A Novel Software Platform for Medical Image Processing and Analyzing

Jie Tian, *Senior Member, IEEE*, Jian Xue, Yakang Dai, Jian Chen, and Jian Zheng

***Abstract*—The design of software platform for medical imaging application has been increasingly prioritized as the sophisticated application of medical imaging. With this demand, we have designed and implemented a novel software platform in traditional object-oriented fashion with some common design patterns. This platform integrates the mainstream algorithms for medical image processing and analyzing within a consistent framework, including reconstruction, segmentation, registration, visualization, etc., and provides a powerful tool for both scientists and engineers. The overall framework and certain key technologies are introduced in detail. Presented experiment examples, numerous downloads, extensive uses, and practical applications commendably demonstrate the validity and flexibility of the platform.**

***Index Terms*—C++ ToolKit, medical imaging, software platform, visualization.**

I. Introduction

MODERN medical imaging devices, such as computed tomography (CT), MRI, and electronic endoscopy, provide tremendous benefits for easy disease diagnoses. Corresponding computer technologies play important roles in processing and analyzing medical images, including computer graphics, pattern recognition, virtual reality, etc. Three main research fields on which medical image processing and analyzing focuses are structural imaging, functional imaging, and molecular imaging. Past two decades have witnessed many algorithms developed in these fields by scientists and engineers, and recently, new algorithms have been emerging continuously [1]–[3].

Consolidating existing algorithms to stimulate the development of new technologies, many softwares have been designed. These softwares can be divided into two categories: algorithm toolkit and application system. Visualization Toolkit (VTK, www.vtk.org) [4]–[6] and Insight Segmentation and Registration Toolkit (ITK, www.itk.org) [7] are the most notable algorithm toolkits that aim to provide an algorithm library for scientific research and software development. 3DVIEWNIX [8] is a typical example for an application system, whose purpose is to provide wieldy assistance tools for more precise diagnosis. Many other useful algorithm toolkits and application systems are based on VTK and ITK, such as Medical Imaging Interaction Toolkit [9] and VolView [10].

Although VTK and ITK are the most famous and popular in medical image processing, there are several bottlenecks that limit their applications. First, ITK does not provide the function of visualization by itself, so we have to apply both ITK and VTK to compose a medical imaging system. Nevertheless, VTK possesses the classical object-oriented design method whereas ITK possesses generic programming; consequently, the frameworks and coding styles of VTK and ITK are quite different. From a normal user's point of view, it is difficult to handle two sets of large-scale toolkits with very different styles. Second, VTK is not specially designed for medical data visualization; thus, it seems too large and complicated for medical data visualization, which may increase the learning difficulty and affect the running efficiency. Furthermore, algorithm categories in VTK and ITK are not sufficient enough; more specific algorithms for functional imaging [11] and molecular imaging [12] are necessarily required. Third, ITK utilizes many modern C++ language features, especially the template, which takes full advantage of the generic programming. It is indeed a very good design for senior C++ developers, but not so good for average researchers because its framework and codes are difficult to understand even for those who are familar with C++. Finally, VTK and ITK do not provide the support to process out-of-core datasets that appear more and more these days. Without an underlying processing framework for out-of-core datasets, researchers have to additionally design special data accessing operations. All these factors restrict the application range of VTK and ITK in medical imaging.

Following VTK and ITK, a particular workshop called Software Development Issues for Medical Imaging Computing and Computer Assisted Interventions (MICCAI) was carried out at the MICCAI 2003 conference. Then, a Visualization Toolkits Session was held at the International Society For Optical Engineering (SPIE) Medical Imaging in 2004. Recently, Medical Image Analysis with ITK and Related Open-Source Software Courses were held in SPIE Medical Imaging in 2006 and 2007. Nowadays, research and development of platforms for medical

Manuscript received October 29, 2007; revised January 25, 2008. First published May 30, 2008; current version published November 5, 2008. This work was supported in part by the Project for the National Key Basic Research and Development Program (973) under Grant 2006CB705700, in part by the Changjiang Scholars and Innovative Research Team University (PCSIRT) under Grant IRT0645, in part by the Chinese Academy of Sciences (CAS) Hundred Talents Program, CAS Scientific Research Equipment Develop Program (YZ0642, YZ200766), 863 Program under Grant 2006AA04Z216, in part by the Joint Research Fund for Overseas Chinese Young Scholars under Grant 30528027, in part by the National Natural Science Foundation of China under Grant 30672690, Grant 30600151, Grant 30500131, Grant 60532050, and in part by the Beijing Natural Science Fund under Grant 4051002 and Grant 4071003.

J. Tian is with the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, and also with the Life Science Center, Xidian University, Xian 710071, China (e-mail: tian@ieee.org).

J. Xue is with the College of Computing and Communication Engineering, Graduate University of Chinese Academy of Sciences, Beijing 100039, China.

Y.-K. Dai, J. Chen, and J. Zheng are with the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITB.2008.926395

imaging is becoming a hot topic. However, there is no uniform software platform for medical image processing and analyzing till date, and better platform solutions need to be investigated urgently.

In this paper, we propose a full platform solution for medical image processing and analyzing, including the Medical Imaging Toolkit (MITK) and the 3-Dimensional Medical Image Processing and Analyzing System (3DMed). MITK is a toolkit being developed with an algorithm library for research and software development, while 3DMed is an application system with an entire application framework. The main innovations and advantages of this platform can be illustrated as follows.

1) This platform is investigated and developed specially for medical image processing and analyses. We integrate the mainstream algorithms in structural imaging, functional imaging, and molecular imaging within a consistent framework, which makes the platform focused and expert for medical imaging.
2) This platform provides an underlying processing framework for out-of-core datasets. All concrete algorithms, including reconstruction, segmentation, registration, visualization, etc., can process out-of-core data through an underlying interface offered by MITK.
3) We employ a data flow model to design computational framework, abstract medical data to volume and mesh, and import many design pattern methods to manage memory automatically. All these features make the platform elegant, easy to understand, and convenient to use.
4) In the platform, MITK is the fundamental algorithm layer and 3DMed is the application layer. They can be combined together to form an extensible medical image processing and analyzing system. Moreover, MITK can also be used solely as an integral algorithm toolkit. This kind of platform structure offers a clear idea and an easy manner for developing complex medical imaging software.

With these efforts, we try to solve the aforementioned problems of existing platforms and provide another excellent choice for both scientists and engineers in the medical imaging field.

In this paper, the detail of medical imaging algorithms in this novel platform is not discussed. We mainly focus on the design and implementation of the platform's framework, which is the greatest contribution of our study. More technical details of algorithms in the platform can be found from some of our papers [13]–[15]. An overview of this platform is demonstrated in Section II. The details for the design and implementation of the toolkit as well as the application system are introduced in Sections III and IV. Examples are provided in Section V. Finally, in Section VI, we draw the conclusion with a future perspective.

II. Platform Overview

Our solution for the platform of medical image processing and analyses is illustrated in Fig. 1. We integrate mainstream algorithms of structural imaging, functional imaging, and molecular imaging within a uniform computational framework to produce a powerful algorithm toolkit on which an advanced application system can be developed conveniently. The implemented software platform consists of two layers, as shown in Fig. 2. The algorithm layer, MITK, [1] is the kernel that provides all the algorithms and the basic visualization and interaction framework for medical image processing. The application layer, which includes 3DMed and a plugin SDK, provides a flexible and extensible application framework for the total platform.

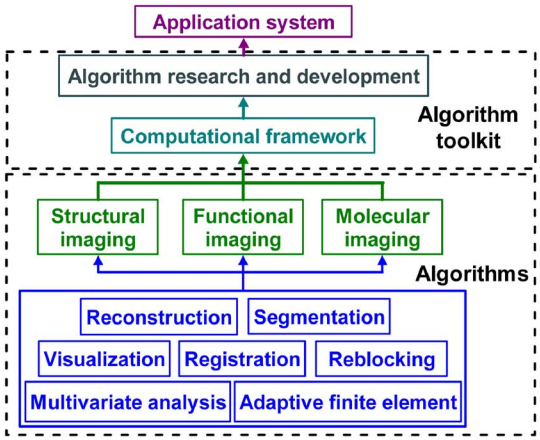


Fig. 1. Solution for the platform of medical image processing and analyses.

