TOWARDS 3D HUMAN-MACHINE-INTERFACES: GENERIC 3D VIEWER EXTENSION FOR THE CONTROL SYSTEMS DISPLAYS AT CERN

P. Golonka, CERN, Geneva, Switzerland

Abstract

The 3D-Viewer component of the PVSS[1] [1] JCOP[2] Framework [2][3] allows PVSS operator consoles at CERN to display three-dimensional, interactive, animated graphics. Being completely generic, the viewer is applicable to a variety of situations, from being used in a synoptic view of the status of a system through to the 3-D customization of an individual HMI element such as a graph or a histogram. The 3D-Viewer is a production system component which showcases the integration of platform-independent technologies, in our case PVSS, Qt[4] and Open Inventor[5]. The viewer moreover gives the operator full control of the content and appearance of the "scene" being displayed, and permits dynamic modification at run-time, through interaction with objects in the scene. The complete functionality of the viewer is visible through the clean, high-level interface of PVSS' graphical objects and scripts, and makes for a seamless integration with the rest of the application.

INTRODUCTION, MOTIVATION

Control and Monitoring systems nowadays often run on PCs, the graphics cards of which are capable of efficient rendering of complex 3-dimensional graphics. Even though the 3D elements have been available in various SCADA products for some time now [6][7][8][9], their use in the HMI seems to be limited.

Studies such as [10] suggest that 3D visualization is not always a cost-effective and optimal solution for applications in industry – the classical 2D user interfaces seem more applicable for typical use in industrial applications. The area where 3D visualization demonstrate its potential superiority is where complex systems need to be presented. The complexity, in turn, is the outstanding feature of many control and monitoring systems for the detectors and accelerators at CERN. Therefore the idea of 3D visualizations used as synoptic views for Detector Control Systems of the large LHC experiments appeared in a natural way already some time ago.

PVSS [1], a commercial SCADA[3] product widely deployed and standardized at CERN, does not provide an out of the box component for 3D visualization. However, it allows the use of third-party user interface extensions on Windows platform through the Microsoft ActiveX component technology. The first 3D visualization for the CMS[4] Detector Control System [11] was implemented as a Java3D applet, embedded in PVSS through a dedicated ActiveX bridge container.

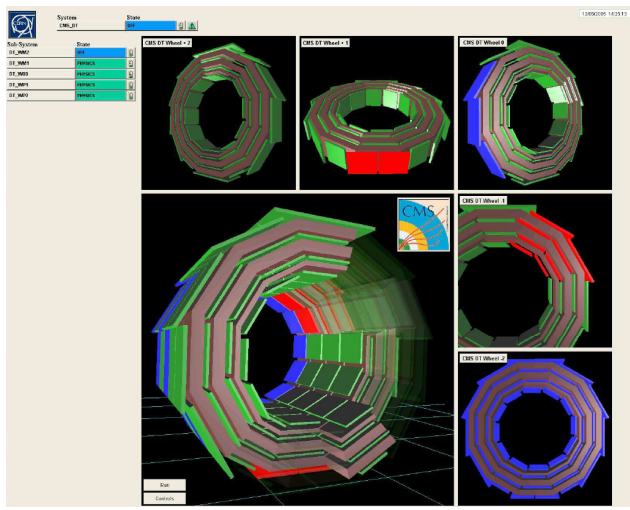

Figure 1: CMS Detector Control System featuring the original Java prototype of the 3D Viewer.

The project found interest in DCS groups of other experiments, yet soon an important limitation was encountered: the control rooms of the LHC experiments run PVSS on the Linux platform, where embedding of ActiveX components is not possible. The portability became a showstopper for the adoption of the project.

The situation changed significantly after the release 3.5 of PVSS: the User Interface part was re-implemented using the Qt[4] libraries, known for their portability and extendibility. PVSS 3.5 introduced a native mechanism for extension of its user interface, called EWO[5]. It allowed the delivery of custom UI elements developed with the Qt libraries in C++. Moreover, the same source code could be used to build the EWO's both for Windows and Linux.

