Appendix A

UGviewer: a medical image viewer

As a complement to this master's thesis, an own medical image viewer was programmed. This piece of software lets the user visualize and compare images. Designing it helps to understand how the medical imaging data is stored in a computer and how this data is transformed into an image that can be displayed on a computer screen.

A.1 Specifications

The piece of software should be able to:

1. Display two studies at the same time, so that they can be easily compared. These two studies could be two CTs or PETs in different respiratory phases, or a PET and a CT from the same patient.
2. Show sagittal, coronal and transverse views.
3. Zoom in the different views.
4. Use different color maps in order to highlight different features in the images.
5. Change the image contrast, with the same purpose of highlighting different tissues.
6. Superimpose the two studies in order have fused images.
7. Draw, erase, open and save regions of interest (ROI).
8. Save views in the most popular 2-D image files, like $*.jpg$ or $*.png$.

A.2 Solution

C++ and Java are currently two of the most popular object oriented programming languages. Both of them offer easy ways of programming graphic user interfaces (GUI). A slight disadvantage with Java is that it is more difficult to program low level operations than in C++, that, on the contrary, keeps all the low level - high speed features of the C programming language. This is desirable in order to read Analyze format image headers, for example (in fact, this header is defined as a C structure). That is why C++ was chosen for programming this viewer.

Qt [45] is "a cross-platform C++ application framework developers can use to write single-source applications that run natively on Windows, Linux, Unix, Mac OS X and embedded Linux". It has been used to build thousands of successful commercial applications worldwide (it is for example the basis of the open source KDE desktop environment). Qt provides the programmer with over 400 C++ classes (and growing), which encapsulate all infrastructure needed for end-to-end application development. Qt was thus a good choice for out piece of software.

Qt has also the advantage of having an open source version. Using this version forces the programmer to automatically agree with that the source of the software will be published according to the GNU GPL license [46]. Using the Open Source Edition means that one agrees with that the source of the software one writes will also be published according to this license. This open source version is specially prepared for working with the C++ MinGW compiler [47], which is the one that actually has been used.

It is finally important to say that the program is meant to work with Analyze format images. This is a standard format that can be used by most of the medical imaging applications nowadays.

A.3 User Guide

A.3.1 Main Window

When the application is started, and after a splash screen, the main window is shown (figure A.1). The screen is divided into an upper region for the first image and a lower region for the second one. There are three different important areas in each region: the image control menu, the contrast and slice number sliders, and the images themselves.

