A VIRTUAL REALITY SYSTEM FOR MEDICAL IMAGING

Giuseppe De Pietro, Ivana Marra and Carmela Vanzanella
Institute for High-Performance Computing and Networking, CNR, Via P. Castellino111, Naples, Italy

Keywords: OSG, Immersive Medical Imaging, VR Juggler, 3D interaction.

Abstract: In recent years significant advances in the field of Immersive Medical Imaging Analysis have made it essential to many disciplines related to medicine, such as radiology, neurology, cardiology, radiotherapy. Starting from three dimensional (3D) image datasets produced by computed tomography (CT) or magnetic resonance imaging (MRI) scans, for these areas it can be very useful to use simulated visualizations of human organs. By viewing the inside of anatomical structures and interacting with them, doctors can better understand the data of interest for medical training, surgical simulations, examination and diagnosis aims.In this paper we present a virtual reality system, that involves new software components based on top of open-source and cross-platform libraries; it consists of a set of services implementing immersive 3D navigation and interaction of virtual representations of human organ structures, starting from DICOM data. These features have been developed to be integrated into a medical imaging component-based architecture, which at present is under development.

1 INTRODUCTION

The field of Virtual Reality for Medical Image Analysis has been strongly investigated over the past decade. Innovations in imaging and immersive technologies and techniques have enabled many improvements in 3D visualization of complex anatomical models (Bartz, 2005). However, existing software tools have only basic support for interaction and immersive features. Moreover, many commercial applications for medical imaging are platform dependent and freeware ones are often developed for specific fields of medicine on the top of proprietary libraries (Rosset, Spadola, Ratib, 2004) so to make it difficult to easily implement new features when needed.

In this paper we propose an implementation of an open-source software components system able to provide immersive visualization and interaction with 3D models (Angel, 2002) which are reconstructed from DICOM image series (Suetens, 2002); the system is essentially based on Open Scene Graph (OSG www.openscenegraph.org) and Virtual Juggler (VR Juggler www.vrjuggler.org) open-source and cross-platform libraries.

OSG is a graphic library used to organize data into scene graph structures, and to provide a set of methods to efficiently handle 3D models. VR Juggler allows to visualize, navigate and interact with 3D models through different devices such as data gloves, HMDs, sensors, and any Immersive Projection Technology (IPT).

The developed system provides some of the most used medical imaging functionalities such as the selection of Region of Interest (ROI), Segmentation, Co-Registration, Fusion and measurements on the synthetic model, besides standard 3D visualization and interaction features.

Another important characteristic of the developed system is that it can be easily integrated within the most popular freeware medical imaging visualization toolkits.

Moreover, we developed a set of OSG-based VR Juggler components which invoke some OSG classes methods in order to encapsulate the advanced features of both libraries.

The virtual reality system is based on a set of components which have been tested by integrating them into a software architecture we developed for medical imaging applications (Von Rymon-Lipinski,

De Pietro G., Marra I. and Vanzanella C. (2007).
A VIRTUAL REALITY SYSTEM FOR MEDICAL IMAGING.
In *Proceedings of the Second International Conference on Computer Graphics Theory and Applications - AS/IE*, pages 215-220
DOI: 10.5220/0002075002150220

Jansen, Hanssen, Lievin, Keeve, 2002). This open-source and cross-platform libraries, combined and extended in order to support medical imaging processing, basic 3D visualization and interaction functionalities.

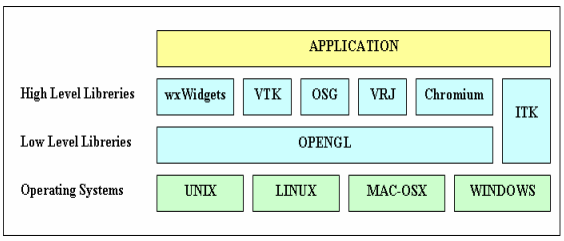

Figure 1: Software Architecture for Immersive Medical Imaging Applications.

In particular, wxWidgets is used for GUI implementation, VTK for 2D and 3D visualization, while ITK for realizing image processing functionalities.

In addition, if virtual reality is required, the Open Scene Graph toolkit is used to organize data into scene graph structures, and the VR Juggler framework to provide the necessary interface to virtual reality devices.

architecture, which is shown in figure 1, is based on

Additional C++ code provides efficient integration of the above mentioned libraries and adds new functionalities.