The availability of the EWO technology made it possible to come back to the idea of 3D visualization in PVSS UI. Based on the concepts of the original Java3D prototype, the 3D Viewer EWO for PVSS was developed as a part of the JCOP Framework project.

In addition to the fascinating 3D visualization in PVSS-based control systems, the project became a showcase for the integration of standardized technologies and code reuse.

This article introduces the JCOP Framework 3D Viewer component, discusses the main concept and presents the

1PVSS: Process Control and Visualization SCADA [1]

2JCOP: Joint COntrol Project

3SCADA: Supervisory Control And Data Acquisition

4CMS: Compact Muon Solenoid detector at CERN

5EWO: Enhanced Widget Object

interesting use cases. In the appendix additional background material about the technologies employed, namely the Open Inventor[5] is provided.

JCOP FRAMEWORK 3D VIEWER

The JCOP Framework 3D-Viewer component [12][13] allows the extension of the PVSS user interface panels with a new programmable UI element (i.e. a widget), capable of displaying a 3-dimensional, interactive view of a *scene* composed of various geometrical objects (also called *shapes*).

The content of the scene is fully dynamic: the shapes can be added, removed, or their properties (such as geometry/size, colour, transparency) modified at runtime. These actions can be linked to events received by PVSS, or to user commands. The user naturally navigates through the scene using the mouse and the keyboard. The camera can move in all directions, and change the orientation in response to user navigation or to programming commands. The widget is also capable of passing the mouse-click events to the PVSS user interface.

Both the modification of the scene contents and properties and the event passing are interfaced with the native mechanisms of PVSS; this makes the integration of this widget easy for any PVSS application developer.

Apart from its portability, what makes our project distinct from other 3D components is that it is generic. Instead of binding it to a particular data file format, or CAD application to construct the content of the 3D scene, we opted for creation of a simple-to-use, high level API[6], allowing for dynamic creation and manipulation of the scene content. It is then the role of the PVSS application developer to get the geometry data from some source, and use it to create the scene. The geometry information can be obtained from an external source, such as a database or a XML file, using the functionality already available in PVSS. It can also be generated adhoc (e.g. bars of a histogram), or based on some model. An example of such a model can be a simplified view of a detector with assumed geometrical properties, where the exact geometry is not of importance, and where only the main functional parts need to be plotted.

Even though the 3D Viewer is a generic tool, we focussed our efforts on an implementation that would fit with the requirements for the main use case: generating animated synoptic views of the detector- or accelerator-subsystems being controlled. With this in mind, we implemented the set of shape types that are typically used in computer geometry description for high-energy physics. Therefore, we were able to reuse the code of the HEPVis package[14].

The Programming Interface of PVSS Side

An important step in the project was to define the programming interface exposed to PVSS for the 3D Viewer and decide on the data model used. The functions are organized around the concepts of shapes, shape types and properties.

A *shape type* defines the class of geometrical objects being available for the construction of the scene, such as boxes, tubes or trapezoids.

The scene displayed in the 3D Viewer contains the instances of shape types: the *shapes*. Each shape is identified by a unique name, which is used to refer to it. New shapes may be added to the scene or removed from it dynamically.

Each shape has a set of *properties*. There is a common set of properties available for all shapes: geometry (i.e. position and rotation), colour and transparency. In addition, each shape type has specific properties that ultimately define its figure. Examples of these properties are the radius for spheres and cylinders, the edge-lengths for the boxes, etc.

The current value of any property of any shape can be queried, and altered at any time. This allows the animation of the shapes. For example, a box that represents a bin of a histogram can be resized, to visualize the change in bin contents. The colour of a shape representing a piece of a detector can be changed to red to mark its alarm or error state. A group of shapes may be set to be semi-transparent to uncover the part of the detector that needs attention instead of hiding it. This allows to visually enhance a particular area of interest (see Figure 2) while maintaining the overall orientation and view of the complete detector.