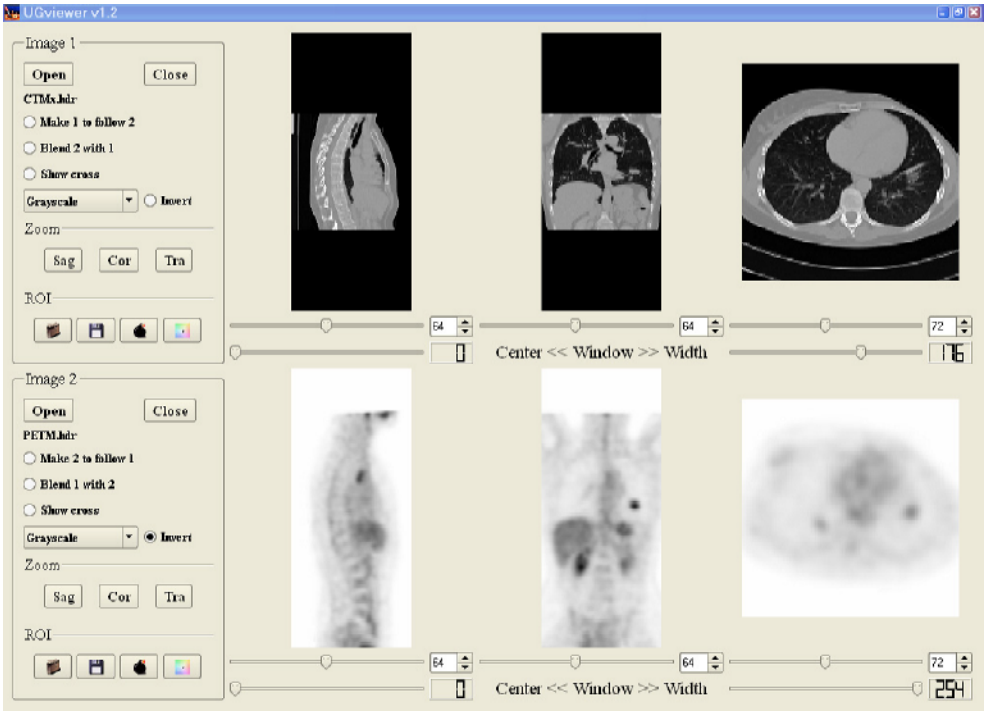

Figure A.1: Main application window.

The volumes

The volumes are represented as a collection of sagittal, coronal and transverse views. The three views are presented at the same time, scaled to be shown as large as the screen allows the application. The current slice for each view can be changed with the corresponding slider. It is also possible to click on one view in order to surf around the other two.

Contrast slider

There is also a pair of contrast slices for each image. The voxel values are scaled from 0 to 254 (255 is kept for ROI voxels). The voxels whose values are around a certain one can be highlighted by placing a window around it (figure A.2). All the voxels with a value smaller than the lower window threshold are mapped to 0, while all the ones with a value bigger that the higher threshold are mapped to 254. The thresholds are also limited by 0 and 254. The values inside the window are then mapped linearly.

The window's center is controlled by the slider on the left, and its width (center to both thresholds) by the slider on the right. An example of how it can be used with a CT image is shown in A.3, where the window's center is

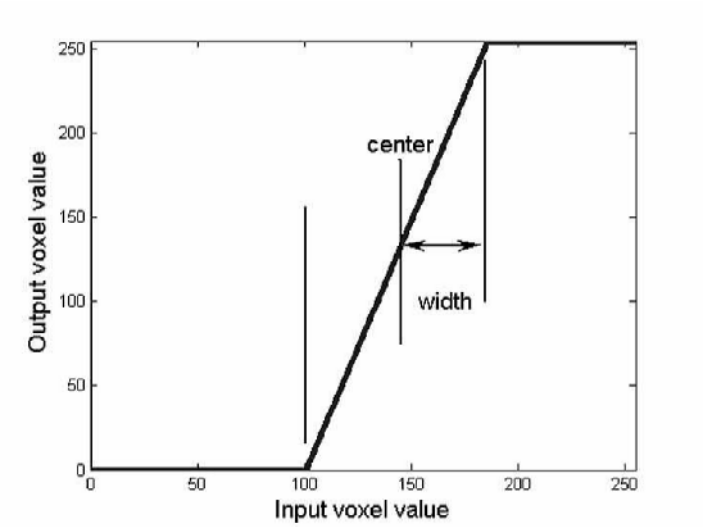

Figure A.2: Voxel value transformation depending on the contrast window's center and length. Mind the definition of width.

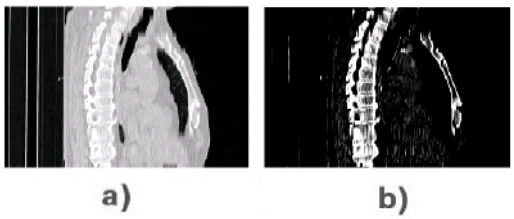

Figure A.3: a) Normal contrast CT image, sagittal view b) Same image but with the contrast stretched around the typical bone voxel value.

placed around the bone's value and the width made quite small. This way, the bones are highlighted.

Image control menu

Some other interesting controls are placed on the left of the images. If the synchronize ("make 1 to follow 2") button is activated, the slice numbers for one image follow automatically the other image's, so that the same views are shown for both volumes.

After the blend control, that will be explained afterwards, the "cross" button shows a crossbar on each view. These crossbars point out the location of the other two slice planes in the current view, as shown in figure A.4.

