Collaborative Scientific Visualization Workbench Manual

Comp 991 Paper

Spring 2011

by Alexis Yee Lyn Chan

1. Introduction

This is a manual for the Collaborative Scientific Visualization Workbench Manual created as part of a Comp 991 project at the University of North Carolina at Chapel Hill (UNC Chapel Hill). "3.User Manual" is written for students, faculty and staff in Computer Science or other scientific fields, who have access to equipment at UNC Chapel Hill. "4. Setup and Maintenance Manual" is written for those with Computer Science background who may be interested in implementing similar projects or extending the work done in this project.

The project is implemented using ParaView, a data analysis and visualization software (Kitware Inc., 2009)

2. Objectives

The objectives of this system are:

1. to visualize scientific data sets as virtual objects by providing the single user's head-tracked stereo, "world-in-hand" manipulation of the data set, and peripheral vision provided by a secondary display at right angles with the primary display
2. to enable collaborative observation of scientific data sets by two users
3. to enable occasional visualization modification of scientific data sets while in shared state

3. User Manual

3.1 Using ParaView

This is a brief overview of how to use ParaView to visualize scientific data, i.e. convert the raw data collected in scientific experiments or simulations into "a form that is viewable and understandable to humans" (Kitware Inc., 2009). Refer to http://www.vtk.org/Wiki/images/6/65/ParaViewTutorial36.pdf for an in-depth tutorial (Kitware Inc., 2009).

3.1.1 Data File Formats

ParaView can be used to visualize spatially embedded, multi-variable 2D and 3D data, and 2D images. Each record within a data set can contain values of several variables for the particular point in the data set geometry. Many common scientific data file formats are supported by ParaView, for e.g.:

- EnSight (.case, .sos)
  - commonly used in computational fluid dynamics (CFD) and computer-aided engineering (CAE)
- Protein Data Bank (.pdb)
  - commonly used in molecular biology
- Cosmology Files (.cosmo)
  - commonly used in cosmology
- Stereo Lithography (.stl).
  - commonly used in stereolithography
- ParaView Data file format (.pvd)
- Visualization Toolkit file format (.vtp, .vtu, .vti, .vts, .vtr)

The data sets loaded into ParaView are referred to as "readers". Refer to http://paraview.org/Wiki/ParaView/Users_Guide/List_of_readers for the full list of file formats that can be loaded onto ParaView (Kitware Inc., 2011).

3.1.2 Data Set Mesh Types

Spatially embedded data set can be sub-sampled using mesh sampling. ParaView supports the following mesh types:

1. Uniform Rectilinear (Image Data)
   a. 1-, 2- and 3-D array of data with points orthonormal to each other and regularly spaced in each direction.
2. Non-uniform Rectilinear (Rectilinear Grid)
   a. 1-,2- and 3-D array of data with axes orthonormal to each other. Spacing may vary along each axis.
3. Curvilinear (Structured Grid)
   a. Each point can be placed at an arbitrary coordinate – provides more compact memory footprint and variation in mesh shape.
4. Polygonal (Poly Data)
   a. Basic rendering primitives, consisting of points, lines and 2D polygons. Connections between cells can be arbitrary or non-existent.
5. Unstructured Grid

a. Consists of points, lines, 2D polygons, 3D tetrahedral and nonlinear cells. Similar to polygonal data except that they can represent 3D tetrahedra and nonlinear cells which cannot be directly rendered.

Refer to section *"1.2 Basics of Visualization"* of the *The ParaView Tutorial* for further information (Kitware Inc., 2009).

When a data set is loaded into the Collaborative Scientific Visualization Workbench (or ParaView), an entry will be created in the Pipeline Browser. In the following example, a data set "SAS.res_t2564_21.vtu" is loaded:

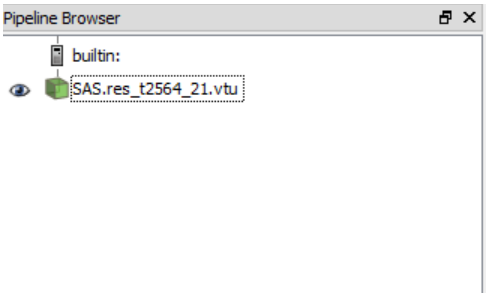


Figure 1: Data set entry in the Pipeline Browser

To see the data mesh type of the data set:

- Click on the data set entry in the Pipeline Browser panel.
- Click on the Information tab of the Object Inspector panel

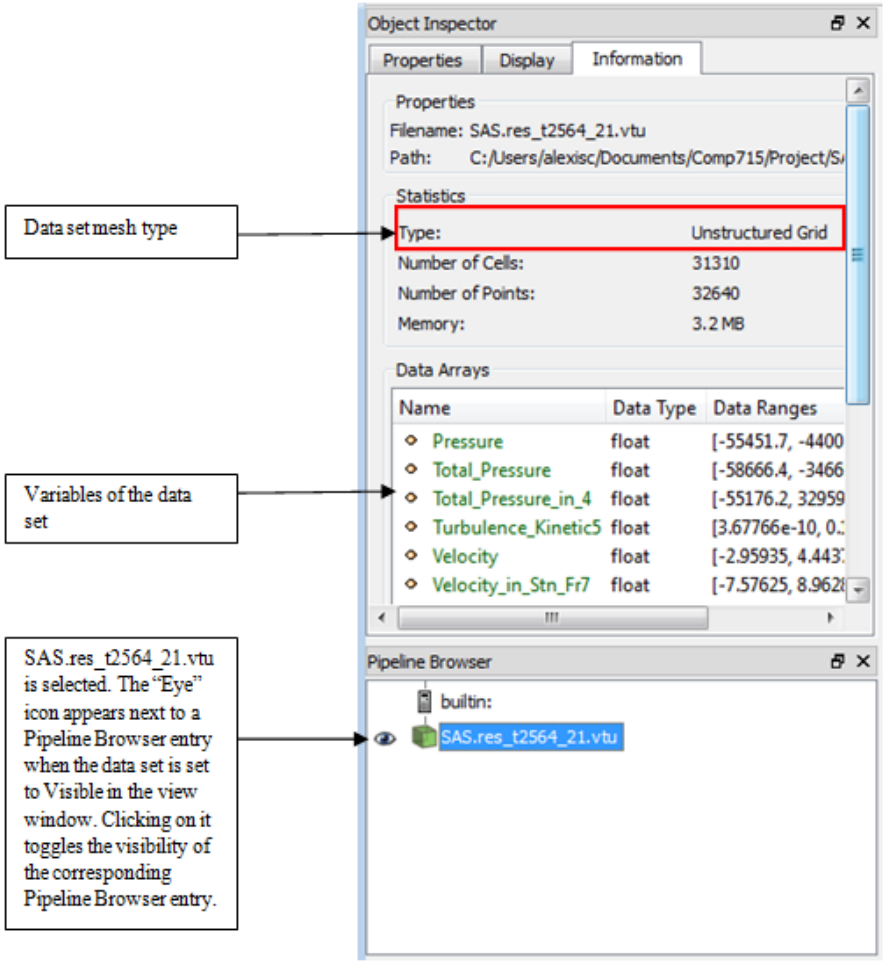


Figure 2: A data set entry is selected in the Pipeline Browser. The data set's mesh type is shown in the Object Inspector Panel. Refer to section "2.1 User Interface" of the *The ParaView Tutorial* for a tutorial on how to use the Pipeline Browser and Object Inspector panels. (Kitware Inc., 2009).

3.1.3 Visualization Filters

ParaView filters are "functional units that process the data to generate, extract, or derive features from the data" (Kitware Inc., 2009). Data set meshes represent surfaces – filters allow one to probe different scalar and vector variables within the surfaces.

ParaView filters can be applied to data sets loaded into ParaView readers, sources (i.e. standard ParaView objects such as cylinders) and other filters to "form a visualization pipeline" (Kitware Inc., 2009).

Here are several examples of filters listed in section *"2.4 Filters"* of the *The ParaView Tutorial* (Kitware Inc., 2009):

- **Calculator**
  - Evaluates a user-defined expression on a per-point or per-cell basis.
- **Contour**
  - Extracts the points, curves, or surfaces where a scalar field is equal to a user-defined value with a tolerance of 5 decimal points. This surface is often also called an isosurface.
- **Clip**

- Intersects the geometry with a half space. The effect is to remove all the geometry on one side of a user-defined plane.

- **Slice**
  - Intersects the geometry with a plane. The effect is similar to clipping except that all that remains is the geometry where the plane is located.
- **Threshold**
  - Extracts cells (volumetric for a 3D data set and planar for a 2D data set) where a scalar field is equal to a user-selected variable and user-defined range with a tolerance of 5 decimal points (default range is the entire range of values of that variable within the data set).
- **Extract Subset**
  - Extracts a subset of the data set by defining either a sub-volume or a sub-sampling rate.
- **Stream Tracer**
  - Seeds a vector field with points and then traces those seed points through the (steady state) vector field.
- **Programmable Filter**
  - Filters the data set according to a Python script given by the user.

To apply a filter to a data set, click on the data set entry in the Pipeline Browser (see Figure 2). In the menu toolbars, select Filters→Alphabetical→ [name of filter].

3.1.4 Color Map

To simultaneously view multiple variables of a same data set, it is common to apply a filter to a variable of the data set and map another variable to a set of colors (referred to as the color map). In the following example, "Log AR" variable of a 2D data set is filtered using "Warp by Scalar", which maps Log AR's magnitude to height in Z axis. Log M is mapped to the Red-White-Purple color map (as seen in the color map legend on the right).

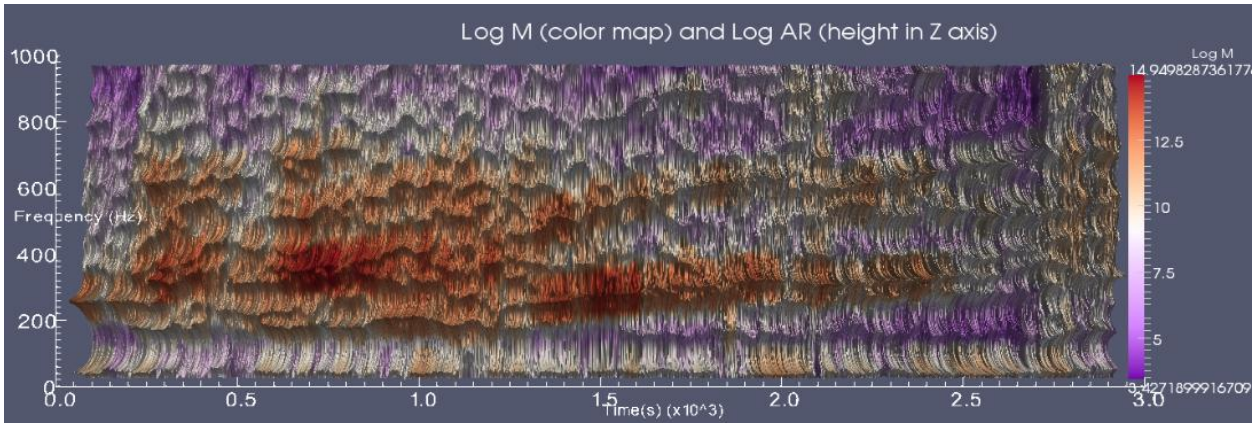


Figure 3: The variable "Log AR" of a data set is mapped to height in Z-axis using the "Warp by Scalar" filter and the variable "Log M" of the same data set is mapped to the Red-White-Purple color map.

To choose the variable by which a data set is colored with:

- Click on the data set entry in the Pipeline Browser panel.
- In the Right panel of the Collaborative Scientific Visualization workbench, click on the drop-down selection menu and select the desired variable. See Figures 9 and 14 in "3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization".

3.1.5 Testcases: Creating and Editing ParaView Readers, Sources & Filters

1. Download the test data sets from http://www.paraview.org/Wiki/images/5/5d/ParaViewTutorialData.tar.gz
2. Perform Exercises 2.1 – 2.7 and Exercises 2.9 – 2.10. (Exercise 2.8 is excluded because multiple view windows are not allowed in the Collaborative Scientific Visualization Workbench).

3.2 Operating the Workbench

This section assumes that you have access to the 3rdTech Wide-Area Tracker system and the computer, sutherland.cs.unc.edu, both of which are located in the Effective Virtual Environments laboratory at UNC Chapel Hill. Otherwise, refer to section Setup and Maintenance Manual" section for instructions on setting up the workbench in your laboratory.

3.2.1 Initializing the System

3.2.1.1 Starting the head-tracking system

1. Reserve the 3rdTech Wide-Area Tracker Ceiling with the EffectiveVirtual Environments group (eve@cs.unc.edu)
2. Log into the computer: tracker1-cs.cs.unc.edu
3. The *3rdTech HiBall-3000 Wide-Area Tracker User Manual* can be found in C://Hiball/Config/HiBallManual_1.5f/ (3rdTech, Inc., 2002).
4. Make sure that two headband-mounted Hiballs with the labels "3008" and "3012" respectively are connected to the CIB (Refer to the Wide-Area Tracker user manual).
5. Turn on the Wide-Area Tracker by pressing the red switch of the CIB.
6. Double-click on the Windows Desktop Shortcut with the label "3008+3012".

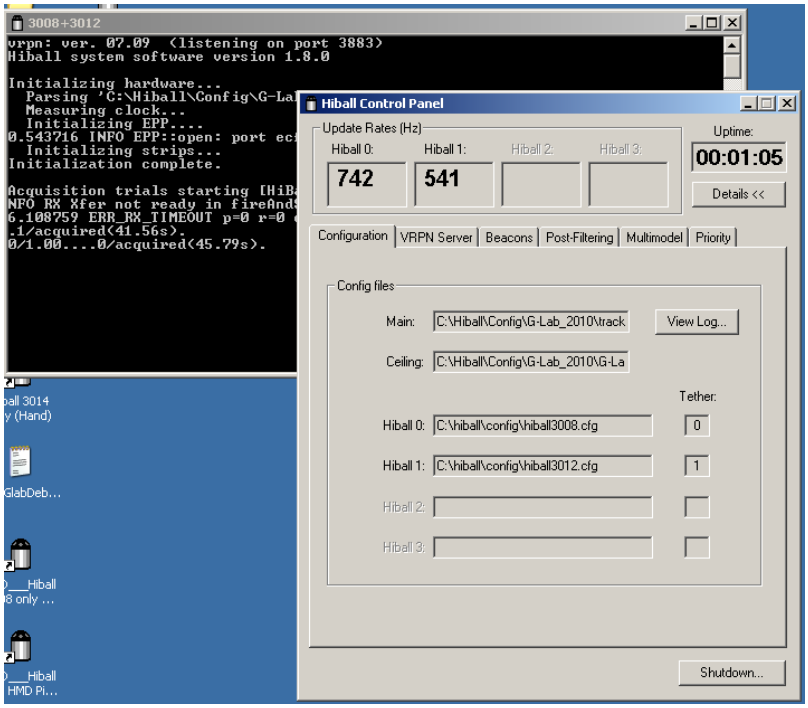


Figure 4: The 3rdTech Wide-Area Tracker running correctly

3.2.1.2 Starting other VRPN Devices

1. Make sure that the following are connected to the computer: sutherland.cs.unc.edu
   a. TNG-3B Serial Interface
   b. SpaceNavigator
   c. Phantom Omni Device
2. If any of the devices in Step 1 are not connected to the computer, refer to "4.1.2 External devices" for connection details.
3. Log into sutherland.cs.unc.edu
4. Double-click on the Windows Desktop Shortcut with the label "vrpn_server.bat"

3.2.1.3 Starting the Workbench

1. On sutherland.cs.unc.edu, double-click on the Windows Desktop Shortcut with the label "ParaViewR0.bat" to start the Workbench for the first user. This user should wear the headband with the Hiball "3008"
2. Double-click on the Windows Desktop Shortcut with the label "ParaViewR1.bat" to start the Workbench for the second user. This user should wear the headband with the Hiball "3012".

