Medical Data Visualization: A Simple Approach Based on the Visualization Toolkit *

Samuel Silva, Paulo Dias

***Abstract* – Data visualization is an important subject. In order to provide a greater insight on data, visualization systems are built which allow the user to manipulate, explore and analyse it.**

Building a visualization system may be a difficult task specially for those who are not graphical experts. The Visualization Toolkit (VTK) is an open source library which provides several classes and objects that can be used to build visualization systems for different kinds of data.

The presented work shows how it is possible to build a simple visualization system for medical data using only the VTK standard functionalities and how it is possible to integrate VTK with an interface development library in order to provide a more user-friendly interface.

***Keywords* – Data Visualization, Visualization Toolkit, Medical Data.**

I. INTRODUCTION

Data visualization is a subject of great importance. With the constant advances in technology and measuring devices, more and more data is generated/measured and has to be analysed. The advances in technology, and in computer systems in particular, also brought the chance of having visualization tools in a common desktop computer.

In Medical Imaging, the amount of data generated from exams (e.g., CT, fMRI, etc.) requires, nowadays, more than a simple slice by slice visualization. One must also provide ways of gathering the data and rebuilding the analysed volume thus giving a better context to the data being analysed. The question is which tool may be used to process and visualize this data.

Several tools have appeared in the literature for the purpose of data (or information) visualization. IRIS Explorer [1] and IBM's OpenDX [2] are tools which provide a set of modules that can be linked using visual programming to create the visualization pipeline [3]. A library called Visualization Toolkit (VTK) [4]-[7] provides an enormous of-the-shelf quantity of functionalities (through hundreds of objects and classes) which enable fast development of visualization systems for several kinds of data. It is a widely used library (see the works of Kanellopoulos et al. [8] and Sadleir et al. [9] for examples) developed in C++ with several interpreted interface layers including Tcl/Tk, Java and Python, which allows the development of applications directly in any of these languages. Its possible integration with the Insight Toolkit (ITK) [10] (for image segmentation and registration) is also an advantage in Medical Imaging scenarios. A recent effort in enhancing the combination between these two libraries is the Medical Imaging Interaction Toolkit (MITK) [11], [12] which adds more complex interaction capabilities to them. Do not confuse with the Medical Imaging Toolkit [13] (curiously also MITK) which tries to provide an alternative to the joint usage of VTK and ITK.

The work presented on this article tries to explore the functionalities of VTK in order to build a simple system for medical data visualization. On the following sections the main objectives of this work are presented. Then, the application pipeline is explained and some details are given about its development. After this, the integration of VTK functionalities in a user interface developed using Fox Toolkit is described. Throughout the article some application examples are presented by visualizing medical data using the developed environments. The article finishes with some conclusions and ideas for future work.

II. OBJECTIVES

This work, developed as the final project for the Computer Graphics Integrated Lab. course (part of the MSc in Electronics and Telecommunications) had two main objectives:

1. To use VTK functionalities to develop a simple visualization environment which allowed the visualization/exploration of medical data – this required an understanding of VTK's main architecture and pipeline and the exploration of its standard capabilities as explained on section III. Is was also intended to experiment with some kind of pseudo-haptic feedback technique applied to the user interaction with the data.
2. To integrate VTK with an interface development library in order to provide a better way of using the developed features – this required searching for means of integrating widgets such as the rendering window provided by VTK on the developed interface and understand how VTK allows manipulating data inside its data structures in order, for example, to convert it to a suitable format for integration, as explained on section IV.

III. QUICK PROTOTYPING: THE USE OF VTK STANDARD FUNCTIONALITIES

On this section a visualization system for medical data developed with VTK is presented and it is shown how it was used to view lung and bubble data (although the system can

*Work developed as the final project for the Computer Graphics Integrated Lab. course, part of the MSc in Electronics and Telecommunications.

read other kinds of medical data as it will be shown ahead).

A. Application Pipeline

There are several classes and widgets provided by VTK that can be used to rapidly develop a visualization system prototype with many interaction and visualization capabilities. Figure 1 shows the simplified functional pipeline of the developed prototype which will be explained next.

