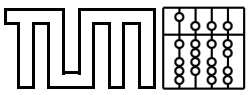

Systems Development Project Documentation (SEP)

OsiriX Segmentation Plugin Development

Brian Jensen

July 13, 2009

Advisor: Prof. Dr. Nassir Navab
Supervisor: Ralph Bundschuh

Technische Universität München
Department of Computer Science
Chair for Computer Aided Medical Procedures and Augmented Reality
http://campar.in.tum.de

Technische Universität München
Department of Nuclear Medicine
University Hospital Klinikum Rechts der Isar
http://www.nuk.med.tu-muenchen.de/

Contents

1 Implementation

This section covers the theoretical background on the segmentation algorithms offered in the NMSegmentation plugin as well as a discussion of the plugin's architecture. The companion document *ITK Configuration Guide* covers the necessary steps for setting up a workstation for the development of an OsiriX plugin using the ITK framework, as such that information is not covered here.

1.1 Segmentation Algorithms

There are four segmentations implemented in the NMSegmentation plugin. All four are variants of a segmentation technique referred to as region growing. Region growing algorithms start with a seed voxel or region that is determined to be within the region that is to be segmented. Starting with the seed all neighboring voxels are visited to determine if they belong in the segmented region. If they are included then their neighbors are also evaluated. This process continues until there are no voxels to be included. Region growing algorithms have proven themselves to be well suited for a wide variety of segmentation tasks.

The implementation of the different segmentation algorithms was accomplished using the Insight Registration and Segmentation Toolkit (ITK) [1]. ITK offers a large variety of image processing tools in addition to offering several segmentation algorithms. The NMSegmentation plugin uses three standard ITK segmentation algorithms as well as one custom implemented algorithm, also using the ITK framework.

1.1.1 Connected thresholding

The simplest method for implementing a region growing algorithm is to combine it with an image processing technique known as thresholding. In thresholding only image voxels whose intensity values fall within a certain interval are included in an image. When used in a region growing segmentation, only neighboring voxels whose intensity value is within the interval would be included in the region. The interval is typically specified by the user as an upper and lower threshold. More generally the thresholding based region growing criterion can be formulated as

$$I(X) \in [lower, uppper]$$

where $I(X)$ represents the intensity value of the image at voxel location X.

ITK offers several variations of thresholding based region growing algorithms, two of which are used by the plugin. The simplest implementation is the connected threshold image filter, implemented in the class `ConnectedThresholdImageFilter`. This class implements the region growing approach described above, it accepts a user specified seed

voxel as well as the lower and upper thresholds, and returns the segmented region from those parameters.

In the NMSegmentation plugin's implementation the seed voxel is determined when the user selects a location in the image. The thresholds can be set either completely manually or fully automatically. The fully automatic variant works by searching a region around the seed voxel for the maximum intensity value. The size of the region is specified by the user in the plugin's interface. The lower threshold is then set to a certain percentage of the maximum value, where the percentage is also set in the interface.

1.1.2 Neighborhood connected thresholding

Neighborhood connected thresholding is almost identical with the connected thresholding approach. The main difference between the two lies in the region inclusion criterion. In connected thresholding, a voxel is included if it is within the interval, in neighbor connected thresholding a voxel is included only if all of its neighbors are also inside of the interval.

This functionality is implemented in the class `NeighborhoodConnectedThresholdImageFilte` This class takes the same parameters as `ConnectedThresholdImageFilter`, with the addition of a neighborhood size. The neighborhood size controls how large the neighborhood in each direction should be evaluated.

1.1.3 Confidence connected

Confidence connected region growing also uses a thresholding approach for the inclusion criterion, but in contrast with the two previous implementations it uses image statistics for setting the lower and upper thresholds. The confidence connected algorithm begins with a seed voxel and an initial neighborhood size. The algorithm then calculates the mean and standard deviation of the voxels in the initial neighborhood. All neighboring voxels are included if they fall within the range of the standard deviation around the mean value. This can be formulated more generally as

$$I(X) \in [m - f\sigma, m + f\sigma]$$

where m is the mean value, σ the standard deviation and f is a constant factor. When no further voxels can be included the algorithm recomputes the standard deviation and mean value including the newly added voxels. This process is then repeated until the maximum amount of iterations has been exceeded.