3.2.2 Using the External Devices

In the following sections, the data set in the "Testcase" sections refers to the computational fluid dynamics (CFD) simulation loaded upon initialization of the Collaborative Scientific Visualization Workbench. (See 3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization). The instructions in the "Testcase" sections apply to any other data set.

3.2.2.1 Head-tracker

The headband with Hiball "3008" should be worn by the user viewing the application started by "ParaViewR0.bat"; the headband with Hiball "3012" should be worn by the user viewing the application started by "ParaViewR1.bat". The knob of the headband should be at the back of the user's head.

Try to keep the Hiball in a mostly upright position. Occasionally, the ceiling trackers may lose tracking of the Hiball. When this happens, the visualization "freezes". In this scenario, you should stop or stabilize the movement of your head until head-tracking is restored.

3.2.2.1.1 Testcase

Wearing the headband with Hiball "3008", start "ParaViewR0.bat". Move your head to the right of the screen. When looking at the monitor, you should get the sense that you are looking through a window to the data set and your head is moving to the right of the data set.

3.2.2.2 SpaceNavigator

This controls the rotation and translation of the data set.

In the following description, "pushing" and "tilting" actions apply to the "joystick" part of the SpaceNavigator (marked with a red square), not the entire SpaceNavigator.

To rotate the data set around the up direction, rotate the SpaceNavigator around the up direction. Similarly, to pitch the data set forward, tilt the SpaceNavigator forward; to roll the data set to the left and right, tilt the SpaceNavigator to the left and right.

To bring the data set towards you, pull the SpaceNavigator towards you. To bring the data set to the right/left, push the SpaceNavigator to the left/right; to bring the data set upwards or downwards, push the SpaceNavigator upwards or downwards.

Figure 5: "Joystick" part of the SpaceNavigator.

3.2.2.2.1 Testcase

1. Tilt the SpaceNavigator to the left. The data set should tilt to the left.
2. Push the SpaceNavigator away from you. The data set should away from you.
3. Pull the SpaceNavigator upwards. The data set should move upwards.

3.2.2.3 Stereo Glasses

This is the easiest device to use. Just put it on the glasses, and press the Button if you do not see the data set rendered in stereo. Refer to "3.2.2.4 Stereo Input Dial" for the testcase.

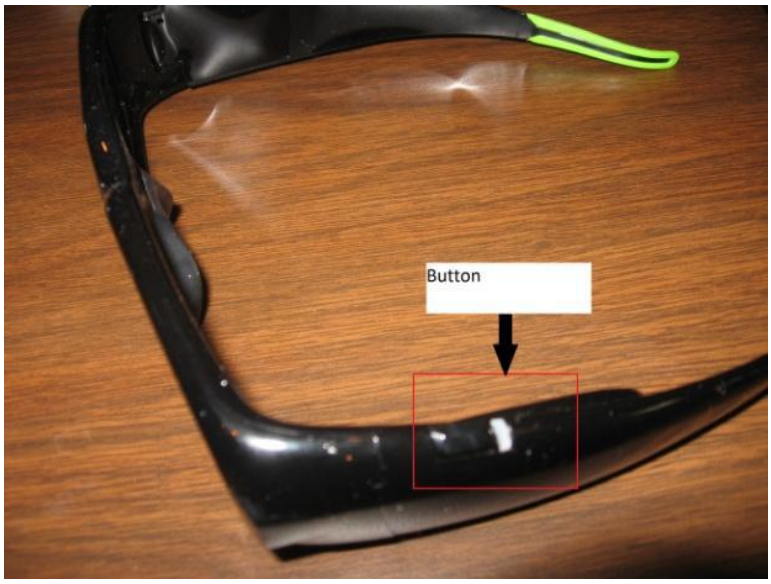


Figure 6: The On/Off button of Nvidia 3D Stereo glasses

3.2.2.4 Stereo Input Dial

A dial (actually a rotary potentiometer, see "4.1.2.4 TNG-3B Serial Interface") is connected to this blue box. This dial will change the separation of images rendered on the screen for stereo viewing. The user using "ParaViewR0.bat" and "HiBall 3008" should use the dial marked with "1" on the TNG-3B Serial Interface and the user using "ParaViewR1.bat" and "HiBall 3012" should use the dial marked with "2".

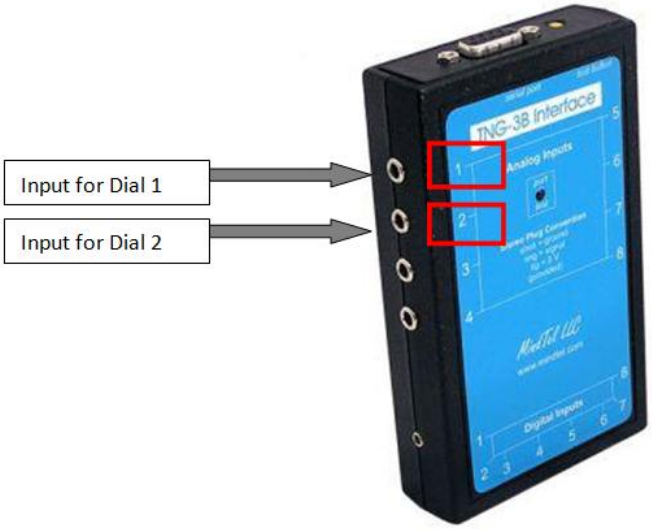


Figure 7: Input ports for Dial 1 and Dial 2 of the TNG-3B Serial Interface

3.2.2.4.1 Testcase

Start "ParaViewR0.bat". Turn Dial 1 clockwise. The separation between the two images (one for the left eye and one for the right eye) should increase. Put on the Stereo Glasses. The data set should appear in 3D.

3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization

A special use case that was implemented for the first user (use "ParaView0.bat" and "HiBall 3008") is the Vortex Visualization workbench. This use case showcases how the Phantom Omni Device can be used to probe computational fluid dynamics (CFD) data via interactive seeding of stream tracers with the Phantom Omni Device.

The current implementation is tailored to a preprocessed version of the sets of data provided by the Institute of Applied Mechanics, Clausthal University, Germany (Dipl. Wirtsch.-Ing. Andreas Lucius) for the IEEE Visualization Contest 2011 at http://viscontest.sdsc.edu/2011/data set/data set.html.

The data set is loaded upon initialization of the workbench.

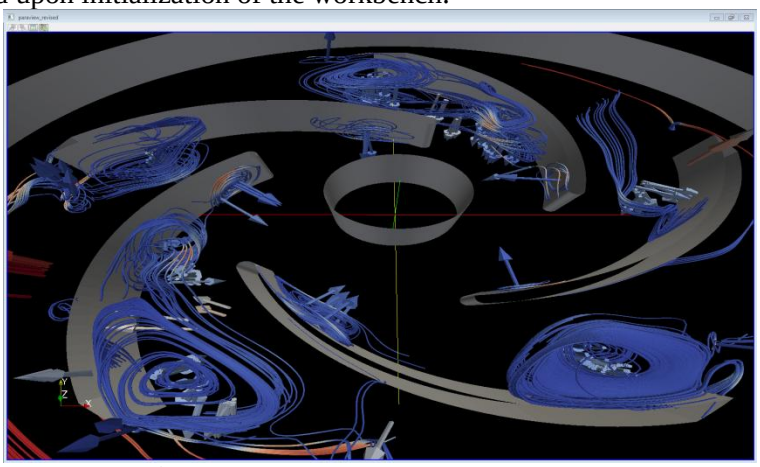


Figure 8: Screenshot of the View Window. Vortex visualization data is loaded upon initialization

All the other VRPN devices work for the first user for the Vortex Visualization Workbench. The center and right panel of the Graphical User Interface window is tailored for this use case.

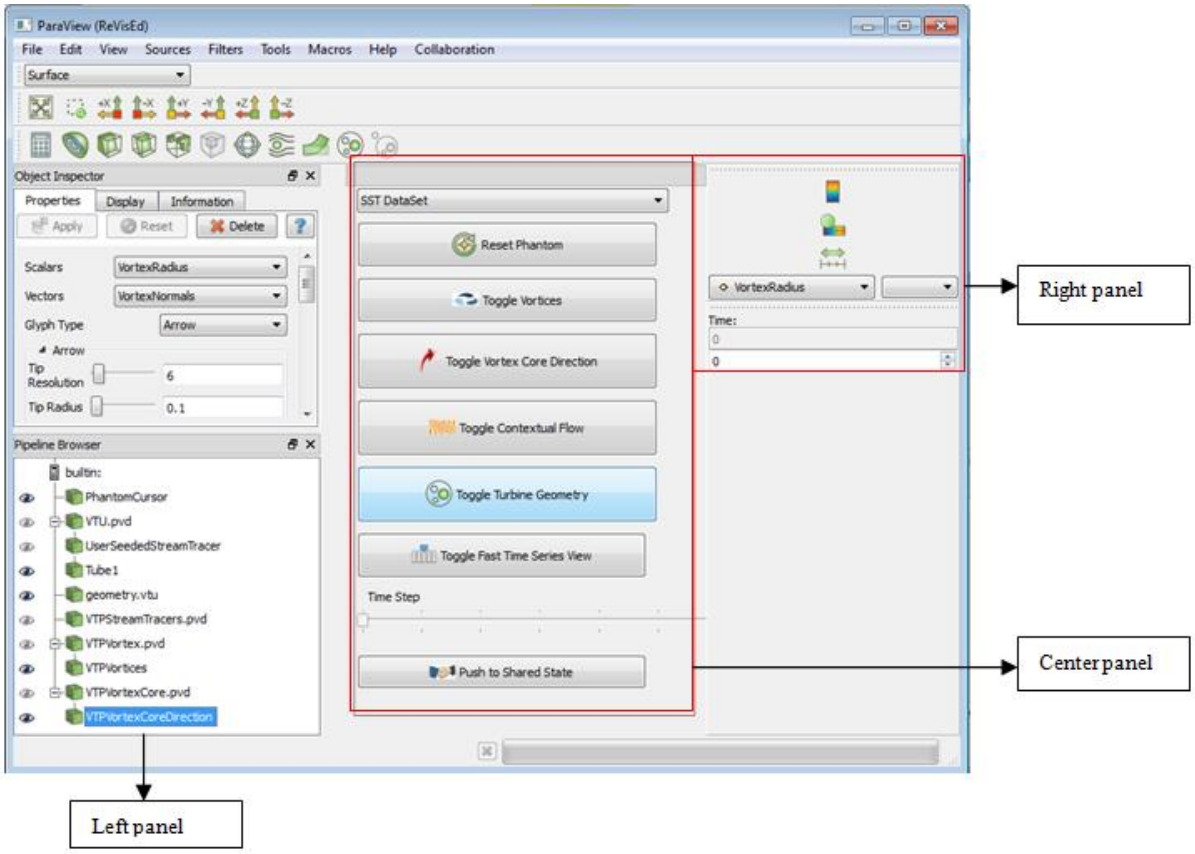


Figure 9: Left, Center and Right panels of the Graphical User Interface window

However the Object Inspector and Pipeline Browser panels on the left are applicable for all types of data sets. Refer to *Appendix 3: Vortex Visualization Workbench Manual* of *Visualization of Flow Instability in Centrifugal Pumps* for a guide on how to use the graphical user interface (Lee, 2011).

Future plans include incorporating the preprocessing stage of the data into the Collaborative Scientific Visualization Workbench and fixing the Phantom interaction for two users.

3.2.2.5.1 Phantom Omni Device Instructions/Testcase

Figure 10: Phantom Omni Device

The Phantom Omni Device controls the position of the Phantom Cursor (the Sphere drawn in the View Window). To move the sphere around, hold the Phantom device like a pen.

Figure 11: Hold the The Phantom Omni Device stylus like a pen

Press the first button (the one closest to the lower end of the Phantom Device) in order to generate a stream tracer at the position of the sphere. You can move the sphere around and generate new stream tracers at other positions.

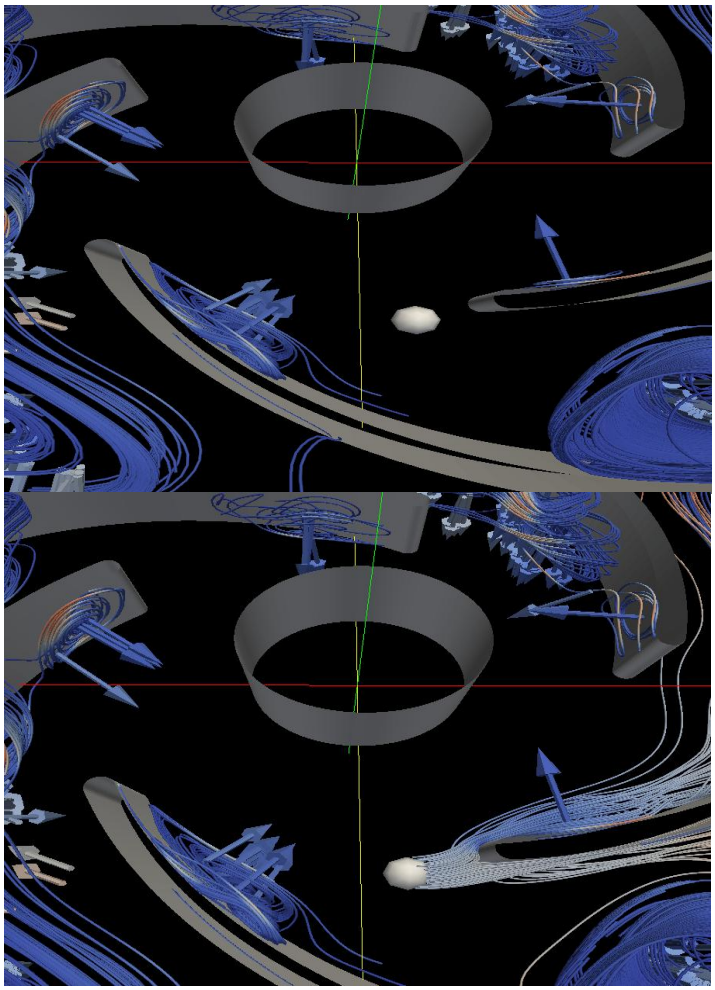


Figure 12: Top: Sphere-shaped cursor for the Phantom Omni Device. Bottom: Stream-tracer tubes generated starting from the position of the Phantom Cursor when the Phantom button is pressed

CAUTION:

a. Please use small movements and avoid jerkiness when operating the **3Dconnexion SpaceNavigator**. This may cause the sphere (Phantom Cursor) to be lost in space.

b. If the Sphere disappears, switch the Data set type to another data set and switch it back to the original one that you were looking at. This is a workaround.