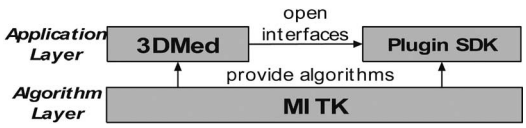


Fig. 2. Total framework of the platform.

III. MITK

We have developed a new algorithm toolkit, i.e., MITK, to attempt the problems of VTK and ITK mentioned in Section I. MTIK is not based on VTK and ITK, but is a novel consistent toolkit that provides the function of reconstruction, segmentation, registration, visualization, etc. Some excellent features of VTK and ITK are used for reference, while the whole framework and underlying medical image processing algorithms of MITK are designed, implemented, and optimized completely by ourselves. Table I presents the comparison of MITK with VTK and ITK. It is obvious that MITK enriches the available toolkits and provides another great option for the medical imaging society.

A. Overall Design

1) Design Goals: MITK pursues clear high level design goals throughout [16].

a) *Consistent design style*: Design patterns [17] are used intensively to get a consistent, flexible, and reusable overall framework.

[1]MITK is novel and completely different from the Medical Imaging Interaction Toolkit (MITK) that was developed by German Cancer Research Center based on VTK and ITK.

TABLE I
COMPARISON OF MITK WITH VTK AND ITK

Feature\ Platform	VTK	ITK	MITK
Specialism	General visualization	Segmentation and registration of medical image	Medical image processing and analyzing for structural imaging functional imaging and molecular imaging
Framework	No Out-of-Core support	No Out-of-Core support	With underlying processing framework for Out-of-Core datasets
Main algorithms	Visualization	Segmentation and registration	Reconstruction, segmentation, registration, visualization, reblocking, multivariate analysis, adaptive finite element
Program style	C++ with object-oriented method	C++ with Generic Programming	C++ with object-oriented design method, and design pattern method

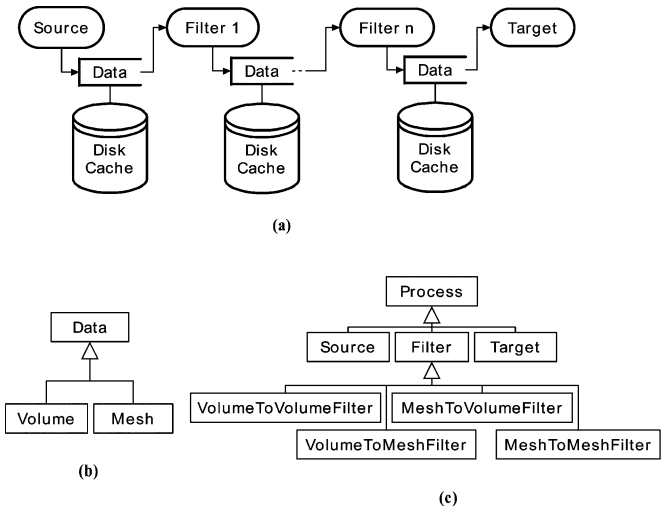


Fig. 3. Computational framework of MITK. (a) Pipeline of data and algorithms. (b) Data model. (c) Algorithm model.

b) *Focused goals*: The specific domain of medical imaging is targeted.
c) *Portability*: It is written in the American National Standards Institute (ANSI) C++ and the system -dependent codes are as separated and minimized as possible.
d) *High performance*: Hardware-accelerated algorithms are used for medical image visualization.

2) Computational Framework: Similarly to VTK and ITK, MITK employs the data flow model to design the computational framework. Particularly, MITK applies a more simplified and traditional model to generate a small and consistent overall framework.

As shown in Fig. 3(a), medical data are abstracted to a Data class, while a medical processing algorithm is abstracted to a Filter class that receives input data and generates output data. A series of algorithms can be connected into a pipeline and form a consistent computational framework. This is basically the same as the visualization model implemented in VTK, except that MITK does not provide the support of network topology, network feedback, and network execution. During the pipeline, every Filter executes immediately after it is member function *Run*() is called. Each Data is also connected to the disk cache and encapsulates the operations of data exchange between internal and external memory. It gives the capability of processing out-of-core data to this computational framework, which is not supported by most mainstream algorithm toolkits yet.

The meanings of Data, Source, Filter, and Target in Fig. 3(a) are explained in detail as follows.

a) *Data*: Data abstracts the attributes and methods of medical image data. We can generate the concrete data class derived from Data for different types of medical image data, and the implementation details for out-of-core data accesses are well encapsulated in these subclasses; therefore, each medical processing algorithm only needs to call the unified application programming interfaces (APIs) defined in their superclass to access the medical data and does not need to know what kind of dataset (in-core or out-of-core) it is processing.
b) *Source*: Source is a type of algorithm. Representing the source of one algorithm pipeline, it only produces output data. The purpose of Source is to generate the initial Data to start the execution of the whole pipeline. The examples include reading data from disk and generating data by using certain algorithms.
c) *Filter*: Filter is a type of algorithm. Representing the data processing algorithm, it has both input and output data. Most algorithms in medical image processing and analyses can be expressed as a Filter.
d) *Target*: Target is another type of algorithm that represents the end of one algorithm pipeline. It only receives input data. The purpose of Target is to put the final Data to appropriate location and finish the execution of the whole pipeline. The examples include writing the final result to disk files or displaying the final result on the screen.

Considering the characteristic of data processed by the algorithms, we specify the data model of MITK, as shown in Fig. 3(b). Volume and Mesh are two concrete subclasses of Data and represent two different kinds of data, respectively.

Volume is a concrete data class to demonstrate the medical image data obtained by imaging devices. It provides an abstract for the multidimensional (1, 2, 3), multimodal (CT, MRI), and regular dataset. The internal data and attributes are exposed to the algorithm object through the interface of Volume. Volume is one of the kernel classes in MITK.

Mesh is a concrete data class to represent geometrical data. It provides an abstract for 1-D lines, 2-D vector graphics, and 3-D triangular meshes. Mesh does not directly correspond to medical image data, but is an intermediate result generated by certain algorithms. For the efficiency of Mesh processing algorithms, a

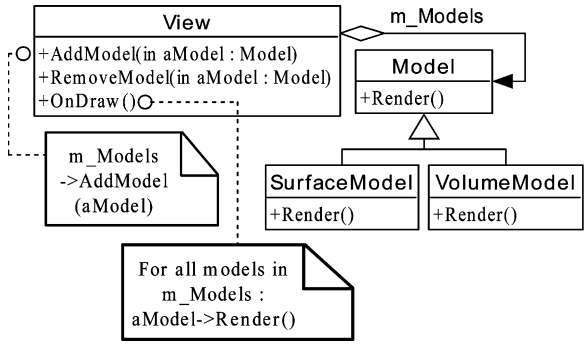


Fig. 4. Rendering framework of MITK.

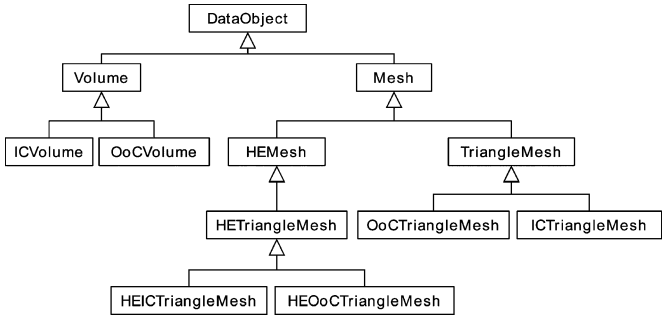


Fig. 5. Inheritance hierarchy of the data classes.

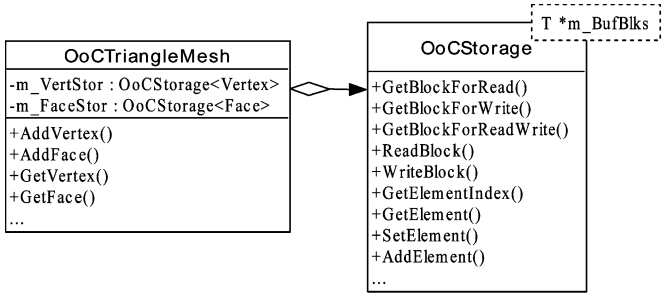


Fig. 6. Out-of-core storage.

half-edge is used in MITK as an internal data structure of Mesh. The internal data and attributes are exposed to the algorithm object through the interfaces of Mesh. Mesh is also one of the kernel classes in MITK.

