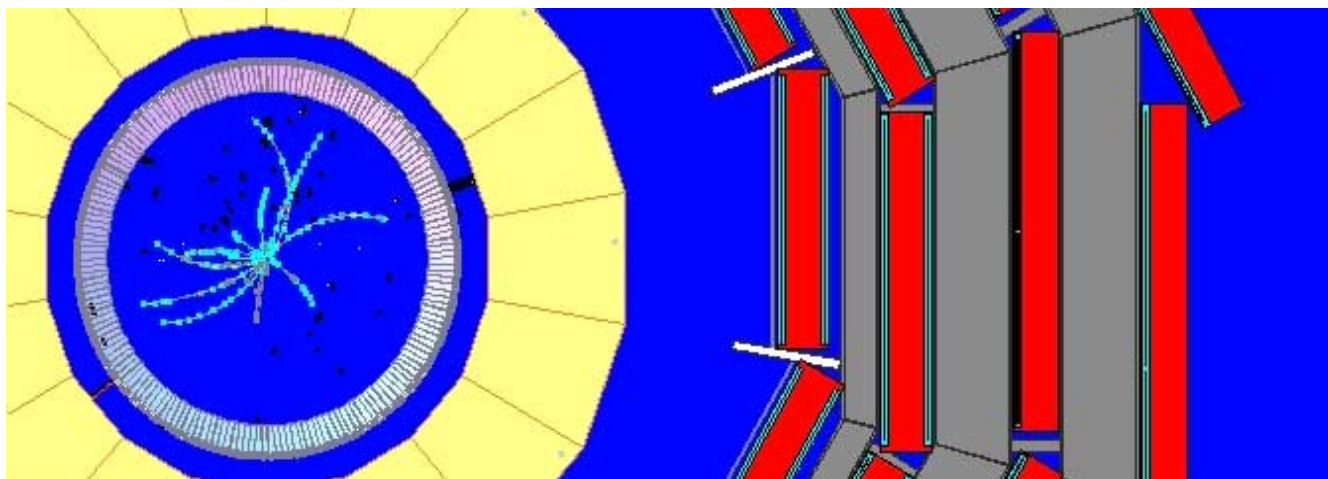


# IGUANA Tutorial

Ianna Osborne

*Northeastern University,  
Boston*



## Introduction

You are about to get familiar with the visualisation subsystem of the ORCA project. This tutorial will help you to start working with it. If you run into problems, have questions, or new ideas, please do not hesitate to contact me ([Ianna.Osborne@cern.ch](mailto:Ianna.Osborne@cern.ch)).

## 2. Getting Started

The visualisation executable **OrcaVisMain** is built along with the ORCA release. A user can run it directly from the release area or rebuild it in his or her developers area. The latter allows customization and optimization of the visualisation program according to the user needs.

### 2.1 Recipe for an Unpatient User

```
> cd /afs/cern.ch/cms/Releases/ORCA/ORCA_5_1_2/src
> eval `scram runtime -csh`
> cd - (back to your working area)
> source /afs/cern.ch/cms/Releases/ORCA/ORCA_5_1_2/
src/Visualisation/OrcaVis/test/testfed.csh (in one line)
> OrcaVisMain
```

### 2.2 Guidance for the Rest of Us

To optimize or customize visualisation it is recommended to create your own copy of

the ORCA Visualisation subsystem. It can be done in a few steps described below. Firstly, create a SCRAM developers area for ORCA project. Secondly, checkout the source code of the Visualisation subsystem from the CVS repository. Thirdly, change the source code or the BuildFiles, if desired, and rebuild the libraries and executable. A developers area can be created from a project release area and it relies on the release area. Note: to get the list of all released ORCA versions on your site use the command `scram list ORCA`. Here are the detailed commands:

### 2.2.1 Create a Developers Area

```
> project ORCA
> scram project ORCA ORCA_5_1_2
> cd ORCA_5_1_2/src
> cvs co -r ORCA_5_1_2 Visualisation
```

