The Medical Imaging Interaction Toolkit (MITK) – a toolkit facilitating the creation of interactive software by extending VTK and ITK

Ivo Wolf, Marcus Vetter, Ingmar Wegner, Marco Nolden,
Thomas Böttger, Mark Hastenteufel, Max Schöbinger, Tobias Kunert, Hans-Peter Meinzer

Deutsches Krebsforschungszentrum (DKFZ)
Div. Medical and Biological Informatics,
Im Neuenheimer Feld 280, D-69120 Heidelberg, Germany

ABSTRACT

The aim of the Medical Imaging Interaction Toolkit (MITK) is to facilitate the creation of clinically usable image-based software. Clinically usable software for image-guided procedures and image analysis require a high degree of interaction to verify and, if necessary, correct results from (semi-)automatic algorithms. MITK is a class library basing on and extending the Insight Toolkit (ITK) and the Visualization Toolkit (VTK). ITK provides leading-edge registration and segmentation algorithms and forms the algorithmic basis. VTK has powerful visualization capabilities, but only low-level support for interaction (like picking methods, rotation, movement and scaling of objects). MITK adds support for high level interactions with data like, for example, the interactive construction and modification of data objects. This includes concepts for interactions with multiple states as well as undo-capabilities. Furthermore, VTK is designed to create one kind of view on the data (either one 2D visualization or a 3D visualization). MITK facilitates the realization of multiple, different views on the same data (like multiple, multiplanar reconstructions and a 3D rendering). Hierarchically structured combinations of any number and type of data objects (image, surface, vessels, etc.) are possible. MITK can handle 3D+t data, which are required for several important medical applications, whereas VTK alone supports only 2D and 3D data. The benefit of MITK is that it supplements those features to ITK and VTK that are required for convenient to use, interactive and by that clinically usable image-based software, and that are outside the scope of both. MITK will be made open-source (http://www.mitk.org).

Keywords: MITK, ITK, VTK, interaction, visualization

1. INTRODUCTION

It becomes more and more common sense that standardization is essential for real progress in medical imaging software development (and computer science in general, as demonstrated by the Boost.org initiative for C++). In the medical imaging field, several aspects needs to be addressed regarding standardization (the list does not claim to be exhaustive):

- Transmission and storage of medical imaging data, which is standardized by the DICOM Standards Committee – at least for commercial products. In the research field, DICOM is often regarded as too complicated (and proprietary formats are used) or it is insufficient (e.g., there is still no final DICOM standard for 4D data).
- Algorithms to process the original image data or data derived from the original images as well as the combination of algorithms. For the most important image processing tasks, i.e., segmentation and registration, the Insight Toolkit (ITK, http://www.itk.org) is on the way of becoming the de facto standard.

Further author information (Send correspondence to I.W. or to mitk@mitk.org)
{I.Wolf, M.Vetter, I.Wegner, M.Nolden, T.Boettger, M.Hastenteufel, M.Schoebinger, T.Kunert, H.P.Meinzer}@dkfz.de

- The visualization of the data (original images and processed/derived data). The Visualization Toolkit (VTK, http://www.vtk.org) can be regarded as the de facto standard, at least for 3D-surface-based visualization.
- Storage of derived data, e.g., polygonal surface meshes. A variety of standards exists, some induced by the success of software products (like Analyze) or libraries (like VTK).
- Standard datasets for testing – and comparison of – the performance of algorithms. There are several initiatives aiming in this direction, e.g., the "BIRN Data Grid" or the "Dermatology Image Atlas".
- Interaction with the data (original images and processed/derived data), including manipulation and correction of generated results.

There is another aspect becoming more and more obvious: Medical imaging techniques can only be proved to be successful – or even relevant – if they can be successfully *applied in a clinical setting* and not by only testing them on (a few) test datasets.

Interactivity is a central prerequisite for software that is applicable in a clinical setting. While the algorithmic and visualization aspects are widely covered by ITK and VTK, there is currently no open-source toolkit sufficiently supporting the creation of interactive, clinical useful medical imaging software.

Currently under active development, MITK aims at supporting the development of medical imaging software with a high degree of interaction. It combines VTK and ITK and adds those features that are most important for developing interactive medical imaging software and that are covered neither by VTK nor ITK. This includes:

- multiple, consistent views on the same data. For example, assume you have three orthogonal 2D views and a 3D view and data consisting of a green surface, then the surface will be visible and green in all views (as contour lines in 2D, as a surface in 3D). When you move the surface, it will move in all views. If you try to realize this with basic VTK, you will need to coordinate the position/orientation/color/... of all the views – exactly this coordination work can MITK do for you.
- organization of data objects in a tree at run-time, e.g., to represent logical dependencies, physical interrelationships (e.g., a heart cavity *is a part of* the heart) and/or to control the rendering process
- interactions that create and/or modify data (not only actors like in basic VTK)
- complex interactions with multiple states. Even the rather simple case of building a polygon and subsequently interacting with it has multiple states (building-in-progress, point-is-selected, point-is-being-moved, etc.). MITK has a unique concept to describe and configure states and state-transitions in a consistent way.
- a concept for undo/redo of interactions and data processing
- handling different types of input devices in a unified way (mouse, graphics tablets, tracking systems, etc.)
- geometry information objects to a) define the position of data objects in space and b) define the area or volume to be rendered. This allows correct handling of data not acquired in uniform Cartesian coordinate systems, e.g., 3D ultrasound data (free-hand or rotational/cylindrical acquisition)
- rendering properties of arbitrary type, which can be assigned to data objects contained in the data tree and defined either to be used for all renderers or only a specific renderer
- visualization and interaction with 3D+t data (basic VTK can handle only 3D data and ITK, which can handle arbitrary dimensions, is not designed for visualization and interaction).
- Additionally, MITK offers some support on the application-level. One example is the structured combination of modules (so-called *functionalities*), e.g., for combining and switching between one functionality for segmentation and another functionality for registration.

