IGUANA INTERACTIVE GRAPHICS PROJECT: RECENT DEVELOPMENTS

G. Alverson, G. Eulisse, S. Muzaffar,
I. Osborne, L. Taylor, L.A. Tuura,
Northeastern University, Boston, USA

Abstract

This paper describes recent developments in the IGUANA (Interactive Graphics for User ANAlysis) project. IGUANA is a generic framework and toolkit, used by CMS and DØ, to build a variety of interactive applications such as detector and event visualisers and interactive GEANT3 and GEANT4 browsers.

IGUANA is a freely available toolkit based on open-source components including Qt, OpenInventor (Coin3D) and OpenGL and LCG services.

New features we describe since the last CHEP conference include:

- multi-document architecture;
- user interface to Python scripting;
- 2D visualisation with auto-generation of slices and projections from 3D data;
- per-object actions such as clipping, slicing, lighting or animation;
- correlated actions (e.g. picking) for multiple views;
- production of high-quality and compact vector postscript output from any OpenGL display, with surface shading and invisible surface culling (together with the gl2ps project).

We compare the IGUANA rendering, memory performance, and porting issues for various platforms including: Linux on x86, Windows, and Mac OSX with its native Quartz-Extreme rendering system.

INTRODUCTION

IGUANA defines a generic object model and a framework for interactive 2D and 3D visualisation. It provides a number of services and tools to generate and manipulate those objects and to manage user interactions. IGUANA is based on the CMS build system, SCRAM [1]. IGUANA development is driven mostly by user requests. IGUANA object model extensively used by the CMS visualisation framework [2].

The enhancements to IGUANA detailed in this paper were, in the main, developed either due to feedback from users concerning the manipulation of complex scenes or for the requirement to display multiple views simultaneously. Other enhancements were proposed in the original IGUANA design [3], but have taken some time to implement in a performant manner.

Overall, the adoption of a multiple document interface structure with centralized scene control has provided a considerably more flexible and useful visualisation framework for use with a variety of visual tasks.

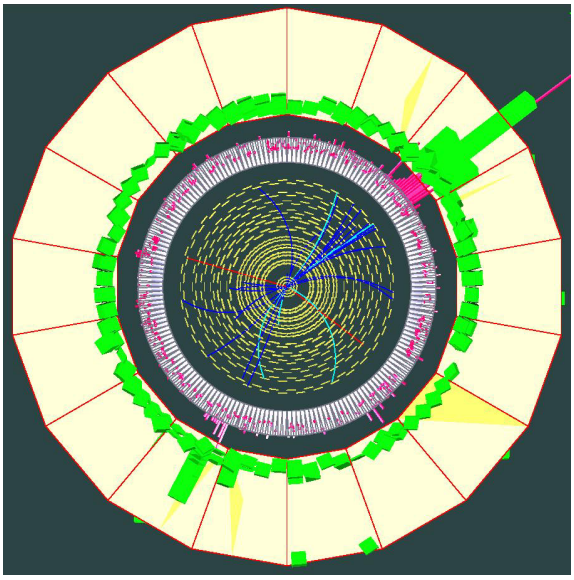

Figure 1: 2D X-Y view of the CMS detector and the event.

MULTI DOCUMENT ARCHITECTURE

With the release of version 5, IGUANA Studio underwent a face lift. It now takes advantage of Qt toolkit features such as dockable widgets and workspaces to provide a multi-document enabled graphical user interface.

It is now possible to create different views (e.g. 2D, 3D, Lego) of the same model or even to have different models observed in different ways through different views. The GUI is now fully customizable with mouse: all the panels and toolbars can be rearranged using drag and drop. The architecture is highly modular and takes full advantage of SEAL [4] plugins, *e.g.* if you only want a 3D view the 2D one is not loaded and load time and runtime resources are reduced.

IGUANA also provides services to manage context switching between one views so that the only controllers exposed are the ones relative to the current view. A simple framework for embedding external custom Qt widgets has also been added.

