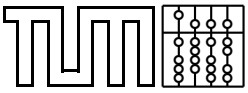

Software Entwicklungs Projekt

User Interface for Visualization and Realignment of 3D Modalities

Henning Herbers

February 27, 2006

Technische Universität München
Departmentof Computer Science
Chair for Computer Aided Medical Procedures and Augmented Reality

wwwnavab.cs.tum.edu

Abstract

For both visualization and navigation in 3D modalities, the existing means often turned out to be insufficient. Especially in concern of realignment, there proofed to be a lack of usability. Therefore an advanced visualization system was to be developed, based on a common 3-Slice-View. One main objective was to integrate the ability to define rigid transformations, further the possibility to load numerous volumes and visualize them in a fused overlay. Thus the system serves as a powerful tool for diagnosis and manual registration. One particular application is the alignment of heart CT data along the heart's major axis.

Contents

1 Introduction

There are two main objectives that had to be solved within the course of development of this project. First, an existing 3-Slice-View had to be extended to fit the requirements, second the main frame of it's user interface had to be modified to make it possible to load a numerous set of volumes and visulalize them in the new view.

The framework of the project was given by a slice visualization interface developed in a medical visualization lab course in 2004. This interface already featured a view for the visualization of three orthogonally orientated slice renderers plus an integrated view that showed the slices together as they were orientated in the volume. However, the view allowed the user only to define vertical and horizontal translations of the differnt renderers, while it was not possible to rotate the slices. This abilty is required for realigning a volume.

The main motivation of the project was to develop a system suitable for realigning a CT data volume of the heart along it's main axis. Cardiologists demand this alignment to find their orientation within the heart when shifting trough the slices. Existing means, such as state of the art software distributed with the CT hardware, often revealed a lack in usability concerning this alignment. Therefore a simple visualization and navigation tool was asked, that includes the possibility to define rigid transformation while orthogonallity is preserved. Cardiologists could then realign the slices by visible features.

Further, the mainframe was to be modified, allowing the user to load different volume and manage their visualization. The slices rendered from the volumes were to be shown as a fused overlay in the 3-Slice-View, whereby the user could define the intensity at which each volume was faded and transform each of them seperatly. This would make it possible to manually registrate corresponding volumes and thus make the tool more powerful.

2 The User Interface

2.1 General

The user interface has to handle two major interactive tasks. Primarily, a medical volume has to be visualized, whereby transformations defined by user input have to be taken into account.

This core part of the system is realized in the SliceViewGL class that holds the actual Open GL content and the methods applied on the volume. It's corresponding GUI is provided by the class SliceViewUI that intercepts all user input and calls the according functions of the SliceViewGL object. Further, all mouse interaction within the visualized Open GL content is handled by the SliceViewGLWindow class in order to keep the SliceViewGL totally independent of the actual UI structure.

The SliceViewGL receives it's input from the UI classes and decides on how to handle this input to modify the visualization. The position of the actual image data, along with it's properties that define the visualization are stored in an object (further referred to as ImageNode) that is linked in as a node within a list (further referred to as ImageList). The properties are changed by the methods of the SliceViewGL object when a user interaction occurs. On callback the node renders a set of three byte images from the according volume, based on its constraints.

To organize these ImageNodes is the second major task of the user interface. The user has to be able to load numerous medical volume images (CT, MR, Pet...) and afterward toggle through them to select which one is to be displayed in what manner in the SliceViewGL. The MainFrameUI serves as the graphic interface, whilst the class MainFrame handles the input passed on by the MainFrameUI.

2.2 The MainFrame

The MainFrame was adopted from the lab course 2004, though it's structure had to be changed. Objective of the application developed in the lab course was to load several images and then create different instances of views to visualize the image data either through three orthogonal slice renderer or through a volume renderer. The MainFrame and it's UI allowed the user to toggle through the loaded images and it's different views.
The new system now works on one single view rather than on different instances. If more than on image is loaded, these should be visualized as a fused overlay in the OpenGL context of this single view. Therefore the MainFrame contains one instance of the SliceViewUI, that is initialized once the first image is loaded. Further, in a box are listed all loaded images, whereby the user has the possibility to toggle through them to select which image is to be effected by an input. A flag on the right hand side of each item let's the user switch on and off whether the image is to be visualized or not.