The ITK class `ConfidenceConnectedImageFilter` implements the confidence connected algorithm. This class takes three main parameters in addition to the seed point. The initial neighborhood size has to be specified, as well as the factor applied to the standard deviation and the maximum number of iterations. All of the parameters are set by the user in the interface.

1.1.4 Gradient magnitude thresholding

Gradient magnitude thresholding is an alternative approach for evaluating the inclusion criterion in region growing algorithms. In contrast with thresholding based algorithms,

the voxel's intensity value alone is not used as grounds for inclusion into the segmented region, instead the gradient magnitude ratio of the voxel at that location is used.

The main reason for choosing this different type of strategy for an inclusion criterion has to do with the target image types of this algorithm. This segmentation method was designed to be carried out on images obtained from nuclear imaging modalities, such as PET or SPECT. In these image types the most common segmentation task is to isolate regions of significantly higher intensity compared to their surroundings. In oncological studies these regions are mostly caused by tumorous lesions, which exhibit specific intensity distribution characteristics. One of the characteristics is a steep intensity decline at the border of the lesion. This is in large contrast to the region within the lesions borders, where intensity values are more homogenous. The idea behind this fourth segmentation algorithm was to develop a region growing variant that uses relative changes in intensity values as the criterion for inclusion into the segmented region.

The concrete approach for implementing this idea involves two main steps. Since this is a region growing based algorithm, the process will begin with a user selected seed voxel. For each voxel the algorithm first estimates the image gradient magnitude at that location. Since it is generally not possible to analytically determine the gradient at any point in an image, a three dimensional sobel convolution kernel is used to approximate the gradient in each direction, given by the following following formulas for each direction

$$G_x = \underset{z_{-1}}{\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}} \quad \underset{z}{\begin{bmatrix} -3 & 0 & 3 \\ -6 & 0 & 6 \\ -3 & 0 & 3 \end{bmatrix}} \quad \underset{z_{+1}}{\begin{bmatrix} -1 & 0 & 1 \\ -3 & 0 & 3 \\ -1 & 0 & 1 \end{bmatrix}}$$

$$G_y = \underset{z_{-1}}{\begin{bmatrix} -1 & -3 & -1 \\ 0 & 0 & 0 \\ 1 & 3 & 1 \end{bmatrix}} \quad \underset{z}{\begin{bmatrix} -3 & -6 & -3 \\ 0 & 0 & 0 \\ 3 & 6 & 3 \end{bmatrix}} \quad \underset{z_{+1}}{\begin{bmatrix} -1 & -3 & -1 \\ 0 & 0 & 0 \\ 1 & 3 & 1 \end{bmatrix}}$$

$$G_z = \underset{z_{-1}}{\begin{bmatrix} -1 & -3 & -1 \\ -3 & -6 & -3 \\ -1 & -3 & -1 \end{bmatrix}} \quad \underset{z}{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}} \quad \underset{z_{+1}}{\begin{bmatrix} 1 & 3 & 1 \\ 3 & 6 & 3 \\ 1 & 3 & 1 \end{bmatrix}}$$

where x, y, z are the image coordinates of the voxel [4]. Each of these directional gradients is then combined to form the cumulative gradient magnitude using the next formula

$$G = \sqrt{(G_x)^2 + (G_y)^2 + (G_z)^2}.$$

In the next step the average intensity value from the voxel and its 26 neighbors is calculated. The gradient magnitude is then weighted against the average intensity value, and if the ration exceeds the user specified threshold the voxel is discarded. If the ratio is below the threshold then the voxel's face connected neighbors are marked for evaluation. The algorithm terminates when no more voxels can be included into the segmented region. The class `itkGradientThresholdImageFilter` implements this functionality.

Initial tests have shown this alternative approach for segmenting lesions in PET images to be relatively successful, although a thorough quantitive analysis still needs to be performed.

1.2 Plugin architecture

This sections covers aspect of the NMSegmentation plugin's implementation that might not be apparent to a developer from the source code. In general anyone wishing to work on the plugin's segmentation algorithms should have a solid understanding of objective-c and cocoa, as well as C++. For the user interface the developer should be familiar with the OsiriX source code and the plugin development guide [3].