By default IGUANA provides the following views:

- 2D View: a 2D view where every object has been sliced by a plane: XY, ZX, or ZY;

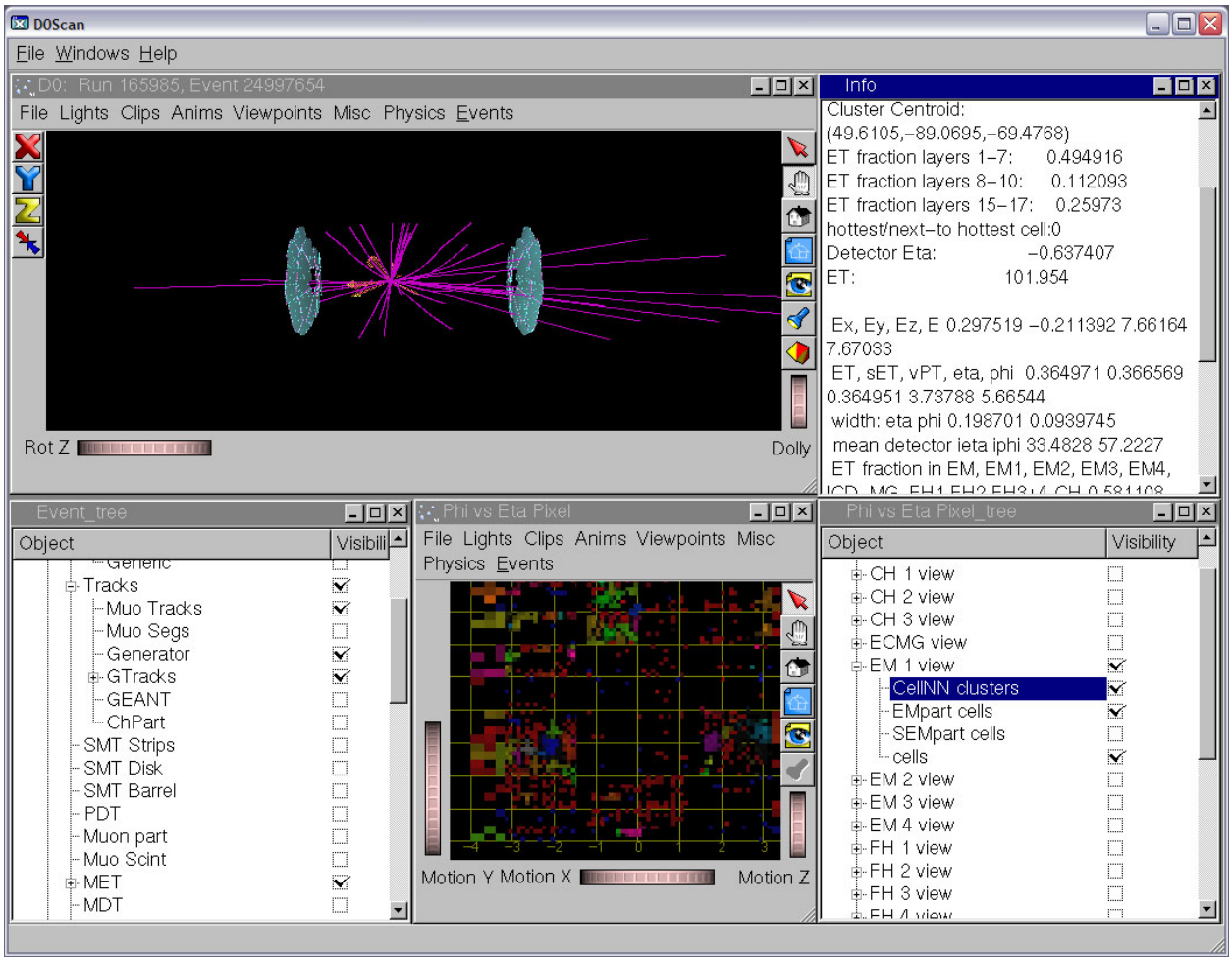

Figure 2: DØdetector and the event. Multi document interface with various views.

- 3D View: the usual 3D view where a detector geometry and event are shown in the 3D space;
- Lego View: allows the display of Lego plots associated to some of the twigs;
- Open Inventor View: displays a generic .iv file as a separate twig tree;
- Python Shell: opens an interactive python [5] shell.

2D MODEL

IGUANA supports 2D views where objects can be grouped and stacked or layered as one would expect from any 2D drawing system. The 2D views use a scene graph of arbitrary-precision vector surfaces with lighting and material properties like the 3D views.