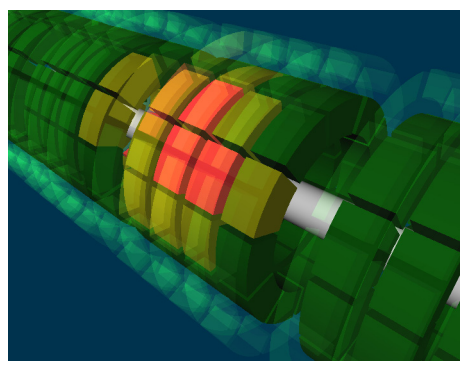

Figure 2: An example of detector visualization with 3D Viewer; colours are used to display the status of elements; transparency applied to the outer layer allow to uncover the internal part, which needs attention.

The shapes can be arranged in groups and identified by a unique name. Property modification operations acting on colour of transparency can then be applied to a group and the change is propagated to all of the shapes belonging to the group. This simplifies the animation code: instead of applying the colour-change code to a large set of shapes, it is sufficient to apply it to the group. In the current implementation, a shape may (but doesn't need to) belong to only one group.

The 3D Viewer widget can operate in two modes: *navigation* or *interaction*. In the navigation mode the mouse is used to navigate through the scene: it is possible to rotate, pan and zoom into the scene in an intuitive way.

[6]API: Application Programming Interface

This corresponds to the modification of camera position and direction.

In the interaction mode, the mouse is used to interact with currently visible shapes. The camera remains steady, and mouse clicks select a shape. Whenever a shape is clicked, an asynchronous notification is posted to PVSS, and a custom script can be executed, The name of the shape that was clicked is passed to the script as an argument. This follows the standard programming techniques for the native PVSS widgets such as tables, push buttons, etc.

The fact that the mouse-click event is passed to PVSS opens the possibility to trigger different types of interactions; for example, in response to a click an action could be executed on the piece of hardware represented by the shape or a new PVSS window can be opened to display the detailed information about the state of the device associated with clicked shape.

For non-interactive displays, where the camera position should be fixed the position and rotation of the camera in the scripting code. The current view can also be smoothly changed using the *seek to shape* functionality: the camera will fly to a position and an orientation where the specified shape is shown in the centre of the view.

APPLICATIONS AND USE CASES

Detector Synoptic View

The initial driving force for the development of the 3D Viewer was to enable the creation of interactive, animated 3-dimensional synoptic views of sub-systems of the large LHC experiments. We expected that modelling the geometry of objects of such a complexity from scratch solely to visualize them would be a prohibitively tedious task.

Surprisingly, the visualization of the on-line Detector Control System (DCS) of large parts of the CMS and the ATLAS detectors became possible using geometry data which had been prepared for the off-line data analysis.

Following the original concept of the CMS Detector Control System Team [11], the detailed data describing the physical and logical volumes composing the parts of the detector is retrieved from an Oracle database, using a database-access PVSS extension, already implemented in the scope of the JCOP Framework project. The mapping between the shapes and the corresponding entities in the DCS system (such as the nodes of the Finite State Machine[7], or devices) is also retrieved, and the shapes are added to the 3D Viewer, with appropriate naming and grouping. The convention used to name the shapes and groups allows for the bi-directional mapping between the control system and the 3D model and allows the activation of the mechanisms linking the two. Once everything is set up, the display reacts to the changes of device properties (such as alerts) or operational states, by changing the look (colour, transparency) of the related shapes. Similarly, a link in the opposite direction is established: clicking on a specific shape in the 3D display opens the operational panel for the associated device, allowing for intuitive selection of the device on which a operation needs to be performed.