### 2.2.2 Build the Libraries

```
> cd Visualisation
> scram build
```

### 2.2.3 Build the Executable

```
> cd Visualisation/OrcaVis/test
> scram build bin
```

### 2.2.4 Set the Run-time Environment

To run the executable a user needs to set up the SCRAM run-time environment:

```
> eval `scram runtime -csh`
```

and set the **OO\_FD\_BOOT** environment variable that will point to the federated database you will be working with. You also have to have a **.orcarc** configuration file. An example of how to do this in one step is given in `Visualisation/OrcaVis/test/testfed.csh` file. All what the user needs to do is:

```
> cd Visualisation/OrcaVis/test
> source testfed.csh
```

Now the user can start the executable:

```
> OrcaVisMain
```

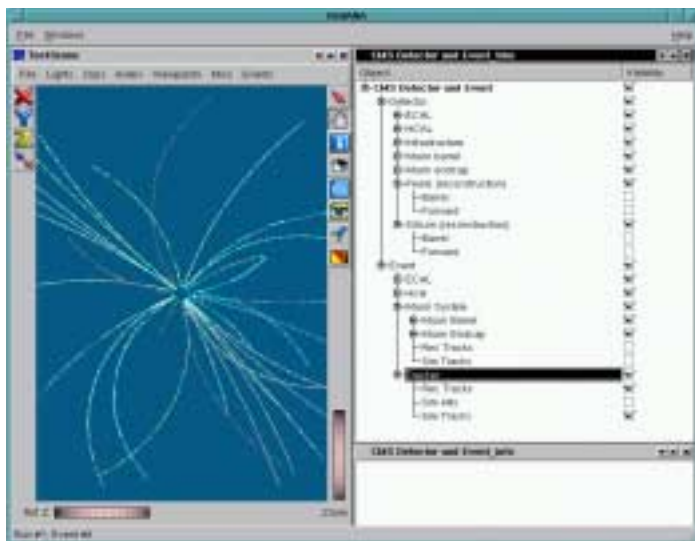
## 3. Visualisation Anatomy

When the application starts a user sees the usual COBRA printout informing the user about all stages of COBRA initialization. After the initialization is finished, the GUI pops up. The default GUI consists of three MDI (multi-document interface) windows: a 3D view window on the left, a controller window on the right, and an information window at the bottom right. The windows are tiled in the workspace with a menu bar. The menu bar contains only three menus: **File**, **Windows**, and **Help**.

The *File menu* item **New Scan** starts another set of windows: a 3D view, a controller, and an information window in the same workspace. The File menu item **Exit** ends the session and closes the GUI. The *Windows menu* shows names of all windows in the

workspace and allows to activate one of them, or to arrange the windows - tile or cascade.

The *Help menu* provides a short information on the version of the program (menu item **About**) and the context help - **What's This**.



The 3D view window has a menu bar with different menus and controls, many other controls on the left and right sides, and wheel manipulators. See chapter 6 for more details.

To start with the 3D view is empty - black, because no objects have been requested to be constructed. The first event is dispatched automatically. The status bar shows the run and event number.

The controller helps to manipulate visibility, switching it on and off for the selected objects (see chapter

5). If the object hasn't been constructed - it will be taken care of (see chapter 7). All selected objects are updated when the next event is requested and the status bar shows respective run/event numbers.

## 4. Environment (Re-) Configuration

The `testfed.csh` script defines several Objectivity-related environment variables, including `OO_FD_BOOT`, executes the command to set SCRAM run-time environment, and writes out the `.orcarc` configuration file.

To change the federated database you have to quit the OrcaVisMain and set `OO_FD_BOOT` pointing to a valid federated database (in one line):

```
> setenv OO_FD_BOOT
cmsc01.cern.ch::/shift/cmsc01/data22/cmsprod/btauProd/
UserFed510/ORCATEST.boot
```

The next step is to modify `.orcarc` *InputCollections*:

```
InputCollections =
/System/RecHits/barrel100bb_d/barrel100bb_d
```