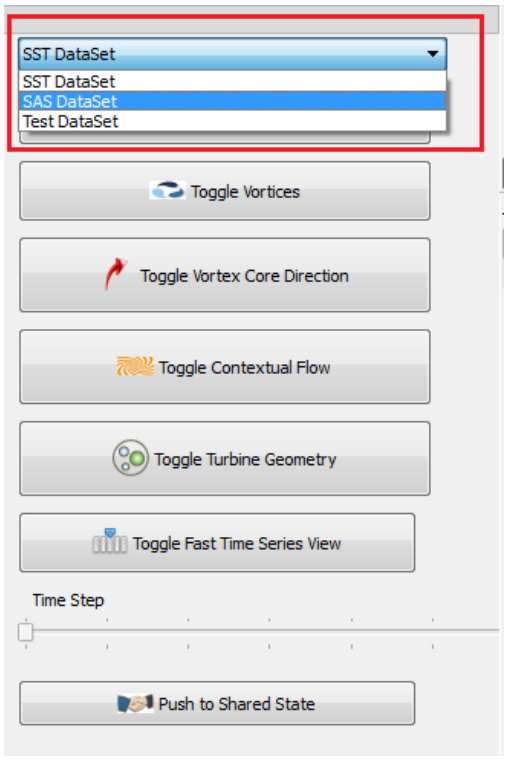


Figure 13: Switching vortex visualization data sets

c. The stream tracer will take a few seconds to be generated (the green progress bar will appear on the UI).

d. Occasionally the color map (i.e. the map of colors representing a variable of the data set) of the stream tracer may be reset to the first variable instead of the user-selected variable. You would have to change the color map.

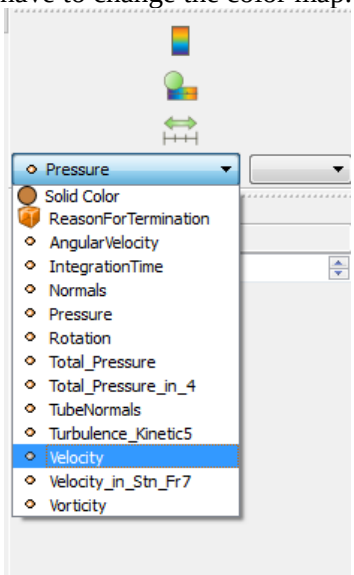


Figure 14: Changing the variable used for the color map

3.2.3 Operating ParaView in Shared Mode

3.2.3.1 External Devices

1. Each user has an independent view of the data set at all times because each user has a headband-mounted HiBall tracker.
2. Users can independently control the stereo separation of their views because each user has a TNG-3B Serial Interface dial (or rotary potentiometer).
3. The SpaceNavigator affects the "world-in-hand" manipulation of the data set for both users.

The afore-mentioned behavior of the external devices holds true even when each user is independently applying filters and modifying the data set using the Graphical User Interface. The modifications will not be automatically shared between the users.

3.2.3.2 Push to Shared State

To share your modification with the other user, click on the UI button "Push to Shared State". The other user's workbench application will discard all its current modifications and load your modifications. The other user has to do likewise to share his/her modifications. This requires clear communication between users before any modification is pushed to the shared state.

4. Setup and Maintenance Manual

4.1 Project Setup

The project is built for the Windows 7 operating system on a 32-bit system. The hardware required include:

- HiBall-3000 Wide-Area Tracker for head-tracking; the Nvidia 3D Vision bundle and two compatible monitors for stereo
- the TNG-3B Serial Interface for manipulating stereo separation
- 3Dconnexion SpaceNavigator to provide "world-in-hand" data set manipulation
- optionally, the Phantom Omni Device for Vortex Visualization (see "3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization").

Note: In the original system, the display consists of two Samsung SyncMaster 2233 LCD monitors that are Nvidia 3D Vision-compatible. The Nvidia Quadro cards are chosen over Geforce cards since only Quadro cards provide stereo for OpenGL applications and many scientific visualization software products are written in OpenGL (Nvidia Corporation, 2009).

Each monitor is connected via a dual-linked DVI cable to an Nvidia Quadro 5000 card. Because Quadro cards only have one DVI output port, it takes two cards to support the dual-display. The two cards are connected with an Nvidia SLI Connector, with the "SLI Mosaic Mode" turned on to enable stereo on both displays at the same time.

4.1.1 Libraries

Install and compile the following libraries:

1. Microsoft Visual Studio 2008 Professional Edition (available free-of-charge for students at https://www.dreamspark.com/)
2. 32-bit Qt – compile from source using the Microsoft Visual Studio 2008 previously installed. Instructions can be found here: http://dcsoft.wordpress.com/2010/01/30/how-to-setup-qt-4-5-visual-studio-integration/
3. Microsoft DirectX SDK 32-bit: http://www.microsoft.com/downloads/en/details.aspx?displaylang=en&FamilyID=3021d52b-514e-41d3-ad02-438a3ba730ba
4. Python 2.7.1 - install from binaries: http://python.org/ftp/python/2.7.1/python-2.7.1.msi
5. CMake 2.8.4 - install from binaries: http://www.cmake.org/cmake/resources/software.html
6. Git for Windows (select "Full installer for official Git 1.7.4"): http://code.google.com/p/msysgit/downloads/list
    a. Install Git with the following options:
        i. **Adjusting your PATH environment** - Use Git Bash only.
        ii. **Configuring the line ending conversions** - Checkout as-is, commit as-is.
7. VRPN (download vrpn_07_08.zip, refer to "4.1.1.1 Setup VRPN" for installation and compilation instructions): ftp://ftp.cs.unc.edu/pub/packages/GRIP/vrpn/

4.1.1.1 Setup VRPN:

1. Start CMake:
    a. Windows Desktop Start Menu ➔ All Programs ➔CMake 2.8➔CMake (cmake-gui)
    b. "Where is the source code:"

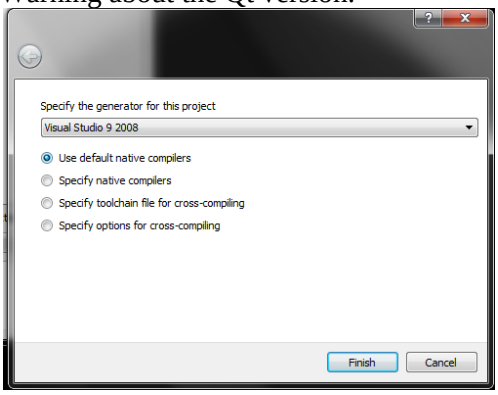
i. Specify the location of the VRPN directory. E.g. C:/Users/alexisc/Documents/EVE/vrpn

c. "Where to build the binaries"

i. Specify the location to store the compiled VRPN project. This should **not** be within the source code directory. E.g. C:/Users/alexisc/Documents/EVE/CompiledVRPN

d. Check "Group","Advanced". Click "Configure". Click "Yes" when prompted to create a new build directory.

e. In the "Specify the generator for this project" pop-up panel, select "Visual Studio 9 2008" from the drop-down selection menu. Click on "Finish". Ignore any CMake Warning about the Qt version.



f. Click "Generate".

2. Start Microsoft Visual Studio 2008

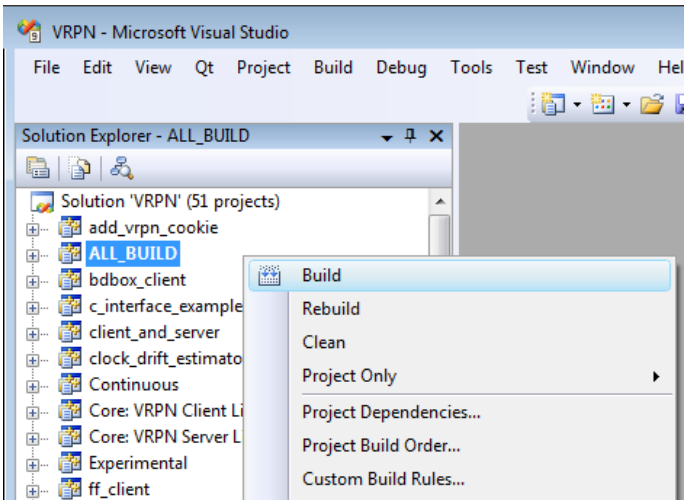
a. Windows Desktop Start Menu ➔ All Programs ➔Microsoft Visual Studio 2008➔ Microsoft Visual Studio 2008

b. Click on the "File" menu item in the top left corner of the application. Click "Open"➔ "Project/Solution"

c. In the "Open Project" dialog, navigate to the location of the compiled VRPN project. E.g. C:/Users/alexisc/Documents/EVE/CompiledVRPN/ . Click the solution file, i.e. VRPN.sln. Click "Open".

d. Click on the "Build" menu item. Click "Configuration Manager". Change "Debug" to "Release" in the drop-down selection menu under "Active solution configuration".

e. In the "Solution Explorer" panel on the left, right click on "ALL_BUILD". Click on "Build".



4.1.2 External devices

4.1.2.1 HiBall-3000 Wide-Area Tracker

This system requires the installation and configuration of the HiBall-3000 Wide-Area Tracker to enable head-tracking. The installation and configuration steps can be found in Chapters 2, 3 and 4 of the *HiBall-3000 Wide-Area Tracker User Manual* (3rdTech, Inc., 2002). The original system uses the Wide-Area Tracker installed in the Effective Virtual Environments lab of University of North Carolina at Chapel Hill.

Refer to *Chapter 6.2 Customizing the HiBall Server* of the *HiBall-3000 Wide-Area Tracker User Manual* for description of the required configuration files. Two 3rdTech HiBalls should each be mounted on headbands that will be worn by the user. In the Tracker Configuration File, make sure that there are two 'tether' lines referring to the HiBall Configuration Files of the headband-mounted HiBalls.

The configuration files that were used in the original system are included in the ParaView/Plugins/VRPN/ directory for reference purposes only. These files are: hiball3008.cfg, hiball3012.cfg, tracker3008+3012.cfg, vrpn3008+3012.cfg.

4.1.2.2 Nvidia 3D Vision

Refer to the *Hardware and Drivers* section of http://cismm.cs.unc.edu/core-projects/visualization-and-analysis/setting-up-a-simple-stereo-system/ for how to install and test the Nvidia 3D Vision setup.

The steps in the subsection, *Setting up Stereo on Multiple Displays***,** are required for this system.

4.1.2.3 3Dconnexion SpaceNavigator

To set up the 3Dconnexion SpaceNavigator, plug in the SpaceNavigator's USB cable to a USB port on the computer. The SpaceNavigator driver should then auto-install on the computer.

4.1.2.4 TNG-3B Serial Interface

To setup the TNG-3B Serial Interface:

1. Connect a Rotary Potentiometer to the TNG3-B Serial Interface's Analog Input 1 and Analog Input 2 (Refer to *TNG-3B Sensors* for a description of Rotary Potentiometers) (SenSyr LLC, 2009).
2. Connect the TNG-3B Serial Interface to the computer via a RS-232 serial port 9-pin connector.

4.1.2.5 Phantom Omni Device

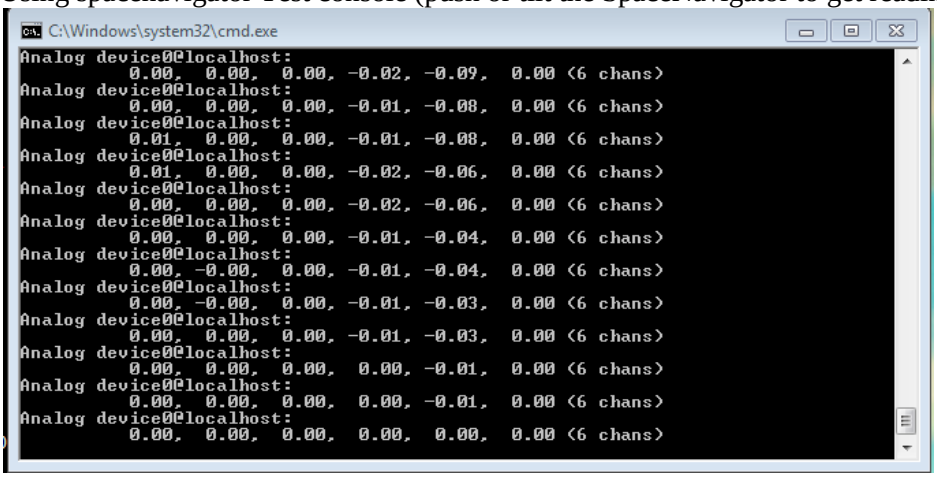
Install the Phantom Device Driver (PDD) and OpenHaptics Toolkit:

1. Visit http://www.sensable.com/products-openhaptics-toolkit.htm
2. Click on "Download the the Academic Edition of OpenHaptics® v3.0 with QuickHaptics™ for no charge today!" link and follow instructions for registration and download.
3. Unzip OpenHapticsAE_v3_0.zip.
4. Within the OpenHapticsAE_v3_0 folder, navigate to **Hardware Documentation**.
5. Open **PHNTMOmni_UserGuide.pdf**. In the **Installing the PHANTOM Device Drivers** section, skip steps 1-3. Instead, navigate to **OpenHapticsAE_v3_0/win32/PHANTOM Device Drivers**. Follow the steps 4-10 of **Installing the PHANTOM Device Drivers**.

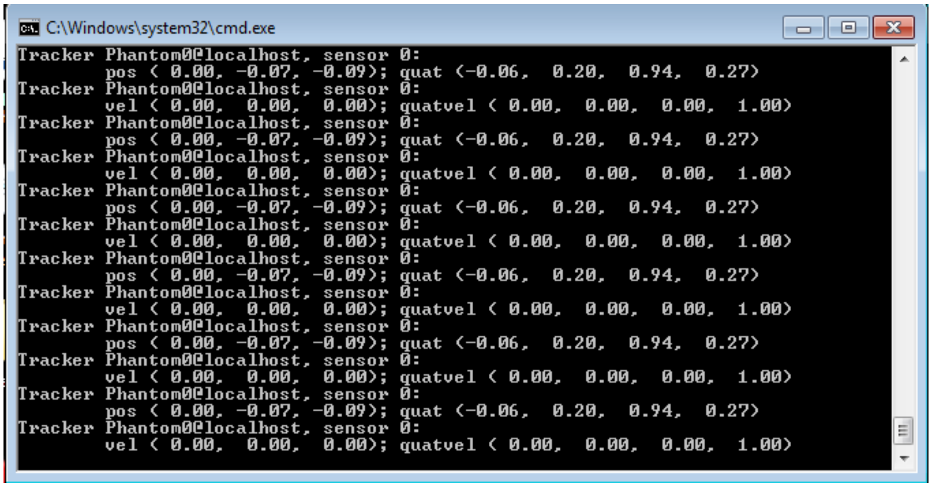
6. Follow the instructions in **Connecting the PHANTOM Omni Device** and **Run PHANTOM Test to Verify Setup** sections of the **PHNTMOmni_UserGuide.pdf**.
7. Navigate to **OpenHapticsAE_v3_0/win32/OpenHaptics Academic Edition**. Double-click on setup.exe to install the OpenHaptics Toolkit and follow the instructions given by the wizard.

