

4. Filtering of displayed volumes

Filtering of volumes is always applied from current top-volume downwards. In table view, when a volume passes the filter, all parent volumes up to the top-node are also shown so that the hierarchy is retained. Display of siblings on the matching path is optional. Only the volumes that actually pass the filter are shown in 3D views after the filtering is complete. By default, all matching volumes are shown irrespective

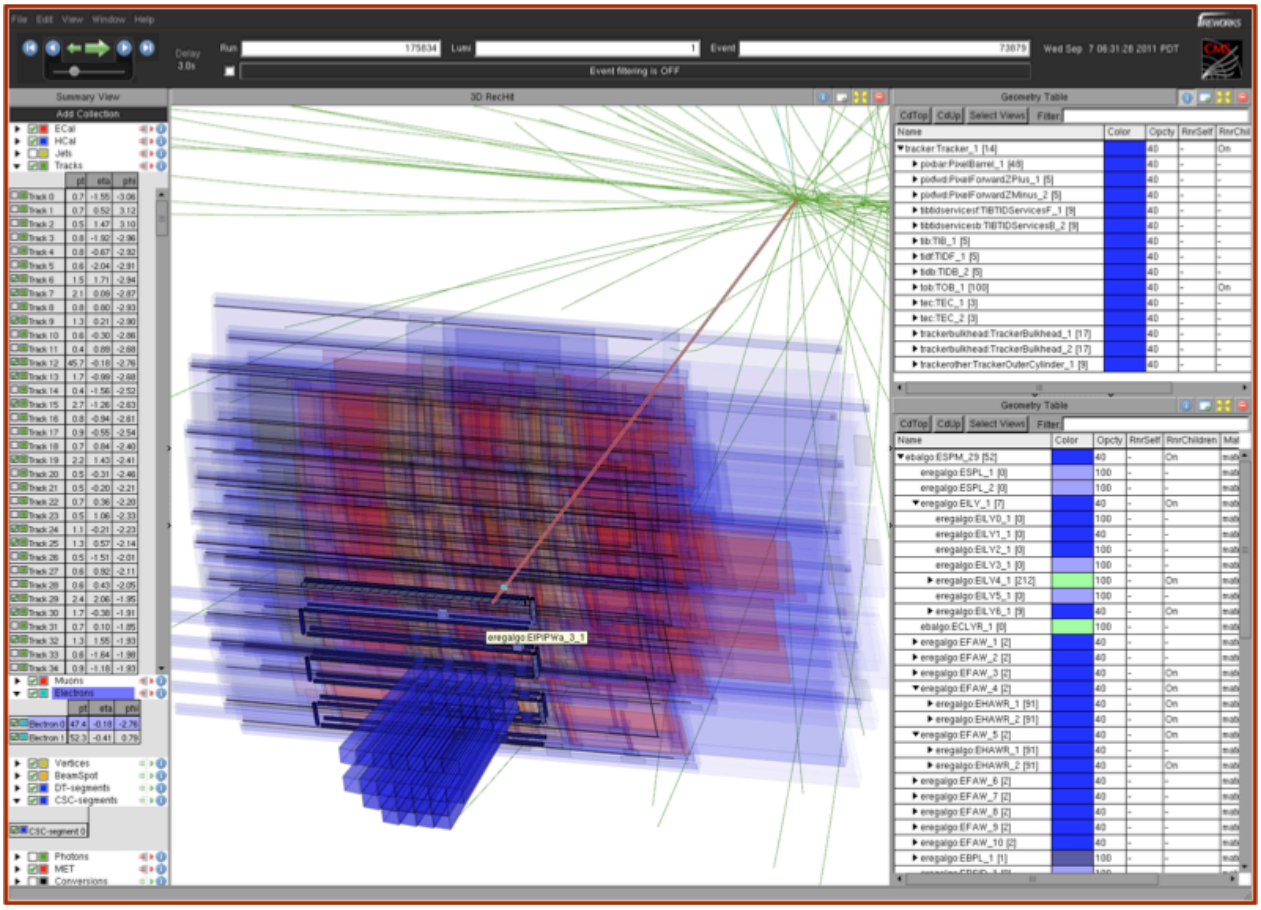

Figure 3. Outer tracker and ECal volumes within 25cm from a point on an electron track.

of their depth in the hierarchy. Optionally, users can force drawing to a specific depth from the top-volume.

4.1. Filtering by volume properties

The following volume properties can be used for filtering:

- volume name, shape name, shape class name or material name;
- name of the chemical element making up more than given mass-fraction of the volume.

All these filters are applied as string searches in the corresponding data-members of TGeo classes representing volumes, shapes and materials. As all these properties are actually shared by all instances of a given logical volume, a hash-map lookup can be used to speed up the filtering process. See figure 2 for an example.