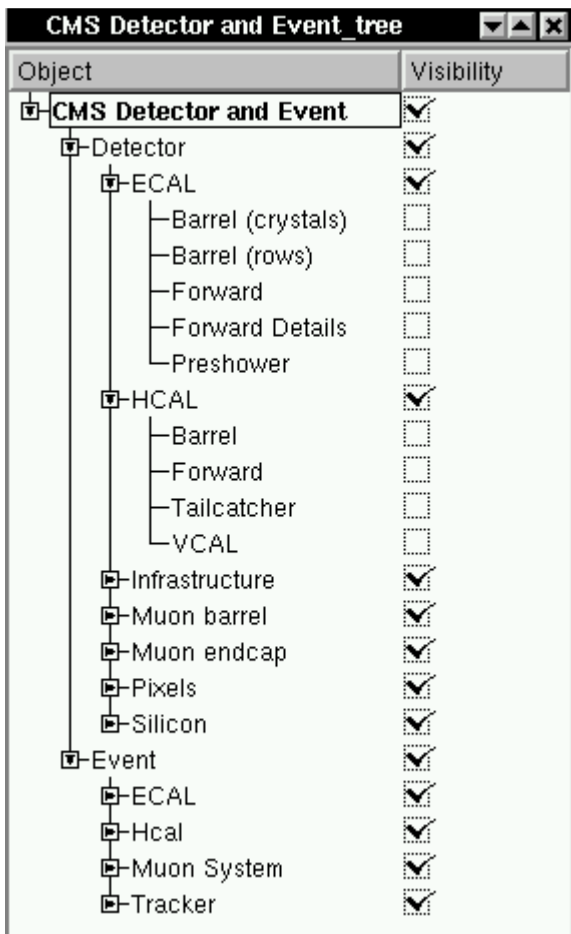
Then restart the program:

```
> OrcaVisMain
```

## 5. Controller

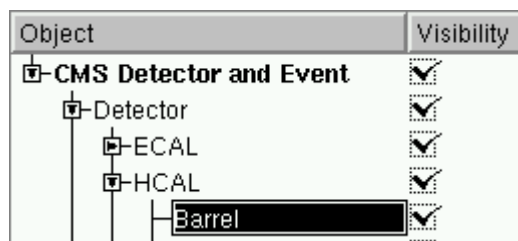
The controller window contains the full logical tree of all loaded objects. It is possible to change it - by modifying the `BuildFile` in the `Visualisation/OrcaVis/test` directory a user can rebuild the executable to load less or more objects. For example, if a user is not interested in the Tracker visualisation, he should remove all libraries related to the Tracker from the `BuildFile`, rebuild the executable (see chapter 2), and run it. The

controller tree branch "Tracker" will not appear and the application itself will be "slimmer".



A user has control over the objects: he can switch visibility of the object requesting at the same time to construct the object if it hasn't been constructed yet (action on demand). The tree-like view groups the objects into different categories. By default all objects are invisible (not constructed) and all parents are set visible. A parent doesn't have any representation. See the picture on the left.

The check-box toggles the visibility of the corresponding object. The object is visible if its check-box is ticked  and its parent(s) are visible .



*Example above:* HCAL Barrel is visible when all groups: CMS Detector and Event, Detector, and HCAL and the Barrel itself are set visible.

**▼ Hint:** To open or close the branches of the tree either double-click with right mouse button on the branch name or click on the small arrow box on the left.

## 6. 3D Viewer Window

The 3D viewer is an MDI window that has its own menu bar and other controls to manipulate the *scene graph* created on demand by controller (a user manipulation with the controller tree).

### 6.1 Menus

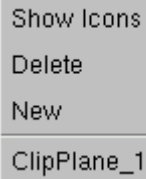
Most of the menu actions are intuitive. The **File** menu provides the following items: **Open** - to open an OpenInventor file into the viewer. The scene graph from the file is added to the existing scene graph. **Save as** saves the displayed scene graph into an OpenInventor file format. The **Print as** saves the displayed scene into a file. It pops up a dialog where a user can select a file name, location, and format of the file. Available formats are: GIF, TIFF, JPEG, PostScript. **Clone Scene** opens another 3D window with the same (cloned) scene graph that is attached to the same controller.