The plugin's base class is `NMSegmentationFilter` which is the plugin's entry point and is only responsible for starting the user interface window. This class inherits from the base OsiriX image processing plugin class `PluginFilter`. The method

```
- (long) filterImage:(NSString*) menuName;
```

is triggered when the plugin is activated by the user from the plugin's menu. This method first searches all open OsiriX windows for a window with the same name as the window panel in the *NMRegionGrowingController* nib file. If one is found and the window's contoller class is the same as the plugin's windows controller class and the viewer settings match, then the matching window is simply redisplay. Otherwise a new window controller class is created and is passed the currently active `ViewerController` object and its registered viewer, if one is present.

The main interface window's controller class is `NMRegionGrowingController`. This class is initialized with the current active `ViewerController` object and a registered viewer object, although the second parameter may be `nil`. Before the class is able to load the user interface window it first has to load the shared user defaults controller with a dictionary containing keys value pairs for each widget in the user interface window that binds to the shared user defaults. This is accomplished by

```
[[NSUserDefaultsController sharedUserDefaultsController]
 setInitialValues:[self getDefaults]];
```

where `[self getDefaults]` returns a `NSDictionary` object containing all the required keys, this concept is explained further in section 1.2.1. After loading the window from the nib file the controller class adds itself an observer of two important events, the `CloseViewerNotification` event that is sent when a `ViewerController` object is about to close, and the `mouseDown` event, which is sent by a viewer when a click event is registered. This is accomplished by adding the current controller object to the default notification center:

```
NSNotificationCenter* nc = [NSNotificationCenter defaultCenter];
[nc addObserver: self
selector: @selector(mouseViewerDown:)
```

```
name: @"mouseDown"
object: nil];

[nc addObserver: self
selector: @selector(CloseViewerNotification:)
name: @"CloseViewerNotification"
object: nil];
```

The segmentation is controlled by the class `NMRegionGrowing3D` that interfaces with the ITK code by maintaining an `ITKImageWrapper` object containing the raw image data for segmentation. If the segmentation is performed on a registered viewer, the object also maintains a size and spacing parameters from the main viewer for resampling purposes.

1.2.1 User interface

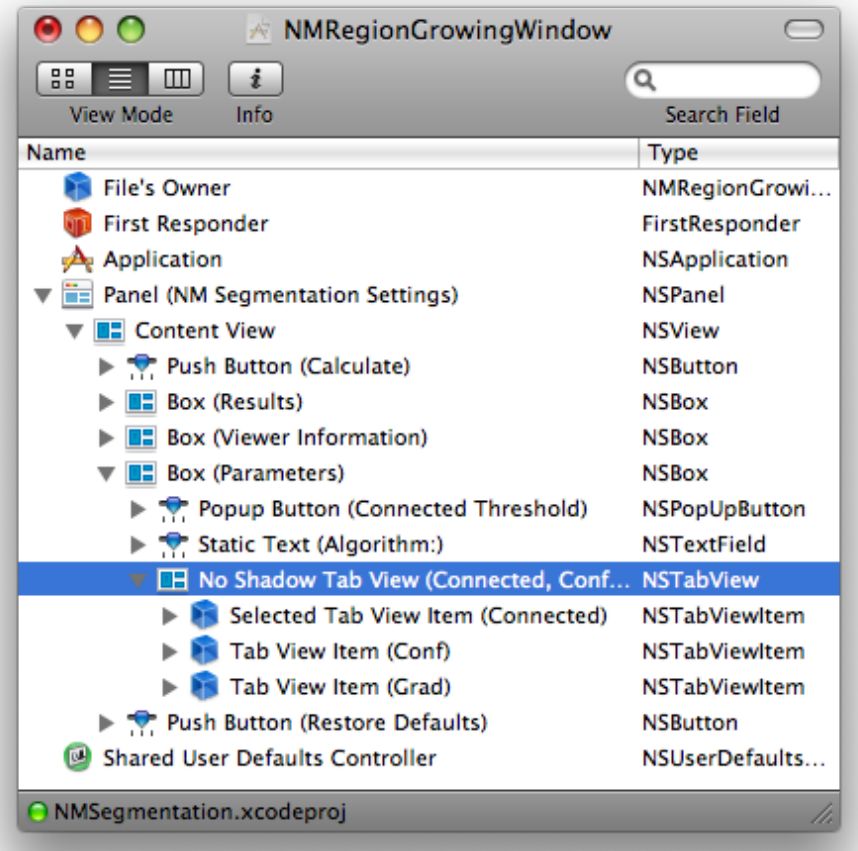

Figure 1.1: The NMSegmentation Hidden tabview element

There are important topics that need to covered in order to fully understand how the user interface is implemented, although the methods discussed are hardly unique. The most important is the method used to implement the algorithm switcher and its dynamic window resizing abilities.