The "colormap" buttons control how a 8-bit integer will be mapped to a

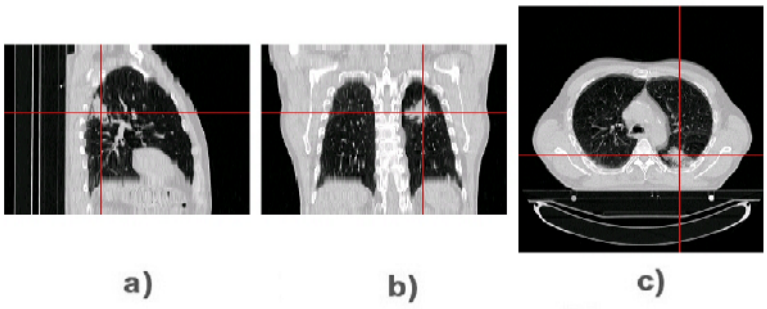

Figure A.4: a) CT sagittal view b) Coronal view c) Transverse view, all of them with the slice position of the other two marked on it, pointing to a lesion.

typically 32-bit display. This is explained more carefully later on.

The zoom buttons open new windows showing a scaled version of a certain view. These new windows have some other options that will be explained afterwards.

The ROI controls let the user work with $*.roi$ files, which work only with this application. The button with a folder is used to load a ROI. The button with a floppy disk is used to save the current ROI. The button with a "colormap" is used to choose the color that will be used to draw the ROI on top of the image. The button with a bomb cleans the ROI from the image.

A.3.2 The zoom window

The zoom window (figure A.5) is the real working window. The main widget should be seen as a preview area, as there is not much room for the different images.

In the right side of the zoom window, the image is displayed, while the left one is used for displaying information and for the control buttons.

The first button is the zoom size button. It lets the user choose between 50%, 100%, 200%, 300%, 400% and "fit to screen" versions of the image. As a reference, a 100% zoom is the result of stretching the image along the biggest pixel size dimension until the right proportion is reached.

After another slice control slider, the main information from the image is displayed. The value and coordinates (both in pixels and in physical length) for the pixel under the cursor in each instant are shown here.

The following button is the "save view" button. It opens a save file dialog in order to save the current displayed view as an image file. The format may be chosen with the control on the right of the button.

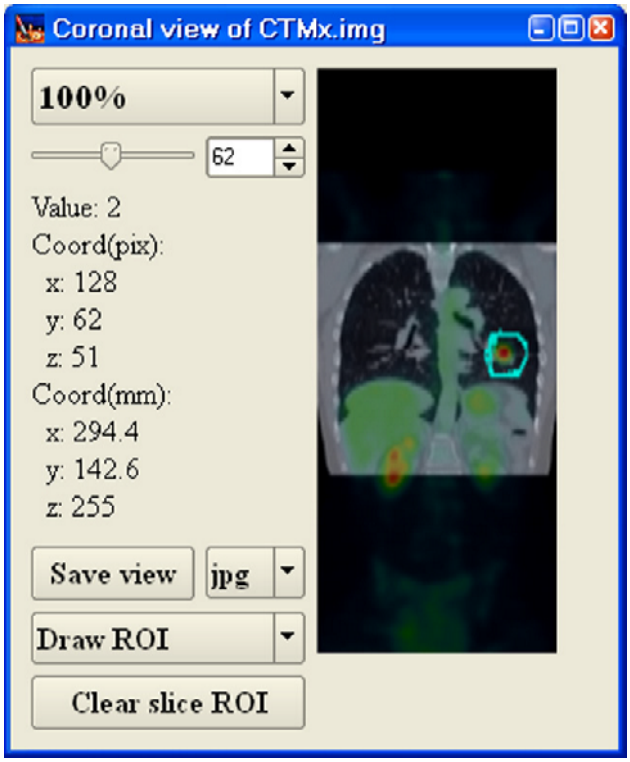

Figure A.5: Zoom window with a coronal view from a patient. A ROI marking a lesion has been drawn.

The last controls are the ROI ones. The draw / erase ROI button lets the user decide wether if he wants to add or take away points from the ROI. This is done just by clicking and dragging the mouse over the image. The drawn ROI for just the current slice can be cleared with the button underneath.