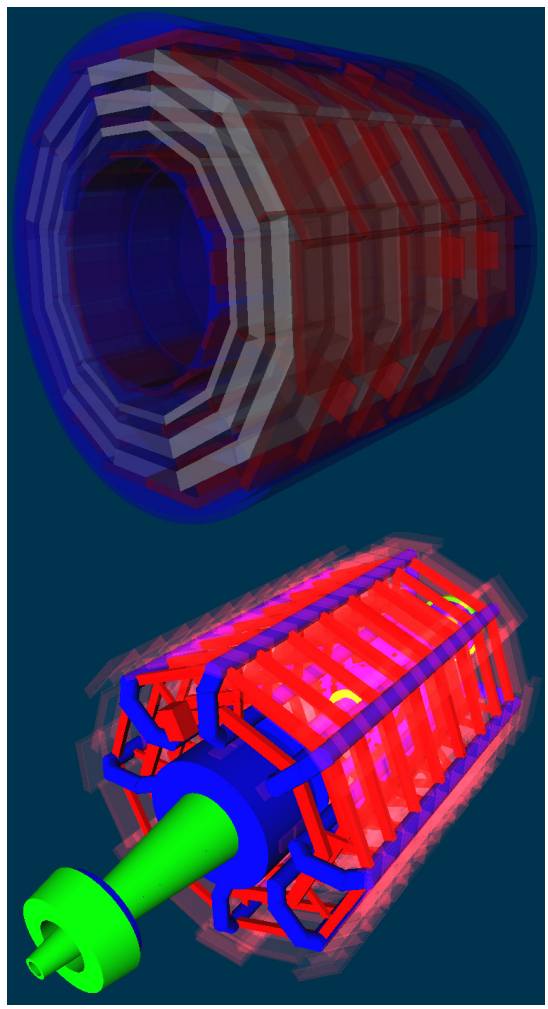

Figure 3: Examples of CMS and ATLAS detector visualizations,based on the data from geometry databases.

Synoptic View of the LHC Accelerator

Like for the case discussed above, an animated 3D synoptic view of the LHC could be constructed for the PVSS-based application like cryogenics, quench protection system, or power converters, replacing the pseudo-perspective view presently used. For the LHC, the detailed geometrical data is available, in the form of engineering drawings, in the CERN Drawing Directory database.

Rack Control Application

One of the applications developed in scope of the JCOP project is the Rack Control Application. Recently, for the CMS experiment, it has been upgraded to make use of the 3D Viewer to display the real-time state of the equipment racks. All the required geometry data, including the real position of the racks inside the buildings, is taken from a construction database; the mapping to the controls items was easily established through a naming convention.

[7]FSM: Finite State Machine tool of the JCOP Framework

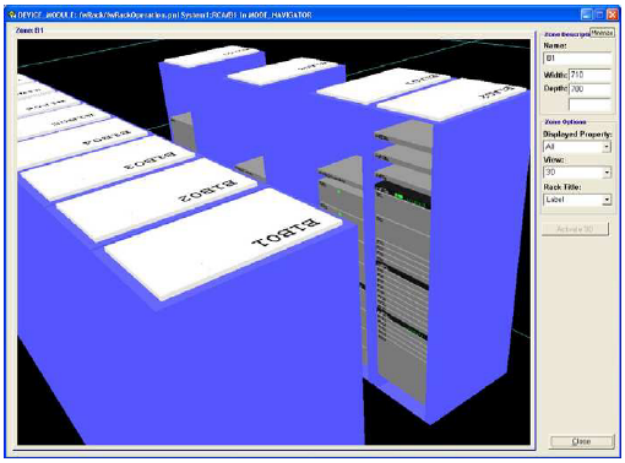

Figure 4: CMS Rack Control Application.

Charts, Graphs, Histograms

The 3D Viewer has also found its applications in the data visualization field. Already one of the early prototypes was used to implement a correlation-showing histogram (2-dimensional plane for parameter space, bin contents in the 3rd dimension). The tight integration of the 3D Viewer with the PVSS native mechanisms made the histogram react to live data changes with no extra effort.