Cocoa in general doesn't offer widgets that automatically resize or hide their contents. This behavior has to be implemented by hand for the most part. There is however a clever trick available that makes implementing such functionality easier. Basically the trick involves placing user elements in a `NSTabView` element using Interface Builder.

In the case of the NMSegmentation plugin a NSTabView was created with three different tabs, one for each different set of options. In the NSTabView options under Attribute Inspector the tab view was set to tabless. This makes the tab view appear embedded in the panel that contains it, but it also prevents the user from directly switching between tabs. Instead the tab switch event is triggered when the chooses a different algorithm from the algorithm pop up menu. Whenever a different algorithm type is selected first the window is resized, then the selected tab item is changed from inside of the controller class. In order to edit the different tab view items inside of Interface Builder you need to switch the selected tab in pane list view as can be seen in figure 1.1.

The other important aspect to take into consideration is the interface's use of the mac shared user defaults system. Each non transient option in the user interface is saved using the user defaults system under the OsiriX domain, so that the user's settings are persistent across instances. This is accomplished using the cocoa bindings' settings from within Interface Builder. For each interface element that should have a persistent state you need to specify that the element bind its value property to the shared user defaults controller in the bindings properties for that element, see figure 1.2. It is important that the **Model Key Path** setting contains a value that is unique OsiriX wide, thats why all names contain the *NMSeg* prefix.

1.2.2 Debugging

To aid in debugging a pair of macros was developed to aid in enabling / disabling of debug messages and settings. The function `DebugLog` can be used in place of `NSlog` and the function `DebugEnable` only executes the code contained within when the macros have been activated. The two macros are activated either when the plugin is compiled with the symbol `DEBUG` defined (Development mode) or when the variable NSDebugEnabled is set (Deployment mode). The advantage of this strategy is that the plugin can be compiled and deployed to a production workspace with debugging information that can be activated later on if problems occur or left deactivated for better performance. For activating the debug macros in a deployed product see section 2.4

```
#ifdef DEBUG
#define DebugLog(...) NSLog(__VA_ARGS__)
#define DebugEnable(...) __VA_ARGS__
#else
#define DebugLog(...) if(NSDebugEnabled) NSLog(__VA_ARGS__)
#define DebugEnable(...) if(NSDebugEnabled) __VA_ARGS__
#endif
```