The menu bar so far includes menu items to load and close images and exit the program.

2.2.1 Functionality

The MainFrameUI is a graphical user interface created with the FLTK library. If an input occurs the according functions of the MainFrame object are called.
If an image is loaded, a new image node is created. It contains a pointer on the location where the image data is stored, along with image information and rendering properties that will be described later. These nodes are organized in a list structure, that is realized by the ImageList class, which is initialized by the constructor of the MainFrame. To each node, and therefore to each loaded image, an item containing the name of the image is shown in a box of the user interface. By mouse input the user can select one of these images and mark it as actual. If this occurs a pointer in the ImageList (further: actual-pointer) is set to point on the selected image. The SliceView later uses this pointer to choose which node's properties have to be changed. Further, there is a check box show next to each item, to turn the visibility flag of each node on and off. Only the nodes within the list with the flag set to true are later visualized. If the image loaded is the first image open, a new SliceViewUI and a new SliceViewGL object have to be created.
Closing an image will removed the node pointed on by the actual-pointer from the list and it's item from the box. If the image closed is the last open image, SliceViewGL and SliceViewUI have to be destructed and the SliceViewUI will be hidden.

2.2.2 The Image List

To organize the various images along with it's display properties a list structure is implemented. This list contains an arbitrary number of nodes. Besides the pointers indicating the first and the last node in the list (first→next = last if list is empty), there is one pointer (act) pointing on the actually selected node and one index pointer (run) used if all nodes in the list have to be addressed.
The nodes itself contain all the necessary information to visualize the according volumes. A pointer indicates where the image data is stored that is rendered by three orthogonal SliceRenderers (camplib) to generate byte images for later visualization in OpenGL. Among other properties, a pose is set for each renderer that can be modified prior to rendering, defining the slice's translation and rotation in space.

2.3 The SliceView

The SliceViewGL is the main part of the project, providing the OpenGL context for the visualization and handling the user input. It goes along with an SliceViewUI object, the receives the input and passes values and callback on to the SliceViewGL class.
The SliceView works in three modes. In the standard mode the OpenGL window is divided into four subwindows, three to display the three orthogonal slices each separately and a fourth to display all three of them together as the lie in the volume. Switching to second mode lets the user display each slide in full view, to have a closer look. The third mode finally shows the slide also separately but within the volume cube to make it easy

for the user to find it's orientation in space.
Main objective of the view is to apply rigid transformation on the slices. Further slice thickness and range can be adapted and Maximum Intensive Projection mode turned on and off. To fuse the visualization of different volumes there is an input on how intense the volume is to be shown.

2.3.1 User interaction

The Three-Slice mode

When in standard, Three-Slice mode, the SliceView shows the three orthogonal slices each separate, whereby a crosshair in each subwindow demonstrates how the other two slices relate to the shown one (Figure 2.1). The fourth subwindow shows the slices as they lie in space.