4.1.2.6 Testing 3Dconnexion SpaceNavigator, Phantom Omni Device and TNG-3B Serial Interface:

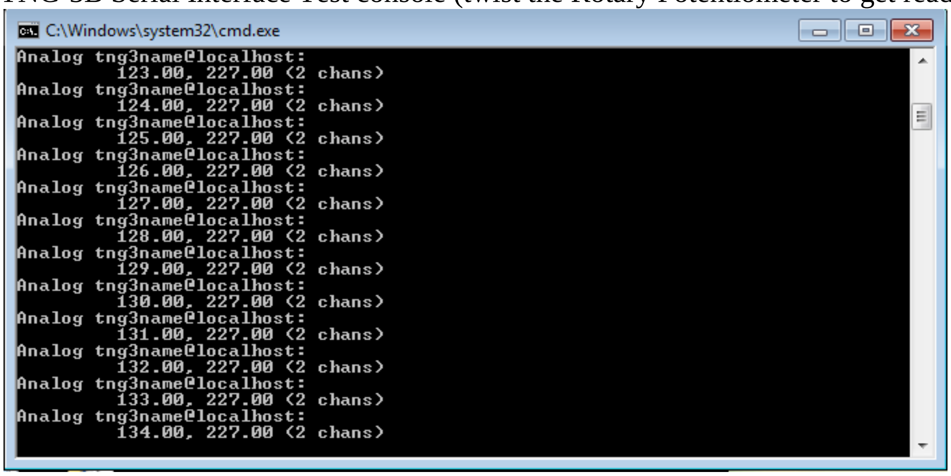
1. Navigate to the "server_src\Release" folder within the compiled VRPN project directory. E.g. C:\Users\alexisc\Documents\EVE\CompiledVRPN\server_src\Release
2. Download https://github.com/alexisylchan/ParaView/tree/simpleTwoParaView/Plugins/VRPN/vrpn.cfg to this directory.
3. Start the vrpn server by clicking on vrpn_server.exe in the directory.
4. Download https://github.com/alexisylchan/ParaView/blob/simpleTwoParaView/Plugins/VRPN/TestParaViewVRPNDevices.bat
5. Modify TestParaViewVRPNDevices.bat by following the instructions between the lines
   : README
   and
   :EndREADME
   (":" marks the start and end of block comments in the TestParaViewVRPNDevices.bat file)
6. The devices are running correctly on VRPN if you see these outputs on the
   a. Using spacenavigator Test console (push or tilt the SpaceNavigator to get readings):



b. Phantom Omni Device Test console:



c. TNG-3B Serial Interface Test console (twist the Rotary Potentiometer to get readings):



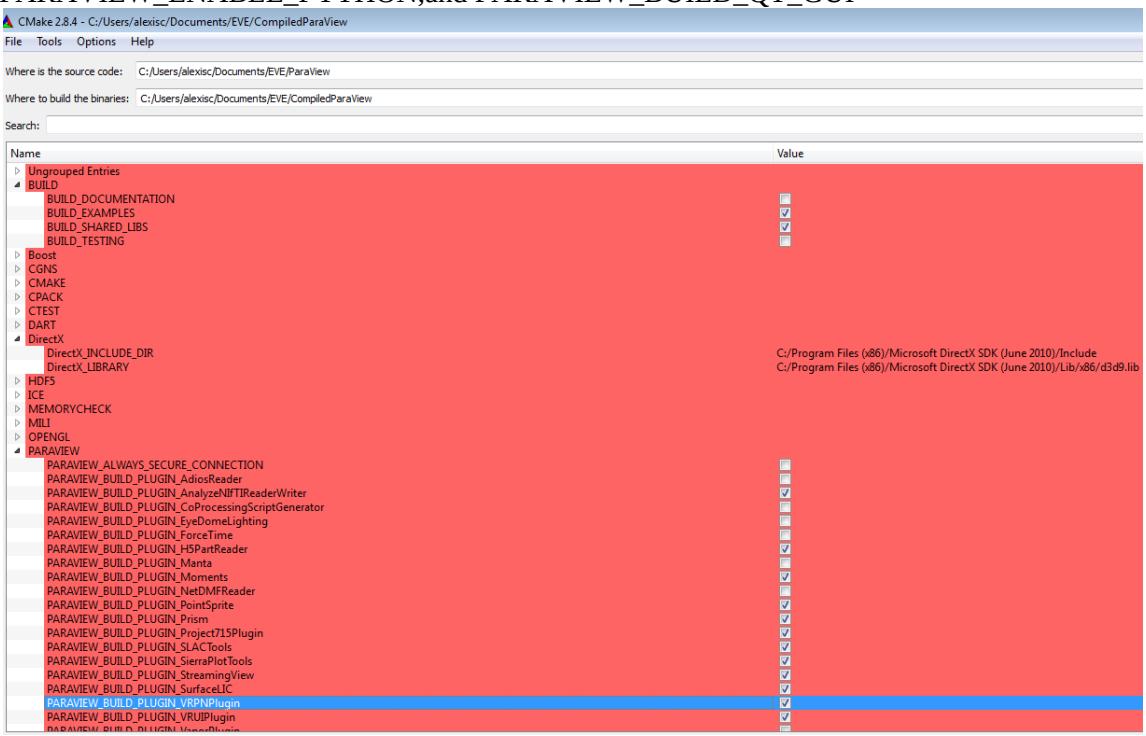
4.1.3 Obtaining the Collaborative Scientific Visualization Workbench

1. Start Git Bash:
   a. Windows Desktop Start Menu ➔ All Programs ➔ Git ➔ Git Bash
2. Type the following commands in the Git Bash window:
   a. cd <your folder path>
      i. E.g., I type "cd /c/Users/alexisc/Documents/EVE/" (without quotes) to store the project in "C:\Users\alexisc\Documents\EVE\"
   b. git clone git@github.com:alexisylchan/ParaView.git -b simpleTwoParaView
   c. cd ParaView
   d. git rm --cached Plugins/VRPN/vtkInteractionDevice
   e. rmdir Plugins/VRPN/vtkInteractionDevice
   f. git submodule add --b phantomManipulateObj git@github.com:alexisylchan/vtkInteractionDevice.git Plugins/VRPN/vtkInteractionDevice/
   g. git rm --cached VTK
   h. rmdir VTK
   i. git submodule add git@github.com:alexisylchan/VTK.git

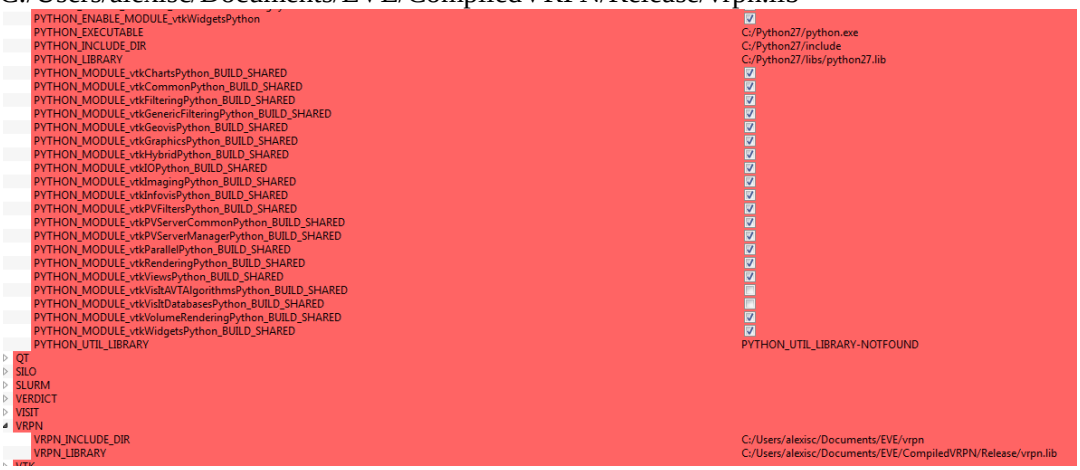
j. git submodule init
k. git submodule update

4.1.4 Setting up the Collaborative Scientific Visualization Workbench

1. Start CMake:
   a. Windows Desktop Start Menu ➔ All Programs ➔CMake 2.8➔CMake (cmake-gui)
   b. "Where is the source code:"
      i. Specify the location of the ParaView directory. E.g. C:/Users/alexisc/Documents/EVE/ParaView
   c. "Where to build the binaries"
      i. Specify the location to store the compiled project. This should **not** be within the source code directory. E.g. C:/Users/alexisc/Documents/EVE/CompiledParaView
   d. Check "Group","Advanced". Click "Configure". Click "Yes" when prompted to create a new build directory.
   e. In the "Specify the generator for this project" pop-up panel, select "Visual Studio 9 2008" from the drop-down selection menu. Click on "Finish". Ignore any CMake Warning about the Qt version.
   f. Check the following entries:
      i. BUILD➔select BUILD_SHARED_LIBS, BUILD_EXAMPLES and BUILD_TESTING
      ii. DirectX ➔DirectX_INCLUDE_DIR, type in the include folder path in your DirectX installation. E.g., C:/Program Files (x86)/Microsoft DirectX (June 2010)/Include
      iii. DirectX ➔DirectX_LIBRARY, the full path of d3d9.lib. E.g., C:/Program Files (x86)/Microsoft DirectX (June 2010)/Lib/x86/d3d9.lib
      iv. PARAVIEW➔select PARAVIEW_BUILD_PLUGIN_VRPNPlugin, PARAVIEW_ENABLE_PYTHON,and PARAVIEW_BUILD_QT_GUI



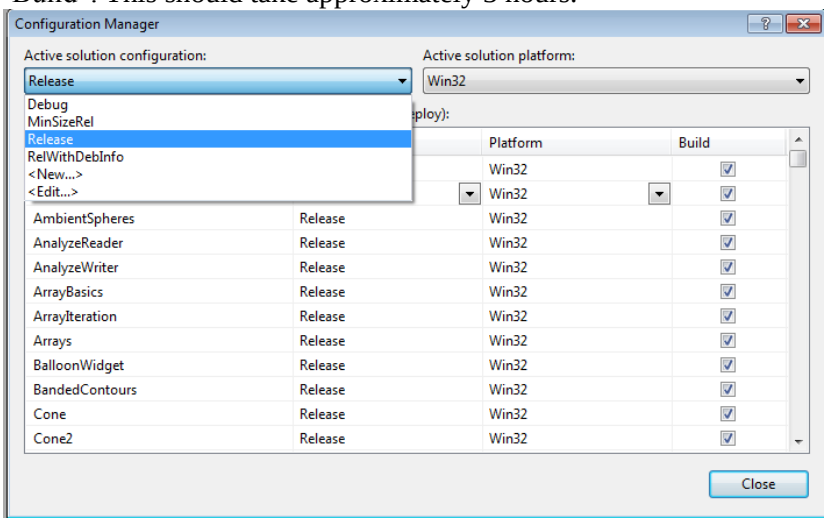
g. Click "Configure".
h. Check the following entries:
   i. PYTHON➔PYTHON_LIBRARY,the full path of the file python27.lib. E.g., C:/Python27/libs/python27.lib
   ii. PYTHON➔PYTHON_INCLUDE_DIR, the folder path in the python installation. . E.g., C:/Python27/include
   iii. PYTHON➔ PYTHON_EXECUTABLE (This may be under "Ungrouped Entries"), full pathname of python.exe, for example,C:/Python27/python.exe
   i. VRPN➔VRPN_INCLUDE_DIR, the location of the VRPN directory. E.g. C:/Users/alexisc/Documents/EVE/vrpn
   ii. VRPN➔VRPN_LIBRARY, the location of the file vrpn.lib. E.g., C:/Users/alexisc/Documents/EVE/CompiledVRPN/Release/vrpn.lib



   i. Click "Configure" again and CMake will process everything a second time. This time you should not see any items in red.
   j. Click "Generate".

2. Start Microsoft Visual Studio 2008
   a. Windows Desktop Start Menu ➔ All Programs ➔Microsoft Visual Studio 2008➔ Microsoft Visual Studio 2008
   b. Click on the "File" menu item in the top left corner of the application. Click "Open"➔ "Project/Solution"
   c. In the "Open Project" dialog, navigate to the location of the compiled project. E.g. C:/Users/alexisc/Documents/EVE/CompiledParaView/ . Click the solution file, i.e. ParaView.sln. Click "Open".
   d. Click on the "Build" menu item. Click "Configuration Manager". Change "Debug" to "Release" in the drop-down selection menu under "Active solution configuration:".

e. In the "Solution Explorer" panel on the left, right click on "ALL_BUILD". Click on "Build". This should take approximately 3 hours.



3. Modify Program startup batch file
   a. In Windows Explorer, navigate to the location of the ParaView directory. E.g. C:\Users\alexisc\Documents\EVE\ParaView\
   b. Click on Examples➔CustomApplications➔Clone1
   c. Right-click ParaViewR0.bat. Click "Edit"
   d. Edit the file based on the instructions between the lines (":" marks the start and end of block comments in the ParaViewR0.bat file)
      : README
      and
      :EndREADME
   e. Repeat steps 3c and 3d for ParaViewR1.bat (The directories that you use to modify the tags within the ParaViewR0.bat and ParaViewR1.bat files should be the same between both files).

4.2 Software Implementation

4.2.1 ParaView Custom Application

The Collaborative Scientific Visualization Workbench consists of two instances of a Custom ParaView Application written based on guidelines in http://www.vtk.org/Wiki/Writing_Custom_Applications and extended from the Clone1 example in ParaView 3.11.2. Each application consists of a Graphical User Interface window and a View window (See Figures 8 and 9).

4.2.1.1 Graphical User Interface Window

The Graphical User Interface (GUI) is implemented by myMainWindow which extends QMainWindow from the Qt library.

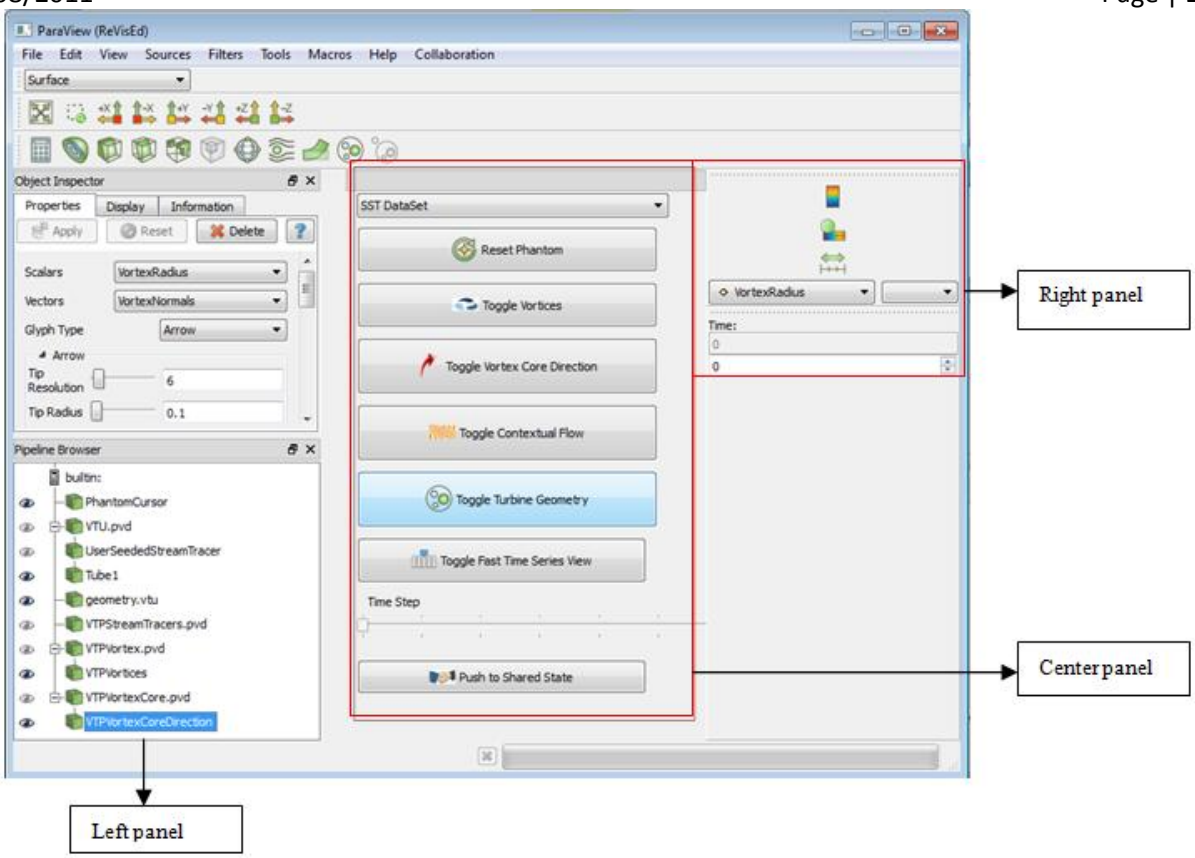


Figure 9 (reproduced here for easy lookup) : Left, Center and Right panels of the Graphical User Interface window

4.2.1.1.1 Menus and Toolbars

The File, Edit, Sources, Filters and Tools menus and all toolbars in ParaView are enabled using pqParaViewMenuBuilders.
The "Active Variables Control" toolbar (which consists of the Color Map menu items) and the Time Step Index menu item's positions were dragged to the Right Panel position (see Figure 9) to bring them to the user's attention for the Vortex Visualization Workbench. Refer to "3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization" for a description of the Vortex Visualization Workbench. The "Common" menu toolbar is retained at the top of the UI in case the user would like to apply common ParaView filters to the data set. Other toolbars were hidden from the user's view because the system lacks robust testing to ensure that the toolbars' functionalities are not impacted at all by the customizations.