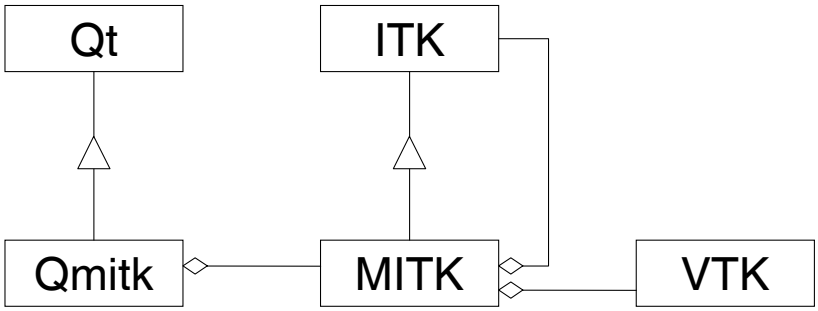

Figure 1. Relations of MITK to the toolkits ITK, VTK and Qt (symbolic). Many MITK classes are derived from ITK, whereas VTK classes are used and not derived from. MITK itself is independent of the GUI library. The GUI library dependent part contain optional add-ons and is currently implemented for Qt (Qmitk).

MITK is a *toolkit* and *not* an application as existing interactive medical imaging software like, for example, the "3D Slicer" (http://www.slicer.org[1]), "Analyze" (http://www.analyzedirect.com) or the "Julius Framework" (http://www.julius.caesar.de[2]). Thus, it allows the construction of specifically tailored applications and can be used within existing software.

MITK re-uses virtually anything from ITK and VTK. Thus, it is not at all a competitor to ITK or VTK, but an extension, which eases the combination of both and adds the features required for interactive medical imaging software. MITK will be made open-source (go to http://www.mitk.org).

2. DESIGN PRINCIPLES

The basic strategy in the design of MITK was to re-use as much as possible of already existing successful designs and implementations of these designs. In our opinion, the most successful designs are currently those of ITK and VTK. MITK does not only use these toolkits, but also re-uses their design principles. This makes it easy for people already familiar with ITK or VTK to get accustomed to MITK. The relations of MITK to ITK and VTK are illustrated in Fig. 1.

Like ITK and VTK, MITK is an object-oriented, cross-platform library implemented in C++. Most MITK classes are derived from top-level classes of ITK (like `itk::LightObject`, `itk::Object`, `itk::ProcessObject` or `itk::DataObject`). This allows to re-use the smart-pointer-, time-stamp-, pipeline- and notification-mechanisms of ITK.

MITK uses a data-centered approach, similar to the *model-view-controller* design pattern. The central elements representing the *model* are *data objects* and *data trees* (see Sec. 3). A data object has a specific type (e.g., image, surface, list of points). A data tree allows to hierarchically organize multiple data objects, which may have different types.

Often several data objects have to be presented to the user. Therefore, what is rendered into a window (or "widget"), is a complete data tree or a branch of a data tree. Multiple, consistent *views* on the same data are possible, since a (branch of a) data tree may be rendered into several windows and all information about the data that is required for rendering (including position, orientation and other properties of the data object like color and opacity) is included in the data tree. Different *types* of views do *not* require different types of windows in contrast to the classical model-view-controller design pattern (see Sec. 4).

The *controller* aspect consists of two parts in MITK. *Rendering controllers* control the camera for 3D views and which slice is displayed for 2D views (see Sec. 4.1). The interaction framework of MITK controls the interactive creation and modification of data objects and is closely connected to the MITK's undo/redo framework (see Sec. 5).

MITK is *not* intended as an application framework. But there are some optional application-level add-ons to MITK, which further facilitate the creation of some types of applications (see Sec. 6).

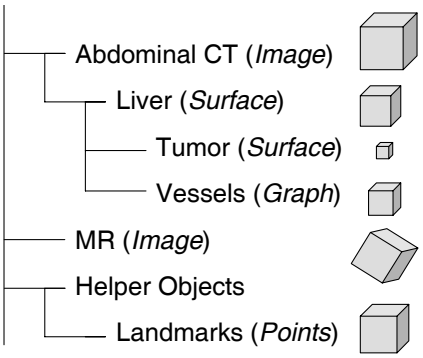

Figure 2. Data tree to hierarchically organize data objects (types in brackets). Geometry information are associated to the data objects to describe their position in space and time (boxes).