Source and Target are two algorithm abstracts with special purpose, while Filter is the main data processing algorithm abstract. According to input and output datasets, we demonstrate the algorithm model in Fig. 3(c). VolumeToVolumeFilter, VolumeToMeshFilter, MeshToMeshFilter, and MeshToVolumeFilter are four abstract subclasses of Filter and represent four different kinds of algorithms, respectively.

VolumeToVolumeFilter is an abstract algorithm class whose input and output data are both Volumes. These algorithms include image processing, segmentation, and registration; VolumeToMeshFilter is an abstract algorithm class which characterizes that input data are Volume and output data are Mesh. These algorithms include surface reconstruction and image segmentation; MeshToMeshFilter is an abstract algorithm class which demonstrates that input and output data are both Meshs. These algorithms include mesh simplification, mesh fairing, mesh subdivision algorithms, etc.; MeshToVolumeFilter is an abstract algorithm class which represents that input data are Mesh and output data are Volume. These algorithms include distance-field-based visualization and implicit surface algorithms. Each of these abstract filters specifies the interfaces that the subclasses must implement.

3) Visualization Framework: The general visualization framework is shown in Fig. 4, in which View displays the result images or 3-D graphics onto screen. View maintains an array of Models and provides the interfaces for adding a Model into the array and removing a Model from the array. In the member function *OnDraw*(), which is called when the View is updated, each model in the array is visited and its virtual function *Render*() is called to display itself to the screen. The concrete classes of Model should implement the virtual function *Render*() to display its contents.

B. Implementation of Key Technologies

The design of the platform involves different technologies, and the implementation of some key technologies are introduced in the following sections.

1) Out-of-Core Support: In order to gain transparent access to the medical datasets, we design a set of uniform interfaces in the data classes (Volume and Mesh) and separate the different implementations into their subclasses for accessing the in-core and out-of-core datasets, respectively. The inheritance hierarchy of the data classes in MITK is shown in Fig. 5. The classes with the prefix "OoC" are designed for containing the out-of-core datasets.

Furthermore, as shown in Fig. 6 (taking OoCTriangleMesh for example), a template class OoCStorage is designed to encapsulate the common functions for the underlying management of the out-of-core raw data, including the buffer management, data transfer between internal and external memory, etc., so as to get a more flexible framework.

2) Volume Rendering Framework: Volume rendering algorithm is one of the most important algorithms in scientific visualization [18], [19]. It is difficult to implement an efficient and flexible algorithm framework because of its complexity and flexibility. Our volume rendering framework is improved from VTK, and is enhanced by integrating the transfer function generation algorithm with multidimensional transfer function support into the framework.

VolumeModel is a concrete subclass of the Model and it is the main component of volume rendering framework. A Volume is visualized by the volume rendering algorithm in the implementation of the *Render* function. The Volume's two subclasses, OoCVolume and ICVolume, encapsulate the access functions for the out-of-core and in-core datasets, respectively. Since there are many kinds of volume rendering algorithms, and many parameters are adjustable, the VolumeModel plays a very important role in the whole volume rendering framework. Its structure is shown in Fig. 7.

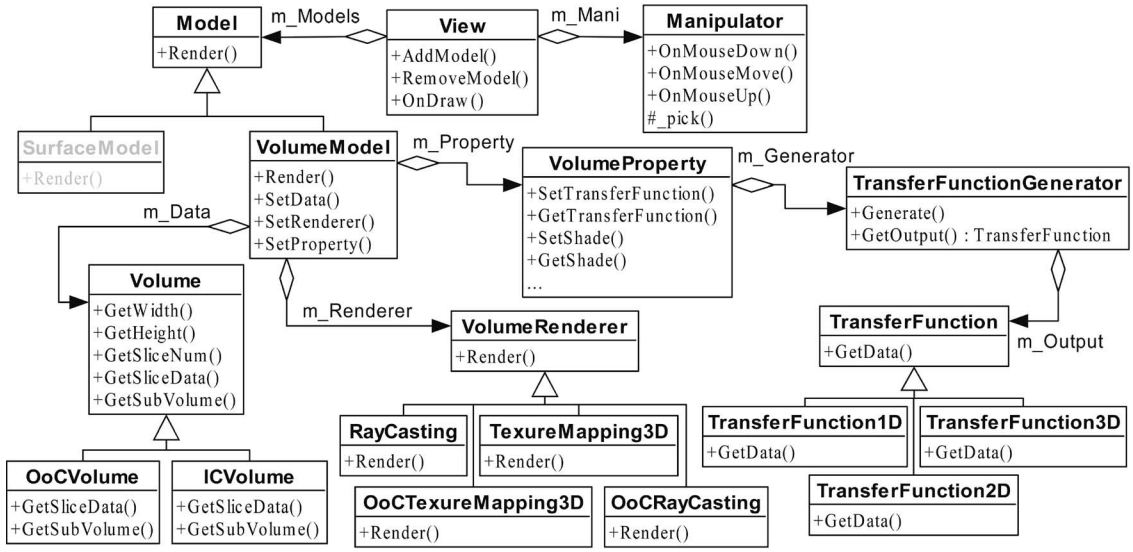


Fig. 7. Volume rendering framework of MITK.

VolumeModel has three class members, i.e., Volume, VolumeProperty, and VolumeRenderer. Volume provides the access of medical image data. VolumeProperty provides the parameters required by volume rendering algorithms, especially the opacity transfer functions. VolumeRenderer provides the actual rendering algorithms. In fact, in the implementation of *Render* function of VolumeModel, the rendering is delegated to the *Render* function of VolumeRenderer.

VolumeRenderer is an abstract class whose concrete subclasses implement different volume rendering algorithms by overriding the virtual function *Render*() defined in the VolumeRenderer. Currently we have implemented the classical ray casting algorithm (RayCasting) [20], splatting algorithm (Splatting) [21], shear warp algorithm (ShearWarp) [22], and the modern Gnutella processing unit (GPU) and texture-mapping-based algorithms (TextureMapping3D) [23], [24]. We have also developed some new out-of-core algorithms for the volume rendering of very large datasets (e.g., OoCRayCasting [15], OoCTexureMapping3D, etc.). New algorithms can be added easily by deriving new classes from VolumeRenderer and overriding the *Render* virtual function.

In addition to the parameters of shading, the VolumeProperty mainly offers a flexible framework to provide VolumeRenderer the parameters of an opacity transfer function. To approach this, VolumeProperty has a class member of TransferFunctionGenerator, which is an abstract class for generating different transfer functions. The output of TransferFunctionGenerator is an object of TransferFunction that includes TransferFunction1D, TransferFunction2D, and TransferFunction3D as three concrete subclasses to support the multidimensional transfer function [25]. For this purpose, VolumeRenderer must be aware of the dimension of transfer function and deal with it specifically.

3) 3-D Interaction: The entire interaction framework of our platform is based on 3-D widgets [26]–[29]. In our platform, WidgetModels represent the 3-D widgets and act as the kernel elements in the interaction framework. According to some issues in designing 3-D widgets advanced in the works of Snibbe *et al.* [30], the design and implementation of WidgetModels should conform to the following rules [13].

a) A WidgetModel's appearance should intuitively reflect the behavior of this WidgetModel without losing its precision. For example, the line widget should take no balls but cones as its control points, so as to precisely locate the ends of the line.
b) Keep the responses of WidgetModels the same as the user expects so as to control the widgets conveniently. For example, keep the control point of the widget following the cursor of the mouse when it is being used.
c) Design the WidgetModel in a set of uniform interfaces and reduce the coupling with other modules so as to simplify the maintenance and extension of the whole framework.

In addition to the WidgetModel itself, some other classes are necessary for the framework to accomplish the whole interacting process.

First of all, the DataModels are the objects controlled by widgets. There are some correlative connections between the WidgetModel and the DataModel. Generally, a WidgetModel is always connected to one DataModel (indicated by m_SourceModel, a member variable of the WidgetModel) as it should be. However, a DataModel can be connected to a group of WidgetModels, and they are maintained by m_WidgetModels, an array member of DataModel. These connections do not always exist. For example, a WidgetModel may be connected to no DataModel.