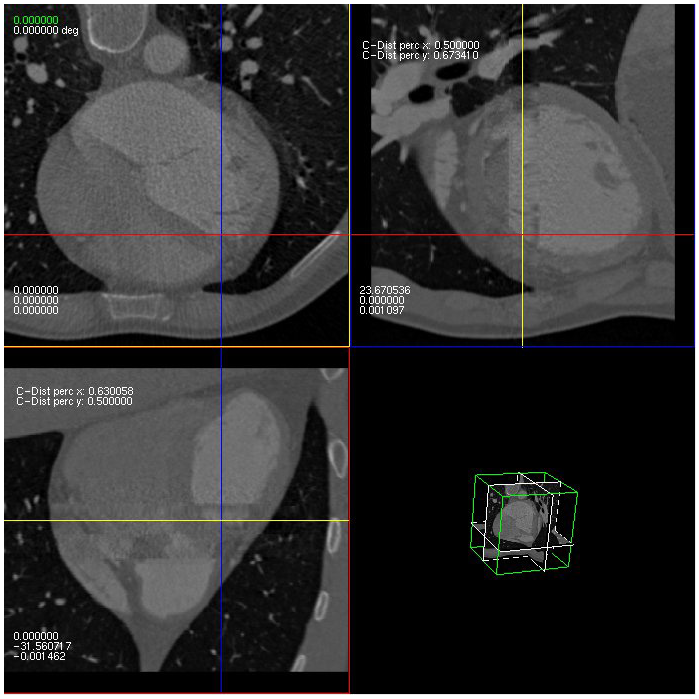

Figure 2.1: OpenGL content of the Slice View in 3-Slice-Mode

By direct mouse interaction the user has the possibility to modify the position of the respective slices. The mouse navigation is adapted to the GLCamera class (camplib). To perform rigid transformation, the user can set the position of the crosshair by right-clicking the desired position and thereby define where the according renderers are to be places. Further, the middle-mouse button lets the user rotate the crosshair and define a rotation of the according renderers. Once the mouse button is released, the respective renderer is rotated relatively around its z-axis to reestablish orthogonality.
Dragging the mouse within a subwindow while the left mouse button is pressed modifies the rotation of the renderer shown in the subwindow around it's x-axis (mouse movement

in y-direction) and it's y-axis (mouse movement in x-direction). Simultaneously the other two renderers are rotated along to keep up the orthogonality.
What slice is actually rendered from the volume is defined by a vector of float values containing the pose of the renderer, 3 entries for the translation from the center of the volume and three entries for the eukledian rotation around the respective axis. This pose can be changed in two manners, absolutely and relatively. For setting the pose absolutely each renderer possesses a vector of float values (renpose) that can be changed arbitrarily and be applied (setPose function of the SliceRenderer) prior to rendering. A relative change to the pose on the other hand is applied directly to the renderer.
Setting the crosshair and rotating it define absolute translation and rotation of the respective renderers. The position of the crosshair in a subwindow in relation to the viewport relates to the translation of the according renderers in relation to dimension and size of the volume in the respective dimension. The angle of the crosshair defines the absolute rotation of the renderers.
A direct modification by using the left mouse button defines a relative change of the pose.
The lower right subwindow shows a cube illustrating the boundaries of the volume and a visualization of each slice as it lies in space. One can rotate, translate and zoom the whole cube analogous to the mouse navigation of the CameraGL class.
Double clicking the left mouse button in a certain subwindow switches to the Fullview mode for the respective renderer.

The Fullview Mode

The Fullview mode shows one single renderer or the cube containing the slice in space separately and in full window dimension. This mode serves for better visualization and diagnosis, once orientation is defined and a stack of slices is to be examined. (Figure 2.2)

Mouse navigation in the Fullview mode is analogous to the GLCamera class. Dragging the mouse with the right button clicked defines a relative translation in x and y directions, the left mouse button let's the slice rotate around the axis orthogonal to the mouse movement. Moving the mouse up and down holding the middle button pressed modifies the z-pose of the renderer and shifts through the stack of slices, a left and right movement rotates all renderers around the the z-axis of the slice shown in fullview.
The UI of the SliceView in this mode serves the same purposes, there are three rollers for the rotations and a slider to shift through the stack. Further the pose of the slice is printed in a box on the right hand side of the view.
All changes in this mode are applied relatively and to all renderers to keep up the orthogonality.
Double clicking the left mouse button will switch back to the 3-Slice mode. Double clicking the right mouse button on the other hand switches to a third mode, the Cubeview mode.

The Cubeview Mode

When the user switches from Fullview to Cubeview mode the renderer priorly shown in Fullview is warped into the volume cube without blending in the other two slices. This helps the user to orientate where exactly the slices can be found in the volume. Interaction