- ***Data Reading and Conversion*** – In this module data can be read from a particular format and then converted to a raw image data format which is a format understandable by `vtkImageReader2`. This function then receives the file and parameters regarding data dimensions and scaling factors (in x, y and z) and generates a `vtkImageData` structure containing all the data.
- ***Data Exploration*** – In order to provide better data exploration, three `vtkImagePlaneWidgets` are used which allow sectioning the data along the three axes (axial, sagital and coronal). As input, each one of these widgets receives the `vtkImageData` object containing all the read data. On the pipeline the comment *3D* means that all data slices are present. Each of the widgets returns a `vtkImageData` object (through the `getResliceOutput()` method) containing the information regarding the sectioning of the data. The comment *2D* means that only one data slice is returned.
- ***Volume Rendering*** – This module is the responsible for processing the read data present in the `vtkImageData` object creating a 3D volume that can be viewed by the user. The pipeline of figure 1 shows the usage of a `vtkVolumeTextureMapper` object to do the volume rendering of the data. With this mapper it is possible to associate color (through the definition of a `vtkColorTransferFunction`) and opacity (through the definition of a `vtkPiecewiseFunction`) to the grayscale levels present on the data used for the volume rendering. These functions are concatenated on a `vtkVolumeProperty` object and, along with the data coming from the mapper, a `vtkVolume` is created containing the obtained volume.
  There are other options for the volume rendering like the `vtkVolumeRayCastMapper` which provides a ray cast method.
- ***Visualization and Interaction*** – This module is responsible for providing the user with the visualization results and means of interacting with them. The `vtkVolume` and the three `vtkImagePlaneWidgets` are associated with a `vtkRenderer` object and a `vtkRenderWindow` is used to show both the rendered volume and the `vtkImagePlaneWidgets`. Notice that the `vtkRenderWindowInteractor` defined is common to the render window and to the image planes, thus allowing full interaction with both. There are particularities which, for the sake of simplicity, are not depicted in the pipeline of figure 1:
  1. The three `vtkImagePlaneWidgets` were all defined with a common `vtkPicker`. This allows a correct selection and manipulation of each widget when they are simultaneously presented on screen.
  2. By adding an observer to each `vtkImagePlaneWidget`, triggered by the interaction events and sharing the same callback, it was possible to synchronize all the three widgets, i.e., picking a point in one of the widgets changes the position of the others, thus making all the planes pass through that point.

Notice that, apart from a possible data conversion to a raw data format understandable by `vtkImageReader2` and defining the callback which allows the widgets synchronization, all the environment was created using off-the-shelf VTK classes.
On the following section the developed environment is shown and used to visualize/examine simple lung and bubble data.

B. Visualization of Lung and Bubble Data

In order to test the developed environment some lung and bubble data were used. These data were provided in separate text files containing the active voxels for each structure (lungs and bubbles). Because the purpose was to visualize the lungs with a degree of transparency and the bubbles inside, in a different color, and due to the fact that the data format was not suitable to be read by `vtkImageReader2`, lung and bubble information had to be joined and converted to a proper format. A different level of gray was associated with the lung (128 level) and bubble (255 level) active voxels which allowed a distinction between them during the volume rendering.

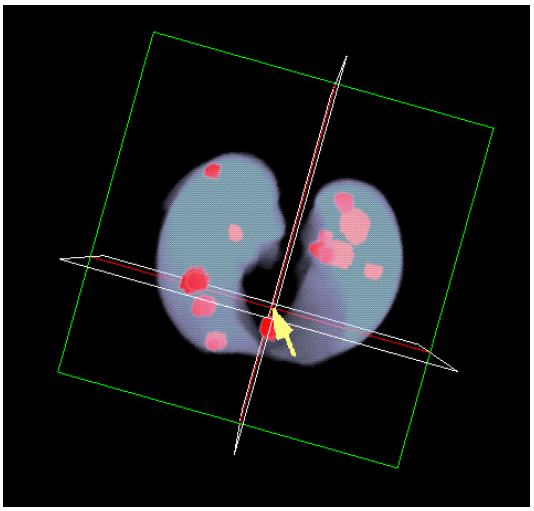

Figure 3 - Picking over a vtkImagePlaneWidget: The other two widgets are moved to the slice where the picked point is.