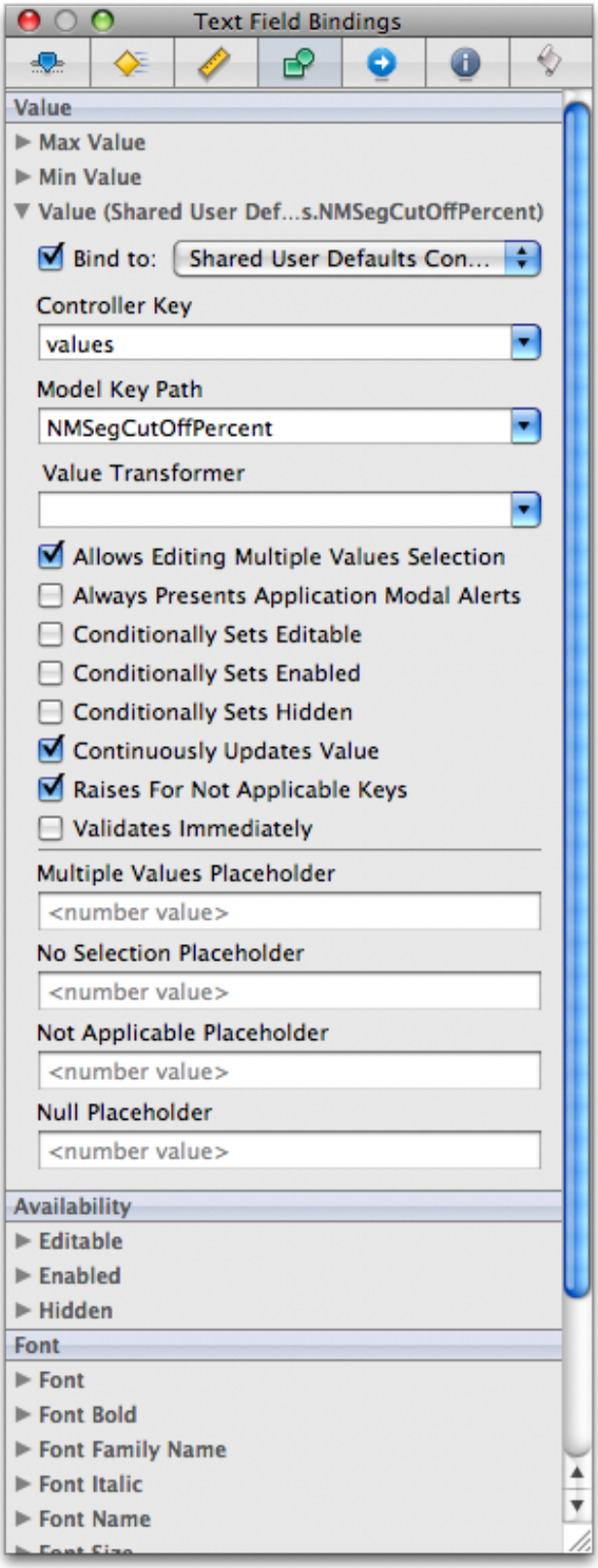

Figure 1.2: Shared user defaults binding settings

2 User's Guide

This section covers the basic usage of the plugin and its various features. OsiriX is a extremely capable program and it is beyond the scope of this document to cover its features in detail, only the features relevant for the plugin usage are covered. For more information on OsiriX see the Osirix Handbook [2].

2.1 Installation

Installation of the plugin is a fairly straight forward procedure with only one prerequisite, a working installation of OsiriX version 3.5 or newer. If have not attained a copy of the plugin it can be downloaded from http://campar.in.tum.de/Students/SepOsiriXSegmentation.

There are two options for installing the plugin, systemwide for all users or only for a specific user. To install the plugin system wide for all users simply double click the plugin icon. This should cause OsiriX to open and request an administrators password, depending upon the current security settings. Alternatively the plugin can be copied by hand in to the directory *Macintosh HD/Library/Application Support/OsiriX/Plugins*. To install the plugin just for a specific user, simply copy the plugin into the directory *HOME/Library/Application Support/OsiriX/Plugins* where *HOME* is the user's home directory. You can verify that the plugin is installed correctly by selecting the menu **Plugins** / **Plugins Manager...** inside of OsiriX. The plugin manager should appear and you should see the NMSegmentation plugin listed.

2.2 The user interface

2.2.1 Starting the plugin

After verifying the plugin has been successfully installed, you can begin using it on any data set of your choice. To begin simply open a data set you want to perform segmentation on, ideally this should be a data set containing a registered CT and PET image pair, although singular image data sets will also work properly. If OsiriX did not open the data sets in fused mode, you can manually activate the fused image mode by selecting the viewer window containing the CT data set, and right clicking the corresponding PET data set in the study navigation bar on the left side of the viewer window. After making sure the viewer setup you wish to perform segmentation on has the focus, select the menu **Plugins** / **ROI Tools** / **NMSegmentation Region Growing**.

The plugin user interface panel should then appear as can be seen in figure 2.1. The user interface is grouped into three sections, the viewer information and settings, the current segmentation parameters, and the result settings.