2 SYSTEM ARCHITECTURE OVERVIEW

As said in the previous section, we are currently developing a component-based framework for immersive medical imaging, which essentially implements two services (fig. 2):

1) *the Medical Imaging Service,*
2) *the Virtual Reality Service*

The first service provides the user with typical functionalities for visualization and image processing of medical data coming from DICOM images, such as segmentation, co-registration, fusion, and ROI.

Moreover, 3D models can be reconstructed both through a surface and direct volume rendering.

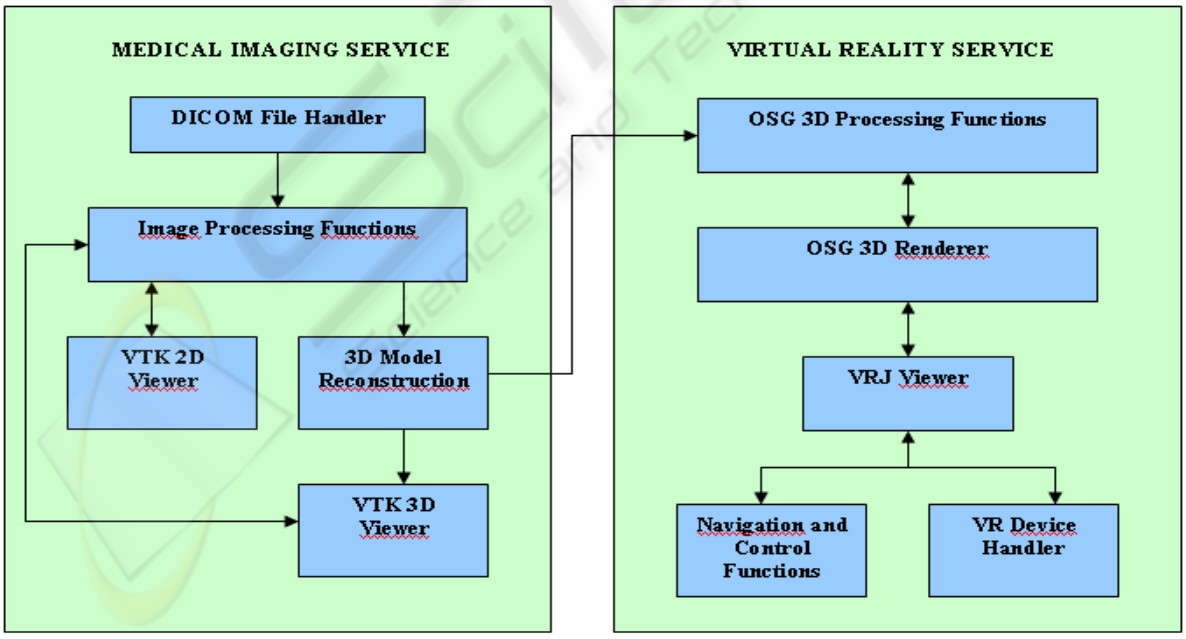

Figure 2: Component-based Architecture.

The *Virtual Reality Service* gives the user all the functionalities for immersive visualization and interaction with the 3D model (Forsberg, Kirby, 2000). This service consists of five components:

1) the *VRJ Viewer* which is the component for immersive visualization of the synthetic model. It provides functionalities for different navigation devices, active and passive stereoscopic systems, and cave systems;

2) the *OSG 3D Renderer* which provides both surface rendering and direct volume rendering algorithms, to be applied on the model deriving from the 3D reconstruction of volumetric data. It has been designed and implemented as an independent module from the VRJ Viewer, because they usually need different computational resources;

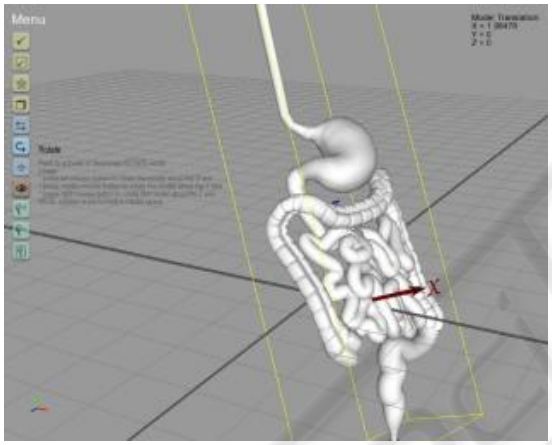

Figure 3: OSG user-interface.

