Experiences with Open Inventor *

Joseph Boudreau *University of Pittsburgh, Pittsburgh, PA, 15260 USA*

Abstract

The Open Inventor(OI) library is a toolkit for constructing, rendering and interacting with a 3D scene. In an object-oriented environment it provides a powerful and low-cost way to construct fully functional event display programs. By subclassing shapes from the OI toolkit, the HEP community can develop HEP-specific shapes of a general nature and further reduce the cost of developing event displays.

Keywords: Open Inventor; Event Display; Visualization; C++

1 Introduction

An event display program is a major cost to a HEP experiment; the time to develop and maintain such a program runs into many man-years as more features are added at the request of users. In the past these programs have been developed using low-level graphics libraries, such as the X Toolkit, GPHIGS, GKS, or GL. These libraries all function at the level of graphics primitives, in two or three dimensions.

Open Inventor (OI) [1] is a C++ toolkit for building 3D scenes. Its class tree contains shapes, materials, transformations, and viewers that encapsulate a large part of the behaviour of a scene, including user interaction. This layer of software has traditionally been written by HEP programmers. OI appears to be unique in the world of computer graphics, and is used widely in industry. The primary advantage of OI is that it allows HEP programmers to assemble complete event displays with unprecedented economy.

This report is organized as follows: in section 2 we describe general problems in HEP visualization; in section 3 we describe the Open Inventor solution; in section 4 we describe a prototype event display based upon OI and HEP-specific extensions; in section 5 we discuss some of the technical issues of concern to potential users, and in the conclusion we recommend a course of action for the HEP community in order to make the most efficient use of this promising technology.

2 HEP Visualization in an Object-Oriented Environment

In future HEP environments the principle objects of experimental particle physics will be coded as C++ objects. In many cases the visualization of these objects is immediate: the object has geometry, and it suffices to render that geometry within a scene. Examples are detector elements and trajectories. Other cases require some creativity to turn abstraction into geometry; for example, energy deposits are sometimes represented by elongated calorimeter crystals. This kind of visualization aims, usually, to be as intuitive as possible.

Because the objects of physics are never far from a geometrical interpretation, we find abundant examples in first generation class libraries of physics objects that can draw themselves. “Draw” methods are useful to application programmers because they allow them to visualize the state of the program at any point during its execution, even (potentially) with spectacular graphics and full control over the camera position and angle (local interactivity).

Simply drawing objects is not enough, however, because users demand far more than to be able to browse events. They need full interactive control over *what* parts of the scene are drawn, and they expect to interact with the image. The purpose of their interaction can be to retrieve information, or even to modify physics objects. Implementing the user interaction with the scene in general is far more challenging than perfecting the quality of images on the screen.

* Open Inventor is a registered trademark of Silicon Graphics, Inc.

3 The Inventor Solution

Because of the need for user interaction, the application programmer should be able to organize *objects* into a scene, to display the scene while giving the user full control over viewpoint, to save and retrieve the scene, to locate a scene object after a pick action, to reliably navigate from a part of composite scene object to the whole, and to set and modify the spatial, geometric, or material properties of scene objects. This is a layer that OI supports and that the low-level graphics libraries do not. In addition, OI provides a base class for shapes of any kind. This makes it easy to define HEP-specific scene objects derived from OI shapes, to map the scene objects in a flexible way into physics objects, to retrieve information from the physics objects, and possibly to modify the physics objects *through* the scene object. Open Inventor uses a good low-level graphics library (Open GL[2]) internally for rendering. The main distinction between a mature graphics library and Open Inventor is functionality, not graphics quality.

3.1 Open Inventor Basics

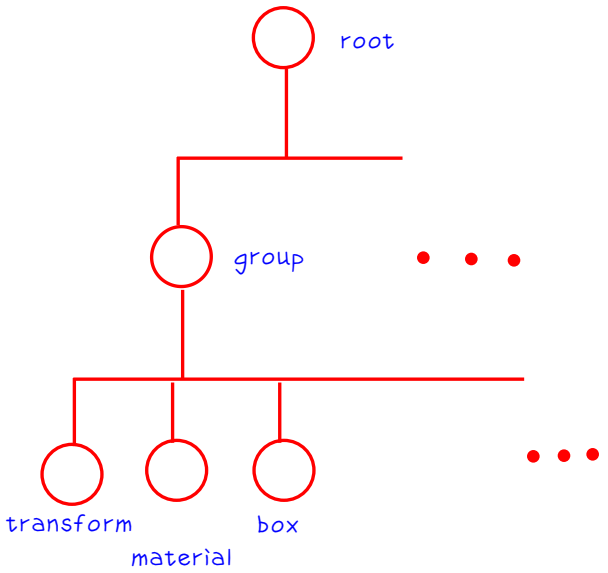

Figure 1: Graph of part of a scene representing one wafer of silicon.