Second, a Manipulator is needed to drive the WidgetModel to control the DataModel. Therefore, the Manipulator must have the ability to select the WidgetModel currently pointed by the mouse from the View and transfer the control to the WidgetModel. Meanwhile, the WidgetModel should provide a

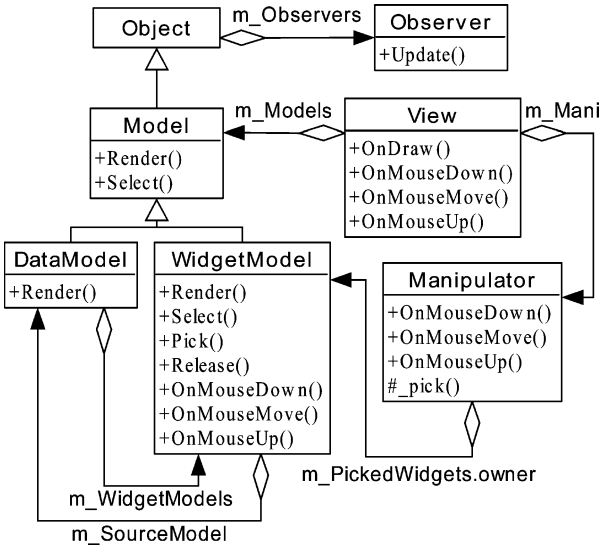


Fig. 8. 3-D interaction framework of MITK.

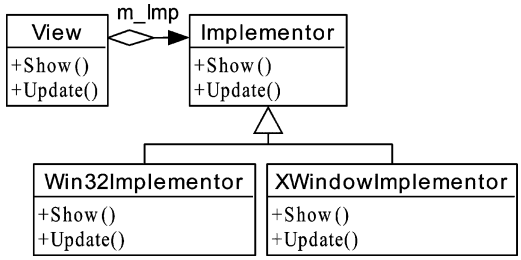


Fig. 9. Implementation of cross-platform.

Select() interface to the Manipulator for selection, and some corresponding interfaces such as *OnMouseDown*(), *OnMouseUp*(), and *OnMouseMove*() to receive and perform the transferred controls. Besides that, the WidgetModel must also provide *Pick*() interface and *Release*() interface to the Manipulator in order to update the WidgetModel's status.

Third, an open Observer module should be available to show the physical parameters obtained from a DataModel by the WidgetModel in appropriate ways. Observer is a highly abstract class and contains only a pure virtual function: *Update*(). Each class derived from Object is able to add multiple Observers (except for the Observer itself), and so does the WidgetModel. Every time when the status changes, *Update*() will be called to inform associated Observers to update the shown data. The actual way to display the data is implemented in *Update*() of the concrete Observer class. Therefore, the whole interaction platform is independent of the user interface (UI) system.

Finally, View constructs a stage for the DataModel and WidgetModel to display. At the same time, View contacts the operating system, captures the messages emitted by mouse and keyboard, and calls the corresponding interface of Manipulator according to different messages so as to start up the whole interaction process.

The earlier modules and their cooperation make up the entire 3-D human–computer interaction framework, as shown in Fig. 8.

In Fig. 8, we can clearly illustrate the whole interaction process as follows.

1) View displays each Model in the 3-D scene, captures mouse and keyboard messages, and drives Manipulator upon different messages.
2) At an appropriate occasion [e.g., *OnMouseDown*() is called], the Manipulator performs the select operation. If certain WidgetModels are chosen, the Manipulator transfers the control to them; otherwise, it does the routines.
3) The selected WidgetModel performs as defined before, and the final results are reflected in View and Observers.

The whole process of interaction is the continuous repetition of 1), 2), and 3).

4) Cross-Platform: All the codes are written in ANSI C++, thus ensuring portability. Furthermore, for some operating-system-specific codes, such as windows management, event processing, and virtual memory management, we must write one set of codes for every operating system. In order to get an elegant solution to encapsulate these system-specific codes, we use Bridge as the design pattern.

In our platform, View is the only class that depends on the GUI. It provides a screen window to display the images or 3-D graphics, which is definitely based on the specific operating system. The system-specific codes should be separated, and adding the support of a new operating system should not affect the client codes. To achieve this purpose, the structure of View is designed as in Fig. 9.

View maintains a pointer of Implementor and delegates all the system-specific codes to the Implementor. Implementor is an abstract class and its concrete subclasses, such as Win32Implementor and XWindowImplementor, override the virtual functions defined in Implementor to implement the system-specific parts by calling the API functions of a specific operating system. In this structure, clients only contact View, and they do not know the existence of the Implementor. When adding support for a new operating system, we only need to derive a subclass from the Implementor and do not need to change the client codes.

5) Memory Management: In the design of MITK, we must carry out an efficient memory management scheme as an infrastructure to maintain the stability and robustness of the toolkit. For the data object, both Volume and Mesh support the loading and the access of out-of-core dataset by using the operating-system-provided memory-mapped file and manage the virtual memory directly. Another level of memory management is to ensure that memory leakage does not occur during run-time. We employed two design patterns, smart pointer and reference counting, to guarantee that the memory of an MITK object is deleted when it is no longer necessary for any other MITK object. In addition, we also implement a simple garbage collection mechanism to ensure that all the MITK objects are deconstructed at the end of one application.

IV. 3DMED

A. Overall Design

1) Design Goals: There are high-level design goals of 3DMed [31].

a) *Supports cross-platform*: It is based on cross-platform libraries and coded with ANSI C++.
b) *Powerful extensibilities*: It provides a flexible plugin framework. Users can develop their own plugins according to the specifications defined by 3DMed.
c) *Easy to obtain*: It is released as freeware and can be downloaded from the Internet freely.

2) Main Functions: The basic functions of 3DMed include data input/output (I/O), 2-D manipulation, medical image segmentation and registration, 3-D visualization and measurement, virtual cutting, etc. The 2-D manipulation, virtual cutting, and 3-D measurement are implemented in kernel and relatively settled. The kernel also involves the basic visualization functions including surface rendering and volume rendering. Other functions are dynamically loaded through plugins. Each function is briefly introduced as follows.

a) *Medical data I/O*: Adopt plugin mechanism to support various image data types, i.e., BMP, JPEG, TIFF, DICOM, etc. The input or output function of each data type is in absolute plugins and can be loaded at any time.
b) *2-D manipulation*: Including image browsing, animation playing, window width/height adjusting, geometric transforming, pseudocolor displaying, 2-D measuring, marking, etc.
c) *Medical image segmentation*: Use plugin mechanism to provide various segmentation algorithms. Users can add new algorithms by their own plugins.
d) *Medical image registration*: Use plugin mechanism and provide a modularized framework to assemble new registration algorithms.
e) *Surface rendering*: Use an enhanced marching cubes algorithm (based on segmentation) to get iso-surfaces and render them by OpenGL with hardware acceleration. The functions are implemented in MITK.
f) *Volume rendering*: Use the volume rendering algorithm based on ray casting implemented in MITK and provide a set of wieldy interfaces to adjust transfer functions.
g) *Virtual cutting*: Support virtual cutting in both surface rendering and volume rendering with arbitrary planes, which enable users to have a clear view of the tissue and organs.
h) *3-D measurement*: Based on 3-D interaction framework implemented in MITK and provide an intuitive and direct manipulation of 3-D objects.

B. Implementation of the Plugin Framework

3DMed is a large and complicated system that involves medical image segmentation, registration, visualization, etc. Each part itself is a sophisticated system with different algorithms and various adjustable parameters. Therefore, 3DMed employs a plugin mechanism to reduce the coupling among its different modules and provides users an open, extensible framework.

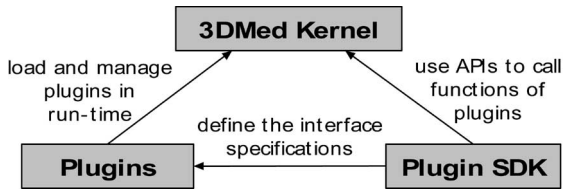


Fig. 10. Plugin framework of the application system.

Plugin is a dynamic link library written with certain specifications and can be dynamically loaded by a 3DMed kernel. To implement the plugin mechanism, three necessary parts must cooperate together, as shown in Fig. 10. 3DMed kernel is the core of the system with the responsibility of loading, managing, and calling each plugin. Plugin SDK provides the interface specifications for writing plugins. With these obligatory interfaces, the Plugins that is the actual plugin for practical functions can be identified and loaded by the kernel.