A.3.3 Color maps

One of the most interesting aspects of the application is the possibility of using different color maps. 255 voxel values (the top one is reserved for representing ROIs) are to be mapped to the 2^{32} different colors that can be represented on the screen (assuming that a 32-bit display is used). In that case, each pixel is described by four 8-bit components: red channel, green channel, blue channel and alpha channel (which indicates the color's transparency). This is the internal machine representation, but in this program the HSI (hue, saturation, contrast) model will also be used. HSI can be easily translated to the RGB one with Qt. The transparency will be set to the maximum to represent the normal views.

Seven different color maps are available (l represents the voxel value):

1. Greyscale: $R = G = B = l$, that is, 255 gray levels from black to white.
2. Red: $R = l, G = B = 0$, that is, 255 red levels from black to pure red.
3. Green: $G = l, R = B = 0$, that is, 255 green levels from black to pure green.
4. Blue: $B = l, R = G = 0$, that is, 255 blue levels from black to pure blue.
5. Hue 0-254: $H = l, S = 255, I = 128$, that is, full saturated colors from red (0°) to blue(255°).
6. Fire: $H = 0, S = l, I = 128$, that is, red tones from grey (saturation 0) to pure red (maximum saturation).
7. Ocean: $H = 255, S = l, I = 128$, that is, blue tones from grey (saturation 0) to pure blue (maximum saturation).

Different color maps highlight different features. There is a "invert colormap" button that turns the high values into low ones and viceversa, using a transform:

$$y = 254 - x$$

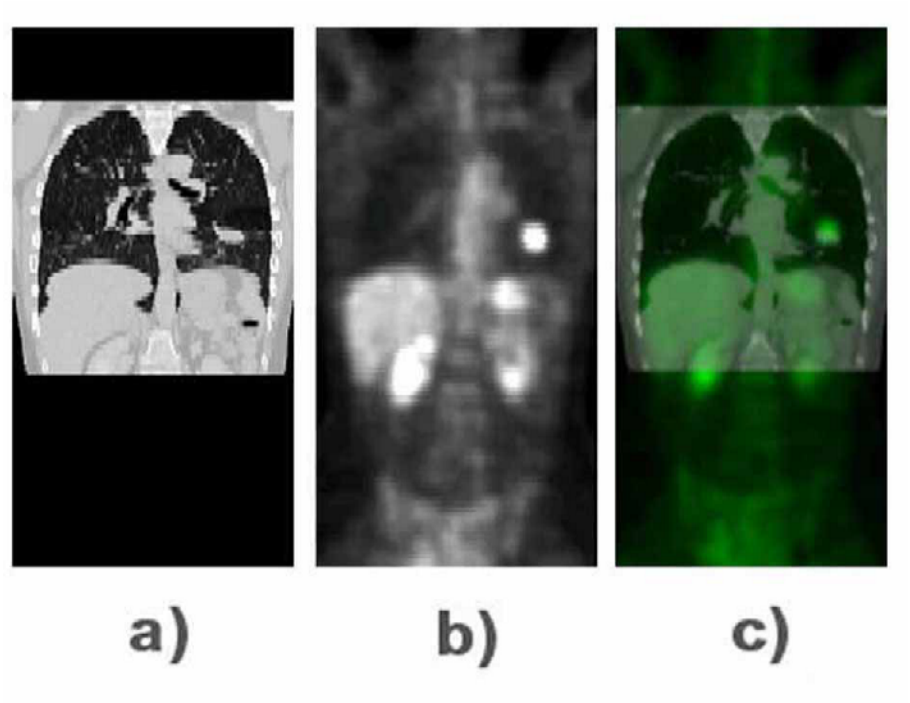

Figure A.6: a) CT coronal view b) PET coronal view c) Fused image. There is clearly a lesion in the right lung.

Color maps are are especially useful when combined with the program's last feature: the "blend" button. This button superimposes both images, assuming that they have the same size. The idea is that one image's alpha channel is made equal to the intensity for every voxel, and then laid on the other image. That way, the voxels are blended according to their "importance" (their value). A sample view is shown in figure A.6, where a greyscale CT is fused with a "Hue 0-255°" PET.