3) the *OSG 3D Processing Functions* is the key component of the Virtual Reality Service. Through a user-friendly interface (fig. 3) it is possible to choose among several functionalities, such as 3D model transformations, component selection, bounding box extraction. All of them allow the user to directly interact with the synthetic scene, so to select a part of the object, to move the camera in any direction within the scene, zoom in and out, pan left or right, or change the 3D model coordinates;

4) the *Navigation and Control Functions* component which gives all the functionalities to get a natural real-time interaction with the virtual environment. A user can navigate around a virtual scene using a control menu, through some pointing device. For this aim we have used the VR Juggler configuration manager paradigm combined with OSG setting parameters, so to allow the application to run in the virtual system described in the following section;

5) the *VR Device Handler* supplies an interface to easily integrate most of commonly used virtual reality devices (HMD, Data Gloves, MoCap, mouse, joystick, etc.) into applications. It is based on the VR Juggler proxy paradigm, which hides hardware details of the selected devices.

The above illustrated components offer the end-user many functionalities, as described in the use-case diagram of figure 4.

The most important ones are:

1. Navigate 3D world
2. Render mode
3. Transform object
4. Show bounding box
5. Show object

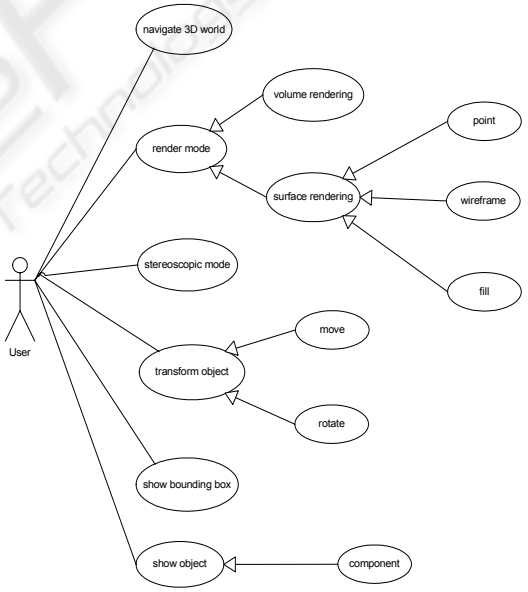

Figure 4: Use Case Diagram.

The *Navigate 3D world* allows the user to move through a scene modifying its point of view by implicitly changing camera movements. Moreover, it is possible, with the same mechanism, to implement rotate pan and scale functions.

The *Render mode* function allows to select between two rendering techniques: Volume and Surface. In particular, for Surface rendering it is