Object representations for the 2D view can be generated in one of three ways. The default is to generate the 2D representation by slicing the object's 3D representation with an arbitrary plane, such as the X-Y or Y-Z plane. Alternatively the 3D object can be retained untouched but displayed squashed (rendered without perspective) into a 2D-layer.

Finally, when neither of the preceding solutions is satisfactory, a truly custom 2D representation can be created. However much of the geometry and event data can typically be handled by one of the two automatic solutions.

The 2D slicing uses a constructive solid geometry algorithm which performs a boolean AND operation on the 3D object and an infinite plane. The algorithm uses as a hybrid BSP-Tree and Octree representation to calculate the 2D surface the 3D object cuts from the plane. While this algorithm is still experimental and being optimised, it is already able to address the vast majority of 3D-to-2D conversions; it is currently unable to handle surfaces with ill-defined orientation. The same algorithm could also be easily applied for any generic 3D-3D boolean operation.

CONTROL CENTRE

The Control Centre provides a central set of widgets to control in a consistent way many of the aspects of displaying visual information inside an IGUANA application. Based on a modular framework where different control categories can be hosted, it exposes through icons only those control categories which were registered for a particular view. A hierarchical organization makes it easy to navigate

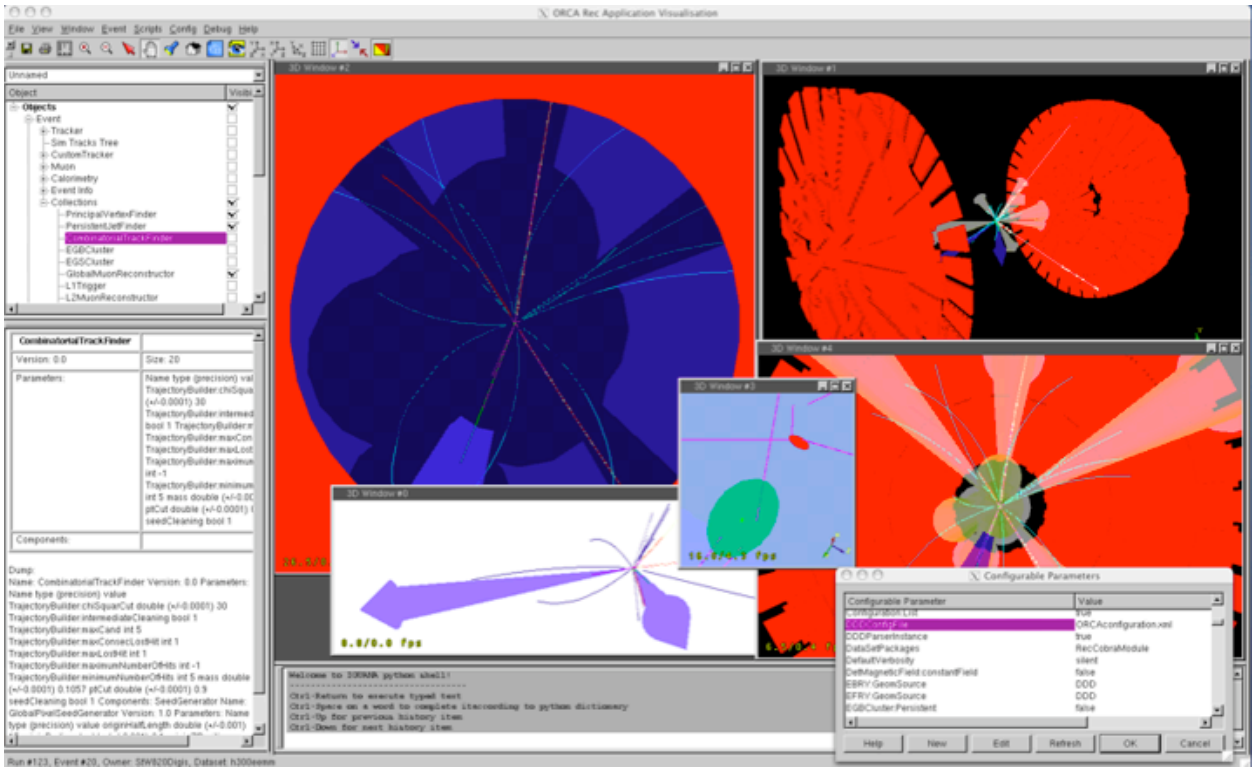

Figure 3: CMS detector and the event. Multi document interface: various views, python shell.

for the user.