4.2.1.1.2 Panels

The Object Inspector and Pipeline Browser panels are retained from the Clone1 example. These UI panels are child elements of a QDockWidget docked in the left "panel" shown in Figure 9.
The other ParaView panels originally shown in the Clone1 example are removed from the Qt UI XML file (for example the Selection Inspector and Animation View panels) . See http://www.digitalfanatics.org/projects/qt_tutorial/chapter09.html for a practical introduction to XML editing for Qt applications. Alternatively, you can download a WYSIWYG (what you see is what you get) Qt Microsoft Visual Studio Add-In designer from http://qt.nokia.com/downloads/visual-studio-add-in. Refer to http://doc.qt.nokia.com/vs-add-in-1.1.7/index.html for a tutorial on how to use the add-in.

The center "panel" shown in Figure 9 is a QDockWidget docked in the right area of the application. (QDockWidgets cannot be docked in the center area of a QMainWindow. To form the right "panel" that is situated to the right of this center "panel", a vertical QToolBar is placed to the right of this QDockWidget. See "4.2.1.1.1 Menus and Toolbars")

4.2.1.1.3 GUI Buttons for the Vortex Visualization Workbench

As ParaView is written based on Qt, the UI buttons in the center "panel" are implemented using Qt signals and slots. Refer to http://doc.qt.nokia.com/4.7/signalsandslots.html for an introduction to Qt signals and slots.

The Qt UI buttons raise Qt signals which are connected to myMainWindow class' Qt slots in myMainWindow.cxx. The UI button actions are implemented in these Qt slots.
The following example sets the icon for the Qt GUI Button "Toggle Vortex Core" and connects the button's Qt signal to myMainWindow's Qt slot:

```
//Set Button Icon Size and Image Source
this->Internals->ToggleVortexCore->setIconSize(QSize(24,24));
this->Internals->ToggleVortexCore->
setIcon(QIcon("C:/Users/alexisc/Documents/EVE/ParaView/Qt/Components/Resources/Icons
/vortexcore.png"));
//Connect Button's Qt Signal to myMainWindow's Qt Slot
QObject::connect(this->Internals->
ToggleVortexCore,SIGNAL(clicked()),this,SLOT(vortexIdentification()));
```

All the UI Buttons except for the "Pushed to Shared State" button are used to toggle the visibility of a component of the centrifugal pump data set.

4.2.1.1.4 Toggling Data Set Visibility

Refer to "4.2.2.5 Creating/Editing/Deleting ParaView Objects" for how to obtain the pqDataRepresentation object used in this section.
The following code is used to make a ParaView object visible:

```
pqDataRepresentation* inputRepr = source->getRepresentation(view);
        if (inputRepr)
          {
                        inputRepr->setVisible(true);
        }
        vtkSMRenderViewProxy *proxy = vtkSMRenderViewProxy::SafeDownCast( view-
>getViewProxy() );
        proxy->GetRenderWindow()->Render();
```

The following code is used to make a ParaView object invisible:

```
pqDataRepresentation* inputRepr = source->getRepresentation(view);
        if (inputRepr)
          {
                        inputRepr->setVisible(false);
        }
        vtkSMRenderViewProxy *proxy = vtkSMRenderViewProxy::SafeDownCast( view-
>getViewProxy() );
        proxy->GetRenderWindow()->Render();
```

4.2.1.1.5 Saving To Shared State

When the "Push to Shared State button is pressed, the ParaView state of the application is saved into a pre-defined state file. Refer to "3.2.3.2 Push to Shared State" for a description of the state-sharing mechanism.

The following code for saving into the state file is adapted from pqSaveStateReaction::saveState(const QString& filename).

```
QString filename  =
QString("C:/Users/alexisc/Documents/EVE/CompiledParaView/bin/Release/StateFiles/1.pvsm");
  pqApplicationCore::instance()->saveState(filename);
  pqServer *server = pqActiveObjects::instance().activeServer();
  // Add this to the list of recent server resources ...
  pqServerResource resource;
  resource.setScheme("session");
  resource.setPath(filename);
  resource.setSessionServer(server->getResource());
  pqApplicationCore::instance()->serverResources().add(resource);
  pqApplicationCore::instance()->serverResources().save(
    *pqApplicationCore::instance()->settings());
```

4.2.1.1.6 Data-Set-Switching Pull Down Menu

Refer to *3.2.2 ParaView Sources and State Files* of the *Visualization of Flow Instability in Centrifugal Pumps document* for context (Lee, 2011).

The data-set-switching pull down menu is implemented using QComboBox.

- The Qt signal raised by the QComboBox is connected to a myMainWindow class' Qt slot, which raises another Qt signal.
- The Qt signal raised by myMainWindow is connected to a Qt slot in the VRPN Plugin's pqVRPNStarter.
    - The data-set-switching implementation has to happen within the VPRN Plugin (i.e. pqVRPNStarter) instead of the Custom Application (i.e. myMainWindow) because it requires reloading the state files and resetting the application, which in turn, requires reinitializing the VRPN devices managed by the VRPN Plugin.
    - For a description of the implementation within the pqVRPNStarter's Qt slot refer to "4.2.2.7 Switching Data Sets for Vortex Visualization".

4.2.1.1.7 Time Slider

The Time Slider in the right "panel", which enables interactive exploration of data set across multiple time steps, is implemented using Qslider.

The Qt signal raised by the QSlider is connected to a myMainWindow class' Qt slot, which raises another Qt signal containing the slider's time index. That Qt signal is connected to pqAnimationScene::setAnimationTime, which takes the slider's time index as an input.

This is the implementation within myMainWindow's initialization code:

```
//Connect QSlider's Qt signal to myMainWindow's Qt slot
QObject::connect(Slider0, SIGNAL(valueChanged(int)),this,
       SLOT(sliderTimeIndexChanged(int)));

//Get the active scene
QPointer<pqAnimationScene> Scene =  pqPVApplicationCore::instance()->
       animationManager()->getActiveScene();
//Connect myMainWindow's Qt signal to pqAnimationScene's Qt slot
```

```
QObject::connect(this, SIGNAL(changeSceneTime(double)),
                 Scene, SLOT(setAnimationTime(double)));
```

The slider's time index range has to be updated whenever the time index range of the Animation Scene is changed:

```
QObject::connect(Scene, SIGNAL(timeStepsChanged()),
      this, SLOT(onTimeStepsChanged()));
```

This is how the slider's time index is sent to the pqAnimationScene Qt slot:

```
if (pqPVApplicationCore::instance()->animationManager()->getActiveScene())
   {
      pqTimeKeeper* timekeeper = pqPVApplicationCore::instance()->animationManager()->
            getActiveScene()->getServer()->getTimeKeeper();
      emit this->changeSceneTime(
      timekeeper->getTimeStepValue(value));
   }
```

4.2.1.2 View Window

The View window is extracted from the GUI window and set to full screen in order to maximize the users' field of view.

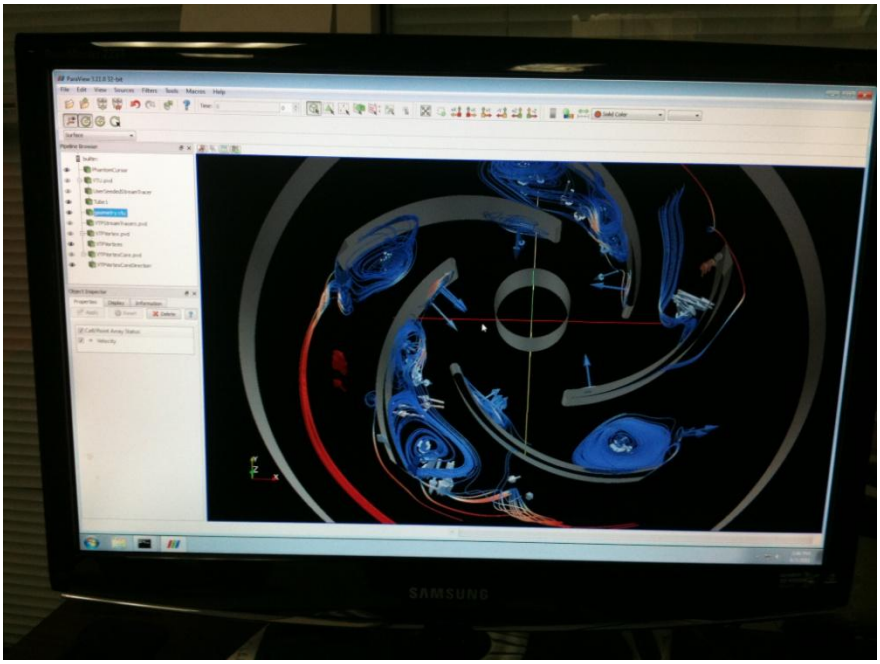


Figure 15: Original ParaView. The View Window is embedded in the GUI Window

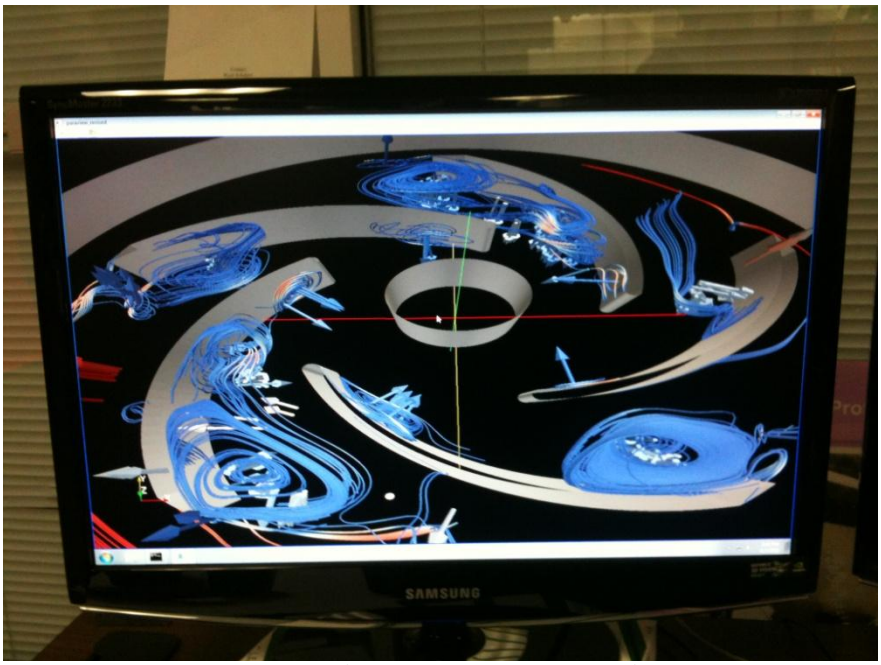


Figure 16: The View Window extracted from the GUI Window and set to full screen upon initialization

This is done by extracting the pqViewManager Qt custom widget from the QMainWindow's central widget and wrapping with with a QDockWidget set to "docked".
Original xml entry:

```
<widget class="QMainWindow" name="pqClientMainWindow">
  …
  <widget class="QWidget" name="centralwidget">
   …
     <widget class="pqViewManager" name="MultiViewManager" native="true"/>
…
  </widget>
```

Current xml entry:

```
<widget class="QMainWindow" name="pqClientMainWindow">
  …
  <widget class="QDockWidget" name="proxyTabDock1">
   <property name="floating">
    <bool>true</bool>
   </property>
   <set>QDockWidget::DockWidgetFloatable|QDockWidget::DockWidgetMovable</set>

   <widget class="pqViewManager" name="MultiViewManager">   …
…
  </widget>
```

To overwrite the ParaView behavior, pqMultiViewFrame, pqViewManager and pqMultiView classes are edited to disable view splitting and view maximization or minimization. Refer to http://www.paraview.org/ParaQ/Doc/Nightly/html/classpqMultiView.html for a description of relationship between the classes.

(Ideally, new classes extending the original ParaView classes should have been written and integrated into the project using steps documented at http://www.vtk.org/Wiki/Extending_ParaView_at_Compile_Time).

The Qt UI buttons for splitting the view into two views (vertically or horizontally), maximizing the view, and closing the view are removed.

Figure 17: Original ParaView. The Qt UI buttons are marked with the red box. From left to right: button for splitting the view vertically; button for splitting the view horizontally;button for maximizing the view; button for closing the view.

Figure 18: The Collaborative Scientific Visualization Workbench's View Window. The Qt UI buttons are removed (see red box).