A scene can be represented as a graph such as that in Figure 1, which might describe a wafer of silicon. It is constructed programmatically through statements such as:

```
SoSeparator *root     = new SoSeparator();
SoGroup *wafer        = new SoGroup();
root->addChild(wafer);

SoTransform *transform= new SoTransform();
wafer->addChild(transform);
SoMaterial  *silicon = new SoMaterial();
wafer->addChild(silicon);
SoCube *box = new SoCube();
wafer->addChild(box);
```

Once the scene is constructed, it can be viewed. Several kinds of standard viewers provide various flavours of advanced local interactivity: one simulates flight, another uses a virtual trackball, a third provides planar projections. Different viewers can be simultaneously active. Most can provide stereoscopic viewing through active eyewear.

3.2 Subclassing

Open Inventor classes are designed to be subclassed. This provides a useful way to define physics scene objects inheriting the behaviour of Inventor shape nodes. For example one could define `SoHelicalTrack` class ("So" is for "Scene Object") to represent helical tracks, they could be added to the scene like any other node. Inventor supports a "LevelOfDetail" node that can automatically switch to a less detailed rendering of a shape when its area on the screen becomes smaller than a programmable limit. One might use it to switch automatically between alternate representations a physics object.

3.3 Locating the Physical Object

When the user picks an object, through a mechanism that we won't describe here, Inventor passes the *path* of the object to a callback function provided by the application programmer. Hash tables (or maps) are container class objects that can be used to navigate from the scene object a physical object. This mapping is at the discretion of the application programmer. The STL[3] map class requires a declaration like:

```
map<SoTrack *, PhysicalTrack, less<SoTrack*> > MyPhysicalTrack;
```

The following line would be used to bind the scene object to a physical object:

```
MyPhysicalTrack[GraphicalTrack]=APhysicalTrack;
```

In the callback function, the physical track is indexed by the scene object:

```
MyPhysicalTrack[GraphicalTrack]
```

An event display has been referred to as an index into an event[4]; this example is a pure transcription of this idea into C++.

4 The Prototype

A small prototype event display has been constructed. First a handful of HEP-objects were implemented as subclasses of OI shapes, including: Tracks, Wafers, Hits, Measured-Vertices and Beam-Profiles. This takes some effort but one quickly gains proficiency. Once the classes exist, programming the rest of the event display is relatively simple. The viewers are regular X Toolkit widgets and can be placed within a widget hierarchy. The display is shown in figure 2. The adornments along the bottom and right-hand side are provided entirely by OI for local interaction. The remaining buttons were coded by hand, but could have been programmed with a GUI-builder. The menus along the top control visibility of silicon wafers, hits, and tracks. Along the side the buttons "N" and "Q" allow the user to proceed to the next event, or quit the application. Clicking on a track prints its transverse momentum.
Two novel features were incorporated. First, the user can interactively rescale the spatial dimensions, by attaching a "tab box" to the scene as in figure 3 and figure 4. This can be convenient in certain situations, such as when detectors with low-angle stereo hits are examined. Second, the user can

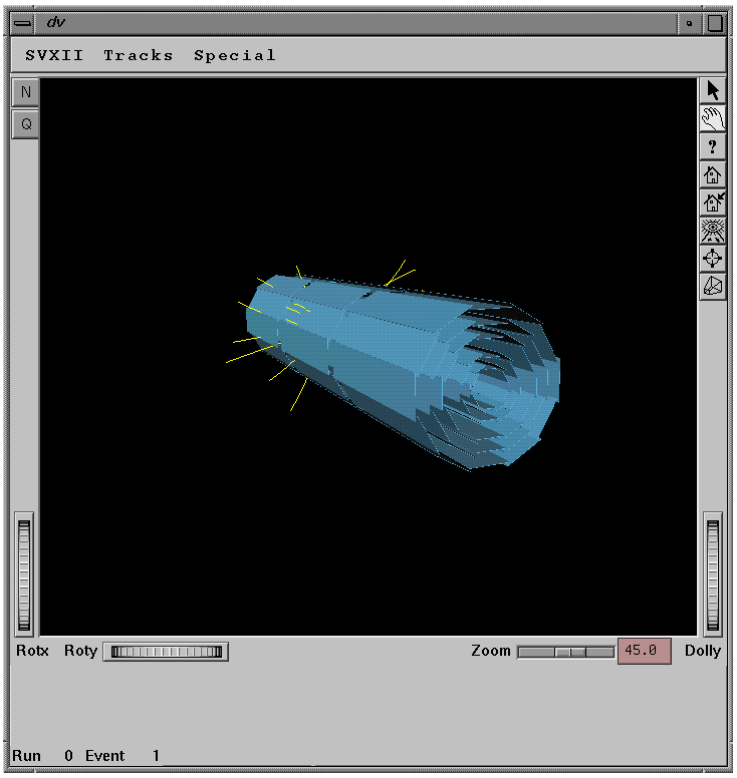

Figure 2: A scene as viewed using one of the standard viewers. Buttons along the bottom and right-hand side control perspective et cetera and are provided by the viewer.