The developed environment can be seen on figure 2 (on the next page) with the image viewers on the left and the render window on the right showing the lungs and bubbles. Figure 3 shows how the picking of a point on one of the

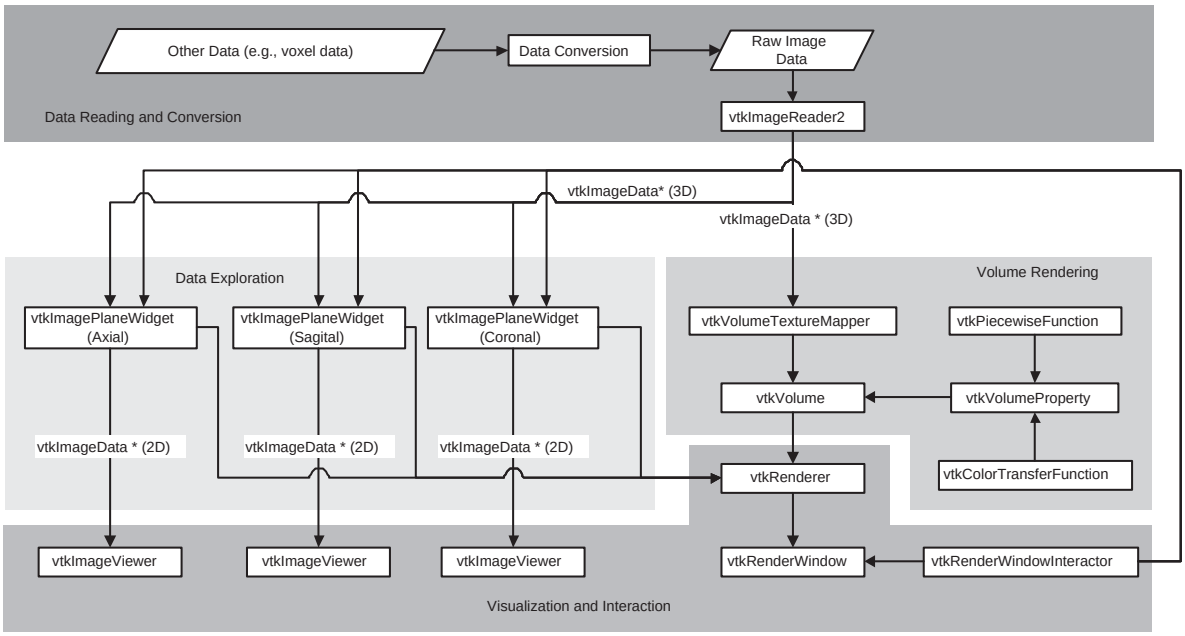

Figure 1 - Functional pipeline of the developed visualization environment.

