The University of Sydney

COMP5424 - Information Technology in Biomedicine

Topic E

3D Region of Interest Segmentation (3D-ROI)

Max Schultz - Student ID: 440176697

May 28, 2017

Contents

List of Figures

Introduction

This module implements a 3D region of interest (ROI) segmentation method. This is divided into three main components, the first visualizing a 3D region of interest, and the second implementing a grow algorithm to cut a point of interest (POI) within the ROI. Finally the third implements an automatic test and a reload button to ensure functionality. The interface consists of 6 main section, the first two *Enter ROI Coordinates* and *Turn On ROI Visualization* allowing the visualization of 3D region. The next two *Cut ROI* and the module *Selection Criteria* allowing for the segmentation and model creation of the ROI. The *Selection Criteria* slider controls the specificity of the growing algorithm. Finally the *Test Module* and *Reload* buttons allow the user to reload and check both the Script and User Interface.

Youtube link for module: https://youtu.be/X-k9C6xRLPw

Code Components

Setup

This section instantiates six buttons. The view for this can be found in *Appendix 2.1 - User Interface*.

Button Name	**Description**	**Method Called**	**Function**
roiButton	Enter ROI Coordinates	self.onSeedRoi	Allows user to input a ROI using the mouse.
visButton	Turn On ROI Visualisation	self.onVisualise	Gives a 3D visualisation of the user's ROI
cutButton	Cut ROI	self.onCut	Generates a 3D model of a relevant features based on an initial seeding point.
reloadButton	Reload	self.onReload	Reloads module and attached scripts.
selCriteria	Selection Criteria	growCutAlgo.onApply()	Calls growCut algorithm.
testButton	Test Module	self.onTest	Allows user to test module function.

Table 1: Objects Generated in Setup

```
        #method counters
        self.roiMethodCounter = 0 #increments every time generate ROI or visualise is called
        self.roiNodeIncrement = 0 #takes value of roiMethodCounter on Generate ROI
        self.modelMethodCounter = 0 #increments every time new model is generated for id String

        #strings for node IDs
        self.roiIDString = ""
        self.modelIDString = ""

    def setup(self):
        # Instantiate and connect widgets ...

        ##############################TEMPLATE SET UP#################################
        # Collapsible button
        self.sampleCollapsibleButton = ctk.ctkCollapsibleButton()
        self.sampleCollapsibleButton.text = "Topic E"
        self.layout.addWidget(self.sampleCollapsibleButton)

        # Layout within the sample collapsible button
        self.sampleFormLayout = qt.QFormLayout(self.sampleCollapsibleButton)

        ##############################INPUT FRAMES####################################

        self.inputFrame = qt.QFrame(self.sampleCollapsibleButton)
        self.inputFrame.setLayout(qt.QHBoxLayout())
        self.sampleFormLayout.addWidget(self.inputFrame)
        self.inputSelector = qt.QLabel("Input Volume: ", self.inputFrame)
        self.inputFrame.layout().addWidget(self.inputSelector)
        self.inputSelector = slicer.qMRMLNodeComboBox(self.inputFrame)
        self.inputSelector.nodeTypes = ( ("vtkMRMLScalarVolumeNode"), "" )
        self.inputSelector.addEnabled = False
        self.inputSelector.removeEnabled = False
        self.inputSelector.setMRMLScene( slicer.mrmlScene )
        self.inputFrame.layout().addWidget(self.inputSelector)

        ##############################OUTPUT FRAMES###################################

        self.outputFrame = qt.QFrame(self.sampleCollapsibleButton)
        self.outputFrame.setLayout(qt.QHBoxLayout())
        self.sampleFormLayout.addWidget(self.outputFrame)
        self.outputSelector = qt.QLabel("Output Volume: ", self.outputFrame)
        self.outputFrame.layout().addWidget(self.outputSelector)
        self.outputSelector = slicer.qMRMLNodeComboBox(self.outputFrame)
        self.outputSelector.nodeTypes = ( ("vtkMRMLScalarVolumeNode"), "" )
        self.outputSelector.setMRMLScene( slicer.mrmlScene )
        self.outputFrame.layout().addWidget(self.outputSelector)


        ##############################wIDEGET SET UP##################################

        # Add a seed button ROI
        roiButton = qt.QPushButton("Enter ROI Coordinates")
        roiButton.toolTip = "This allows you to select an ROI for visualisation"
        self.sampleFormLayout.addWidget(roiButton)
        roiButton.connect('clicked()', self.onSeedROI)

        # Add a visualisation button ROI
        visButton = qt.QPushButton("Turn On ROI Visualisation")
        visButton.toolTip = "This visualises the ROI selected using Enter ROI Coordinates"
        self.sampleFormLayout.addWidget(visButton)
        visButton.connect('clicked()', self.onVisualise)

        # Add a growCut algo button
        cutButton = qt.QPushButton("Cut ROI")
```

```
65          cutButton.toolTip = "After placing a seeding point in Editor, this button allows you ...
                to cut out that POI"
66          self.sampleFormLayout.addWidget(cutButton)
67          cutButton.connect('clicked()', self.onCut)
68
69          # Add a test button
70          testButton = qt.QPushButton("Test Module")
71          testButton.toolTip = "Test all modules features"
72          self.sampleFormLayout.addWidget(testButton)
73          testButton.connect('clicked()', self.onTest)
74
75          # Add a reload button for debug
76          reloadButton = qt.QPushButton("Reload")
77          reloadButton.toolTip = "Reload this Module"
78          self.sampleFormLayout.addWidget(reloadButton)
79          reloadButton.connect('clicked()', self.onReload)
80
81          # Set local var as instance attribute
82          self.reloadButton = reloadButton
83          self.roiButton = roiButton
84          self.visButton = visButton
85          self.testButton = testButton
86          self.cutButton = cutButton
87
88          # Add vertical spacer
89          self.layout.addStretch(1)
```