From a design point of view, MITK itself is *not* limited to a specific GUI library and can be extended to other GUI toolkits (like FLTK) or even script languages (e.g., similar to ITK by using the CABLE wrapping package, http://public.kitware.com/Cable). The software process of MITK is adapted from ITK and VTK (usage of CMake, DART, doxygen, cvs).

3. DATA OBJECTS AND DATA TREES

Data objects represent and provide access to data, for example images, surfaces, points, etc. Data objects are derived from `itk::DataObject` and thus can be created and updated by process objects (derived from `itk::ProcessObject`) connected together to a data processing pipeline.[3] MITK defines some additional data objects. For handling data in interactive applications, non-templated data objects are more convenient, because the data type and dimension of the data the user wants to load and manipulate often can not be known in advance (especially for images). The MITK data types also allow an easier handling of 3D+t data, for example to update and access only a single point in time (or a single slice of an image) and interpreting the result as a 3D data object (or a 2D image, respectively) without the duplication of memory. The general idea is to use these MITK data types only for data handling and the templated ITK data types for writing algorithms. Filters allow to convert between the templated and non-templated versions without the duplication of memory.

Data objects may be hierarchically organized in a tree-structure at run-time, the *data tree* (see Fig. 2). The hierarchy can be used to represent logical dependencies and physical interrelationships. For example, the liver vessels are *located within* the liver. Also artificial objects like bounding boxes or measuring instruments may be added to this object hierarchy. An MITK renderer and the MITK interaction framework takes a data tree or a branch of a data tree rather than single data objects to allow rendering of and interacting with multiple data objects (see Secs. 4 and 5). Like all data objects, data objects within the data tree may be created and updated by process objects. A pipeline may be constructed, the resultant data object added to the data tree and all other references to the constituents of the pipelines (resultant and intermediate data and process objects) may be discarded. This is very convenient, for example, to "intelligently" load data. The resultant data object of a file reader process object is added to the data tree, but data will actually only be read when required for visualization, interaction or other processing.

Data objects added to the data tree are objects in space and time. The data values are stored in intrinsic coordinates, e.g., integer pixel/voxel or time indices. The information required to convert these intrinsic coordinates into a physical world coordinate system, with coordinates in millimeters and milliseconds, is called *geometry* information. Geometry information are encapsulated in geometry classes in MITK, see the following Sec. 3.1, and can be associated to data objects.

Optionally, properties like color, opacity, visibility, etc. can be added to a data object within a data tree. These properties apply to all views on the data, rendered into different windows, by default, but may be changed for each views if required. For example, a data object may be visible in all but one view.

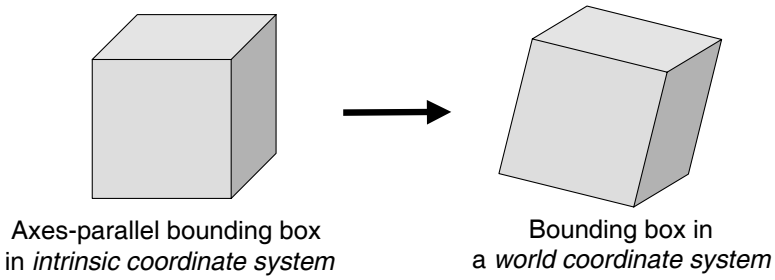

Figure 3. Geometry classes contain at least a bounding box, which is parallel to the coordinate axes of the intrinsic (data object) coordinate system (often integer indices of pixels), and a transform to convert intrinsic coordinates into world-coordinates.

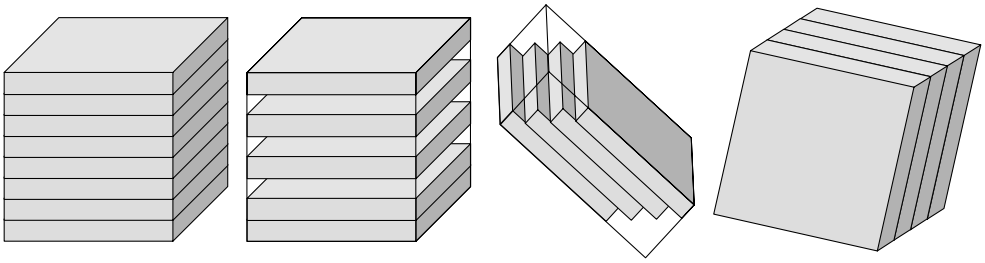

Figure 4. Sliced-geometries describe data composed of slices. Besides the overall bounding box, the bounding boxes of the contained slices are stored. Examples from left to right: geometry of a standard CT acquisition, CT acquisition with some slices missing, tilted CT acquisition, mostly frontally acquired MR data.

3.1. Geometry information

Each data object within the data tree associate a geometry object. The base geometry class holds a bounding box which is axes-parallel in intrinsic coordinates (often integer indices of pixels) and a transform to convert intrinsic coordinates into a world-coordinate system with coordinates in millimeters and milliseconds (floating point values), see Fig. 3.

These information are, in principle, also contained in ITK's spatial objects.[3] The extension in MITK is that the information are encapsulated in separate geometry classes, which hence can be *used and extended independently* of the data object itself. Thus, a geometry class contains *at least* information about a bounding box and conversion from intrinsic to world-coordinates, but it may contain more information.