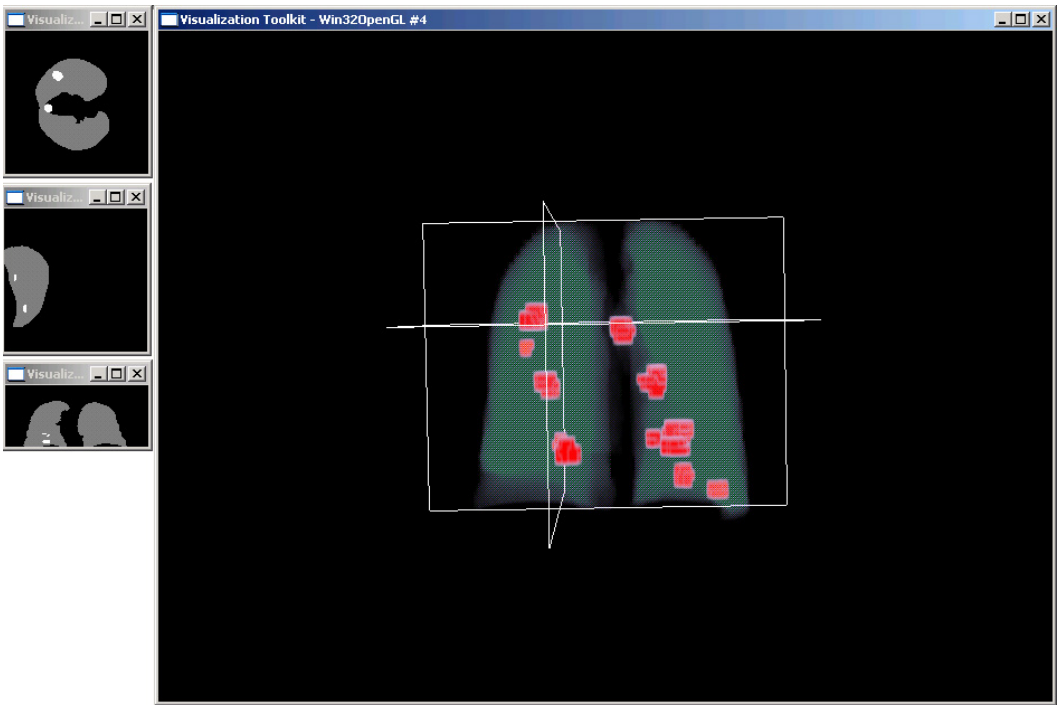

Figure 2 - Developed environment showing the lungs with bubbles inside, the three ImagePlaneWidgets, and the images corresponding to the three sections (axial, sagital and coronal).

planeWidgets results in the other two being moved to the slice where the picked point is.

C. Pseudo-haptic Feedback: A First Attempt

Haptic interfaces [14] can be used to simulate texture in many applications. However, due to their high price and complexity they are not widely used. It is possible to develop an interacion technique which, by combining a passive input device (e.g., a mouse) and visual feedback of a computer screen can be used to simulate textures. This is what is called pseudo-haptic feedback [15], [16].

This concept can be applied to the visualization of medical data by providing the user with some sort of pseudo-haptic feedback based on the kind of tissue the mouse pointer is on. An attempt was made based on the gray level of that point varying the mouse speed accordingly. To change the mouse speed a Windows API exists, called `SystemParametersInfo()` (see MSDN Library [17] for more details) which must be used with the following arguments:

```
SystemParametersInfo}(SPI_SETMOUSESPEED,
                      NULL,
                      (void*)speed,
                      SPIF_SENDCHANGE)
```

where `speed` is an integer variable containing the desired speed. There is a particularly important detail: notice how the third parameter passed to the function is (strangely) an integer with a cast to a pointer to void. This information is not given in the MSDN library (and even the example they present is wrong) and resulted in many failed attempts of using this function. The mouse speed parameter can be changed in a range between 1 and 20.
Using the lung data, a speed was established for each of the greyscale levels present (0, 128 and 255), making the mouse go faster out of the lungs and really slow when on a bubble. This allows the user to have more precision when moving the mouse on a bubble area.
It was noticed that, if the mouse movement is fast, the mouse speed is not changed at the precise moment the mouse enters a slower speed region. This can have something to do with the way VTK catches events and particularly with situations where the mouse pointer moves faster than the "picking cross" which appears over the plane widgets due, for example, to the volume display refresh.

IV. USER INTERFACE DEVELOPMENT: INTEGRATING VTK WITH FOX TOOLKIT

The second part of this work consisted in trying to integrate elements of the environment described above in a user interface which would then allow, for example, choosing the file to load or the volume rendering method.
Many options could be made at this point: among them were, for example, Microsoft Foundation Classes (MFC), Fox Toolkit [18], Qt [19], etc. It was considered that Fox Toolkit would be a good option due to its multi-platform nature and due to the author's greater experience with it than with any of the other.
On the following section a brief description of the integration process is provided.

A. Integration

Two main widgets add to be integrated in the interface: the render window and the image viewers. A library called *vtkFOX* [20], developed by Doug Henry, provides a modified FOX Toolkit canvas which contains an interactor object. This enabled VTK to draw on the FOX environment and input events to be redirected to VTK. This allowed an easy integration of the render window on the FOX environment but some issues did not allow maintaining full interaction capabilities: when the `vtkImagePlaneWidgets` were associated with the interactor, a VTK render window started to appear on the taskbar as a "ghost" window with no content refresh. The full scene was still rendered in the FOX environment and interaction with the plane widgets was possible. Figure 4 shows a VTK render window integrated in a FOX interface allowing full interaction capabilities with the volume and with the plane widgets. In order to avoid the appearance of this "ghost" window the interactor associated with the render window had to be different than the one used in the image planes, which resulted in the impossibility of direct manipulation of the planes on the render window. To manipulate the planes position a control window was created which allows their activation and movement (through a slider) (see the right side of figure 6). This was an easy task because VTK provides methods for enabling/disabling the planes and moving them to particular slices of the data.