In pqMultiView, the code for connecting pqMultiViewFrame's Qt signals to pqMultiView's Qt slots are commented out:

```
//QSignalMapper* CloseSignalMapper = new QSignalMapper(frame);
  //QSignalMapper* HorizontalSignalMapper = new QSignalMapper(frame);
  //QSignalMapper* VerticalSignalMapper = new QSignalMapper(frame);
  //QSignalMapper* MaximizeSignalMapper = new QSignalMapper(frame);
  //QSignalMapper* RestoreSignalMapper = new QSignalMapper(frame);

  //CloseSignalMapper->setMapping(frame, frame);
  //HorizontalSignalMapper->setMapping(frame, frame);
  //VerticalSignalMapper->setMapping(frame, frame);
  //MaximizeSignalMapper->setMapping(frame, frame);
  //RestoreSignalMapper->setMapping(frame, frame);

  //// connect close button
  //QObject::connect(frame, SIGNAL(closePressed()),
  //                 CloseSignalMapper, SLOT(map()));
  //QObject::connect(CloseSignalMapper, SIGNAL(mapped(QWidget*)),
  //                 this, SLOT(removeWidget(QWidget*)), Qt::QueuedConnection);

  //// connect split buttons
  //QObject::connect(frame, SIGNAL(splitHorizontalPressed()),
  //                 HorizontalSignalMapper, SLOT(map()));
  //QObject::connect(HorizontalSignalMapper, SIGNAL(mapped(QWidget*)),
  //                 this, SLOT(splitWidgetHorizontal(QWidget*)));
  //
  //QObject::connect(frame, SIGNAL(splitVerticalPressed()),
  //                 VerticalSignalMapper, SLOT(map()));
  //QObject::connect(VerticalSignalMapper, SIGNAL(mapped(QWidget*)),
  //                 this, SLOT(splitWidgetVertical(QWidget*)));
  //
  //QObject::connect(frame, SIGNAL(maximizePressed()),
  //                 MaximizeSignalMapper, SLOT(map()));
  //QObject::connect(MaximizeSignalMapper, SIGNAL(mapped(QWidget*)),
  //                 this, SLOT(maximizeWidget(QWidget*)));

  //QObject::connect(frame, SIGNAL(restorePressed()),
  //                 RestoreSignalMapper, SLOT(map()));
  //QObject::connect(RestoreSignalMapper, SIGNAL(mapped(QWidget*)),
  //                 this, SLOT(restoreWidget(QWidget*)));

  //// Connect decorations signals.
  //QObject::connect(this, SIGNAL(hideFrameDecorations()),
  //  frame, SLOT(hideDecorations()));
  //QObject::connect(this, SIGNAL(showFrameDecorations()),
  //  frame, SLOT(showDecorations()));
```

In pqMultiViewFrame, the code connecting the Qt UI Button's signals to pqMultiView are commented out:

```
//this->connect(this->ActiveButton->defaultAction(), SIGNAL(triggered(bool)),
  //              SLOT(setActive(bool)));
  //this->connect(this->CloseButton->defaultAction(), SIGNAL(triggered(bool)),
  //              SLOT(close()), Qt::QueuedConnection);
  //this->connect(this->MaximizeButton->defaultAction(),
  //              SIGNAL(triggered(bool)),
  //              SLOT(maximize()), Qt::QueuedConnection);
  //this->connect(this->RestoreButton->defaultAction(),
  //              SIGNAL(triggered(bool)),
  //              SLOT(restore()), Qt::QueuedConnection);
  //this->connect(this->SplitVerticalButton->defaultAction(),
  //              SIGNAL(triggered(bool)),
  //              SLOT(splitVertical()), Qt::QueuedConnection);
  //this->connect(this->SplitHorizontalButton->defaultAction(),
  //              SIGNAL(triggered(bool)),
  //              SLOT(splitHorizontal()), Qt::QueuedConnection);
```

In pqViewManager, code for showing the Qt UI Buttons are commented out:

```
 /*frame->MaximizeButton->show();
  frame->CloseButton->show();
  frame->SplitVerticalButton->show();
  frame->SplitHorizontalButton->show();*/
```

In myMainWindow, the View Manager is set to full screen on default:

```
   this->Internals->MultiViewManager->toggleFullScreen();
   this->Internals->MultiViewManager->getFrame(this->Internals->MultiViewManager->
                                               getActiveView())->
                                               setMenuAutoHide(true);
```

4.2.1.3 Other Behaviors

The pqFixPathsInStateFilesBehavior::blockDialog is set to true to hide the “Fix File Path” dialog box that usually appears upon loading and reloading ParaView state files:

```
   pqFixPathsInStateFilesBehavior::blockDialog(true);
```

All other default ParaView behaviors are enabled using the pqParaViewBehaviors class.

4.2.2 ParaView Plugin

4.2.2.1 Input Options

Users can turn on each of the following feature independently:

- head-tracking
- "world-in-hand" data set manipulation
- stereo separation control
- Phantom stream tracer seeding (see “3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization”).

In the following sections, User1 refers to the user using ParaViewR0.bat and User2 refers to the user using ParaViewR1.bat. These files are included in the “Appendix” and located in ParaView/Examples/CustomApplications/Clone1 in the source code tree.

4.2.2.2 Viewpoint Setup

vtkCamera's capability to provide head-tracked perspective for Cave Automatic Virtual Environments (CAVE) is capitalized upon to set up the two user viewpoints. This is implemented in pqVRPNStarter::initializeEyeAngle.

4.2.2.2.1 Enable Head-tracking

- Invoke vtkCamera::SetHeadTracked.
- Set Eye Angle to 0 to allow the user to gradually tune up the stereo separation. See "4.2.2.8 Stereo Separation Control using TNG-3B Serial Interface"

4.2.2.2.2 Define the Display Surface

- The surface orientation of the display is defined with the lower left coordinate, lower right coordinate and upper right coordinate of the screen in ParaView space. For an example of how the code is implemented for CAVEs, see vtkCaveSynchronizedRenderers:: vtkCaveSynchronizedRenderers, which defines the display screen coordinates in ParaView\cave.pvx
- In the ParaView VRPN Plugin of this system,
    - for User1, the following surface orientation is used:
        - lower left coordinate: -1, -1, 1
        - lower right coordinate: -1, -1, -1
        - upper right coordinate: -1, 1, -1
        - This is the surface orientation commonly used for the front wall of the CAVE. Therefore, the display surface (and projection plane) for User1 is parallel to the projection plane of the actual vtkCamera.
    - for User2, the following surface orientation is used:
        - lower left coordinate: -1, -1, -1
        - lower right coordinate: 1, -1, -1
        - upper right coordinate: 1, 1, -1
        - This is the surface orientation commonly used for the left wall of the CAVE. Therefore, the vtkCamera's projection plane has to be rotate 90 degrees counter-clockwise to form the display surface (and projection plane) for User2.

4.2.2.2.3 Set the Display Configuration

- The surface orientation coordinates from "4.2.2.2.3 Define the Display Surface" are used to compute the parameters required by vtkCamera::SetConfigParams. The code for this is copied directly from vtkCaveSynchronizedRenderers::SetDisplayConfig
- The parameters are used in vtkCamera to compute head-tracked perspective-correct viewpoints and view frustums.

4.2.2.3 Updating VRPN Devices with QTimer

Within a QTimer callback, input data is collected from the following VRPN devices. After the collection, the visualization of the data set is updated with the new information.

4.2.2.4 Rendering in ParaView

To issue a render call to ParaView:

- Obtain the Server Manager Model:

```
pqServerManagerModel* serverManager =
        pqApplicationCore::instance()->getServerManagerModel();
```

- Loop through View objects; obtain each View object's View Proxy's Render Window. Invoke the Render Window's render command.

```
pqView* view = serverManager->getItemAtIndex<pqView*>(i);
vtkSMRenderViewProxy *proxy =
        vtkSMRenderViewProxy::SafeDownCast( view->getViewProxy() );
proxy->GetRenderWindow()->Render();
```

4.2.2.5 Creating/Editing/Deleting ParaView Objects

4.2.2.5.1 Editing An Object

- Refer to "3.1 Using ParaView" for explanation of ParaView Reader, Source and Filter objects.
    - **Representation**
        - To modify properties that affect how an object (data set reader, source or filter) is rendered, e.g. "position", "orientation", "scale", etc., edit the Representation (i.e. pqDataRepresentation).
        - This is shown in the following examples.
        - Modifications done to the Representation is only done to the corresponding view window representation of the object.
        - Refer to ParaView\Servers\ServerManager\Resources\views_and_representations.xml for each Representation's list of properties.
    - **Source**
        - To modify properties that affect the geometry of a source, e.g. "radius", "length", (geometrical mesh) "resolution", etc., edit the Pipeline Source.
        - Replace pqDataRepresentation in the following examples with pqPipelineSource.
        - Refer to ParaView\Servers\ServerManager\Resources\sources.xml for each Source's list of properties.
    - **Filter**
        - To modify properties that affect the geometry of a filter, e.g. "radius", "length",etc., or the properties determining the source-to-filter conversion, e.g. "input vector", "random sampling", "scale factor", etc., edit the Pipeline Filter.
        - Replace pqDataRepresentation in the following examples with pqPipelineFilter.
        - Refer to ParaView\Servers\ServerManager\Resources\filters.xml for each Filter's list of properties.
    - **Color, Opacity, Color Map Scale**
        - pqPipelineRepresentation provides several methods for changing the variable by which a Reader, Source or Filter can be colored by, the type of color map used, the opacity of the rendering, etc.
- Obtain the ParaView Representation

```
pqDataRepresentation *object = pqApplicationCore::instance()->
    getServerManagerModel()->
    getItemAtIndex<pqDataRepresentation*>(object_index);
```

    - Note: this method only works if you know the order in which objects are created in the ParaView pipeline. It is not recommended in cases where the user is allowed to delete all objects and recreated objects in different orders.

- Here is another method that might be less performant, with lower chances of obtaining the wrong Representation, but requires knowledge of the name of the object in the ParaView pipeline. The fact that ParaView names the objects by the object type and order in which it is created makes this fairly reliable. For example, a Tube filter will be named "Tube1". (If the user modifies the name of the object in the ParaView Pipeline Browser upon object creation this will not work) :

```
pqDataRepresentation* object = pqApplicationCore::instance()->
    getServerManagerModel()->findItem<pqDataRepresentation*>("Name1");
```

- Obtain the Representation Proxy

```
vtkSMPVRepresentationProxy *repProxy = 0;
repProxy = vtkSMPVRepresentationProxy::SafeDownCast(object->getProxy());
```

- Edit Representation Proxy's property. (In this example, Position is used).

```
vtkSMPropertyHelper(repProxy,"Position").Set(newPosition,3);
```

- Issue Update VTK Objects

```
repProxy->UpdateVTKObjects();
```

4.2.2.5.2 Creating An Object

- **Creating a Reader**
  - This is not used in the current ParaView VRPN Plugin, but included for general knowledge.
  - Create vtkXMLReader class (or one of its subclasses)
  - Set the data set file name.
  - Invoke vtkXMLReader::Update
- **Creating a Source**
  - To create a custom source, sources.xml is edited directly (search for PhantomCursorSource in the file).
  - Ideally, this should be inserted in a separate xml file located in the Plugin directory.
  - See http://www.itk.org/Wiki/ParaView/Plugin_HowTo#Adding_a_Reader . This would require modifications to the CMakeLists.txt file and regeneration of the project via CMake. Refer to "4.1.4 Setting up the project".
  - Invoke ParaView Object Builder's CreateSource method.
  - In the following example, "sources" is the ProxyGroup value in the xml file; "PhantomCursorSource" is the SourceProxy name value.

```
pqApplicationCore* core = pqApplicationCore::instance();
pqServerManagerModel* serverManager = core->getServerManagerModel();
pqPipelineSource* pipelineSource =
    core->getObjectBuilder()->createSource("sources",
    "PhantomCursorSource",pqActiveObjects::instance().activeServer());
```

  - Display the source in all views
    - Loop through View objects (See “4.2.2.5 Rendering in ParaView”)
    - For each View, obtain the ParaView Display Policy (refer to the pqDisplayPolicy ParaView class).

```
pqView* view = serverManager->getItemAtIndex<pqView*>(i);
vtkSMRenderViewProxy *proxy =
    vtkSMRenderViewProxy::SafeDownCast( view->getViewProxy() );
```

```
pqDisplayPolicy* displayPolicy =
pqApplicationCore::instance()->getDisplayPolicy();
```

- For each output port of the Source, create the preferred representation as per the current Display Policy.
  - (This code is copied directly from pqObjectInspectorWidget::show)

```
pqDisplayPolicy* displayPolicy =
pqApplicationCore::instance()->getDisplayPolicy();
for (int cc=0; cc < pipelineSource->getNumberOfOutputPorts(); cc++)
{
    pqDataRepresentation* repr =
    displayPolicy->createPreferredRepresentation(
        pipelineSource->getOutputPort(cc), view, false);
    if (!repr || !repr->getView())
    {
        continue;
    }
    pqView* cur_view = repr->getView();
    pqPipelineFilter* filter =
qobject_cast<pqPipelineFilter*>(pipelineSource);
    if (filter)
    {
        filter->hideInputIfRequired(cur_view);
    }
}
```

- **Creating a Filter**
  - To modify the default properties upon filter creation, filters.xml is edited directly. Ideally a new filter in a separate xml file located in the Plugin directory should be created. See http://www.itk.org/Wiki/ParaView/Plugin_HowTo#Adding_a_Filter .This would require modifications to the CMakeLists.txt file and regeneration of the project via CMake. Refer to "4.1.4 Setting up the project". The following code is modified from pqFiltersMenuReaction::createFilter.
  - Create the filter proxy from a ParaView prototype. "filters" is the ProxyGroup value in the xml file; "TubeFilter" is the SourceProxy name value.

```
vtkSMProxyManager* pxm = vtkSMProxyManager::GetProxyManager();
vtkSMProxy* prototype = pxm->GetPrototypeProxy("filters","TubeFilter");
```

  - Obtain the output port of the Source or Filter that will be processed by the new filter.

```
pqPipelineSource* item =
    pqApplicationCore::instance()->getServerManagerModel()->
    getItemAtIndex<pqPipelineSource*>(sourceIndex);
pqOutputPort* opPort = qobject_cast<pqOutputPort*>(item);
pqPipelineSource* source = qobject_cast<pqPipelineSource*>(item);
if (source)
{
    outputPorts.push_back(source->getOutputPort(0));
}
else if (opPort)
{
    outputPorts.push_back(opPort);
}
```

  - Assign the output port to the input port of the newly created Prototype Filter Proxy. This code from pqFiltersMenuReaction::createFilter creates a dialog if the proxy contains more than 1 input port.

```
QMap<QString, QList<pqOutputPort*> > namedInputs;
QList<const char*> inputPortNames =
    pqPipelineFilter::getInputPorts(prototype);
namedInputs[inputPortNames[0]] = outputPorts;

// If the filter has more than 1 input ports, we are simply going to ask the
// user to make selection for the inputs for each port. We may change that
// in future to be smarter.
if (pqPipelineFilter::getRequiredInputPorts(prototype).size() > 1)
 {
        vtkSMProxy* filterProxy = pxm->GetPrototypeProxy("filters",
                                                  name);
        vtkSMPropertyHelper helper(filterProxy, inputPortNames[0]);
        helper.RemoveAllValues();

        foreach (pqOutputPort *outputPort, outputPorts)
        {
           helper.Add(outputPort->getSource()->getProxy(),
                              outputPort->getPortNumber());
        }
   pqChangeInputDialog dialog(filterProxy, pqCoreUtilities::mainWidget());
        dialog.setObjectName("SelectInputDialog");
        if (QDialog::Accepted != dialog.exec())
        {
           helper.RemoveAllValues();
           // User aborted creation.
           return -1;
        }
        helper.RemoveAllValues();
        namedInputs = dialog.selectedInputs();
 }
```

- Invoke ParaView Object Builder's CreateFilter method. In the following example "filters" is the ProxyGroup value in the xml file; "TubeFilter" is the SourceProxy name value. (A filter can be cast as pqPipelineSource or pqPipelineFilter).

```
pqPipelineSource* createdSource = pqApplicationCore::instance()->
        getObjectBuilder()->createFilter("filters", "TubeFilter",
        namedInputs, pqActiveObjects::instance().activeServer());
```

Display the filter in all views. (Refer to "Creating a Source: Display the source in all views" in "4.2.2.5.2 Creating An Object").

4.2.2.5.3 Deleting An Object

- Obtain the object

```
pqPipelineSource* source = pqApplicationCore::instance()->
  getServerManagerModel()->findItem<pqPipelineSource*>("Name1");
```

- Invoke pqObjectBuilder::destroy

```
if (source)
{
        pqApplicationCore::instance()->getObjectBuilder()->
        destroy(source);
}
```

4.2.2.6 3Dconnexion SpaceNavigator and TNG-3B Serial Interface

These are tracked as vrpn_Analog_Remote classes from the VRPN library http://www.cs.unc.edu/Research/vrpn/vrpn_Analog_remote.html .The callback for these classes are implemented in the pqVRPNStarter.cxx, but are not part of the class.