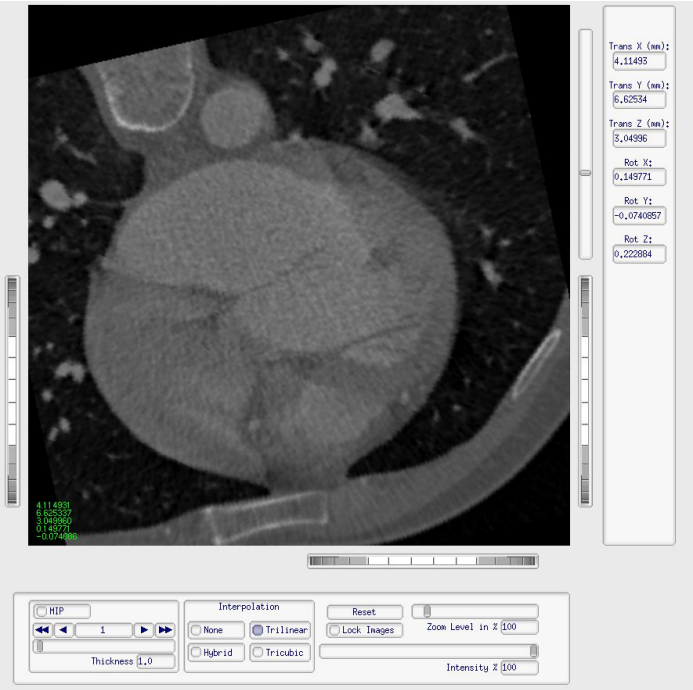

Figure 2.2: The Slice View in Fullview Mode

in this mode is the same as in fullview mode. It would be desirable to blend in a surface renderer of the volume on demand in a further course of development.

2.3.2 Functionality

The SliceViewGL holds the actual OpenGL content for drawing the image content. Based on user defined properties a byte image is rendered out of the volume data by the according image node. This byte image is passed on to the SliceViewGL for visualization, whereby numerous images can be merged to be shown as a fused overlay. All rendering is done by the nodes them self, so once an image is rendered it can be visualized and modified at low computational cost within the SliceViewGL as long as now new rendering has to be done. One can easily switch through different display modes or manipulate how the image is do be displayed (e.g. intensity, relative offset to reference image).
In order to use the feature of fused visualization for manual registration, it is important to differentiate between a pose that is applied on a single slice to define it's pose within the volume and a second pose that is applied to the whole volume to shift and rotate it in reference to secondary image data.(Figure 2.3 illustrates how a CT scan of the heart might relate to a the scan of the same patients thorax)
If two or more volumes are loaded, the user can select in the MainFrame user interface which image should be modified when interaction occurs. All changes made in the 3-Slice-Mode affect the three orthogonal slices as described above. When in Fullview mode on the other hand, a translation defined by shifting the slice with the right mouse button in x- and y-direction is applied to the whole volume momentarily selected to enable manual