Figure 4 - VTK render window integrated in a FOX Tollkit interface, allowing full interaction capabilities.

Integrating the `vtkImageViewer` objects was also possible using the modified canvas provided by vtkFOX but the same problem of "ghost" windows appeared. In this situation, in order to avoid the ghost windows, the option was to convert the image data contained in the `vtkImageData` objects returned by the image planes and show it on a FOX Toolkit widget. The data conversion was accomplished by obtaining a pointer to the scalar data vector using the method `GetScalarPointer()` and obtaining the size of the data through `GetDimensions()`.

B. fMRI Data Visualization

To test the developed visualization tool fMRI data of a head was used. To load the data, a dialog box is provided (see figure 5) which allows the user to choose a data file and specify its dimensions and applicable scale factors. It is also possible, in this dialog box, to define the type of data (VTK supports a larger number of types but, at the moment, only unsigned char and unsigned short are available). After loading the data, the cutting planes are positioned automatically at the middle slice in each dimension (although they have to be enabled to obtain a valid `getResliceOutput()` to fill the image windows, they are then disabled) and the image windows show the result. Figure 6 shows the developed interface. On the left the im-

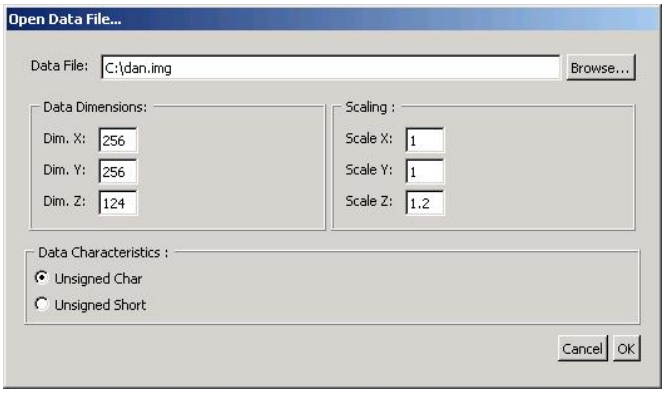

Figure 5 - File loading dialog box: the user can browse for a data file and then define several parameters regarding the data characteristics.