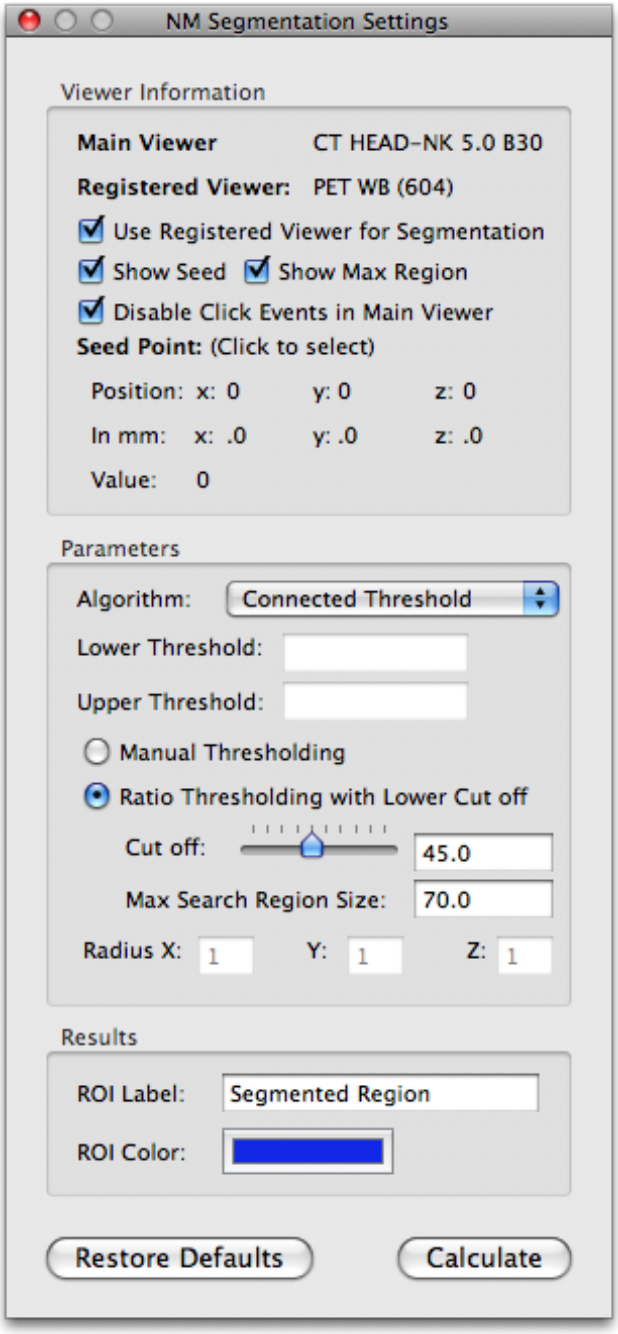

Figure 2.1: The NMSegmentation interface after startup.

2.2.2 Viewer options

The **Viewer Information** box contains information and options concerning the viewer setup. Most important is the information about which viewers the plugin is operating on, and information about the seed point.