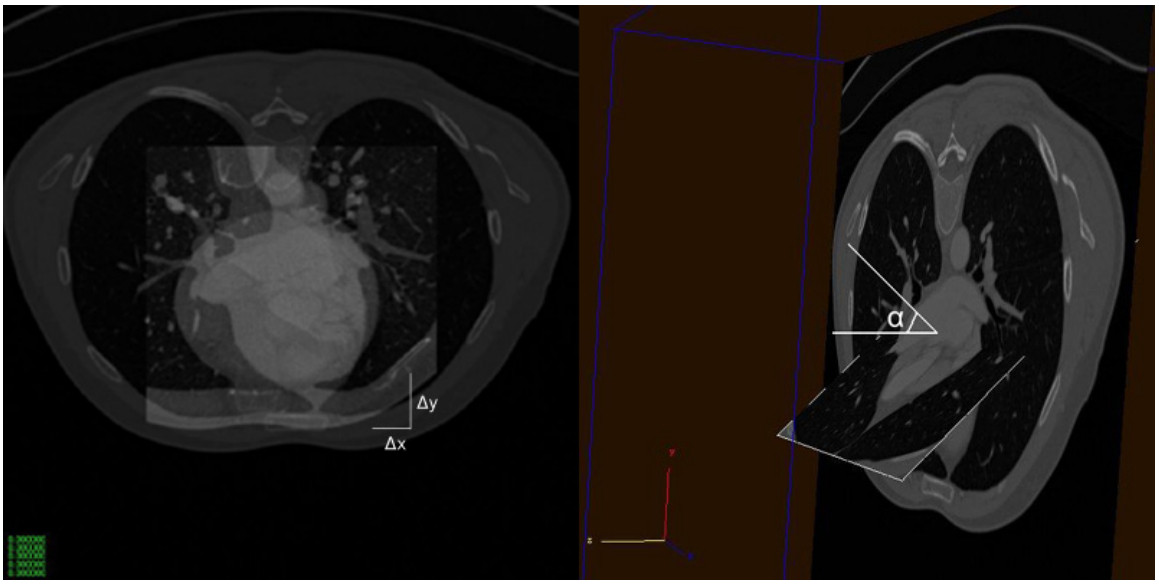

Figure 2.3: Relative translation and rotation between two images

registration.
All modifications affect only the selected volume unless the "Lock Images" is set. In this case all images are modified respectively.

Relationship between two volumes expressed through relative pose

Each slice renderer contains a pose vector with six entries to define its position within the volume. The first three entries describe the translation of the center of the slice in respect to the center of the volume while the last three entries describe its eukledian rotation. Therefore the pose vector describes a rigid transformation matrix

$$H_i = \begin{bmatrix} R_i & \vec{t_i} \\ 0\vec{T} & 1 \end{bmatrix} \text{ , for each renderer } i \in \{1 \dots 3\} \tag{2.1}$$

This homography can be split into a part that is valid for the whole model (rotation and translation) and a translational part dependent of the renderer. This individual translation describes the leaf through the stack of slices in the z-direction of the renderer.
Since all three slices always stand orthogonal to each other there is one global rotation matrix that, multiplied by the respective permutation of the canonical coordinate system gives the rotation matrix for each renderer.

$$R_i = R_M B_i \text{ , } i \in \{1 \dots 3\} \tag{2.2}$$
$$R_M = rot_x rot_y rot_z$$
$$B_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [b_1 b_2 b_3] \quad B_2 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = [b_3 b_1 b_2] \quad B_3 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = [b_2 b_3 b_1]$$

The translational part consists of a global translation t_M for the whole model and a out of plane translation that is specific for each renderer. (Figure 2.4)

$$\vec{t_i} = \vec{t_M} + R_i \vec{t_i'} \tag{2.3}$$

$$\vec{t_i'} = \begin{bmatrix} 0 \\ 0 \\ \lambda_i \end{bmatrix}, \lambda_i = \text{out of plane translation for renderer i, } i \in \{1 \dots 3\}$$

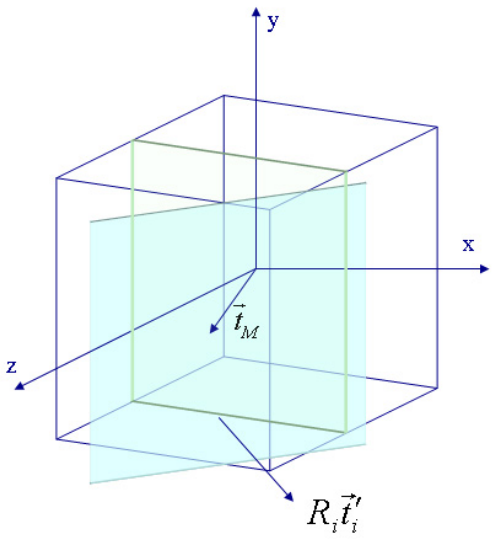

Figure 2.4: Global and slice specific out of plane translation of a slice renderer

The relative part of the transformation matrix is consistently kept, as it states the transformation that has to be applied to a volume to registrate it with a reference volume. Mathematically it can be described as

$$H_{rel} = \begin{bmatrix} R_M & \vec{t_M} \\ \vec{0^T} & 1 \end{bmatrix} \tag{2.4}$$