1) Implementation of the Plugin SDK: Plugin SDK plays an important role in the whole framework. It not only provides the call interfaces of plugins for the system kernel, but also defines the interface specifications for the developers to write their own plugins. To achieve these purposes, 3DMed must expose its internal data to the outer user through certain interfaces. Two kinds of data type, medVolume and medMesh, which represent the volume data and mesh data, respectively, expose the actual data and attributes to the plugin developers by some *Get* functions.

Furthermore, all the plugins of 3DMed are divided into five categories in plugin SDK: I/O plugins, filter plugins, registration plugins, segmentation plugins, and visualization plugins, as shown in Fig. 11. They are all rooted in the same base class, medPlugin, and the actual functions are implemented in the overrided virtual function *Show*(). I/O plugins provide input and output functions for reading and writing data in different formats. For the input plugin, it uses a *GetOutput* function to return the loaded data to the kernel in the form of medVolume or medMesh. For the output plugin, it uses a *SetInput* function to get data from kernel and write them out. Filter plugins provide filter algorithms for medical data. They get data from kernel through *SetInput* function, process the data in *Show*() function, and return the results to the kernel through *GetOutput* function. According to different data types, there are two kinds of filter plugins: medVolumeFilterPlugin and medMeshFilterPlugin. Most image processing algorithms and digital geometry algorithms can be assembled into 3DMed via these two kinds of plugins. The rest segmentation plugins, registration plugins, and visualization plugins directly correspond to various segmentation, registration, and visualization algorithms, respectively.

2) Implementation of the Plugins: A plugin is a dynamic link library that is functionally associated to a certain algorithm. The precondition of implementing a practical plugin is to confirm the correct category. For example, a plugin for reading a series of bitmap (BMP) image files should be derived from medVolumeImportPlugin with the reading function implemented in the virtual function *Show*().

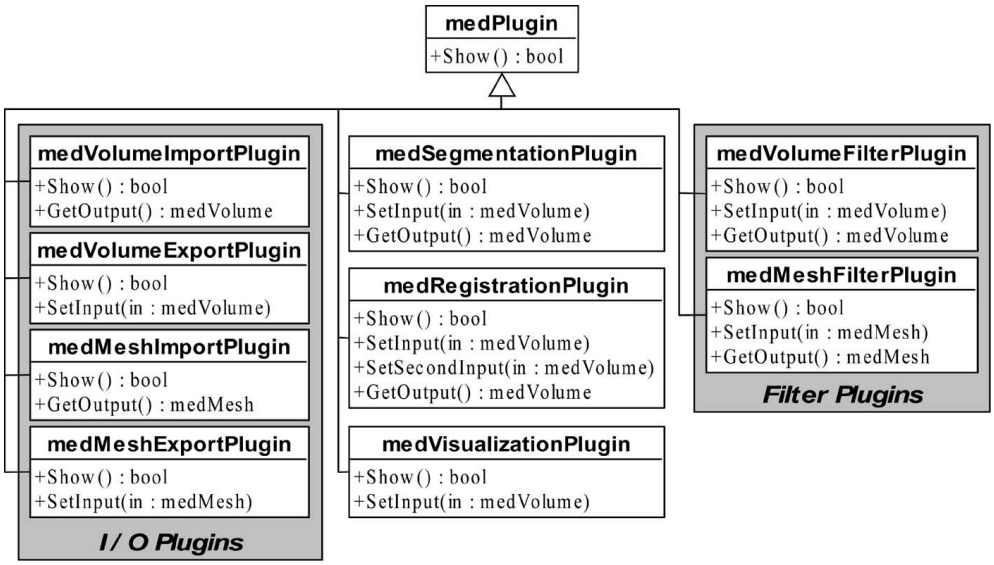


Fig. 11. Class diagram of plugin SDK.

In addition, each plugin must provide a C function *MakePlugin* for the kernel to create its instance dynamically. For the plugin of reading BMP files mentioned before, its *MakePlugin* function is declared and implemented as follows:

```
medVolumeImportPlugin* MakePlugin()
{ return new medBMPImportPlugin; }
```

3) Implementation of the Kernel: The task of 3DMed kernel is to load various plugins dynamically at run-time, create corresponding menu items, call the function of the plugin when its menu item is activated, and clean up the loaded plugins when the system exits.

To dynamically load plugins, each plugin must provide necessary information for the kernel to identify and create a corresponding menu item. In 3DMed, each plugin must provide following C functions for the kernel:

```
const char* GetTypeName();
const char* GetClassname();
const char* GetMenuDescription();
```

Among these functions, *GetTypeName*() gives the type of the plugin; *GetClassname*() gives theclass name of this plugin; and *GetMenuDescription*() gives the caption of the corresponding menu item. In order to simplify the process of developing plugins, plugin SDK provides a macro to generate these functions including *MakePlugin*() automatically.

After loading the plugins, 3DMed kernel uses the design pattern called abstract factory to manage them in a flexible and graceful manner. As shown in Fig. 12, medFactory is a template class and the template parameter T can be one of the nine plugin categories. Inside medFactory, a map is used to manage plugins, and the name of the plugin is used as the reference key. *Add* function registers a new plugin to the factory; *Create* function gets the plugin's actual pointer from its name; and *Clear* function destroys all the registered plugins.

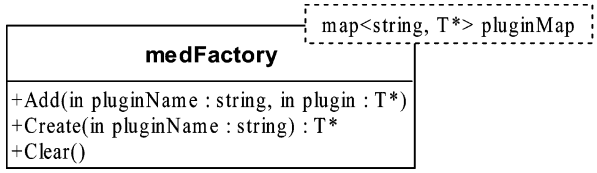


Fig. 12. Plugins factory in 3DMed kernel.

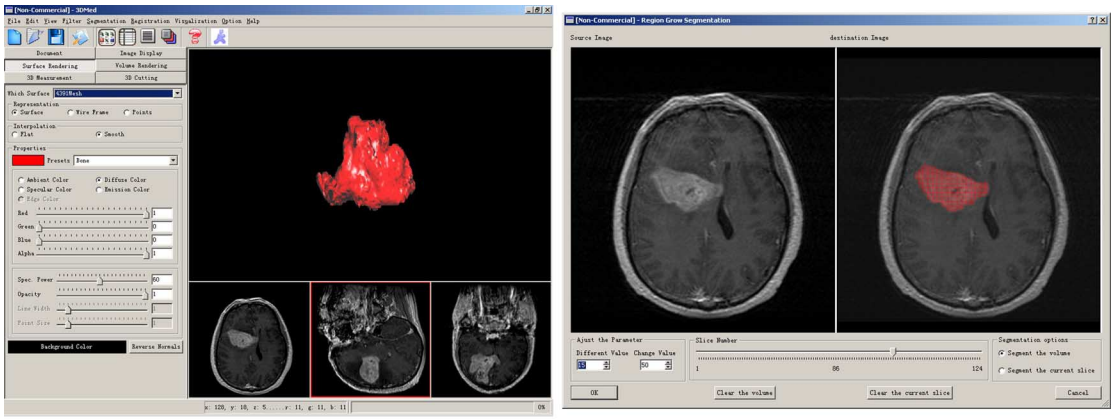
The pseudocodes in Algorithm 1 give the flow of loading and managing various plugins at run-time:

Algorithm 1: Loading and managing various plugins at run-time.

```
foreach file in plugin directory do
    if the plugin functions are found in this file then
        typeNameStr ← GetTypeName();
        classNameStr ← GetClassName();
        menuDescriptionStr ← GetMenuDescription();
        Decide pluginType according to typeNameStr;
        pluginType newPlugin ← MakePlugin();
        medFactory⟨pluginType⟩::Add(classNameStr,
        newPlugin);
        Make dynamic menu item with the title
        menuDescriptionStr;
        Set the message handler of the menu item to
        newPlugin::Show();
    end
end
```

V. Experiment Examples

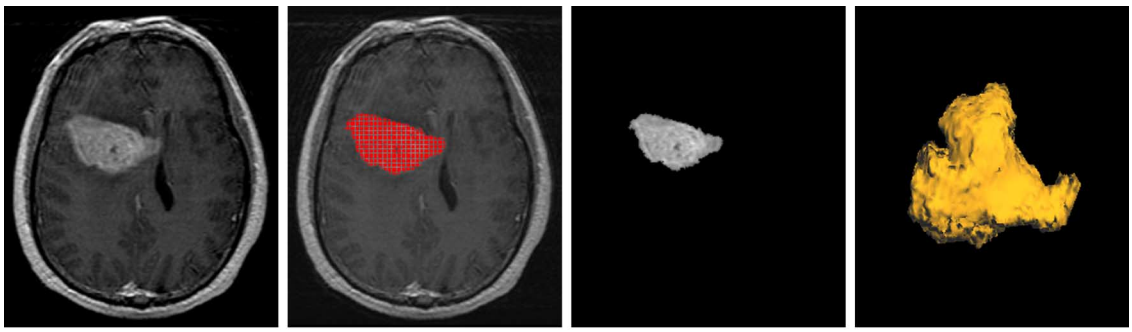
Fig. 13 shows the GUIs of 3DMed. Figs. 14– 18 demonstrate some experiment examples using our platform, including segmentation, surface rendering, volume rendering, and 2-D/3-D



(a) (b)

(c)

Fig. 13. GUIs of 3DMed. (a) Main GUI. (b) Segmentation GUI. (c) Registration GUI.