Block 1. Setup instantiates GUI objects

Enter ROI Coordinates

This section allows the user to input a ROI around a section in an scene. This is done to give a preliminary view of the ROI, before it is segmented into a model. The view for this can be found in *Appendix 2.2 - ROI Coordinate*. A counter *roiMethodCounter* is incremented every time this method is called, and is assigned to *self.roiNodeIncrement*. This is done as the Volume Render module creates another unnecessary ROI node when called. As such, a counter is required to keep track of which ROI nodes contain the correctly segmented data.

```
    #method to get ROI
    def onSeedROI(self):

        selectionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLSelectionNodeSingleton")
        # place rulers
        selectionNode.SetReferenceActivePlaceNodeClassName("vtkMRMLAnnotationROINode")

        interactionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLInteractionNodeSingleton")
        placeModePersistence = 1
        interactionNode.SetPlaceModePersistence(placeModePersistence)
        # mode 1 is Place, can also be accessed via slicer.vtkMRMLInteractionNode().Place
        interactionNode.SetCurrentInteractionMode(1)

        self.roiMethodCounter += 1
        self.roiNodeIncrement = self.roiMethodCounter #counter only increments when region is ...
            segmented, means visalisation only occurs for this node
```

Block 2. Allows users to input ROI coordinates using a mouse

Turn On ROI Visualization

This section allows the user to visualize the ROI designated in the previous section. This is done to give a preliminary view of the ROI, before a specific object is cut out. The view for this can be found in *Appendix 2.3 - ROI Segment Visualization*. A string *self.roiIDString* is constructed from *self.roiNodeIncrement* to ensure that the correct ROI node is visualized. The variable *self.roiMethodCounter* is incremented after this method to account for the unnecessary ROI node generated by the Volume Render module.

```
1       #method to visualise segment
2       def onVisualise(self) :
3
4           logic = slicer.modules.volumerendering.logic()
5           volumeNode = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')
6           displayNode = logic.CreateVolumeRenderingDisplayNode()
7           self.roiIDString = 'vtkMRMLAnnotationROINode' + str(self.roiNodeIncrement) ...
                #visualises correct node based on methods
8
9           #crop volumeNode for ROINode and visualise
10          displayNode.CroppingEnabledOn()
11          slicer.mrmlScene.AddNode(displayNode)
12          displayNode.UnRegister(logic)
13          logic.UpdateDisplayNodeFromVolumeNode(displayNode, volumeNode)
14          volumeNode.AddAndObserveDisplayNodeID(displayNode.GetID())
15
16          #visualise ROI node
17          displayNode.SetAndObserveROINodeID(self.roiIDString)
18
19          self.roiMethodCounter += 1
20
21
22
23          return
```

Block 3. Automatically presents a 3D visualization of the ROI

Grow and Cut ROI

This section allows the user to grow and cut an POI within an ROI based on an initial seeding point. This initial seeding point is placed using the Editor module, and the view for this can be found in *Appendix 2.4 - ROI Object Seeding*. While initial sections of the Topic E module implemented a paint brush tool to place the seeding tool, the Editor interface provides an incredibly robust and useful interface for correctly placing these seeding points. As such, it was chosen to retain the use of the Editor module as it comes in-built into Slicer, and to have the Topic E module handle the output of the Editor module effectively.

The file *growCutAlgo.onApply()* is called in this file. This imports the grow algorithm implemented in the tutorials in the ITBRegionGrow.py file. A modified version of this file can be found in *Appendix 5 - Grow Cut Algorithm*. Significant modifications include editing the User Interface to only retain the Selection Criteria for grey-scale control, and thee addition of *self.roiCounter* to allow for more robust node control, so that multiple objects can be generated sequentially from the same seeding point if the Selection Criteria controls are changed.