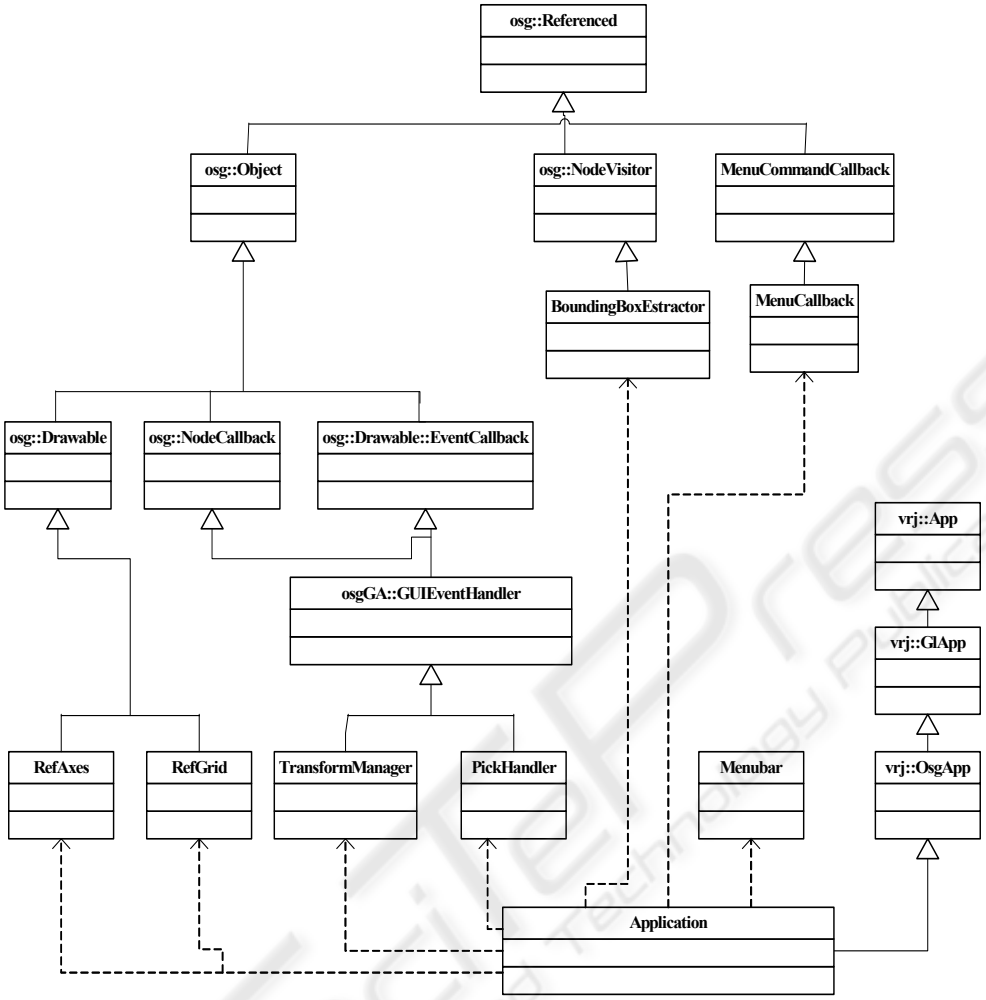

Figure 5: Component Class Diagram.

possible to select between three representations (point, wireframe, fill) in order to visualize a 3D model from abstraction level of detail.

The *Show bounding box* functionality is used to optimize geometrical operations by using simple volumes to contain more complex objects and it is useful to show the actual size of a partially visible part of the model from the actual angle-shot.

The *Transform object* refers to the geometric manipulation process of the object transform matrix, and allows to modify the object coordinates to translate and rotate itself. It should be noted that such transformations are different from the Navigation ones because they are linked to the object itself and not with the camera.

Once the model has been loaded, the user through the *Show object* use case can choose an object within the scene and interact with it.

3 SOFTWARE IMPLEMENTATION

The Visualization and Interaction Components of Virtual Reality Service, presented in the previous section, have been implemented in C++; many support classes derive from OSG and VR Juggler libraries. We tested the implementation of the two software services, the Medical Imaging Service and the Virtual Reality Service, by using some DICOM image data sets, coming from MRI and CT scans. The modular approach used for the implementation is strongly related to Scene Graph (SG) data structure, which is used to efficiently represent

complex scenes in the graphical environment. In figure 5 the diagram of the main classes, used to develop software components, is shown. Most of these classes can be grouped into three categories:

A) Basic Graphic Classes, which provides methods for graphic interfaces and for setting environmental graphic options;

B) Events Classes, which includes all the classes used to handle events generated by user's commands;

C) Application Class, which includes the methods to properly integrate OSG and VR Juggler libraries

3.1 Basic Graphic Classes

BoundingBoxExtractor class calculates the bounding boxes of all objects in the scene during the traversal of the scene graph. The user can select an object and visualize its bounding box to define a bounding box of a scene graph.

MenuBar is used to generate the graphical menu with push-buttons. When an instance of this class is created, it executes one or more calls to the addButton () method specifying the data of the push-buttons to be added; therefore the buildSceneGraph () method is called in order to generate the subgraph of the menu. At this point the instance of the MenuBar class can be destroyed;

RefAxes class is responsible for the visualization of the reference axes. It inherits from osg::Drawable class which allows to design geometries compounded by lines, triangles, etc. An instance of this class visualizes a system of three orthogonal axes in the left inferior angle of the window. The guideline of the axes is calculated for every frame and is updated according to camera orientation;

RefGrid class allows the visualization (if required) of the reference grid. Like RefAxes class it inherits from osg::Drawable class. Instance of this class visualizes a grid with variable dimension and step, layered on a plane with z=0.