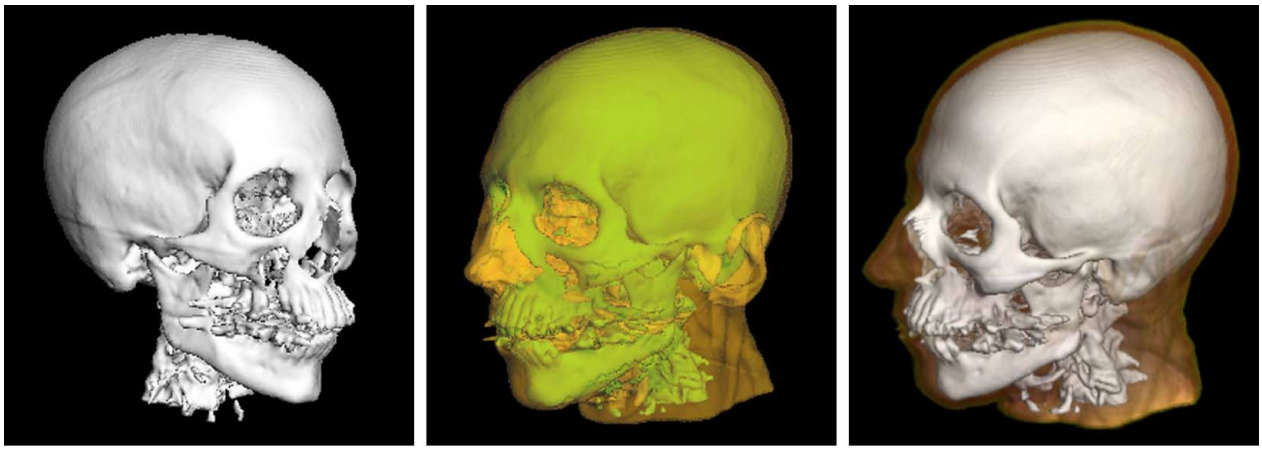


(a) (b) (c) (d)

Fig. 14. Application examples of segmentation. (a) Original image. (b) Segment the focal lesion using Region Grow Segmentation Algorithm. (c) Result of segmentation. (d) Surface reconstruction result of focal lesion.

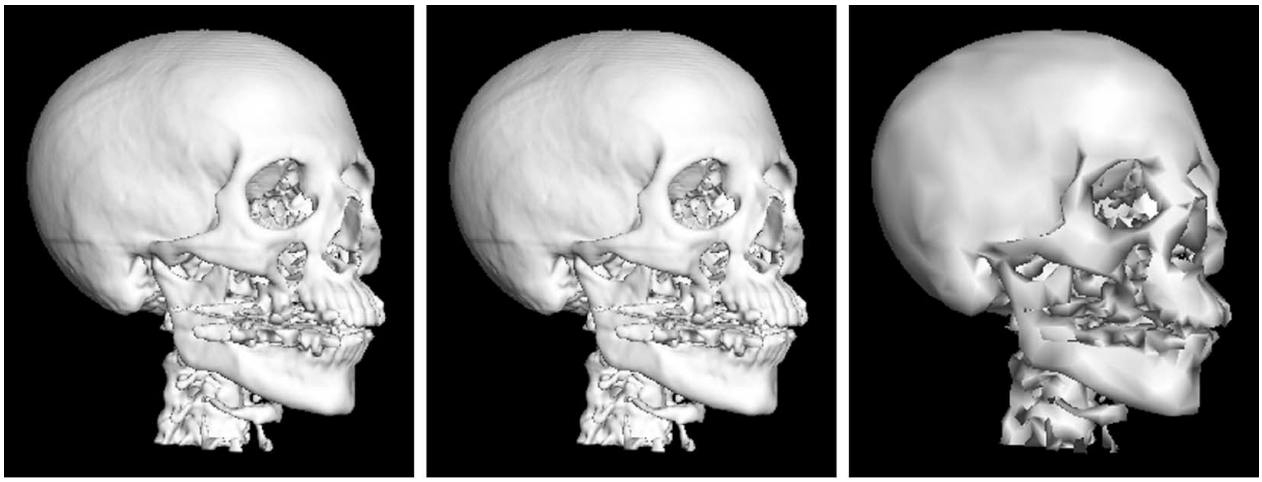
interaction with 3-D widgets. Fig. 19 illustrates some results of the out-of-core volume rendering algorithms developed by us based on MITK, and the test data are the Chinese visible human CT dataset. It contains 1714 slices, and each slice has a resolution of 512 × 512 pixels with 16 bits, which totally occupies 857 MB storage space and can hardly be handled by in-core methods. All the tests are run on a Windows-PC with an Intel Pentium 4 2.8 GHz processor and 1 GB physical memory.[2]

[2]The original data of Figs. 15 and 16 are from http://www.psychology.nottingham.ac.uk/staff/cr1/ct.zip; the original data of Fig. 13(c) are from http://www.isi.uu.nl/Research/Databases/DRIVE; the original data of Figs. 13(a) and (b), 14, 17, and 18 are from Beijing Shougang Hospital; the original data of Fig. 19 are from the Southern Medical University of China.



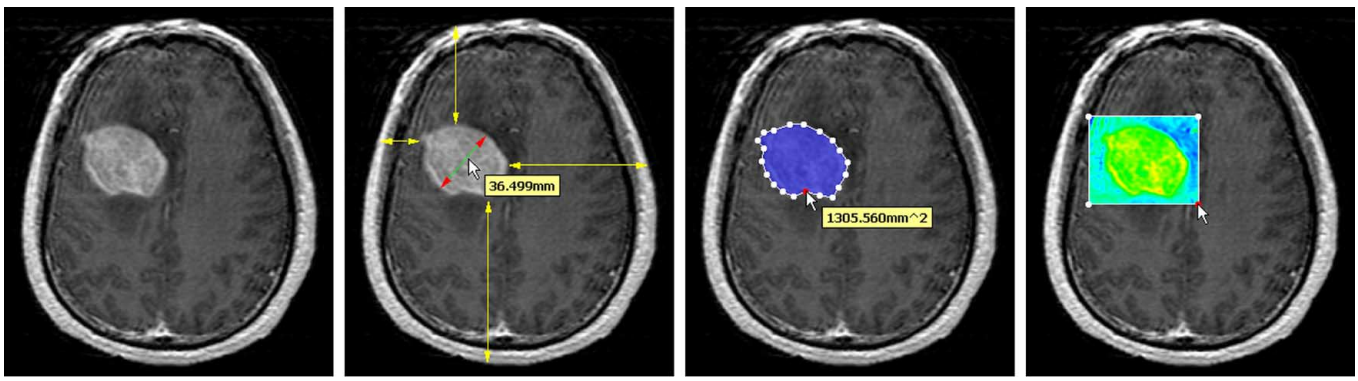
(a) (b) (c)

Fig. 15. Application examples of surface rendering and volume rendering. (a) Surface rendering. (b) Multiple surface rendering. (c) Volume rendering.