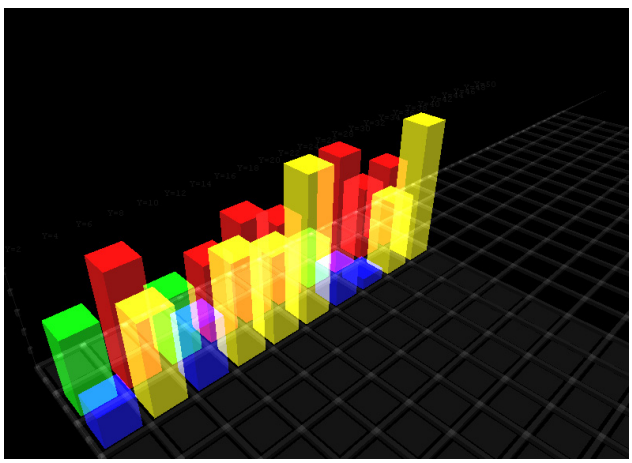

Figure 5: Example of 2D live histogram, with colour and transparency effects

The concept can be extended to a variety of data presentations: bar charts, pie charts and trend plots.

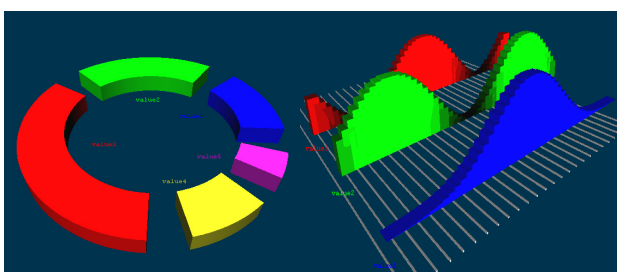

Figure 6: Example of a pie chart and a trend plot.

Others

Many 3D-enabled SCADA applications make use of a virtual-reality based training tools; although this was not needed for the cases described above, virtual-reality based training applications can be implemented exploiting the generic nature of the widget.

Furthermore, the 3D Viewer can be used as a generic widget, playing the role of standard widgets such as push buttons, or LED indicators, adding a non-trivial animation and look thus making the UI experience richer.

ACKNOWLEDGEMENTS

I would like to thank:

- Robert Gomez-Reino (CERN CMS Central DCS Team)
  - for initial concept and the Java3D prototype of the 3D Viewer;
  - for the showcases of use of the widget in CMS DCS System
  - many conceptual discussions leading to the implementation of the 3DViewer.
- Alvar Cuevas i Fajardo, who implemented the first version of the widget under my supervision, during his CERN Technical Student stay.

APPENDIX: TECHNOLOGIES

Open Inventor

Open Inventor [5][15] is a object-oriented 3D graphics programming API. It was designed and implemented in 90s by SGI[8]. It aimed at making the 3D programming more efficient and convenient, by providing a higher-level layer on top of OpenGL, a lower level cross-platform API for 2D and 3D computer graphics.

Whereas OpenGL is targeted for fast rendering of simple lists of polygons in the so called immediate mode, Open Inventor applies the retained mode approach: the client calls do not directly cause the rendering, but instead modify an internal model built of higher-level 3D geometrical objects and maintained in its data space. This allows the optimizations of the actual rendering, which is performed using OpenGL avoiding unnecessary data transfers or occlusion culling (aka hidden surface removal: avoiding the rendering of objects that are entirely behind other opaque objects).

As a result, by using Open Inventor it is possible to fit the 3D visualization program in a few hundred lines of C++ code, offloading the programmer's efforts significantly, when compared with OpenGL. However, this comes at a price: the rendering implemented by manually-tweaked OpenGL code would often be faster that the one of Open Inventor. In any case, the performance of Open Inventor is sufficient and satisfactory for typical application, and often surpasses the OpenGL implementations coded by non-experts.