4.2. Filtering by physical volume position / proximity

Users are often interested in what geometric elements are present in a vicinity of some point, e.g. sensitive material, a gap in barrel to end-cap transition of calorimeter, or support structures that can cause catastrophic energy loss. To provide for these use-cases, physical volumes can be selected based on their

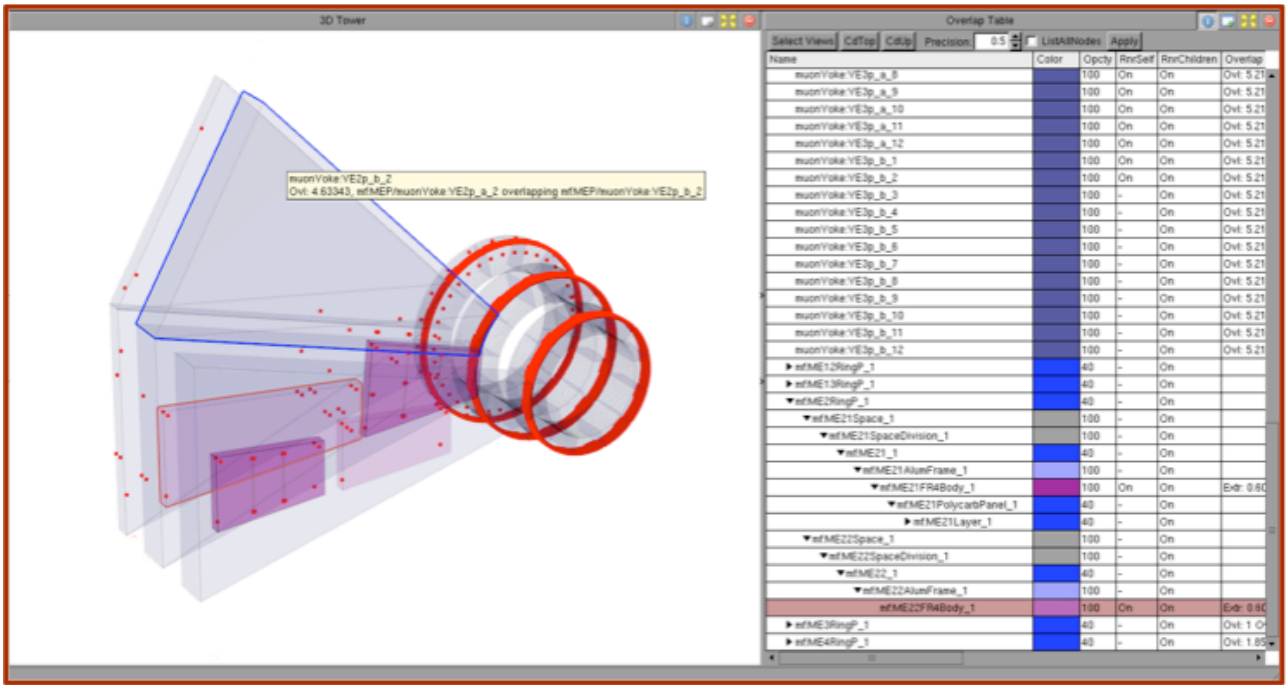

Figure 4. Overlaps with more than 5mm in muon detector end-cap.

distance from a reference point. Figure 3 shows two geometry views selecting tracker and ECal volumes lying within 25cm of an electron-track point of exit from the tracker system.

Distance from point to volume's bounding-box is currently used as the criterion. If needed, filtering can be extended to use more complex reference objects, e.g., tracks.

Containment of daughter volumes within a mother volume can be used to optimize the geometry traversal, as we know that if a mother volume does not match a proximity criterion, neither can any of its daughters.

Boolean operations among filters are currently not available. Implementation of this feature should be rather straightforward when the need for it arises. Each volume-selection can be represented by a bit-vector for which operations of union, intersection and difference are easily provided. Alternatively, a simple filtering expression parser could be implemented to allow construction of composite selections.

5. Visualization of overlaps and extrusions

TGeo overlap checking functionality is used for detection of volume overlaps and extrusions. Overlap table view is just a specialization of geometry table view with the following additional features:

- users can select the top-node and precision used for checking;
- in 3D view the overlapping volumes are shown together with a marker showing the overlap position;
- context menu provides access to detailed information about the overlap.

An example showing overlaps in a part of the CMS muon detector is shown in figure 4 as an example.

There was no overlap checking functionality integrated with CMSSW before and many of the discovered overlaps have already been fixed in the CMS XML geometry description.

6. Conclusion

We have presented a modern multi-view, multi-selection approach to visualization of large simulation geometries and described features and technical details of the implementation that fulfills current needs of the CMS experiment. Further work is expected to focus mainly on user-support and implementation of extensions and tools requested by the users, mostly for work on detector upgrades and for SLHC.

Acknowledgments

This work is sponsored by the US National Science Foundation under Grant No. PHY-0612805 (CMS Maintenance & Operations).

References

[1] CMS Collaboration 2008 *JINST* **3** S08004
[2] Kovalskyi D et al 2010 *J. Phys.: Conf. Ser.* **219** 032014
[3] Bauerdick L *et al* 2011 *J. Phys.: Conf. Ser.* **331** 072039
[4] Tadel M 2008 *PoS* **ACAT08** 103
[5] Antcheva I *et al* 2009 *Comput. Phys. Commun.* **180** 2499
[6] http://twiki.cern.ch/twiki/bin/view/CMSPublic/SWGuideDetectorDescription
[7] Brun R, Gheata A and Gheata M 2003 *Nucl. Instrum. Meth. A* **502** 676