The most important derived geometry classes describe data composed of slices (sliced-geometries), since three-dimensional medical images normally consist of a stack of slices. In practice, the slices are not always parallel to each other or evenly spaced. Fig. 4 shows some examples. Other examples are 3D ultrasound data created by free-hand or rotational/cylindrical acquisition methods. MITK's geometry classes allow to deal with such cases.

Besides describing the geometry of data objects, geometry objects are used to control the rendering, especially for 2D views, see Sec. 4.1.

Like in DICOM, there may be several world-coordinate systems, for example one for each patient or each data acquisition after patient movement. Therefore, each world-coordinate system has an identifier, which is held in the geometry base-class.

4. RENDERING

Instead of building up one scene-graph for each window, a data tree – or branches of a data tree – are passed to the renderer, which renders into the window it is associated to. This allows multiple consistent views (e.g.,

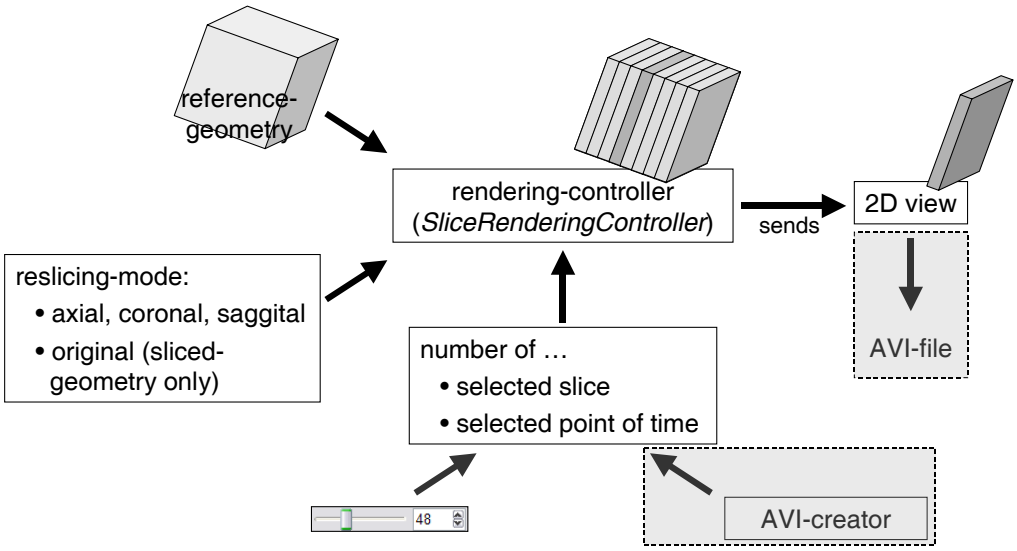

Figure 5. Rendering controller for 2D views. Provided with a reference geometry and the reslicing-mode (axial, coronal, saggital) the rendering controller generates a sliced-geometry (or, in case of "original" as reslicing-mode and a provided sliced-geometry just uses that). The sliced-geometry together with the number of the selected slice and point of time is send to the view every time one of it changes or when explicitly called. Besides being provided by a GUI element, slice and point of time may originate from a general video clip (AVI) exporter (dashed boxes).

several 2D and 3D views) on the same data, since a data tree can be passed to multiple renderers and all information about the position, orientation and other properties of the data required for rendering is contained in the data tree. The visualization itself is done by calling VTK.

MITK follows VTK's philosophy of having only *one kind* of rendering window. A rendering window is a window in a graphical user interface where renderers draw their images. Thus, what is being displayed in a rendering window depends on the renderer, which in turn delegates the rendering of specific data (via *props*/*actors*) to so-called *mappers*. The task of a MITK mapper is to transform a data object in graphics primitives for the VTK scene or OpenGL render process. Different types of visualizations of the same kind of data (2D vs. 3D views, volume vs. surface rendering, etc.) can be realized by different mapper classes. Specific mappers may be used for rendering in different windows. This approach is much more flexible and easer to extend than the alternative, which is, to have specialized windows for each visualization task. Furthermore, it makes possible to write code that is only dependent on the graphics toolkit (e.g., OpenGL), but is independent of the GUI library (Qt, FLTK, etc.).

Since the 2D visualization support of VTK is not as evolved as its 3D counterpart, MITK provides additional classes for 2D visualization. These allow, for example, fast texture based rendering with tiling for huge images. 2D views are defined by a geometry in MITK (see previous Sec. 3 for geometries). The top face of the bounding box contained in the geometry, transformed into world-coordinates, are used as the 2D manifold to cut through the data and display. The geometry is also used to convert 2D display coordinates into 3D world-coordinates. Even strongly deformed views (like, for example, a planar map of a sphere) can be transparently handled.

4.1. Rendering controllers

Rendering controllers control the camera for 3D views and which slice is displayed for 2D views. They standardize the way to control stepping through a sequence of visualizations, for example a stack of slices in 2D or a rotation around (or of) an object in 3D.