[8]SGI® : Silicon Graphics International Corp.

Open Inventor delivers a library of high-level objects such as geometrical shapes, cameras, light sources, etc. It is also possible to extend Open Inventor with custom geometrical objects and mechanisms.

Since August 2000, the code and specification of Open Inventor have been released by SGI under an open source license. However, for the 3DViewer we opted to choose another implementation of the Open Inventor API: the Coin3D library [16], which is backward compatible with Open Inventor 2.1.

Since 2009, the proprietary development and support of the Open Inventor standard is in the hands of an independent entity called VSG[17]. Although it is not widely known as OpenGL, Open Inventor establishes the de facto standard for 3D visualization software for science and engineering. Numerous specialized extensions are also available.

HEPVis

The HEPVis[14] library extends the set of geometrical shape types of Open Inventor with the ones typically needed by high-energy physics applications. It is commonly used by numerous event-display and detector-display programs, like Atlantis (the ATLAS experiment event display) or Iguana (the CMS event display). The parameterization of the geometrical shapes origins from the GEANT detector description and simulation package[18][19] widely used by the high energy physics community for more than a decade now. In particular, the shape types and parameterizations implemented by HEPVis match exactly the ones used by GEANT-based simulations and off-line data analysis software. This makes the use of HEPVis an obvious choice for 3D Viewer's main use cases which is display detector views: all the parameters extracted from the database could be used directly as shape parameterization for the 3D Viewer, with no approximation or transformation.

REFERENCES

[1] PVSS: Process Control and Visualization SCADA, http://www.pvss.com

[2] The JCOP Framework Project , http://cern.ch/en-dep-ice-scd/Projects/Framework/welcome.html

[3] "The JCOP Framework", O.Holme, M. Gonzalez-Berges, P. Golonka, S, Schmeling, Proceedings of 10th ICALEPCS Conference on Accelerator & Large Experiment Control Systems, Geneva, 10-14 October 2005

[4] Qt cross-platform application and UI Framework, http://qt.nokia.com/products/

[5] Open Inventor, http://oss.sgi.com/projects/inventor/

[6] PCVue, http://www.arcinfo.com

[7] "ARC Informatique's Innovation in SCADA" , Arc Tech Brief 2006

[8] Genesis64 suite of 64-bit HMI/SCADA software solutions, http://www.iconics.com/products/genesis64.asp

[9] ProcessLife Operations 3D SCADA, http://www.vrcontext.com/processlife/3d-scada.html

[10] "Use Cases and Concepts for 3D Visualization in Manufacturing" B. Wolf, G. Mofor, J. Rode SAP AG, Lecture Notes in Informatics (LNI) P-110 2007

[11] "CMS DCS Design Concepts", R. Arcidiacono,V. Brigljevic,E. Cano,S. Cittolin,S. Erhan,D. Gigi,F. Glege,R. Gomez, Proceedings of 10th ICALEPCS Conference, Geneva, 10-14 Oct 2005

[12] JCOP Framework 3D Viewer widget, http://cern.ch/en-dep-ice-scd/Projects/Framework/Download/Components/3DViewer/welcome.html

[13] "3D viewer offers another dimension for PVSS" P. Golonka and A. Cuevas i Fajardo , CERN Computer Newsletter, April-May 2008 http://cerncourier.com/cws/article/cnl/34864

[14] HEPVis class library extension to the Open Inventor toolkit, http://openscientist.lal.in2p3.fr maintained in the Open Scientist code base

[15] "The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2", Josie Wernecke, ISBN-10: 0201624958

[16] Coin3D: 3D Graphics Developer Kit by Konsberg SIM AS, http://www.coin3d.org

[17] Visualization Sciences Group, http://www.vsg3d.com/

[18] GEANT - Detector Description and Simulation Tool, http://wwwasd.web.cern.ch/wwwasd/geant/

[19] “Geant4 - a simulation toolkit”, Geant 4 Collaboration,