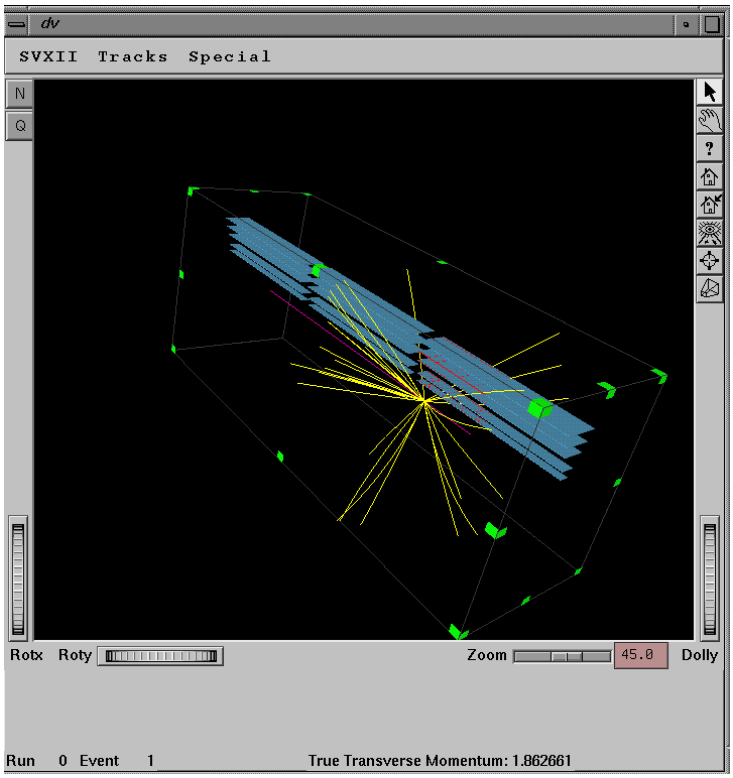

Figure 3: Silicon wafers showing tracks and the beam profile. A “Tab Box” is placed on the wafers to allow the rescaling of spatial dimensions.

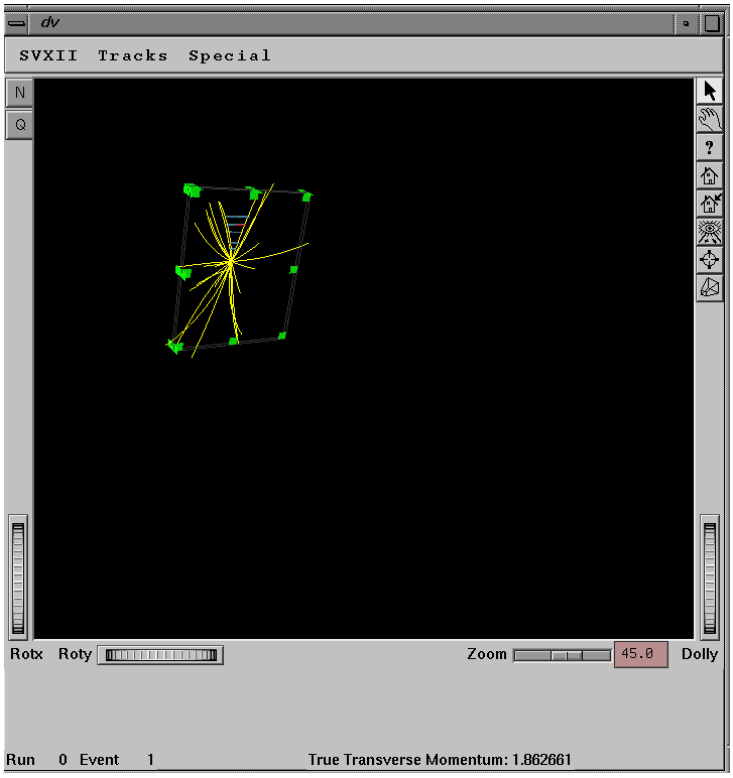

Figure 4: The same scene as figure 3. after rescaling.