Rendering controllers are invisible, GUI library independent objects, which receive the number of the frame in the sequence, and change the display accordingly. The frame number is often set by visible control elements

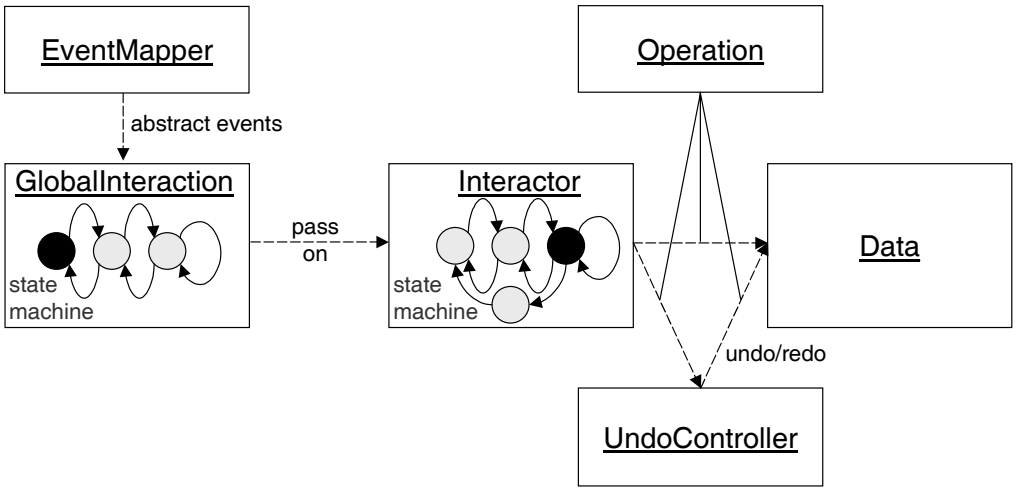

Figure 6. Overview of the interaction concept. Events from different event sources (e.g., graphics tablets, tracking systems etc.) are converted into abstract events by the singleton class `EventMapper`. This class transmits the events to the singleton `GlobalInteraction`. According to its internal state, this state machine delegates the events to the dedicated `Interactor`-objects. The manipulation of the data is achieved through the creation of `Operation`-objects that are send to a specific data object. Here the `Operation` is executed. Additionally, the `Operation` and its associated inverse `Operation` (undo-operation) are send to the `UndoController`, which stores the operations according to the undo-model. In case of an undo, the inverse operation is send to the data and restores the previous status.

like sliders (GUI library dependent). They may also be used to realize a general video clip (AVI) exporter, working independently of the contents (set of 2D slices, rotation of 3D object, real 4D like a beating heart, etc.) of the video clip.

Fig. 5 illustrates the functioning of the rendering controller for 2D views. A reference geometry is provided to the controller, out of which it creates slice-geometry, i.e., a stack of slices. By that, the selection of the view orientation is independent of the displayed contents. Alternatively, a slice-geometry may be provided as a reference geometry and the stack of slices contained therein used for stepping through the data (reslicing-mode "original"). The slice-geometry may be taken from a data object (e.g., a free-hand ultrasound acquisition), or created in any other way, for example to slice the data perpendicular to the centerline of a vessel.

5. INTERACTION

Interaction is an important part for image based medical applications. This is true for intervention planning, assistance systems for medical interventions, for manual and semi-automatic segmentation as well as to adjust automatic processes for segmentation or registration tasks. MITK has a new, generic interaction concept to structure and by that standardize the manipulation of data and artificial objects in 2D and 3D views.

Interactions are reactions of the software on user input. Since there are many types of input devices (e.g., graphics tablets, tracking systems, etc.), the first step to standardization is to create abstract events out of device dependent events. The next step is to interpret the meaning of the event, which often depends on the state of the software. Even the rather simple case of building a polygon and subsequently interacting with it has multiple states (building-in-progress, point-is-selected, point-is-being-moved, etc.). MITK's unique interaction concept is organized by hierarchically defined state machines and allows to describe and configure states and state-transitions in a consistent way in XML-files, which are loaded at run-time. By that, it is possible to load different interaction behaviors used in common commercial medical software (e.g., Siemens Syngo, Philips

EasyVision, GE Advantage). Furthermore it is possible to easily adapt the behavior to specific applications and workflows. Fig. 6 gives an overview of the interaction concept.

5.1. Undo/redo

Undo is one of the important functions of a good user interface in interactive applications. Without undo, correcting a false instruction is often very time consuming. Undo and redo functionality is a fundamental design decision and cannot be easily added at a later time. Thus, the MITK concepts offer undo and redo functionality for interaction and processing tasks.

MITK's general undo-/redo-concept is based on the *inverse command strategy*.[4] The concept requires the interaction objects to issue *operations*, which are send to so-called *operation actors* that actually perform the operation, e.g., modify the data or change the displayed slice. For each operation an associated *inverse operation* is created by the interaction object (e.g., "move backward" for "move forward"). Both, packed together in an `OperationCommand`-object, are send to an `UndoController` object (see Fig. 6). In case of an undo-command issued by the user, the `UndoController` sends the inverse operation to the `OperationActor`, which restores the previous status, or, in case of filter operations, the previous data or parameter set.