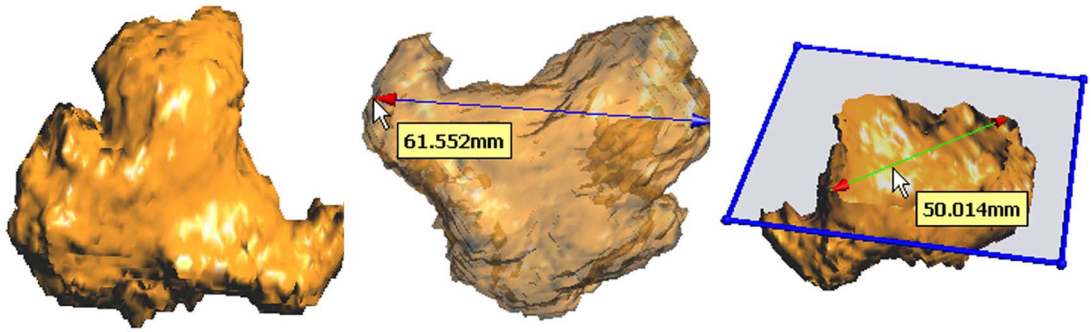
(a) (b) (c)

Fig. 16. Application examples of mesh simplification. (a) Original mesh (406 894 vertices and 813 996 triangles). (b) Simplified to 49 896 vertices and 100 000 triangles. (c) Simplified to 4896 vertices and 10 000 triangles.



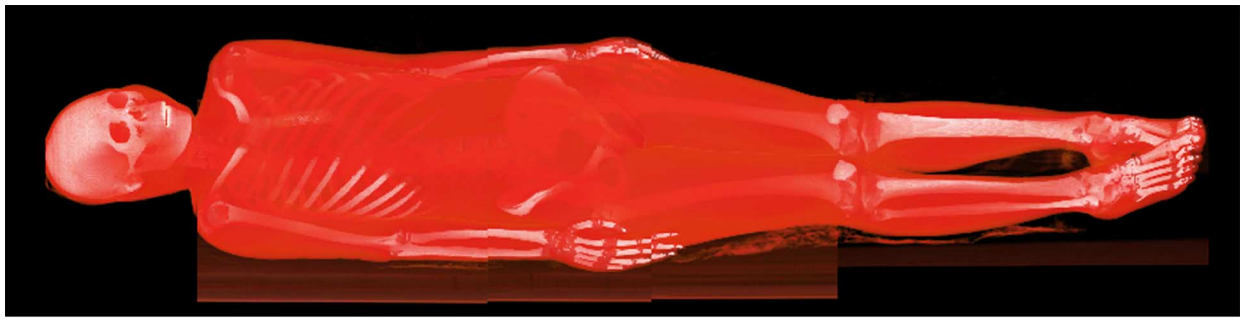
(a) (b) (c) (d)

Fig. 17. Application examples of 2-D widgets. (a) Original image. (b) Locate the position and measure the size of focal lesion. (c) Measure the area of focal lesion. (d) Display focal lesion with pseudo color.

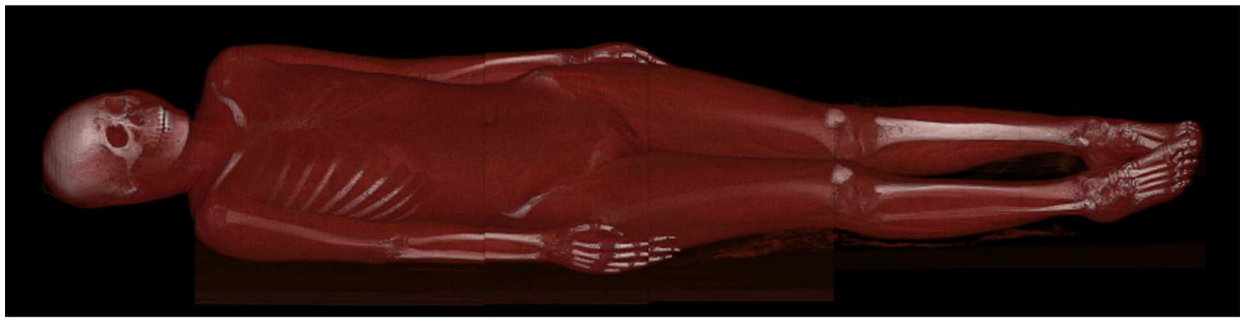


(a) (b) (c)

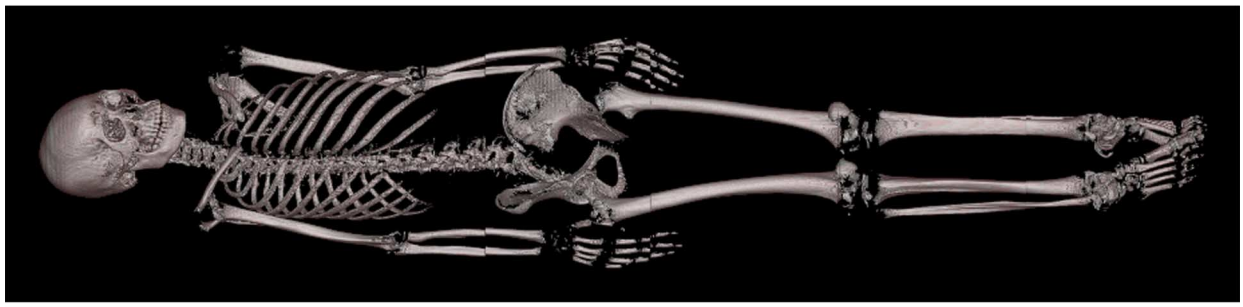
Fig. 18. Application examples of 3-D widgets. (a) 3-D model. (b) Measure the size using LineWidgetModel3D. (c) Measure the section plane using LineWidgetModel3D and ClippingPlaneWidgetModel.



(a)



(b)



(c)

Fig. 19. Some results of the texture-based out-of-core volume rendering algorithms in MITK 2.0. The test data are the Chinese visible human CT dataset (512 $\times$ 512 $\times$ 1714 $\times$ 16 bit, 857 MB). (a) Shading off rendering time: 5.50 s, memory buffer for data: 7.06 MB. (b) Shading on rendering time: 66.2 s, memory buffer for data: 9.41 MB. (c) Shading on rendering time: 14.6 s, memory buffer for data: 9.41 MB.

VI. Conclusion and Future Perspective

This paper introduces a novel software platform that is investigated and developed specially for medical image processing and analyses. The design goals, overall framework, and implementation of a few key technologies are discussed in detail. This platform is designed and implemented independently, not based on other toolkits such as VTK and ITK. It integrates the mainstream algorithms in medical imaging that includes structural imaging, functional imaging, and molecular imaging within a consistent framework. Furthermore, comparing with existing popular medical imaging platforms, this platform provides an underlying processing framework for out-of-core datasets. Using this platform, researchers need not additionally design special data accessing operations. During the platform, MITK is indeed a powerful algorithm toolkit. Besides, it can also be combined with 3DMed to produce an extensible application system for medical image processing and analyzing. Such a seamless platform structure provides an effective method for developing robust medical imaging software.

Currently, we have released MITK 2.0 version (with out-of-core support) at http://www.mitk.net, which can be downloaded free of charge. MITK and 3DMed have exceeded 9000 downloads till now. Users consist of researchers, clinicians, engineers, students, etc., who are distributed worldwide. The real-time download statistics can be found from http://www.mitk.net/downstat.php. Large amount and scope of uses fully testify the validity of our proposed platform. Besides being used for teaching and scientific research, MITK and 3DMed are also employed for clinical applications and engineering projects, such as assistance diagnosis of retinal disease, freehand 3-D ultrasound imaging [14], surgery programming and navigation for ultrasound guided tumor melting, processing, and analyzing for lossless defect detection in industrial CT, etc. These practical applications commendably demonstrate the flexibility of this new platform.

In the future, we will continue to improve the performance and efficiency of the platform; meanwhile, more practical algorithms need to be added such as interactive segmentation and faster visualization, and this will make our platform a valuable tool for the medical imaging society.

References

[1] A. Souza, J. K. Udupa, and P. K. Saha, "Volume rendering in the presence of partial volume effects," *IEEE Trans. Med. Imag.*, vol. 24, no. 2, pp. 223–235, Feb. 2005.

[2] A. Madabhushi and J. K. Udupa, "Interplay between intensity standardization and inhomogeneity correction in MR image processing," *IEEE Trans. Med. Imag.*, vol. 24, no. 5, pp. 561–576, May 2005.