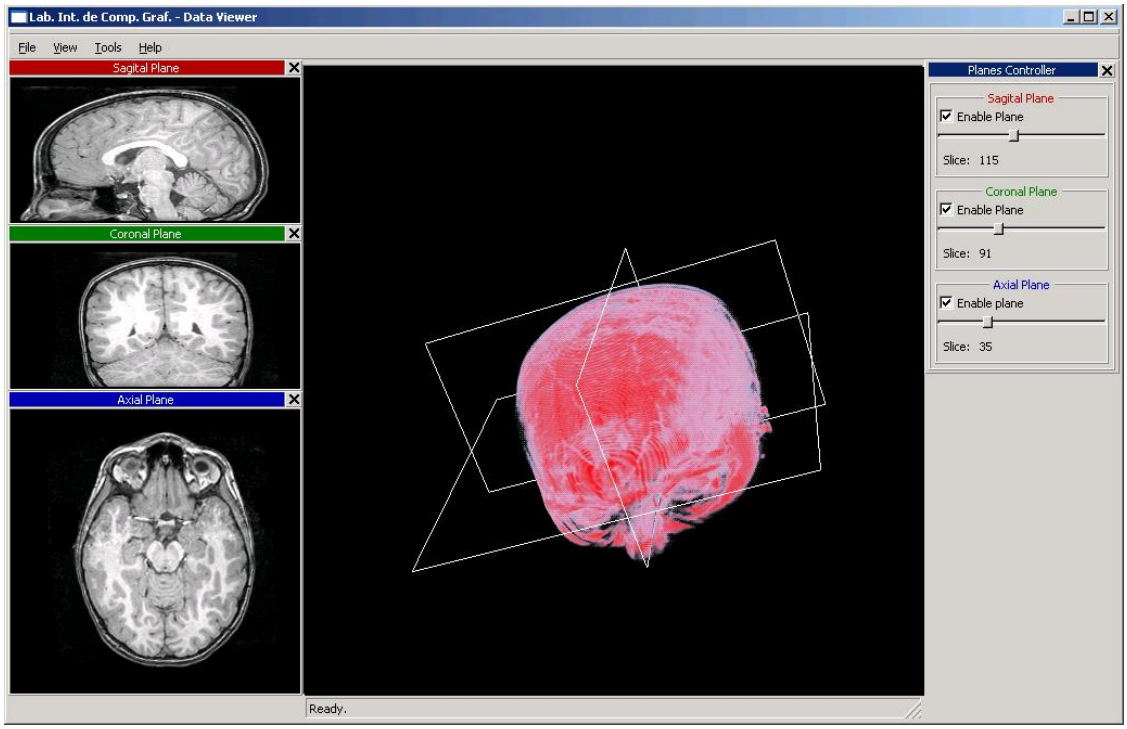

Figure 6 - Fox interface showing fMRI data.

age windows showing the cutting planes content. These windows allow, by using the right mouse button, to save their content to a image file in BMP format. On the middle, the result of applying a volume rendering method (using a `vtkVolumeTextureMapper`) to the data. It is also possible to view the active cutting planes. On the right, the widget containing the controls which allow enabling/disabling the cutting planes and control their position. All these windows are undockable and can be closed or positioned where the user wishes.
The result of using a ray caster (`vtkRayCastMapper`) to perform the volume rendering is shown on figure 7.

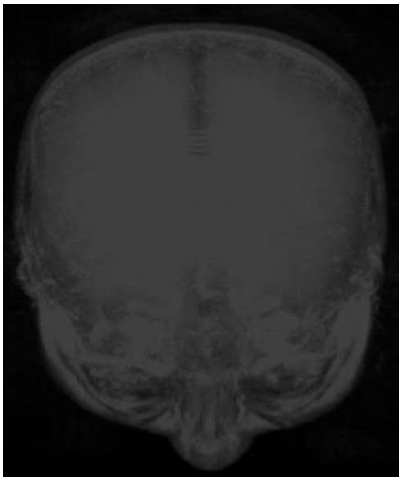

Figure 7 - Volume obtained from the fMRI data using a vtkRayCastMapper (ray cast method).

V. CONCLUSIONS AND FUTURE WORK

The presented work allowed to develop a simple visualization system capable of providing a basic analysis of medical data. It allows volume visualization and axial, sagital and coronal cuts, thus providing several degrees of freedom

in data exploration. The Visualization Toolkit allowed the development of the presented functionalities in a more or less easy way. The only problem found was the large number of classes and functionalities provided by this library, a situation that does not always help on a first time exploration. The environment developed using only VTK provided a reasonable analysis/exploration of the data showing that VTK can be used to perform fast visualization system prototyping. The first attempt of using pseudo-haptic feedback resulted in finding a "natural" (to the user) method of varying the mouse speed which gives, according to some qualitative testing, a kind of "haptic feedback" when exploring the data.

The visualization environment developed by integrating VTK with FOX Toolkit, although with losses regarding some interaction capabilities of VTK, allowed a more versatile system gathering, in one window, all the functionalities (which now included loading data files and changing the volume rendering method in run-time).

But, the developed visualization system (using VTK and FOX) still only provides basic data exploration/analysis. So, much more can be done to enhance it:

- Provide the possibility of picking a point in one of the images and move the other cutting planes in order for them to cross on that point;
- Allow the user to define the designation for each section (sagital, coronal or axial) or change data orientation during the loading process;
- Add a window containing information about the data: dimensions, scaling factors, scalar type, etc.;
- Add a dialog box which allows the user to custom properties of the volume rendering method, e.g., the color transfer function;
- Continue to explore the idea of adding pseudo-haptic feedback to user interaction with the data, namely by testing its applicability on images with a greater number of grey levels distributed erratically (i.e., there isn't, in general, a succession of regions with common grey levels) like fMRI data. The use of this pseudo-haptic techniques must also be evaluated in order to understand if it can provide useful information.