Different undo/redo models are possible, like a limited linear undo/redo-model or a linear instruction tree model. Currently, a restricted linear undo model is implemented.[5] The state machine concept outlined above facilitates the consistent realization of the undo/redo functionality of interactions.

6. ADD-ONS

This section briefly describes some add-ons to MITK, which are not part of the MITK core. Most of them are GUI library dependent. Currently, these add-ons are implemented only for Qt (Trolltech, Oslo, Norway, http://www.trolltech.com) and therefore called *Qmitk*. This includes a server to connect tracking systems, the implementation of a concept facilitating structured combination of modules and integration code for a PACS/Telemedicine system.

6.1. Tracking server

Tracking systems are a specific type of input device providing localization and orientation information with three to six degrees of freedom. MITK's tracking server provides a unified interface to tracking systems. Currently, the following systems can be connected:

- MiniBird (Ascension Technology Corp., Burlington, VT, USA): electromagnetic, 6 degrees of freedom
- Aurora, NDI (Northern Digital, Waterloo, Ontario, Canada): electromagnetic, 5 or 6 degrees of freedom, depending on the sensor
- Polaris, NDI: optical 6 degrees of freedom

The server connects to the tracking system and sends the received localization and orientation data via TCP/IP to the MITK application. The application can use the information to change the transform (included in a geometry) of a data object or the camera. The tracking server can supply the tracking coordinates to multiple applications on different computers.

Some tracking systems need a rather long time for initialization. For these systems, the tracking server makes debugging easer, because no re-initialization of the tracking system is required when re-connecting to the tracking server, which can be kept running during multiple debugging sessions.

6.2. Functionalities

The concept of functionalities facilitates the structured combination of modules. A *functionality* is a module for a specific task, combining a user interface with algorithmic function. It consists of

- an identification (name of the functionality, icon, tooltip, ...),
- a workspace area, where the main interaction is taking place,
- a control area containing GUI elements to set parameters, and
- the algorithmic implementation.

Communication between functionalities is largely based on the data tree. Each functionality accesses the data objects contained in the tree, changes them and/or creates new data objects. Other functionalities can continue to work on the changed and/or newly created data tree entries. By that, functionalities can communicate without becoming dependent on each other. The examples described in Sec. 7 were realized as functionalities and can used within one application or as separate applications.

6.3. PACS/Telemedicine integration

The PACS/Telemedicine system *Chili* (Chili GmbH, Heidelberg, Germany, http://www.chili-radiology.com) provides a plug-in interface for third-party modules. An add-on to MITK allows to write software that can run either stand-alone or as a plug-in within Chili (after recompilation) with a minimum of additional effort. The stand-alone version is more convenient during development and debugging, the plug-in version greatly facilitates the clinical integration of the software.

The radiologist or surgeon using the MITK-based software find it seamlessly integrated into the workspace he/she is used to and has full access to the PACS functionality of Chili. Chili deals with the connection of modalities, DICOM import and export, "unification" of DICOM (DICOM tags are not as standardized as they should be), data transfer, management and archiving of results from image processing and can be used as a teleradiology system.

7. EXAMPLES

The current implementation of MITK includes all major concepts mentioned above and consists of about 120 classes. We have already realized several applications using MITK, including applications for interactive usage of ITK segmentation and registration algorithms, for volumetry of heart cavities, for the definition of landmarks in image-guided procedures, and for resampling of image data on curved surfaces defined by a set of points.

Fig. 7 and 8 show screenshots of the latter application (realized as a functionality, see Sec. 6.2). The code to realize the specific functions of this applications is less than 200 lines (mostly to initialize VTK's thin-plane-spline filter). A curved surface (thin-plate-spline) is fitted through the points displayed as crosses, which can be set on the three orthogonal views. The result (a cut through the teeth) is visible in the 2D views as curved lines (the straight lines show the position of the currently shown MPR views) as well as in the 3D view (bottom right). Thanks to the MITK interaction mechanism, the points can interactively be added, moved and removed, including the possibility to undo/redo any number of interactions. In Fig. 8, the bottom-right view is switched to a 2D view of the resampled data. Consequently, all points defining the resampling surface are visible in this view, demonstrating the capability of MITK to handle arbitrary 2D-geometries. Each point on the screen in a 2D view has a 3D point in real-world coordinates associated with it. By clicking into one of the 2D views (including the bottom-right view), the MPR views are moved in order to contain the associated 3D position. Zooming and panning are also possible in the 2D views, as shown in the orthogonal views (compare the upper two views of Fig. 7 and Fig. 8).