3 Realignment of heart image data

3.1 Standards for tomographic imaging of the heart in cardiac applications

There are several imaging modalities to measure myocardial perfusion, left ventricular function and coronary anatomy for clinical management and research, such as nuclear cardiology, echocardiography, cardiovascular magnetic resonance (CMR), cardiac computed tomography (CT), positron emission computed tomography (PET) and coronary angiography. All these modalities image the myocardium and the adjacent cavity. But since they show technical differences, features like e.g. orientation of the heart, angle selection for cardiac planes, number of segments or slice display and thickness have evolved independently within each field, based on the inherit strength and weakness of the technique and it's practical application. This independent evolution has resulted in a lack of standardization and has made accurate intra- and cross-modality comparisons for clinical patient management and research very difficult.
To overcome this difficulties, it is necessary to standardize the options for all cardiac imaging modalities, whereby the selection of the standardized methods have to be based on certain criterias: Consistency with accepted anatomic and autopsy data has to be maintained, as much as possible existing and accepted approaches to myocardial segmentation and nomenclature are to be utilized, whereby an precise localization using anatomic landmarks has to be allowed. Further an adequate sampling of the left ventricle and coronary distribution has to be provided without exceeding the resolution limits of the imaging modality, and finally linkage of the segments to known coronary arterial topography as defined by coronary angiography has to be allowed.

Based on these criterias, consensus recommendations have been made to optimize and facilitate communication between cardiac imaging modalities for research and clinical applications for among other aspects the orientation of the heart and the selection and thickness of cardiac slices for display and analysis.
In the development of the project focus was set on these two aspects and there technical realization.

3.2 The orientation of the heart

Concerning plane selection and slice orientation for serial myocardial slices generated by cardiac 2D or tomographic imaging modalities standards were defined by the American Heart Association, the American College of Cardiology and the Society of Nuclear Medicine as shown in Figure 3.1.

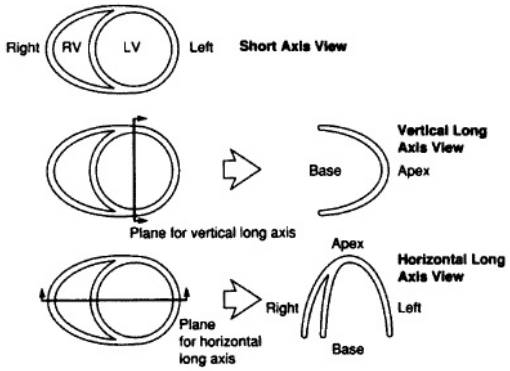

Figure 3.1: Cardiac plane definition and display for tomographic imaging modalities

Digital cross-sectional or tomographic imaging modalities such as body CT and MR traditionally oriented and displayed the body using planes that were parallel or at 90° angles to the long axis of the body, called transaxial or body-plane orthogonal views. The cardiac planes generated by using the long axis of the body do not cleanly transect the ventricles, atria or myocardial regions supplied by the major coronary arteries. Single photon emission computed tomography (SPECT) and 2D echocardiography, the most widely used cardiac imaging modalities, have defined and oriented the heart for display at 90° relative to the long axis of the left ventricle that transects the apex and the center of the mitral valve plane. This approach maintains the integrity of the cardiac chambers and the distribution of coronary arterial blood flow to the myocardium. For these reasons, this approach is optimal for use in research and for clinical patient management involving cardiac perfusion and function.
It is therefore recommended that all cardiac imaging modalities should define, orient and display the heart using the long axis of the left ventricle and selected planes orientated at 90° angles relative to the long axis.
By identifying the apex and the center of the mitral planes from slices captured by a tomographic imaging modality, one can define the orientation of the long axis of the left ventricle and thus realign the image to fit this standard.

Realigning data with the viewer

Based on the aspects stated above, it becomes easy to realign heart data using the fusion viewer. The cardiologist can identify visible features such as the apex in any of the three orthogonal views and set the crosshair to define a plane that includes the heart axis. A second view now shows the slice including the axis, that can once again be defined by setting the crosshair. Due to the consistently kept orthogonality, the third view now shows a stack of slices always in 90° angle to the heart axis.
The pose of the slice that defines the transformation to achieve realignment can be exported and used in other applications working with the Slice Renderer (camplib).

Bibliography

[1] Manuel D. Cerqueira et al. *Standardized Myocardial Segmentation and Nomenclature for Tomographic Imaging of the Heart*. American Heart Association Writing Group on Myocardial Segmentation and Registration for Cardiac Imaging, Circulation 2002;105:539-542, 2002.

[2] Joseph V. Hajnal, Derek L.G. Hill, and David J. Hawkes, editors. *Medical Image Registration*. CRC Press, 2001.