The developed environment is very simple but offers a platform that can be expanded in order to allow more specific data visualization, e.g., electrode information associated with fMRI exams.

ACKNOWLEDGMENTS

The author would like to thank to José Silva for providing the lung data and to José Maria Fernandes for providing the fMRI data used to test the developed systems.

REFERENCES

[1] J. Walton, *NAG's IRIS Explorer*, in Visualization Handbook, Charles D. Mansen, Chris R. Johnson (eds.), Academic Press, 2004.

[2] D. Thompson, J. Braun, and R. Ford, *OpenDX: Paths to Visualization*, VIS Inc., 2000.

[3] M. B. Carmo, J. D. Cunha, and A. P. Claudio, "Ivprototype-an information visualization prototype", in *Proc. 6^{th} International Conference on Information Visualization*, pp. 159-164, 2002.

[4] W. Schroeder, L. Avila, and W. Hoffman, "Vtk tutorial", *IEEE Computer Graphics & Applications*, vol. 20, no. 5, pp. 20–27, 2000.

[5] W. Schroeder, K. Martin, and W. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Kitware Inc., 3^{rd} edition, 2004.

[6] THE VISUALIZATION TOOLKIT, "http://public.kitware.com/vtk/", online Feb. 2005.

[7] Kitware Inc., *The VTK User's Guide*, Kitware Inc., version 4.4 edition, 2004.

[8] I. Kanellopoulos, A. Stein, and M. Turatti, "Visualisation of geographic information in a dynamic 3-dimensional environment", in *Proc. Geoscience and Remote Sensing Symposium 2001 (IGARSS '01)*, pp. 201-203, 2001.

[9] R. Sadleir, P. Whelan, P. MacMathuna, and H. Fenlon, "Informatics in radiology: Portable toolkit for providing straightforward access to medical image data", *in RadioGraphics 2004*, vol. 24, pp. 1193–1202, 2004.

[10] ITK - INSIGHT SEGMENTATION AND REGISTRATION TOOLKIT, "http://www.itk.org/", online Feb. 2005.

[11] I. Wolf, M. Vetter, I. Wegner, M. Nolden, T. Bottger, M. Hastenteufel, M. Schobinger, T. Kunert, and H.P. Meinzer, "The medical imaging interaction toolkit (mitk): a toolkit facilitating the creation of interactive software by extending vtk and itk", in *Proc. SPIE Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display, vol. 5367*, pp. 16-27, 2004.

[12] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, and H.P. Meinzer, "The medical imaging interaction toolkit", *Medical Image Analysis*, vol. 9, no. 6, pp. 594–604, 2005.

[13] M. Zhao, J. Tian, X. Zhu, J. Xue, Z. Cheng, and H. Zhao, "The design and implementation of a c++ toolkit for integrated medical image processing and analyzing", in *Proc. of SPIE Medical Imaging 2004: Visualization, Image-Guided Procedures, and Display, vol. 5367*, pp. 39-47, 2004.

[14] G. Burdea, *Force and Touch Feedback for Virtual Reality*, John Willey & Sons, New York, 1996.

[15] A. Lécuyer, J.M. Burkhardt, and L. Etienne, "Feeling bumps and holes without a haptic interface: the perception of pseudo-haptic textures", in *Proc. of the SIGCHI conference on Human factors in computing systems*, pp.239-246, 2004.

[16] A. Paljic, J.M. Burkhardt, and S. Coquillart, "Evaluation of pseudo-haptic feedback for simulating torque: a comparison between isometric and elastic input devices", in *Proc. 12^{th} International Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems (HAPTICS'04)*, pp. 216-223, 2004.

[17] MSDN Library, "http://msdn.microsoft.com/library", online Feb. 2005.

[18] J. van der Zijp, "FOX TOOLKIT", online Feb. 2005.

[19] QT, "http://www.trolltech.com/products/qt/index.html", online Feb. 2005.

[20] D. Henry, "VTKFOX, http://www.brilligent.com/wikka.php?wakka=vtkfox", online Feb. 2005.