Different categories are available in IGUANA for clipping, slicing, lighting, pre-defined view points, and magnetic field manipulation. There are in addition categories to customize the application and viewer. The print category allows customization of the GL2PS [6] vector postscript printing, *e.g.* enabling or disabling background printing, best root finding, and occlusion culling.

VECTOR POSTSCRIPT

Vector postscript printing through the GL2PS C library (an open source project) has been improved by the addition of 2D occlusion culling and a simple best root finding algorithm. The 2D occlusion culling algorithm uses the 2D BSP culling tree to drop the hidden primitive and has been optimized to work faster and use less memory. Depending on the scene graph and view point, these improvements can make the vector postscript output file 99.9% smaller than previous versions. Support for 2D bitmap marker printing (such as those produced by an SoMarkerSet) is now also available.

All these improvements have been contributed back to the GL2PS project and are available in GL2PS Version 1.2.2.

IGUANA AND PYTHON

IGUANA was not originally designed with the intention of exposing a scripting interface. The advantages of such an approach in regard to inter-operability with other software, however, has led us to adapt Python into IGUANA. In particular Python seemed to be the natural choice for mainly three reasons:

- It is widely available, open source and with many contributors
- It is object oriented and has a simple syntax.
- It is very easy to integrate with other languages, C++ in particular.

Moreover, in the case of IGUANA, the choice of Python is almost the unique one, since CMS simulation and reconstruction software already exposes a Python interface.

The Python environment in IGUANA is made up three components:

- a basic service providing access to the Python environment from C++,
- a framework for the creation of Python bindings,
- a Qt GUI to access the Python command-line from the IGUANA Studio Environment.

The scripting service provides access to the Python interpreter by wrapping the Python C API into C++ classes. It is possible in this way to execute a Python script from C++. The framework also allows the developer to access Python objects created within the Python environment within C++, provided that they are instances of classes whose base classes are actually C++ pure abstract classes.

The framework for creating Python bindings is a set of makefiles to be used within the SCRAM build environment which allows the developer to wrap their C++ classes and functions so that they are accessible from C++ as well. It is based on boost_python and a tool called Pyste. The Python View is a standard plug-in for IGUANA Studio. It pops up a window in the environment which allows the user to type and execute their scripts. This console has standard features (like history) but it also features word completion based on the Python dictionary.

CONCLUSION

IGUANA is a generic software currently available for Linux, Mac OS X platforms. Windows versions have been produced for prior versions. There has been significant effort put into improving the rendering performance, in some cases by re-writing the basic shapes, in others by optimization of the scene graphs.

In this year and a half IGUANA architecture has begun to be exploited in order to finally provide a fully integrated physicists environment. Much still has to be done, but huge steps forward have been made in the direction of providing a framework for the physicist's desktop.

We have found that the modular design of IGUANA has proven to be a very effective way to develop applications step by step, introducing new features without requiring major rewrites of the old, stable, components.

ACKNOWLEDGEMENTS

This work is supported by the NSF.

REFERENCES

[1] S. Ashby, I.Osborne, J.P. Wellisch, C. Williams, "Code Organization and Configuration Management", Proceedings of CHEP 2001, Beijing, China, September, 2001.

[2] V. Innocente, G. Eulisse, S. Muzaffar, I. Osborne, L. Taylor, L.A. Tuura, "Composite Framework for CMS Applications", CHEP'04, Interlaken, Switzerland, September 2004.

[3] G. Alverson, G. Eulisse, S. Muzaffar, I. Osborne, L. Taylor, L.A. Tuura, "IGUANA Architecture, Framework and Toolkit for Interactive Graphics", Proceedings of CHEP'03, La Jolla, USA, March 2003.

[4] P. Mato, et al., "SEAL: Common core libraries and services for LHC applications", Proceedings of CHEP'03, La Jolla, USA, March 2003.

[5] http://www.python.org/

[6] http://www.geuz.org/gl2ps/