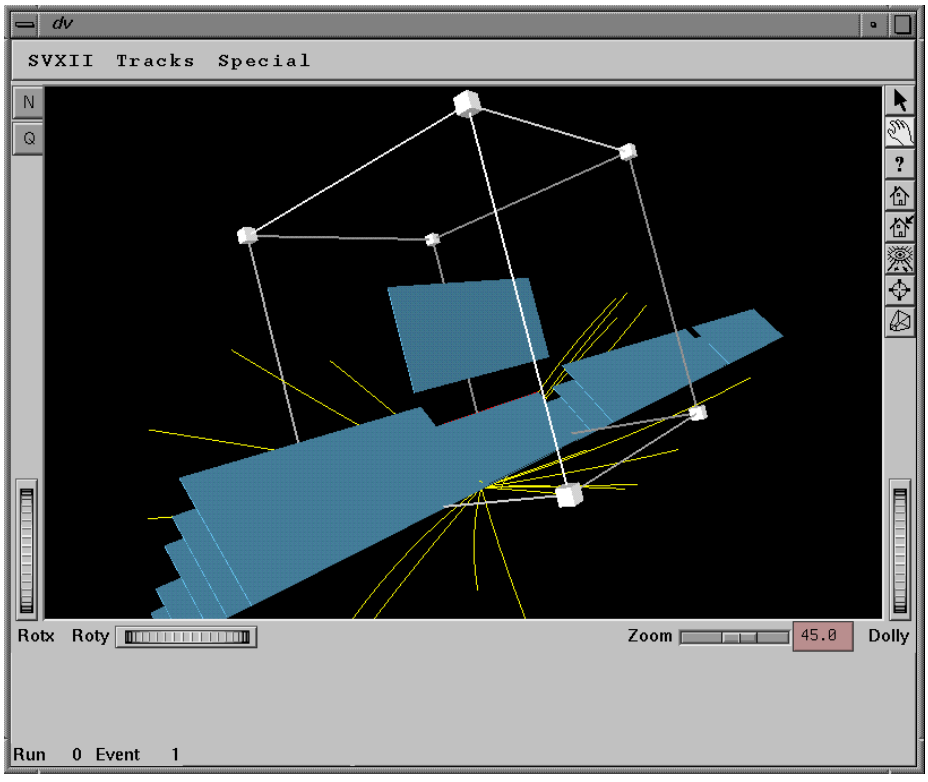

Figure 5: User interaction with Silicon Geometry. The Trackbox can manipulate position and orientation. Various other manipulators are also available.

also interact with the wafer geometry, as in figure 5. Here the user has put a “trackball” manipulator around the wafer, allowing him to change its position. Variations on this technique could be useful some day for detector alignment. The prototype falls well short of a full experimental event display but one can easily imagine scaling it up.

5 Features of Open Inventor

Open Inventor was developed at Silicon Graphics, and is now an open standard distributed by three third-party vendors in addition to SGI. OI can be purchased for virtually all Unix platforms, MacIntosh, Microsoft Windows 3.1, Windows 95, and Windows NT. No version yet supports the use of the gnu compiler. OI developers can call on arbitrary level of graphics acceleration, however they should keep their low-end users in mind by allowing them to turn these features off. OI can run over X-terminals thanks to the MESA library[5], on PC’s, and on high-powered graphics workstations and VR equipment. Open Inventor ASCII File Format is the basis for VRML[6], and a scene can be readily converted to VRML for web browsing.

6 Conclusions

The Inventor Toolkit today provides the most direct and rapid route to well-designed event displays for the next round of collider experiments. HEP projects which ignore this are doomed to re-implement a large fraction of Inventor’s functionality themselves. The development of graphical applications can be made yet more economical if the HEP world could design a reusable set of HEP objects, as extensions to Inventor shape nodes. The design of these classes is relatively unrelated to physics issues and uncomplicated; the main hurdle is to allow optimal performance on all levels of graphics hardware. It is not too optimistic to think that this step will be taken before HEPVis 97.

References

1 J. Werneke and the Open Inventor Architecture Group, *The Inventor Mentor*, Addison-Wesley 1993 J. Werneke and the Open Inventor Architecture Group, *The Inventor Toolmaker*, Addison-Wesley 1994

2 J. Neider *et. al*, *Open GL Programming Guide*, Addison-Wesley, 1993

3 A. Stepanov and M.Lee. *The Standard Template Library*. ISO Programming Language C++ Project. Doc. No. X3J16/94-0095, WG21/N0482, May 1994.

4 H. Stone, this workshop.

5 I. Gaponenko, this workshop.
B. Paul, “The Mesa 3-D graphics library”, http://www.ssec.wisc.edu/ brianp/Mesa.html

6 The VRML Architecture Group, *The Virtual Reality Modeling Language Specification, Version 2.0,* August 4, 1994