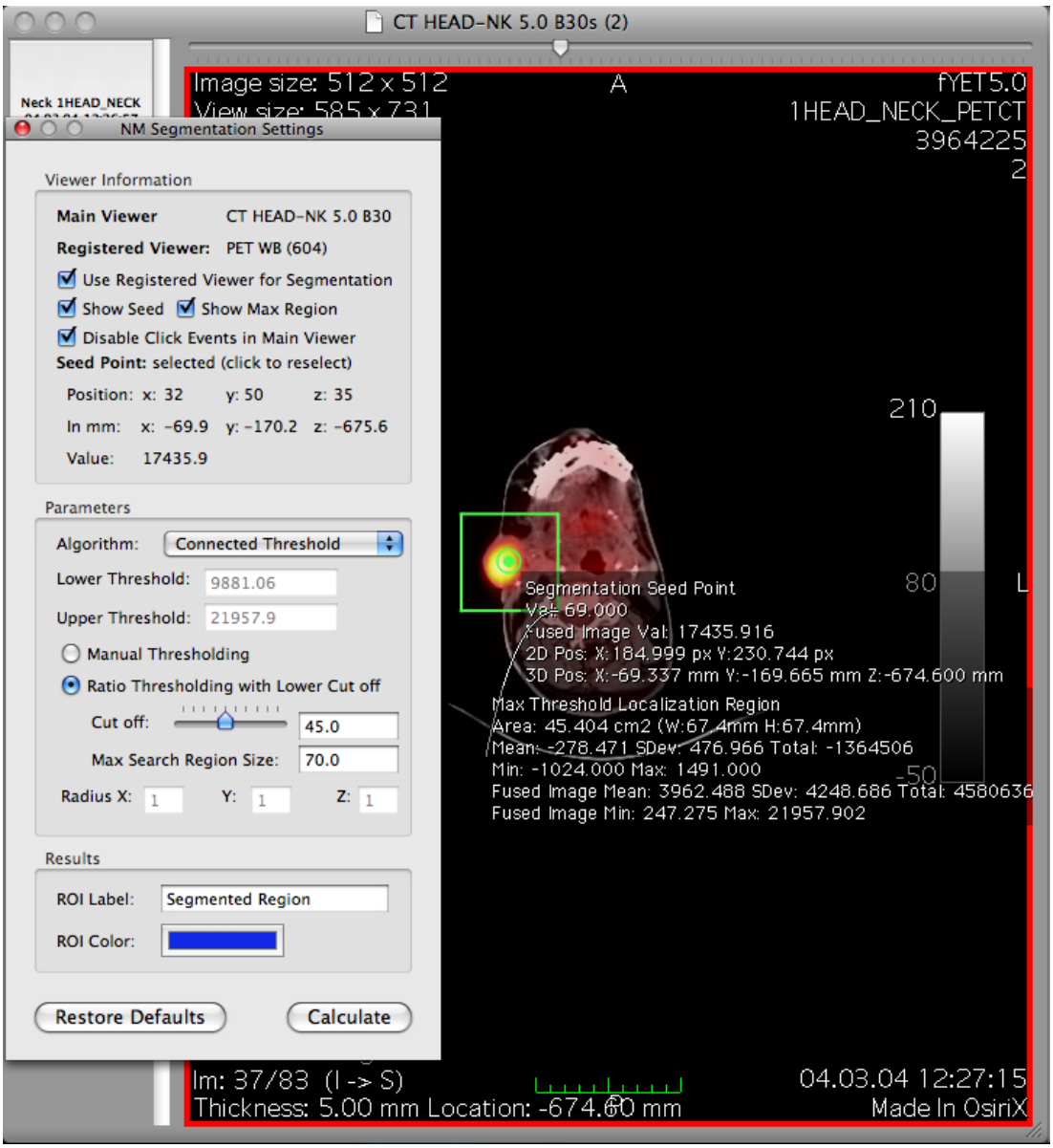

Figure 2.2: NMSegmentation after selecting a seed point in the viewer

Initially the seed point information is empty until the user selects a seed point in the main viewer, after which the interface is updated to reflect the new information as can be seen in figure 2.2. This information is automatically updated each time the user selects a new seed point.

If the plugin was started on a fused image viewer, such as a fused PET / CT image viewer, then the user has the option of performing the segmentation on the main data set, in this case the CT image, or on the registered data set, in this case the PET image. By selecting the *Use Registered Viewer for Segmentation* option the seed point information is automatically updated, as well as any other information that is dependent upon the seed point, such as the lower threshold of the connected threshold segmentation algorithm.

Below the registered viewer option are the seed point and max region ROI drawing options. When the **Show Seed** option is enabled, the Seed point is drawn on the image when the user selects a seed point. Similarly the **Show Max Region** option determines whether the maximum intensity search region is drawn on the viewer for the algorithms that use it (connected threshold algorithms). If the **Disable Click Events in Main Viewer** option is enabled, then the viewer is prevented from reacting to mouse clicks, if it is disabled then the plugin is prevented from acting on mouse events. This option is used to determine whether the plugin or the viewer react to mouse events, so as to prevent unwarranted conflicts.

2.2.3 Algorithm options

The **Parameters** box contains all of the settings relevant to the currently selected algorithm. There are four algorithms available in the NMSegmentation plugin. Their main characteristics are discussed in section 1.1. Since the parameters are specific to the chosen algorithm, only those parameters that are relevant are shown by the interface.

Connected Threshold

The available settings for connected threshold based segmentation can be seen in figure 2.1. In this algorithm the user has the choice between either manually setting the thresholds or semiautomatically using ratio drop off parameter. By default **Ratio Thresholding with Lower Cut off** is selected.

In this mode the lower and upper thresholds are calculated automatically based upon the seed point. The upper threshold is determined by searching a cube shaped region around the seed point for the maximum intensity value. The side length of the cube region is set by the parameter **Max Search Region Size**. This parameter is always specified in millimeters. The region is constructed by spanning from $seedPoint - size/2$ to $seedPoint + size/2$ in each image direction. By searching for the maximum value we can set the upper and lower thresholds relatively independent of the chosen seed point. The lower threshold is calculated using the **Cuf Off** parameter. This parameter specifies what percent of the upper threshold the lower threshold should be.

Neighborhood Connected Threshold

Neighborhood connected thresholding is almost identical to connected thresholding, with the exeption that neihborhood connected thresholding only segments a point if all of its neighbors within a user specified neighborhood are within the threshold. The parameters **Radius** specify the neighborhood size in each image dimension in voxels. As an example a neighborhood size of one in each direction will yield a neighborhood of 26 neighbors.