The **Lights** menu allows to manipulate light floods in the scene graph: show and hide the icons, delete the existing lights and create new

Clips

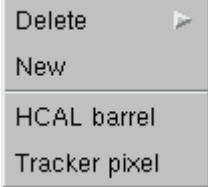
ones, switch the lights on and off.



The **Clips** menu controls the visibility of the clip plane icons, allows to delete clip planes and to create a clip plane with an arbitrary name - a dialog pops up asking for a name of the new plane. It provides control over the existing clip planes in the scene graph - activates or disactivates them.

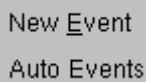
The **Viewpoints** menu allows to save the viewpoints. It helps to navigate into the 3D space - go back to the previous view of the detector and event.

Viewpoints

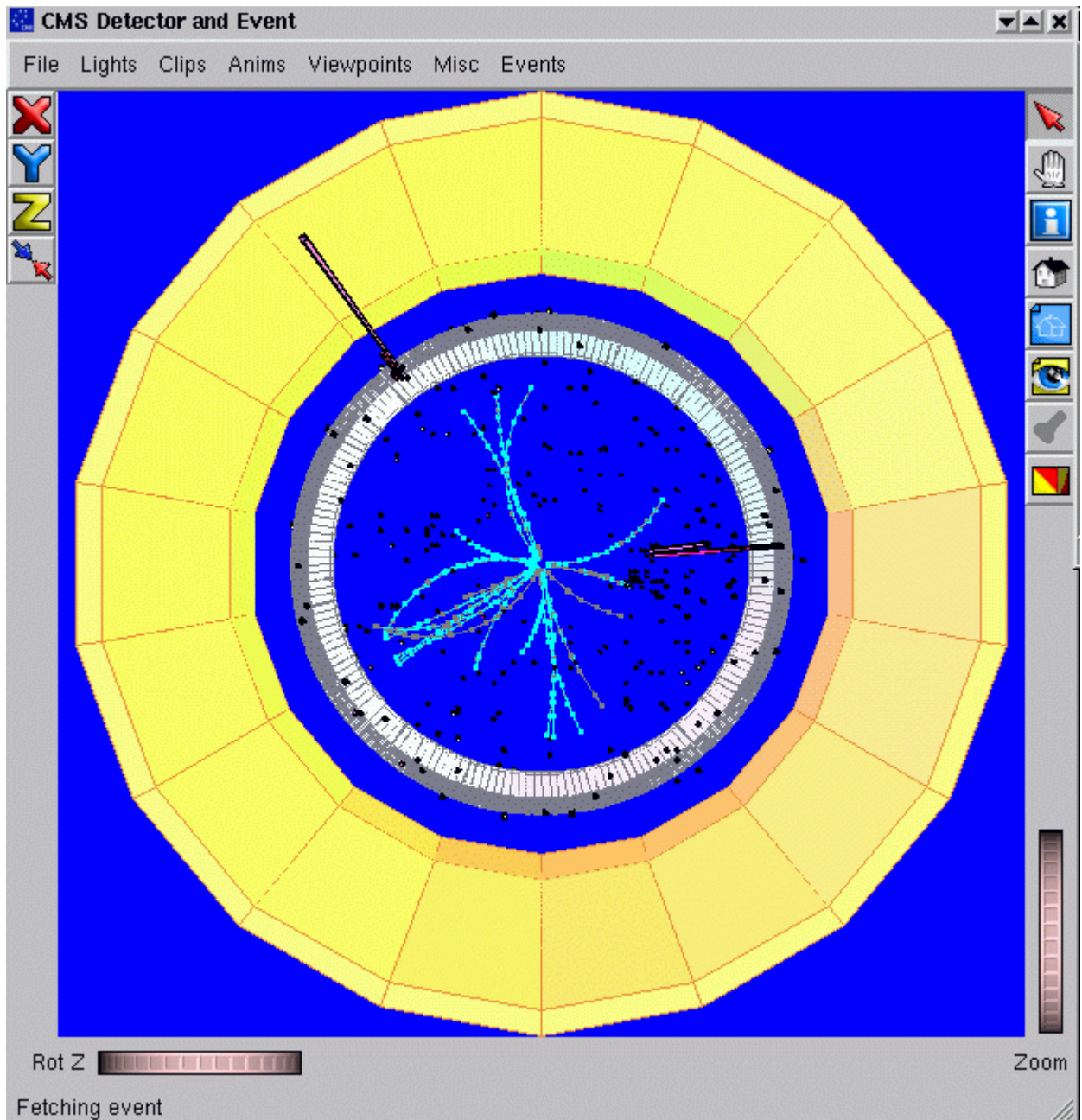


Events

The **Misc** menu provides controls that do not fall into any other category, such as **Z Slice** and **Background Color**.