4.2.2.7 "World-in-hand" Data Set Manipulation using SpaceNavigator

Callback functions:

- SNAugmentChannelsToRetainLargestMagnitude
  - Code adapted from AugmentChannelsToRetainLargestMagnitude callback in ParaViewVRPN.cxx in ParaView 11.8.2
  - Only the channel with the highest value is "read" from the SpaceNavigator.
- handleSpaceNavigatorPos
  - Code adapted from handleAnalogPos callback in ParaViewVRPN.cxx in ParaView 11.8.2
  - This code is written for a specific orientation of the SpaceNavigator. The SpaceNavigator USB wire should be pointing towards User1, i.e. to the left of User2. The SpaceNavigator should be positioned approximately halfway between the two users for better collaboration. This is not needed for the code to work correctly.
  - In the following description, "pushing" and "tilting" actions apply to the "joystick" part of the SpaceNavigator, not the entire SpaceNavigator. Refer to "3.2.2.2: SpaceNavigator" for illustration.
  - **Channel 0:**
    - Positive values correspond to pushing the SpaceNavigator to the right of User1, i.e. towards User2.
      - This is mapped to translating the vtkCamera position and focal point by a positive value along the cross product of the direction of projection and the up vector.
    - Negative values correspond to pushing the SpaceNavigator to the left of User1, i.e. away from User2.
      - This is mapped to translating the vtkCamera position and focal point by a negative value along the cross product of the direction of projection and the up vector.
  - **Channel 1:**
    - Positive values correspond to pushing the SpaceNavigator towards User1, i.e. to the left of User2.
      - This is mapped to translating the vtkCamera position away from the focal point, using vtkCamera::Dolly(pow(1.0,-1*channel_value)). The choice to scale the translation value to the power of 1 is taken from ParaViewVRPN.cxx in ParaView 11.8.2
    - Negative values correspond to pushing the SpaceNavigator away from User1, i.e. to the right of User2.
      - This is mapped to translating the vtkCamera position towards the focal point, using vtkCamera::Dolly(pow(1.0,-1*channel_value)).
  - **Channel 2:**
    - Positive values correspond to pulling the SpaceNavigator upwards.
      - This is mapped to translating the vtkCamera position and focal point by a positive value along the up vector.

- Negative values correspond to pushing the SpaceNavigator downwards.
    - This is mapped to translating the vtkCamera position and focal point by a negative value along the up vector.

- **Channel 3:**
    - Positive values correspond to tilting the SpaceNavigator away from User1, i.e. to the right of User2.
    - Negative values correspond to tilting the SpaceNavigator towards User1, i.e. to the left of User2.
    - The values are mapped to rotation about the cross product of the negative of the direction of projection and the view up vector, with the focal point as the center of rotation.
    - For User1, the axis vector is the first row of the vtkCamera::ViewTransform Matrix. The rotation for User1 is implemented by using vtkCamera::Elevation.
    - For User2, the axis vector is the third row of the vtkCamera::ViewTransform Matrix. The rotation for User2 is implemented by copying the code of vtkCamera::Elevation and adapting it to use the third row of vtkCamera::ViewTransform's matrix as the axis of rotation. This is necessary because User2's view projection plane is actually obtained by rotating the actual vtkCamera's projection plane(which corresponds to User1's view projection plane) by 90degrees counter-clockwise.(See "4.2.2.2 Viewpoint Setup").
    - For both users:
        - Positive values are mapped to clockwise rotation.
        - Negative values are mapped to counter-clockwise rotation.
- **Channel 4:**
    - Positive values correspond to tilting the SpaceNavigator to the right of User1, i.e. towards User2.
    - Negative values correspond to tilting the SpaceNavigator to the left of User1, i.e. away from User2.
    - These are mapped to rotation about the direction of projection using vtkCamera::Roll.
    - Positive values are mapped to clockwise rotation.
    - Negative values are mapped to counter-clockwise rotation.
- **Channel 5:**
    - Non-zero values correspond to rotating the SpaceNavigator around its up vector.
    - Clockwise rotation is mapped to clockwise rotation and counter-clockwise rotation is mapped to counter-clockwise rotation of the vtkCamera about the view up vector centered at the focal point using vtkCamera::Azimuth.

- Suggestions for improvement:
    - Instead of recording the value of the channel with the highest reading and discarding the rest, the system should read all channels and update the "world" translation and rotation with all the values.
    - For the translation values,map the direction of SpaceNavigator translation to direction of the vtkCamera position translation (do not translate focal point, do not invert direction for "Dolly").

4.2.2.8 Stereo Separation Control using TNG-3B Serial Interface

The TNG-3B's callback sets the stereo separation using vtkCamera:: SetEyeOffset.

- Upon TNG-3B device initialization, the initial reading is recorded in the struct that is passed to the TNG callback function.
- The difference between the current reading and initial reading is scaled by 0.001.
- The difference is added to vtkCamera::EyeOffset.

4.2.2.9 3rdTech Wide-Area Tracker and Sensable Phantom Omni Device

pqVRPNStarter uses the VTK Interaction Device library which was adapted from original created by David Borland https://github.com/davidborland/vtkInteractionDevice to map the readings from the Wide-Area Tracker and Phantom Omni Device to changes in the rendering of the scene in ParaView.

The VTK Interaction Device library classes used are:

- vtkDeviceInteractor:
  - manages a list of vtkInteractionDevice and their corresponding vtkInteractionDeviceStyle. pqVRPNStarter invokes vtkDeviceInteractor::Update during the QTimer callback which invokes
    - vtkInteractionDevice::Update
      - vtkInteractionDevice applies Transforms to the VRPN device readings in its record
    - vtkInteractionDevice::InvokeInteractionEvent
      - vtkInteractionDevice invokes a vtkCommand::UserEvent that is listened to by the vtkInteractionDeviceStyle.
      - vtkInteractionDeviceStyle implements the changes in the rendering of the scene in ParaView.
      - This decouples the VRPN device reading thread from the rendering thread.
- vtkVRPNTrackerCustomSensor:
  - extends vtkVRPNTracker which implements vtkInteractionDevice.
- vtkVRPNTrackerCustomSensorStyleCamera:
  - extends vtkVRPNTrackerStyleCamera which implements vtkInteractionDeviceStyle.
- More information about vtkVRPNTrackerCustomSensor and vtkVRPNTrackerCustomSensorStyleCamera is provided in "4.2.2.10 Head-tracking using 3rdTech Wide-Area Trackers"
- vtkVRPNPhantom:
  - adapted from vtkVRPNTracker and implements vtkInteractionDevice.
- vtkVRPNPhantomStyleCamera:
  - adapted from vtkVRPNTrackerStyleCamera and implements vtkInteractionDeviceStyle.
- More information about vtkVRPNPhantom and vtkVRPNPhantomStyleCamera is provided below in "4.2.2.11 Vortex Streamline Seeding with Phantom Omni Device"

4.2.2.10 Head-tracking using 3rdTech Wide-Area Trackers

4.2.2.10.1 vtkVRPNTrackerCustomSensor Initialization

pqVRPNStarter initializes vtkVRPNTrackerCustomSensor with

- the tracker's VRPN server host address specified in ParaViewR0.bat and ParaViewR1.bat
- the "sensor index", i.e. the index that informs ParaView whether this tracker is used for User1 and User2
- the Tracker Space Inverse Translation matrix:

- vtkVRPNTrackerCustomSensor adds this matrix to the position reported by the VRPN device. This subtracts the Tracker position in Tracker Space from the reported position, giving us the translation delta values.

- the Tracker to ParaView World Rotation matrix
  - vtkVRPNTrackerCustomSensor multiplies this matrix to the translation delta. This rotates the translation delta values to align with the ParaView world axes.
  - The axes-aligned translation delta values are set as the vtkVRPNTRackerCustomSensor::Position and used by vtkVRPNTrackerCustomSensorStyleCamera.
  - vtkVRPNTrackerCustomSensor also multiplies the rotation matrix to the rotation quaternion reported by the VRPN device to align the rotation with the ParaView world axes.
  - The axes-aligned rotation quaternion are set as the vtkVRPNTRackerCustomSensor::Rotation and used by vtkVRPNTrackerCustomSensorStyleCamera.

4.2.2.10.2 vtkVRPNTrackerCustomSensorStyleCamera Initialization

pqVRPNStarter initializes vtkVRPNTrackerCustomSensorStyleCamera with

- the corresponding vtkVRPNTrackerCustomSensor.
- the vtkRenderer that it will modify.

4.2.2.10.3 vtkVRPNTrackerCustomSensorStyleCamera

In vtkVRPNTrackerCustomSensorStyleCamera,

- the rotation quaternion from vtkVRPNTrackerCustomSensor::Rotation is converted to a rotation matrix.
- the rotation matrix is concatenated to the translation delta vector from vtkVRPNTrackerCustomSensor::Position to form
  - the Head Transform matrix supplied to ParaView via vtkCamera::SetHeadPose. vtkCamera contains code to utilize this matrix
  - to create perspective correct views.

4.2.2.11 Vortex Streamline Seeding with Phantom Omni Device

The vtkVRPNPhantom is a wrapper for the vrpn_Tracker and vrpn_Button classes that listen to the VRPN's Phantom values.

4.2.2.11.1 vtkVRPNPhantom Initialization

pqVRPNStarter initializes vtkVRPNPhantom with

- the Phantom's VRPN server host address specified in ParaViewR0.bat and ParaViewR1.bat
- the Phantom Space Inverse Translation matrix:
  - vtkVRPNPhantom adds this matrix to the position reported by the VRPN device. This subtracts the Phantom position in Phantom Space from the reported position, giving us the translation delta values.

- the number of vrpn_Button that it is tracking.(The Phantom Omni Device has 2 buttons).

4.2.2.11.2 vtkVRPNPhantomStyleCamera Initialization

pqVRPNStarter initializes vtkVRPNPhantomStyleCamera with

- the corresponding vtkVRPNPhantom.
- the vtkRenderer that it will modify.
- file used to log Phantom positions in ParaView space for testing purposes.

4.2.2.11.3 vtkVRPNPhantomStyleCamera

- This ParaView VRPN Plugin for the Collaborative Scientific Visualization Workbench is built on top of the VTK Interaction Device library, which is built on top of VTK. All other vtkInteractionDeviceStyle implementation classes access VTK classes to modify rendering attributes and do not require access to ParaView classes. (They were also originally built to work for VTK). Therefore, these classes are located within the VTK Interaction Device library. In the case of vtkVRPNTrackerCustomSensorStyleCamera, the camera position and orientation is modified via ParaView's instance of the vtkCamera class. vtkVRPNTrackerCustomSensorStyleCamera accesses the vtkCamera through vtkRenderer that was initialized with in pqVRPNStarter. However, the vtkVRPNPhantomStyleCamera is located in the ParaView VRPN Plugin because it needs to create and modify ParaView objects, which requires access to ParaView classes

- **Phantom Cursor Creation**
  - pqVRPNStarter::createConeInParaView creates the Phantom Cursor object using the PhantomCursorSource. (Refer to "4.2.2.4.2 Creating An Object" for further details.) This was run only once during the development process. The ParaView state file during the run was saved as ParaView\Plugins\VRPN\clean.pvsm
  - clean.pvsm is loaded upon pqVRPNStarter initialization. This is a workaround.
    - Basically all objects in the ParaView pipeline are deleted and the entire application is reset before a ParaView state file can be loaded in order for this Plugin to work correctly. The reason is that if a state is loaded after a source is created in pqVRPNStarter::createConeInParaView (in general, pqObjectBuilder::createSource), the source cannot be rendered using the steps in "4.2.2.4 Rendering in ParaView". It is highly possible that there is a better solution.
    - Note: State-loading occurs when the ParaView application is being reset due to the loading of a shared state (see "4.2.2.12 Push to Shared State (Implementation)") and during data set switching (see "4.2.2.13 Switching Data Sets for Vortex Visualization").

- **Phantom Cursor Position Update**
  - The Phantom position is obtained from vtkVRPNPhantom.
  - This position value is transformed into the vtkCamera space.
    - This is necessary to ensure that the orientation of the Phantom Cursor's movement on the visual display corresponds to the orientation of the user's hand movement regardless of the vtkCamera's orientation, which is user-defined via the SpaceNavigator.
    - The transformation is done by multiplying the position with vtkCamera:: CameraLightTransformMatrix
  - The position value is scaled by the size of the vtkCamera::FrustumPlanes . See vtkVRPNPhantomStyleCamera::ScaleByCameraFrustumPlanes.

        - This is not satisfactory. The Phantom Cursor is occasionally moved to a position that is very distant from the vtkCamera::FocalPoint due to great changes in the vtkCamera's position and orientation via the SpaceNavigator. The Phantom Cursor could not be retrieved because the Phantom Omni Device imposes a limit to the range of motion. When this happens, the workaround is to switch the Vortex Visualization data set (which forces a state reloading and application resetting. See "4.2.2.13 Switching Data Sets for Vortex Visualization".)
    - The Phantom Cursor's position is modified using the steps in "4.2.2.4.1 Editing An Object" for ParaView Representation objects.

- **Vortex Streamline Seeding**
    - Change the StreamTracer position
        - To change the Stream Tracer's seed position, modify the "Center" property of the Stream Tracer's Point Source. (See http://www.vtk.org/Wiki/ParaView/Users_Guide/List_of_filters#Stream_Tracer)
        - The code is copied directly from pqStreamTracerPanel::pqStreamTracerPanel
    - Delete the Tube filter that was previously applied to the Stream Tracer.
    - Create a new Tube filter with the Stream Tracer as the input.

- **Phantom Position Logging**
    - The Phantom position is recorded to a standard ofstream for test data evaluation. See *4. Visualization System Evaluation* of *Visualization of Flow Instability in Centrifugal Pumps* (Lee, 2011).

4.2.2.12 Push to Shared State (Implementation)

Ideally, the modifications to the ParaView objects should be propagated automatically between the two users during the Shared State mode. However, there are no available documentation on how to implement this at this point.

The idea of sharing a single state file between two ParaView applications came from Cory Quammen from the UNC Chapel Hill CISMM group.

This is the current implementation:

- When a user decides to push his/her modifications to the "Shared State", the modifications are inserted into a pre-created Paraview state file. The ParaView application that edited the state file records the time stamp of the file after that particular modification.
- Each ParaView application periodically checks the time stamp of the state file (i.e. within the QTimer callback. See "4.2.2.3 Updating VRPN Devices with QTimer"). If the time stamp differs from the application's recorded time stamp, it means that another ParaView application has modified the state file, and the current application should load the new Shared State.
- Loading the new Shared State:
    - Uninitialize VRPN Devices
        - The QTimer thread is stopped.
        - The following objects are deleted
            - QTimer object
            - SpaceNavigator and TNG-3B Serial Interface vrpn_Analog_Remote instances.
            - vtkDeviceInteractor used for managing the Wide-Area Tracker and Phantom Omni Device
    - Reset Application
        - Refer to pqCommandLineOptionsBehavior::resetApplication

- Load New Shared State
  - pqLoadStateReaction::loadState is invoked with the name of the shared state file. This file name is hardcoded due to time constraints. It should be extracted into a constants file in future implementation.
  - The state file time stamp is recorded in the Plugin.
- Set Up Viewpoint
  - The user viewpoint is set up using the steps in "4.2.2.2 Viewpoint Setup"
- Reinitialize VRPN Devices
  - All VRPN Devices are initialized. See the following sections for details:
    - "4.2.2.7 "World-in-hand" Data Set Manipulation using SpaceNavigator"
    - "4.2.2.8 Stereo Separation Control using TNG-3B Serial Interface"
    - "4.2.2.10 Head-tracking using 3rdTech Wide-Area Trackers"
    - "4.2.2.11 Vortex Streamline Seeding with Phantom Omni Device"

4.2.2.13 Switching Data Sets for Vortex Visualization

Refer to *3.2.2 ParaView Sources and State Files* of the *Visualization of Flow Instability in Centrifugal Pumps document* for context (Lee, 2011).