[3] Y. Zhan and D. Shen, "Deformable segmentation of 3-D ultrasound prostate images using statistical texture matching method," *IEEE Trans. Med. Imag.*, vol. 25, no. 3, pp. 256–272, Mar. 2006.

[4] W. Schroeder, K. Martin, and B. L. Schroeder, *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*, 3rd ed. New York: Kitware, Inc., 2003.

[5] W. J. Schroeder, L. S. Avila, and W. Hoffman, "Visualizing with VTK: A tutorial," *IEEE Trans. Comput. Graph. Appl.*, vol. 20, no. 5, pp. 20–27, Sep.–Oct. 2000.

[6] W. J. Schroeder, K. M. Martin, and W. E. Lorensen, "The design and implementation of an object-oriented toolkit for 3-D graphics and visualization," in *Proc. IEEE Vis. Conf.*, Nov. 1996, pp. 93–100.

[7] L. Ibanez, W. Schroeder, L. Ng, and J. Cates, *The ITK Software Guide: The Insight Segmentation and Registration Toolkit*, (version 1.4). New York: Kitware, Inc., 2003.

[8] J. Udupa, R. Goncalves, K. Iyer, S. Narendula, D. Odhner, S. Samarasekera, and S. Sharma, "3DVIEWNIX: An open, transportable software system for the visualization and analysis of multidimensional, multimodality, multiparemetric images," *Proc. SPIE*, vol. 1897, pp. 47–58, 1993.

[9] I. Wolf, M. Vetter, I. Wegner, T. Bottger, M. Nolden, M. Schobinger, M. Hastenteufel, T. Kunert, and H. Meinzer, "The medical imaging interaction toolkit," *Med. Image Anal.*, vol. 9, pp. 594–604, 2005.

[10] Volview. Kitware. (1999). [Online]. Available: http://www.kitware.com/products/volview.html

[11] J. Tian, L. Yang, and J. Hu, "Recent advantages in the data analysis method of functional magnetic resonance imaging and its applications in neuroimaging," *Prog. Nat. Sci.*, vol. 16, no. 8, pp. 785–795, 2006.

[12] Y. Lv, J. Tian, W. Cong, G. Wang, J. Luo, W. Yang, and H. Li, "A multilevel adaptive finite element algorithm for bioluminescence tomography," *Opt. Exp.*, vol. 14, no. 18, pp. 8211–8223, 2006.

[13] J. Xue, J. Tian, and M. Zhao, "Three-dimensional human computer interaction based on 3d widgets for medical data visualization," in *Proc. SPIE Med. Imag. 2005*, vol. 5744, pp. 697–706.

[14] Y. Dai, J. Tian, J. Xue, and J. Liu, "A qualitative and quantitative interaction technique for freehand 3-D ultrasound imaging," in *Proc. EMBS'06.* New York: IEEE Engineering in Medicine and Biology Society, 2006, pp.2750–2753.

[15] J. Xue, J. Tian, J. Chen, and Y. Dai, "An efficient out-of-core volume ray casting method for the visualization of large medical data sets," in *Proc. SPIE Symp. Med. Imag, 2007*, San Diego, CA, pp. 650920-1–650920-8.

[16] 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. SPIE Med. Imag.*, 2004, vol. 5367, pp. 39–47.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns, Elements of Reusable Object-Oriented Software*. Upper Saddle River, NJ: Pearson Education, 1994.

[18] C. R. J. Charles and D. Hansen, *The Visualization Handbook*. New York: Elsevier Academic Press, 2004.

[19] K.-L. Ma, E. B. Lum, and S. Muraki, "Recent advances in hardware-accelerated volume rendering," *Comput. Graph.*, vol. 27, pp. 725–734, 2003.

[20] M. Levoy, "Display of surfaces from volume data," *IEEE Trans. Comput. Graph. Appl.*, vol. 8, no. 3, pp. 29–37, May 1988.

[21] L. Westover, "Footprint evaluation for volume rendering," in *Proc. ACM SIGGRAPH'90*. New York: ACM Press, pp. 367–376.

[22] P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation," in *Proc. SIGGRAPH'94*. New York: ACM, pp. 451–458.

[23] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl, "Interactive volume rendering on standard PC graphics hardware using multi-textures and multi-stage rasterization," in *Proc. ACM SIGGRAPH/ EUROGRAPHICS Workshop Graph. Hardware*. New York: ACM, 2000, pp. 109–118.

[24] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proc. ACM SIGGRAPH/ EUROGRAPHICS Workshop Graph. Hardware*. New York: ACM, 2001, pp. 9–16.

[25] J. Kniss, G. Kindlmann, and C. Hansen, "Multidimensional transfer functions for interactive volume rendering," *IEEE Trans. Vis. Comput. Graph.*, vol. 8, no. 3, pp. 270–285, Jul.–Sep. 2002.

[26] B. D. Conner, S. S. Snibbe, K. P. Herndon, D. C. Robbins, R. C. Zeleznik, and A. van Dam, "Three-dimensional widgets," in *Proc. Interact. 3D Graph. Symp.*, 1992, pp. 183–188.

[27] M. P. Stevens, R. C. Zeleznik, and J. F. Hughes, "An architecture for an extensible 3d interface toolkit," in *Proc. UIST'94*. New York: ACM, pp. 59–67.

[28] D. A. Bowman, E. Kruijff, J. J. LaViola, and I. P. Jr., "An introduction to 3-d user interface design," *Presence*, vol. 10, no. 1, pp. 96–108, 2001.

[29] R. W. Lindeman, J. L. Sibert, and J. N. Templeman, "The effect of 3-D widget representation and simulated surface constraints on interaction in virtual environments," in *Proc. Virtual Real. Ann. Int. Symp*. Piscataway, NJ: IEEE Press, 2001, pp. 141–148.

[30] S. S. Snibbe, K. P. Herndon, D. C. Robbins, B. D. Conner, and A. van Dam, "Using deformations to explore 3-D widget design," in *Proc. SIGGRAPH'92,* New York: ACM, pp. 351–352.

[31] M. Zhao, J. Tian, J. Xue, and X. Zhu, "3DMed: An integrated 3-D medical image processing and analyzing system," presented at the Proc. RSNA'04, San Francisco, CA, Sep.

Jie Tian (M'03–SM'03) received the Ph.D. degree (with honors) in artificial intelligence from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1992.

During 1995–1996, he was a Postdoctoral Fellow at the Medical Image Processing Group, University of Pennsylvania. Since 1997, he has been a Professor in the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences. His current research interests include medical image process and analysis, and pattern recognition. He is the author or coauthor of more than 60 research papers published in the international journals and conferences.

Dr. Tian is the reviewer of mathematical reviews of the American mathematical society, the Director of the Special Committee of Pattern Recognition and Machine Intelligence of the Chinese Society of Automation, the Beijing Chapter Chair of the Engineering in Medicine, and the Biology Society of the IEEE.

Jian Xue received the Ph.D. degree in computer applied technology from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2007.

He is currently with the College of Computing and Communication Engineering, Graduate University of Chinese Academy of Sciences, Beijing. Since 2003, he has participated in the development of Medical Imaging ToolKit (MITK) and 3-Dimensional Medical Image Processing and Analyzing System (3DMed). His current research interests include out-of-core processing and visualization of large medical datasets.

Yakang Dai received the B.S. degree in electric engineering from the University of Hunan, Hunan, China, in 2004.

He is currently the Team Leader of the Medical Imaging ToolKit (MITK) in the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing, China. Since 2004, he has participated in the development of MITK and 3-Dimensional Medical Image Processing and Analyzing System (3DMed). His current research interests include freehand ultrasound imaging and 3-D visualization.

Jian Chen received the B.S. degree in electric engineering from Beijing Normal University, Beijing, China, in 2003. He is currently working toward the Ph.D. degree in computer science at the Institute of Automation, Chinese Academy of Sciences, Beijing.

Since September 2007, he has been a visiting student of biomedical engineering at Columbia University, New York. His current research interests include cardiac image segmentation and multimodality registration.

Jian Zheng received the B.S. degree in automation from the University of Science and Technology of China, Beijing, China, in 2005.

He is currently a team member in Medical Imaging ToolKit (MITK) in the Medical Image Processing Group, Institute of Automation, Chinese Academy of Sciences, Beijing. Since 2005, he has participated in the development of MITK and 3-Dimensional Medical Image Processing and Analyzing System (3DMed). His current research interests include medical image registration.