3.2 Event Classes

MenuCommandCallback is a virtual class in which are defined the virtual methods for activate or disable the menu commands.

MenuCallback class inherits from the previous class and implements the virtual methods enable() and disable() which are called by an activation/de-activation action from the menu.

PickHandler class intercepts the mouse events to make possible the selection and the highlighting of the nodes of interest of the 3D model. This is obtained through Application::selectNode() and Application::highlightNode() methods.

TransformManager class manages the transformations (translation and rotation) on the 3D model. It is an event handler that concurs to modify the transformation matrix of a MatrixTransform node. These modifications are performed according to the user input device movements.

3.3 Application Class

A very important class, which is not included in the previously mentioned class categories is the *Application* class.

Application is the core class of the whole system. Through the vrj::App interface, the VR Juggler kernel runs an instance of the class and manages all the computations required by the user inputs, updates consequently the 3D scene and can detect collisions among the objects.

In order to have an integrated use of OSG and VR Juggler, the Application class implements the vrj::OsgApp::initscene() method which initializes the SG structure, that represents all the 3D scene elements. The currently active SG is accessed calling the vrj::OsgApp::getScene() method whenever it is required (i. e. for rendering or updating aims).

4 SYSTEM PROTOTYPE

We developed a first prototype of the Virtual Reality System implementing the software architecture described in section 2.

This prototype includes the following hardware devices:

- a graphical bi-processor workstation;
- a Stereoscopic Video Projection System, with two DLP 3000ANSI LUMEN projectors and a 1,5X2 Mt screen;
- a HiRes900 Cybermind Head Mounted Display;
- an HP iPAQ hx2490 Pocket PC with PDA functionalities.

Figure 6: VR System.

In figure 6 it is shown the application used for testing the whole system. It consists of immersive stereoscopic visualization of a 3D model representing a human gut obtained from a series of DICOM slices acquired as CT datasets scans. Through a PDA, the user is able to navigate and interact with the model in a very user-friendly manner.

5 CONCLUSIONS AND FUTURE WORK

In this paper we presented a system for immersive medical imaging.

It is based on visualization and interaction components which have been realized on the top of Open Scene Graph and VR Juggler open source libraries, and integrated into a software architecture for immersive medical imaging applications currently under development.

The modularity of the Virtual Reality System can allow an easy extension of its capabilities. At present, we are working at the implementations of some crucial medical imaging functionalities, such as ROI, Segmentation and Image Fusion, so to make them work directly on 3D model. In particular, to reach this aim we are extending some VTK classes with new methods, and integrating them directly into VR Juggler, without switching between 2D and 3D data representation, whenever some kind of 3D model manipulation is required.

Finally, we are testing effective performance improvements for highly complex scenes through the parallelization (Bajaj, Ihm, Koo, 1999) of the rendering phase.

REFERENCES

Dirk Bartz , "*Virtual Endoscopy in Research and Clinical Practice"*ACM CCS**:** I.3.7 Computer Graphics: *Three-dimensional graphics and realism*, J.3 Computer Applications: Life and Medical Sciences Volume 24 (2005), number 1 pp. 111–126.

Rosset A., Spadola L., Ratib O., "*OsiriX: an open-source software for navigating in multidimensional DICOM images*", Journal of digital imaging: the official journal of the Society for Computer Applications in Radiology., Springer-Verlang .

Von Rymon-Lipinski, B., Jansen, T., Hanssen, N., Lievin, M., Keeve, E., 2002. A software framework for medical visualization. In Poster Compendium of IEEE Visualization_02, pp. 100–101.

P. Suetens, "*Fundamentals of Medical Imaging*", Cambridge University Press.

Andrew S. Forsberg, Robert M. Kirby, David H. Laidlaw, George E. Karniadakis, Andries van Dam, and Jonathan L. Elion*." Immersive virtual reality for visualizing flow through an artery".* In *Proceedings Visualization '00*. IEEE Computer Society Press, 2000.

E. Angel, *"Interactive Computer Graphics"*, ed. Addison Wesley.

Chandrajit Bajaj, Insung Ihm, Gee-Bum Koo, et al. "*Parallel Ray Casting of Visible Human on Distributed Memory Architectures***",** Proc on Data Visualization '99.