When the user switches the data set type, the state file for that particular data set is loaded using the same operations described in "Loading the new Shared State" of "4.2.2.12 Push to Shared State (Implementation)".

4.3 System Design Considerations & Future Work

4.3.1 Interactivity with dataset compared to original Paraview

Once the initial novelty of using Virtual Reality input devices with ParaView fades, interactivity with dataset will become an important factor. Rather than looking at a dataset from multiple viewpoints in order to "perceive" the shape of the structure, scientists may prefer a measurement tool / measurement feature that automatically generates all the values that they are interested in. For example, one who is very familiar with ParaView's keyboard shortcuts may prefer to use the mouse and keyboard to slice, clip and apply other filters to the dataset rather than try to use the headtracking, stereo and SpaceNavigator

1. **Retain ParaView's filtering functionality**
   1. Therefore, retaining all of ParaView's filtering functionality is important for interactivity with dataset. This is already implemented in the current system.
2. **Value addition to original ParaView**
   1. In the original ParaView, seeding streamtracers (integration of vector field to generate streamlines) is a difficult task. The mouse is not suitable for specifying a point in 3-dimensional space. In order to analyze the flow at a critical point, one would have to rotate the object several times to get the position of the streamtracer source correct. The following improvements have been implemented in the current system:
      1. Seeding of streamtracers using the Phantom haptic device for a single user. (**Done**)
      2. The Phantom haptic device can be used in conjunction with the SpaceNavigator, which provides "world-in-hand" manipulation of the data set.
3. **Weight of headtracker**
   1. The weight of the headtracker may become an important factor that influences preference between the
      1.original ParaView

2.current system without headtracking and
3.current system with headtracking

2. Using the workbench with head-tracking for hours for investigating the dataset causes discomfort.

4.3.2 Fidelity of interaction with dataset

A question would be whether fidelity of interaction with dataset (in as much as one could envision interacting with the dataset in real life) would produce new insights, or whether displaying measurements would suffice. To ensure high fidelity, the following should be implemented:

1. Accurate Head-tracking (accurate rendering based on head-tracked position)
2. Haptic-Rendering – the ability to feel the data set using the sense of touch via the Phantom Omni Device

4.3.3 Collaboration

To enable collaboration, there are three stages of implementation:

1. Continuous observation, occasional modification in shared state (**Done)**
2. Continuous modification (continuous sync between two users)
3. Enable switching between viewpoints

5. Appendix

5.1 ParaViewR0.bat

```
: README
GOTO EndREADME

To run the project, modify

PATH=%PATH%;<QT_DIRECTORY>;<PYTHON_DIRECTORY>;<COMPILEDPARAVIEW_DIRECTORY>\
Plugins\VRPN\vtkInteractionDevice\lib\Release;
<COMPILEDPARAVIEW_DIRECTORY>\bin\paraview_revised.exe --tracker --tracker-
address=<TRACKER_SERVER> --tracker-sensor=0 --tracker-origin="8.68,5.3,1.25" --spacenavigator --
spacenavigator-address=<SPACENAVIGATOR_SERVER> --tng --tng-address=<TNG_SERVER> --phantom --
phantom-address=<PHANTOM_SERVER> --stereo --stereo-type="Crystal Eyes"

Replace the following tags:
<QT_DIRECTORY>
with the directory in which you installed Qt.
E.g. C:\Qt\bin

<PYTHON_DIRECTORY>
with the directory in which you installed Python.
E.g. C:\Python27

<COMPILEDPARAVIEW_DIRECTORY>
with the directory of the compiled ParaView.
E.g. C:\Users\alexisc\Documents\EVE\CompiledParaView\



Input Options:

--tracker
Turn on headtracking with 3rdTech Hiball 3000 Wide-Area Tracker

--tracker-address
Replace <TRACKER_SERVER> with the address of the VRPN server that is running the 3rdTech Hiball 3000
Wide-Area Trackers. E.g. Tracker0@tracker1-cs.cs.unc.edu
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --tracker is specified.

--tracker-sensor
Index of HiBall. For this batch file, it should be 0. Only applicable when --tracker is specified.

--tracker-origin
Initial position of the first user. The values are specified in Tracker Space Coordinates. Only applicable when --
tracker is specified.



--spacenavigator
Turn on "world-in-hand" manipulation of dataset using 3Dconnexion SpaceNavigator

--spacenavigator-address
Replace <SPACENAVIGATOR_SERVER> with the address of the VRPN server that is running the 3Dconnexion
SpaceNavigator. E.g. device0@localhost
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --spacenavigator is specified.
```

--tng
Turn on stereo separation control with TNG-3B Serial Interface.

--tng-address
Replace <TNG_SERVER> with the address of the VRPN server that is running the TNG-3B Serial Interface. E.g. tng3name@localhost
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --tng is specified.

--phantom
Turn on Phantom for seeding streamtracers in the Vortex Visualization Use Case. See "3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization" of the "Collaborative Scientific Visualization Workbench Manual"

--phantom-address
Replace <PHANTOM_SERVER> with the address of the VRPN server that is running the Phantom Omni Device. E.g. Phantom0@localhost
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --phantom is specified.

This is an example:

PATH=%PATH%;C:\Qt\bin;C:\Python27;C:\Users\alexisc\Documents\EVE\CompiledParaView\Plugins\VRPN\vtkInteractionDevice\lib\Release;
C:\Users\alexisc\Documents\EVE\CompiledParaView\bin\Release\paraview_revised.exe --tracker --tracker-address=Tracker0@tracker1-cs.cs.unc.edu --tracker-sensor=0 --tracker-origin="8.68,5.3,1.25" --spacenavigator --spacenavigator-address=device0@localhost --tng --tng-address=tng3name@localhost --phantom --phantom-address=Phantom0@localhost --stereo --stereo-type="Crystal Eyes"
:EndREADME

PATH=%PATH%;<QT_DIRECTORY>;<PYTHON_DIRECTORY>;<COMPILEDPARAVIEW_DIRECTORY>\Plugins\VRPN\vtkInteractionDevice\lib\Release;
<COMPILEDPARAVIEW_DIRECTORY>\bin\paraview_revised.exe --tracker --tracker-address=<TRACKER_SERVER> --tracker-sensor=0 --tracker-origin="8.68,5.3,1.25" --spacenavigator --spacenavigator-address=<SPACENAVIGATOR_SERVER> --tng --tng-address=<TNG_SERVER> --phantom --phantom-address=<PHANTOM_SERVER> --stereo --stereo-type="Crystal Eyes"

5.2 ParaViewR1.bat

```
: README
GOTO EndREADME

To run the project, modify

PATH=%PATH%;<QT_DIRECTORY>;<PYTHON_DIRECTORY>;<COMPILEDPARAVIEW_DIRECTORY>\
Plugins\VRPN\vtkInteractionDevice\lib\Release;
<COMPILEDPARAVIEW_DIRECTORY>\bin\paraview_revised.exe --tracker --tracker-
address=<TRACKER_SERVER> --tracker-sensor=1 --vrpn-origin="8.58,5.3,1.3" --spacenavigator --
spacenavigator-address=<SPACENAVIGATOR_SERVER> --tng --tng-address=<TNG_SERVER> --phantom --
phantom-address=<PHANTOM_SERVER> --stereo --stereo-type="Crystal Eyes"

Replace the following tags:
<QT_DIRECTORY>
with the directory in which you installed Qt.
E.g. C:\Qt\bin

<PYTHON_DIRECTORY>
with the directory in which you installed Python.
E.g. C:\Python27

<COMPILEDPARAVIEW_DIRECTORY>
with the directory of the compiled ParaView.
E.g. C:\Users\alexisc\Documents\EVE\CompiledParaView\



Input Options:

--tracker
Turn on headtracking with 3rdTech Hiball 3000 Wide-Area Tracker

--tracker-address
Replace <TRACKER_SERVER> with the address of the VRPN server that is running the 3rdTech Hiball 3000
Wide-Area Trackers. E.g. Tracker0@tracker1-cs.cs.unc.edu
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --tracker is specified.

--tracker-sensor
Index of HiBall. For this batch file, it should be 1. Only applicable when --tracker is specified.

--tracker-origin
Initial position of the first user. The values are specified in Tracker Space Coordinates. Only applicable when --
tracker is specified.


--spacenavigator
Turn on "world-in-hand" manipulation of dataset using 3Dconnexion SpaceNavigator

--spacenavigator-address
Replace <SPACENAVIGATOR_SERVER> with the address of the VRPN server that is running the 3Dconnexion
SpaceNavigator. E.g. device0@localhost
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --spacenavigator is specified.
```

--tng
Turn on stereo separation control with TNG-3B Serial Interface.

--tng-address
Replace <TNG_SERVER> with the address of the VRPN server that is running the TNG-3B Serial Interface. E.g. tng3name@localhost
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --tng is specified.

--phantom
Turn on Phantom for seeding streamtracers in the Vortex Visualization Use Case. See "3.2.2.5 Special Use Case: Phantom Omni Device for Vortex Visualization" of the "Collaborative Scientific Visualization Workbench Manual"

--phantom-address
Replace <PHANTOM_SERVER> with the address of the VRPN server that is running the Phantom Omni Device. E.g. Phantom0@localhost
See Plugin/VRPN/TestParaViewVRPNDevices.bat for further information on VRPN server address.
Only applicable when --phantom is specified.

This is an example:

PATH=%PATH%;C:\Qt\bin;C:\Python27;C:\Users\alexisc\Documents\EVE\CompiledParaView\Plugins\VRPN\vtkInteractionDevice\lib\Release;
C:\Users\alexisc\Documents\EVE\CompiledParaView\bin\Release\paraview_revised.exe --tracker --tracker-address=Tracker0@tracker1-cs.cs.unc.edu --tracker-sensor=1 --vrpn-origin="8.58,5.3,1.3" --spacenavigator --spacenavigator-address=device0@localhost --tng --tng-address=tng3name@localhost --phantom --phantom-address=Phantom0@localhost --stereo --stereo-type="Crystal Eyes"
:EndREADME

PATH=%PATH%;<QT_DIRECTORY>;<PYTHON_DIRECTORY>;<COMPILEDPARAVIEW_DIRECTORY>\Plugins\VRPN\vtkInteractionDevice\lib\Release;
<COMPILEDPARAVIEW_DIRECTORY>\bin\paraview_revised.exe --tracker --tracker-address=<TRACKER_SERVER> --tracker-sensor=1 --tracker-origin="8.68,5.3,1.25" --spacenavigator --spacenavigator-address=<SPACENAVIGATOR_SERVER> --tng --tng-address=<TNG_SERVER> --phantom --phantom-address=<PHANTOM_SERVER> --stereo --stereo-type="Crystal Eyes"

5.3 TestParaViewVRPNDevices.bat

```
: README
GOTO EndREADME

To test that the SpaceNavigator, Phantom Omni Device and TNG-3B Serial Interface are working, modify
<COMPILEDVRPN_DIRECTORY>\client_src\Release\vrpn_print_devices.exe
<DEVICE>@<MACHINE_ADDRESS>
Replace the following tags:
<COMPILEDVRPN_DIRECTORY>
with the directory of the compiled VRPN project.
E.g. C:\Users\alexisc\Documents\EVE\CompiledVRPN\client_src\Release\vrpn_print_devices.exe

<DEVICE>
with the device that is being tested.
To test the SpaceNavigator, use:            device0
To test the Phantom Omni Device, use:       Phantom0
To test the TNG-3B Serial Interface, use:   tng3name

<MACHINE_ADDRESS>
with the network address of the VRPN server. Use: localhost

NOTE: If the device is plugged into a different computer from the one running TestParaViewVRPNDevices.bat:
1. Click the Windows 7 Start Menu Button.
2. In the "Search programs and files" toolbar, type "cmd" and press Enter.
3. Type:    ipconfig /all
4. Scroll to the top of the window. In the second and third lines after "ipconfig/all", locate "Host Name" and
"Primary Dns Suffix".
  Example:
  Host Name .....:Sutherland
  Primary Dns Suffix .....:cs.unc.edu

5. Replace <MACHINE_ADDRESS> with your computer's "Host Name" and "Primary Dns Suffix" in the
following format: "Host Name"."Primary Dns Suffix"
    Example:
  Sutherland.cs.unc.edu

Examples:
SpaceNavigator Test:
C:\Users\alexisc\Documents\EVE\CompiledVRPN\client_src\Release\vrpn_print_devices.exe device0@localhost

SpaceNavigator Test Result:
Analog device0@localhost:
      0.00, 0.01,  0.03, 0.00, 0.06, -0.01  (6 chans)

Phantom Omni Device Test:
C:\Users\alexisc\Documents\EVE\CompiledVRPN\client_src\Release\vrpn_print_devices.exe Phantom0@localhost

Phantom Omni Device Test Result:
Tracker Phantom0@localhost, sensor 0:
      pos  ( 0.00, -0.07, -0.09); quat (-0.13, 0.16, 0.74, 0.64)
Tracker Phantom0@localhost, sensor 0:
      vel (0.00, 0.00, 0.00); quatvel (0.00, 0.00, 0.00, 1.00)

TNG-3B Serial Interface Test:
C:\Users\alexisc\Documents\EVE\CompiledVRPN\client_src\Release\vrpn_print_devices.exe tng3name@localhost

TNG-3B Serial Interface Test Result:
```

```
Analog tn3name@localhost:
      245.00, 124.00 (2 chans)
:EndREADME

<COMPILEDVRPN_DIRECTORY>\client_src\Release\vrpn_print_devices.exe
<DEVICE>@<MACHINE_ADDRESS>
```

Works Cited

3rdTech, Inc. (2002, January 1). HiBall-3000 Wide-Area Tracker User Manual.

IEEE Visualization Contest 2011 Committee. (2011, March 21). Retrieved from IEEE Visualization Contest 2011 Data Set: http://viscontest.sdsc.edu/2011/dataset/dataset.html

Kitware Inc. (2009, September 18). Retrieved from The ParaView Tutorial: http://www.itk.org/Wiki/images/8/88/ParaViewTutorial38.pdf

Kitware Inc. (2010, November 1). Retrieved from How to Write a Custom ParaView Application: http://www.vtk.org/Wiki/Writing_Custom_Applications

Kitware Inc. (2011, June 1). Retrieved from ParaView List of File Readers: http://paraview.org/Wiki/ParaView/Users_Guide/List_of_readers

Lee, J. C. (2011). Visualization of Flow Instability in Centrifugal Pumps.

Nvidia Corporation. (2009, March). *NVIDIA Quadro vs. GeForce GPUs: Features and Benefits.* Retrieved from http://www.nvidia.com/object/quadro_geforce.html

SenSyr LLC. (2009, February 16). Retrieved from TNG-3B Sensors: http://www.sensyr.com/manuals/TNG3BSensors.pdf