Fig. 9 shows a second example, also realized as a functionality. Using the ITK algorithms for level set based segmentation,[6] the new interactive 3D segmentation tool, the *Fast Marching Level Set* tool, was implemented. This tool requires the user to select one or more initial seed points. Starting from these seeds an initial level set

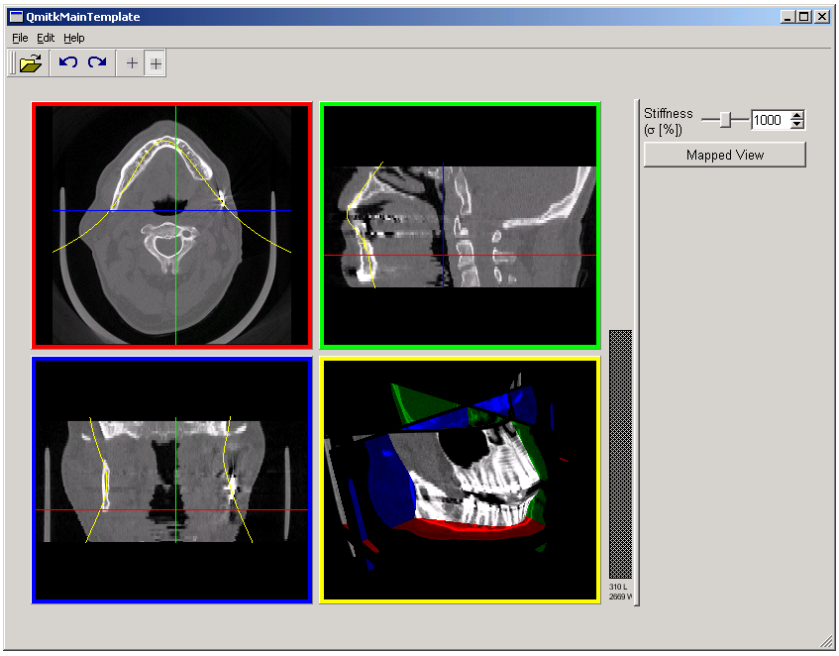

Figure 7: Application for resampling of image data on curved surfaces defined by a set of points. See text for details.

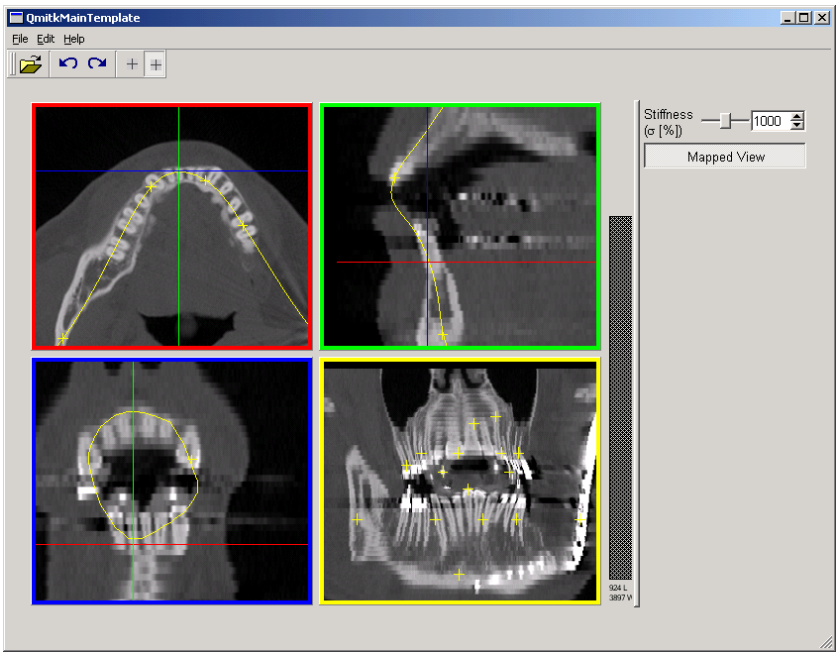

Figure 8. Same application and data as in Fig. 7, but with the bottom-right view switched to a 2D view of the resampled data. See text for details.

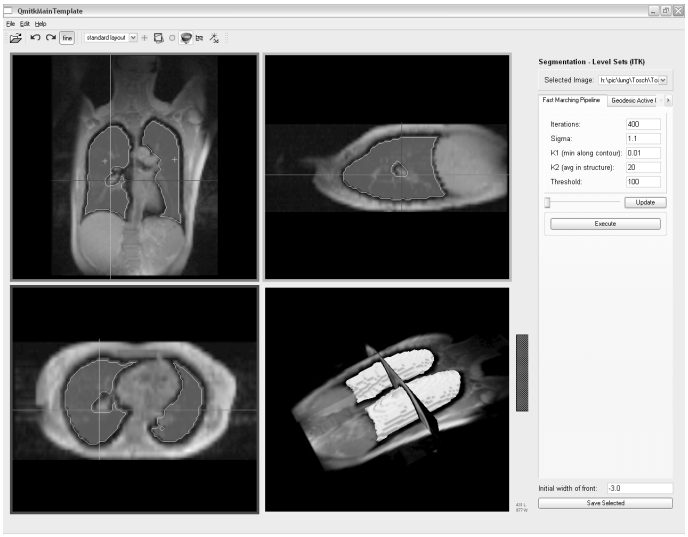

Figure 9. Level-set segmentation of the lungs in a standard anatomical MRI. The ITK-based segmentation algorithm was initialized in this case with one seed-point in each lung (visible as crosses in the upper-left view).