The **Events** menu contains **New Event** item that requests the next event to be fetched, and **Auto Events** item that fetches events automatically within a fixed time interval. The signal from the menu item is connected to



the slot (C++ method) that asks COBRA to provide the next event.

## 6.2 Viewer Buttons

The buttons on the left and on the right sides of the viewer help to navigate in the 3D space. If the Scene Viewer window is too small, not all of these buttons will be visible.



### Select/Pick Button

Selects object manipulation or pick mode (and deselects camera or viewer mode). The cursor shape will change to an arrow. In this mode, the user is manipulating objects in the scene graph.



### View Button

Selects camera or viewer mode (and deselects object manipulation or pick mode). The cursor shape will change to a hand icon. In this mode, the user is moving the camera in 3D space.



### Help

This menu provides help about the application.



### Home Button

Returns the camera to its home position (initial position if not reset).



### Set Home Button

Resets the home position to the current camera position.



### View All Button

Brings the entire scene graph into view.



### Seek Button

Allows the user to select a new center of rotation for the camera. When clicked on (and in viewer mode) the cursor changes to a crosshair. The next left mouse button press causes whatever is underneath the cursor to be selected as the new center of rotation. Once the button is released, the camera either jumps or animates to its new position depending on the current setting of the seek time in the preferences dialog box.



### Camera Alignment Buttons

Select the axis of alignment (X, Y, or Z) of the camera.



### Camera Alignment Button

Inverts the coordinates through origin.



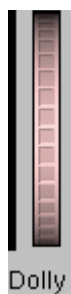
### Projection Button

Selects the type of camera used by the viewer. It toggles between the two available camera types - perspective and orthographic.



### Rotation Thumb Wheel

Rotates the scene clockwise (drag it right) or counterclockwise (drag it left).

