viewitLib – A framework for the development of software for medical images visualization

C. Buchart Izaguirre\textsuperscript{1}, Á. Bertelsen Simonetti\textsuperscript{1}, D. Borro Yáñüez\textsuperscript{1}

\textsuperscript{1} Department of Applied Mechanics, CEIT, San Sebastián, Spain, \{cbuchart, abertelsen, dborro\}@ceit.es

Abstract

Nowadays, many medical imaging packages exist, but most of them are so big that have become very difficult to use by physicians, even for the simplest tasks. There are also several visualization libraries specifically designed for scientific and medical images visualization, but they require a lot of work to create professional applications. In this work we present a new programming framework to help in the design and development process of applications that make intensive use of visualization of medical images.

1. Introduction

There are many software packages for medical image visualization and analysis that cover a high spectrum of visualization tools, support many image formats and even can be extended through the use of plug-ins. Some of the best examples are 3D Slicer \cite{1,2} and OsiriX \cite{3}. However, such software packages have been designed for general purpose visualization and to fulfill many analysis tasks producing a lot of functions are never used by physicians and an unnecessarily complicated workflow.

On the developer side, several libraries have been written in order to provide support for the most common medical image formats, data structures and state-of-the-art visualization algorithms. Some of the best in this field are the Visualization Toolkit (VTK) \cite{4} and Insight Toolkit (ITK) \cite{5}. Other libraries offer support for DICOM (Digital Imaging and Communications in Medicine), e.g., Grassroots DICOM (GDCM) \cite{6}. All these libraries are offered through commercial-capable licenses.

The main problem with these libraries is very similar to the one with visualization packages: generality. They cover so many aspects of medical visualization that the most common features are obscured or require a lot of work before a not-so-complex application can be created.

We present viewitLib, a development framework to quickly create medical imaging applications that share a common workflow and user interface (see Figure 1). This framework is built upon standard visualization libraries (such as VTK and ITK) and Qt to create cross-platform projects. This allows us to easily develop our applications and extend them with new techniques.

2. Overview of viewitLib

viewitLib consists on three main components: widgets, actions and tools. Widgets can be seen as canvases, where the user can visualize and interact with medical images using actions and tools. These component are explained separately in the following sections.

viewitLib also provides support for many medical image formats, including DICOM datasets, for which a special data loader tool has been designed. Current stable version of the framework only provides support for local datasets (or accessible via shared network volumes).

From the developer side, viewitLib is based on a modular paradigm that eases the creation of new functionalities, while increasing code reutilization.

3. Widgets

The basic work unit in viewitLib is the widget. A widget is a working space for the user, where a medical data set is shown and has a defined a set of actions and tools. It also establishes relationships with other widgets and communicates with them, handling events and sharing information. The design of each widget doesn't assume the existence of others. In this sense, an application created with viewitLib can use as many widgets as it requires, not only the four shown in Figure 1 for instance.

3.1. Widgets hierarchy

In order to deal with different visualizations and data representations, widgets are organized in a hierarchy that gradually increases their specialization (see Figure 2). In this way, it is possible to maximize the reuse of their common features.
The topmost level includes the functionality shared across the complete hierarchy, such as a VTK rendering context, screenshot and video features, and all the communication structure needed by viewitLib.

The rest of the hierarchy follows with the visualization mode (2D, 3D). Each one includes proper navigation tools and viewing points (coronal, sagittal, axial). The next level defines the data type (images and volumes), and visualization tools. The last predefined level adds support for sliced imaging, both for 2D and 3D.

### 3.2. Decorators

In order to deal with all the functionality added by each hierarchy level and by any new specialization from third-party developers, viewitLib uses a decorator design pattern to maximize the independence between widgets and their functionality (for more information, please refer to [7]). When a widget is preparing the user interface, it calls its decorator, which enumerates all the associated objects (tools and actions) and constructs an adequate interface. For example, the contextual menu of each widget is built hierarchically, starting from the top-level widget (see Figure 3). Each foregoing level adds its own set of features using its own decorator.

### 5. Tool containers

viewitLib provides a special kind of tool, named Containers, to ease the presence of multiple instances of the same tool. Containers offer both insertion and deletion operations, as well as an automatic tool selection mode under user interaction. Measurement tools, as it will be seen later, are a good use case of containers.

### 5.2. Common tools

As part of its common workflow, viewitLib includes several basic predefined tools for image visualization, measurement and navigation. Some of them are explained in the following lines.

#### 5.2.1 Distances

The distances tool (Figure 6) allows the user to measure the distance between two points in any 2D view of data. An unlimited number of distances can be inserted in a widget.

Figure 6. Distance tool

#### 5.2.2 Navigation

Top-level widgets (2D and 3D) offer common data navigation tools: panning and zooming, and in 3D, rotation. The user can navigate either selecting the appropriate tool (using the contextual menu or the keyboard shortcut), or using an “alternative” tool. This is an alteration of the interaction standard of viewitLib in order to give the user an easier and quicker interaction. All the tools use the left button of mouse for interaction. Instead, these secondary tools use the wheel for zooming and the middle button for rotation.

#### 5.2.3 Window / level

Window / level is an interactive contrast enhancement tool available in many image processing software. It is an expansion of the contrast of the pixels within a given
window range. `viewitLib` offers two versions of the tool: the first one allows window / level changes using 2D mouse dragging over the image. The other tool is a simplification of color transfer functions for volumetric visualization. It works similarly to the 2D version, but modifies the opacity and intensity of voxels.