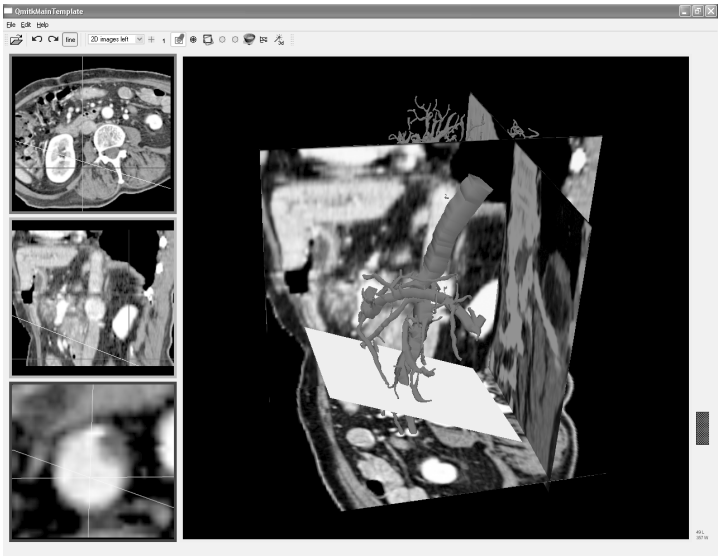

Figure 10. Functionality to reslice image data perpendicular to a vessel. The position of reslicing is visible as a rectangle in the large 3D view and as lines in the 2D views (the other lines represent the position of the other 2D views). The resliced image is shown in the bottom-left view.

is computed and then evolved towards the boundaries. During level set propagation, the user gets a preview of the resulting segmentation at the current time, visible in the orthogonal 2D slices and the 3D view, and by that gets a good impression of the current status of the segmentation.

The third example, Fig. 10, shows a functionality to reslice image data perpendicular to a vessel. After segmentation, the centerline of the vessels were extracted.[7] The user can interactively select two end-points along a vessel, between which the reslicing takes place and then step through the slices. To realize the reslicing with MITK, only appropriate Geometry-objects, representing rectangles perpendicular to the centerline of the

vessels, had to be calculated, added to the data tree (to make the rectangle visible in the 3D view as well as lines in the 2D views) and passed to a rendering controller (see Fig. 5) that controls the bottom-left view to get the actual resliced image in that view.

8. CONCLUSION

The value of medical imaging algorithms can only be judged if they are applied in a clinical context. This requires visualization and interaction with the data. Writing a convenient to use, clinically applicable software for testing just one new algorithm is much too time-consuming.

The described "Medical Imaging Interaction Toolkit" (MITK) facilitates the creation and combination of interactive software. Our goal was not to reinvent anything already existing, but just to add the features not available before. Therefore, we have derived MITK from ITK and use VTK wherever possible. The similar structure should make it easy for developers already familiar with ITK/VTK to get accustomed to MITK.

The traditional approach to support the integration of medical imaging methods into a clinical environment is to extend one application with more and more methods. In contrast, the Medical Imaging Interaction Toolkit MITK is a *toolkit* and *not* an application. Thus, MITK allows the construction of applications specifically tailored for a medical task, providing only those features to the user (physician) that are required. Another advantage is that the toolkit can be used within existing software.

MITK already proved to greatly reduce the time required to write interactive applications for image-guided procedures and medical image processing. We feel confident that other researchers will profit from its use, too. Go to http://www.mitk.org to check for news about MITK.

REFERENCES

1. D. Gering, A. Nabavi, R. Kikinis, W. E. L. Grimson, N. Hata, P. Everett, F. Jolesz, and W. M. Wells 3rd., "Integrated visualization system for surgical planning and guidance using image fusion and interventional imaging," in *MICCAI - Medical Image Computing and Computer-Assisted Intervention*, pp. 809–819, 1999.
2. E. Keeve, T. Jansen, Z. Krol, L. Ritter, B. von Rymon-Lipinski, R. Sader, H. Zeilhofer, and P. Zerfass, "Julius - an extendable software framework for surgical planning and image-guided navigation," in *Fourth International Conference on Medical Image Computing and Computer-Assisted Intervention MICCAI'01*, W. Niessen, ed., *Lecture notes in computer science* **2208**, pp. 1336–1337, Springer, (Berlin), 2001.
3. L. Ibanez, W. Schroeder, L. Ng, and J. Cates, *The ITK Software Guide*, Insight Software Consortium, August 2003.
4. J. E. Archer Jr., R. Conway, and F. B. Schneider, "User recovery and reversal in interactive systems," *ACM Transactions on Programming Languages and Systems* **6**(1), pp. 1–19, 1984.
5. T. Berlage, "A selective undo mechanism for graphical user interfaces based on command objects," *ACM Transactions on Computer-Human Interaction* **1**(3), pp. 269–294, 1994.
6. J. A. Sethian, *Level Set Methods and Fast Marching Methods*, Cambridge University Press, 1999.
7. M. Schöbinger, M. Thorn, M. Vetter, C. F. Cardenas, P. Hassenpflug, I. Wolf, and H. P. Meinzer, "Robust analysis of vascular structures using three-dimensional skeletonization," in *CARS'03 - Computer Assisted Radiology and Surgery*, H. U. Lemke, M. W. Vannier, K. Inamura, A. Farman, K. Doi, and J. H. C. Reiber, eds., p. 1319, Elsevier, (Amsterdam, Lausanne, New York), 2003.