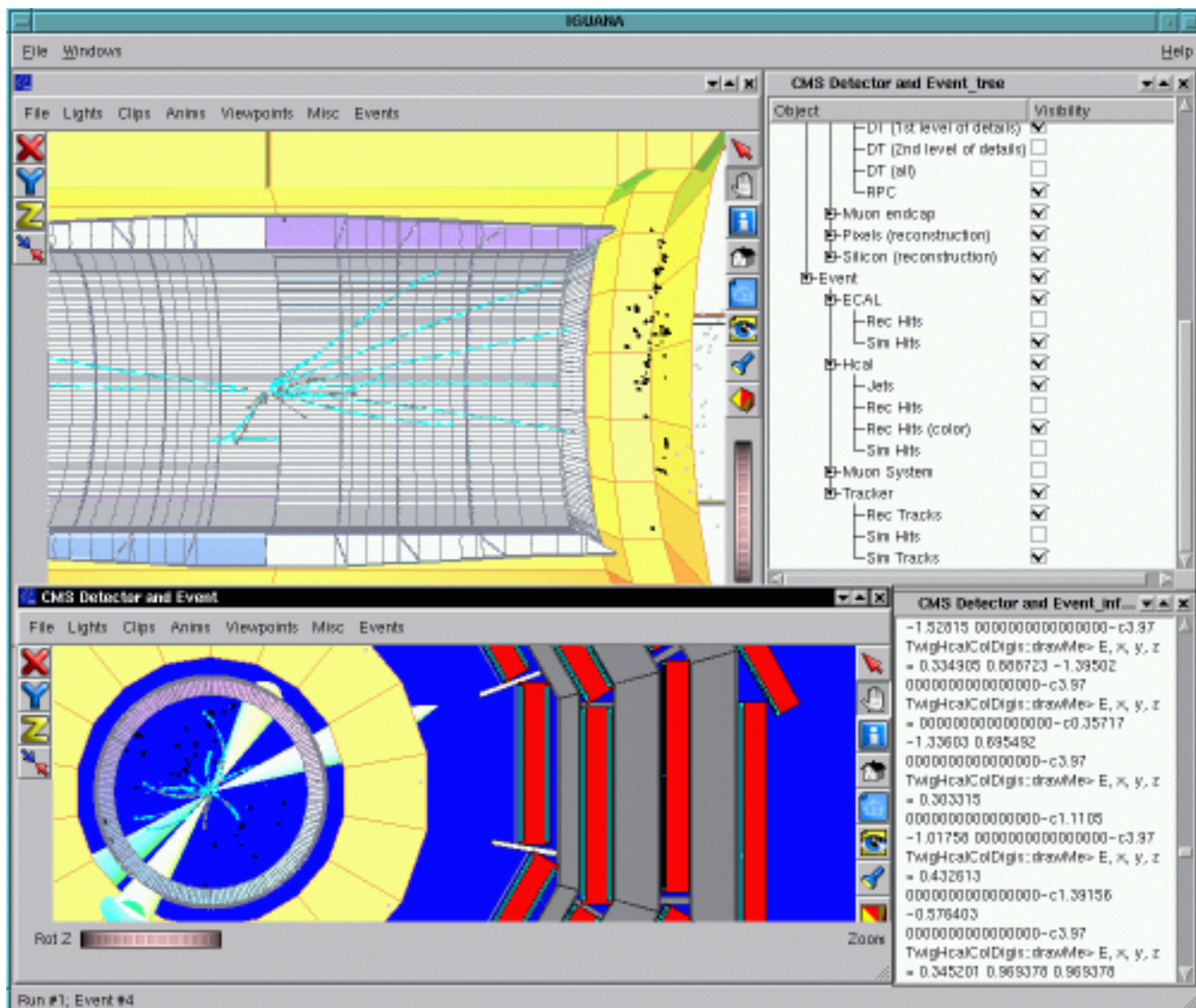


### Dolly Thumb Wheel

is only available to the perspective camera. It brings the camera closer to (drag the wheel down) or further from (drag it up) the scene.

### Zoom Thumb Wheel

is only available to the orthographic camera: drag it down to zoom in, drag it up to zoom out.



## 7. Documentation

[1] ORCA Project: <http://cmsdoc.cern.ch/orca>

[2] IGUANA Project: <http://iguana.web.cern.ch/iguana>

[3] OpenInventor on-line documentation:  
[http://www.tgs.com/pro\\_div/Support/Documentation/OIV3-0/doc/index.htm](http://www.tgs.com/pro_div/Support/Documentation/OIV3-0/doc/index.htm)

[4] OpenGL: <http://www.mesa3d.org>

[5] Qt Toolkit Reference Documentation: <http://doc.trolltech.com>

## 8. Glossary

**Federated Database** Objectivity/DB provides logical storage through persistent objects, databases, and federated databases. The highest level of storage is the federated database. Each federated database logically contains one or more databases. Each database contains objects.

**VRML** - Virtual Reality Modeling Language - is an open standard for 3D multimedia and shared virtual worlds on the internet.

**OpenGL** is a cross-platform standard for 3D rendering and 3D hardware acceleration.

**OpenInventor** is an object-oriented 3D toolkit. It presents a programming model based on a 3D scene database. It is built on top of OpenGL.

**Scene database** is represented through a directed acyclic graph: no path starts and ends at the same vertex.

**Node** is a vertex of a graph.

**Qt Toolkit** is a high-performance object-oriented framework for developing graphical user interface (GUI) applications.

**Signals** and **slots** - concepts provided by Qt is a type-safe mechanism for communication between objects. It allows objects to communicate without any knowledge of each other.

**MDI** - Multi-Document Interface - the ability of an application to show windows giving views of more than one document at a time.

**Orthographic projection** - a method of projection in which an object is depicted using parallel lines to project its outline on to a plane.