5.2.4 3D visualization

3D widgets provide two more tools for image exploration (Figure 7). The 3D slices tool shows the data slices from 2D widgets on the 3D volume (left column). The second tool crops a section of the 3D image using one of the 2D slices (right column), making 3D cropping an easy task.

![Figure 7. 3D slices (left) and 3D cropping tool (right)](image)

6. Other components

Along with widgets, actions and tools, `viewitLib` makes use of different data structures and design patterns for synchronization and communication, as well as to reduce the code complexity in the most common tasks.

6.1. Dictionaries

Most of the synchronization process in `viewitLib` is possible thanks to a set of object containers that allow an easy access and information exchange between instances of different widget types, without exactly knowing the internals of each class. These containers are called Dictionaries, and stores data using a character string as index. Tools, actions, and other data are inserted in their respective dictionaries using a string identifier (usually the same for related tool-action pairs).

In this way, there is no need neither to create specialized getters and setters, nor introduce complicated shared references; a widget can synchronize tools, for example, just using its string identifier.

6.2. Delegates

There are some tasks in `viewitLib`’s workflow that vary depending on the specific context of use. For such tasks, `viewitLib` delegates their implementation to inherited objects, separating common and constant tasks from specific ones (for more information about delegation, please refer to [7]).

For example, widgets delegate the linkage to other widgets so each specialization can perform the required connections, tools perform basic mouse computation but delegate its handling, they also control the update cycle described in Figure 5 but delegate the tool-specific tasks, and actions have a similar workflow for notifications.

6.3. Shared objects

Even when delegates and dictionaries offer a great help for communication, there are situations where the talking graph would become too complicate (e.g., the rotation matrix for all the tools in a 3D widget, or slice positions among all the slices widgets). For these cases, `viewitLib` provides special shared objects, based on reference counting capable structures to reduce the problems associated with the destruction of such objects.

7. Inter-widget communication

Widgets are designed in such way that they do not need to know the existence of others but can communicate with the rest of them in a seamless way. Each widget is linked with others, establishing a communication channel. The bases of this communication are Qt’s signals and slots [8].

When something happens, the affected tool or action publishes its new state. Some of the most used are the “updated” / “executed” and “updatedBy” / “executedBy”; the former communicates with its own widget and, through a notification, with the decorator, while the second one communicates with other linked widgets.

Internally, tools and actions of different widgets are connected using the dictionaries: as they are named equally in all the widgets of the same level, it is possibly to retrieve them in an automated way.

8. DICOM loader

DICOM is the most common medical image format. `viewitLib` uses the common combination of GDCM and ITK to give support to DICOM. When the user imports a DICOM, `viewitLib` provides him/her with a preview window to select the right dataset (Figure 8). Optionally, the user can inspect its tags and values.

![Figure 8. Information of the selected DICOM series](image)

8.1. Name generator

One of the most common problems with DICOM datasets is the fact that there is no an easy way to identify each individual image set: usually there are hundreds of files in the same directory that correspond to a few amount of studies. `viewitLib` presents the name generator, a concept very similar to decorators. When a DICOM dataset is going to be previewed, the name generator provided by the developer is called and it creates a human-readable
string to identify each dataset. The name generator provides an easy access to common DICOM tags, as well as methods to parse such tags (for example, to extract the image type or the dimensions).

9. Field-testing of viewitLib

In this section we will present some applications developed using viewitLib. All of them were developed in a very short timeframe, giving the opportunity to focus in the specific functionality of each application.

Figure 9 shows a good example of the power of viewitLib to quickly create imaging applications and expanding them beyond the original examples. In this case, eight widgets show different views of the same data set, and the slice tools are automatically synced among all of them without any additional developer intervention.

9.1. viewit-Cochlea

viewit-Cochlea (see Figure 10) is a surgical planning tool for implantation of cochlear implants. Over viewitLib it adds a new tool that represents an user loaded 3D model of the cochlear implant, and which uses a cursor based positioning system very similar to the one used by the distances tool to position and orient it.

9.2. isoExtract

This application renders an isosurface visualization in real time, and allows the user to export it as a triangle mesh. It uses a single volume visualization widget, a tool for isovalue selection, and a second tool for triangle mesh quality (see Figure 11).

9.3. viewit-Tractography

This application is used to visualize the neural tracts from DTI (Diffusion Tensor Imaging) sources. The tracts are extracted using the dominant fiber orientation as a color component on the image.

9.4. viewit-Kinect

Finally, viewit-Kinect is an extension to the basic visualization tool provided by viewitLib, which makes use of the Microsoft Kinect to navigate around the image data set. The design of viewitLib based on tools and its communication interfaces made very easy to integrate the Kinect with the already existent navigation tools.

10. Conclusions

In this paper, we have presented viewitLib, an easy-to-use framework for medical applications with a visualization and image analysis workflow, built upon standard libraries. viewitLib maximizes the reutilization of components and reduces the development time, while creating a common user interface, hugely reducing the time needed to learn new applications. viewitLib has been designed in collaboration with physicians, ensuring the user interface is adapted to their necessities.

The roadmap for this project includes the compatibility with PACS systems for remote DICOM import, the addition of new exploration tools such as cropping regions and an easy-to-use transfer function editor.

References