Confidence Connected

Confidence connected segmentation takes an entirely different set of parameters compared to connected thresholding. Figure 2.3 shows the available settings. The parameter **Initial Neighborhood** sets the initial neighborhood size in voxels. The parameter **Confidence Multiplier** specifies the multiplier applied to the deviation for settings the thresholds. The parameter **Number of Iterations** sets the maximum number of iterations the algorithm will perform.

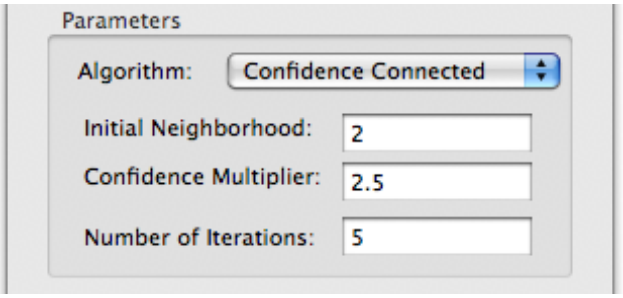

Figure 2.3: The confidence connected algorithm parameters

Gradient Threshold

Gradient magnitude thresholding is controlled by only a single parameter. The parameter **Gradient** sets the maximum factor that the gradient magnitude ratio can reach and still be included in the segmented region. This factor is quite varying depending upon the type of region that needs to be segmented, but a conservative value of 8.0 seems to perform satisfactorily. The parameter **Max Size** sets the maximum amount of the image in percent that the segmented volume can take up. This is meant to act as a emergency stop parameter in the case of over segmentation.

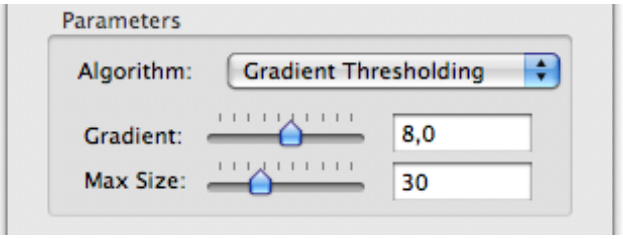

Figure 2.4: The gradient magnitude threshold algorithm parameters

2.2.4 Result options

The result options available are fairly self explanatory. The parameter **ROI Label** sets the title of the segmentation ROI generated. The parameter **ROI Color** sets the color that the generated ROI will receive.

2.3 Performing the segmentation

Performing the segmentation is a fairly straight forward procedure. Once the plugin is started you can click in the main viewer to set the seed point. If the **Show Seed** option is enabled you should also see a marker on the viewer indicating the location of the seed point. If you are satisfied with your seed point selection and have set all the parameters correctly you can start the segmentation by clicking calculate. When the segmentation has finished and there was no error you will see a new ROI on the main viewer. The title will be the **ROI Label** from the interface together with the calculated volume. The ROI can be further processed the same as any other ROI in OsiriX.

2.4 Troubleshooting

If there are problems using the NMSegmentation plugin the main source of information is the system console. This is located under *Applications / Utilities / Console* (and *Programme / Dienstprogramme / Konsole* on german systems). Under Console Messages you will see messages from any running OsiriX instance. If the plugin error is not apparent from the log messages you can put the plugin into debug mode by entering the following command on the terminal:

```
defaults write com.rossetantoine.osirix NSDebugEnabled 1
```

The plugin is compiled with extensive debug messages which are normally deactivated for performance reasons. Any errors related to the plugin should now be contained in the Console logs for OsiriX. To deactivate debug mode again enter this command on the terminal:

```
defaults delete com.rossetantoine.osirix NSDebugEnabled
```

Bibliography

[1] Luis Ibanez, Will, Schroeder, Lydia Ng, Josh Cates, and Insight Software Consortium. *The ITK Software Guide Second Edition*. Kitware Inc, November 2005.

[2] Osman Ratib, Antoine Rosset, and Joris Heuberger. Osirix: the pocket guide. http://www.osirix-viewer.com/Store.html, 2009.

[3] Antione Rosset. Plugin development guide http://osirix.svn.sourceforge.net/viewvc/osirix/Documentation/Developers%20Guide/index.html.

[4] Irwin Sobel. An isotropic 3x3x3 volume gradient operator. August 1996.