Lines 14 through to 24 implement a new Model Maker model based on the node generated in the *growCutAlgo.onApply()*. This POI is then generated and visualized in 3D. The view for this can be found in *Appendix 2.5 - ROI Model Generation*, which shows the model visualized within the ROI segment from the previous sections. An isolated view of just the 3D model can be found in *Appendix 2.6 - ROI Model isolated*. Finally, *Appendix 2.7 - Selection Control* shows that the 3D model can be controlled using the Selection Criteria module. If the values in this module are changed, and the *self.onCut* method is called again, the model is regenerated with the new parameters. This allows for flexibility in model creation. It is recommended that the user starts with low values, and progressively increase the Selection Criteria, as produces a computationally efficient result.

```
1       def onCut(self) :
2
3
4           inputVolume = self.inputSelector.currentNode()
5           outputVolume = self.outputSelector.currentNode()
6
7           growCutAlgo.onApply() #apply growCut Algo
8           print("Iteration Complete! That was iteration number: %i" % self.modelMethodCounter)
9
10          #define ROI vairables for cutting
11          scene = slicer.mrmlScene
12          self.modelMethodCounter += 1
13          self.modelIDString = "MRBrainTumor" + str(self.modelMethodCounter) + "_label_grow"
14
15          #define nodes
16          hie_node = slicer.vtkMRMLModelHierarchyNode()
17          roi_node = slicer.util.getNode(self.modelIDString)
18
19          #generate model
20          scene.AddNode(hie_node)
21          params = {}
22          params['InputVolume'] = roi_node.GetID()
23          params['ModelSceneFile'] = hie_node.GetID()
24          slicer.cli.run(slicer.modules.modelmaker, None, params)
25          mod_node = hie_node.GetModelNode()
26
27          return
```

Block 4. Automatically segments and visualizes a POI in 3D

Test Module

This section allows the user to automatically test the complete functionality of the module. No view for this model has been provided, as it simply automatically executes all the other views. Significant features include lines 21 - 23, which allow the user to customize the ROI section they would like to automatically segment. Significantly, as this is automatic segmentation from a predetermined seeding point, and has no user input, the grow-cut algorithm will simply adopt the shape of the ROI segment defined in these lines.

```
1       #Add button to test all features of Topic E module
2       '''
3       Full reference for slicer unit test scripts found here:
4       https://github.com/Slicer/Slicer/blob/master/Modules/Scripted/EditorLib/Testing/
5
6       full reference for threshold testing found here:
7       https://github.com/Slicer/Slicer/blob/master/Modules/Scripted/EditorLib/Testing/ThresholdThreadingTest.py
8
9       Above git has been used in the development of this metho'd
10      '''
11      def onTest(self) :
12          #Import Sample Data
13          import SampleData
14          sampleDataLogic = SampleData.SampleDataLogic()
15          MRBrainTumor1 = sampleDataLogic.downloadMRBrainTumor1()
16
17          #Define ROI annotation Node
18          roi = slicer.vtkMRMLAnnotationROINode()
19          slicer.mrmlScene.AddNode(roi)
20
21          #ENTER COORDINATES FOR ROI SEGMENT HERE
22          roi.SetXYZ(-17.5, 26, 31)
23          roi.SetRadiusXYZ(20, 20, 20)
24
25          #Apply ROI cropping to node
26          cropLogic = slicer.modules.cropvolume.logic()
27          cvpn = slicer.vtkMRMLCropVolumeParametersNode()
28          cvpn.SetROINodeID( roi.GetID() )
29          cvpn.SetInputVolumeNodeID( MRBrainTumor1.GetID() )
30          cropLogic.Apply( cvpn )
31          croppedHead = slicer.mrmlScene.GetNodeByID( cvpn.GetOutputVolumeNodeID() )
32
33          #Visualise ROI
34          logic = slicer.modules.volumerendering.logic()
35          volumeNode = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')
36          displayNode = logic.CreateVolumeRenderingDisplayNode()
37
38          #crop volumeNode for ROINode and visualise
39          displayNode.CroppingEnabledOn()
40          slicer.mrmlScene.AddNode(displayNode)
41          displayNode.UnRegister(logic)
42          logic.UpdateDisplayNodeFromVolumeNode(displayNode, volumeNode)
43          volumeNode.AddAndObserveDisplayNodeID(displayNode.GetID())
44
45          #visualise ROI node
46          displayNode.SetAndObserveROINodeID(roi.GetID())
47
48          #
49          # create a label map and set it for editing
50          #
51          volumesLogic = slicer.modules.volumes.logic()
52          croppedHeadLabel = volumesLogic.CreateAndAddLabelVolume( slicer.mrmlScene, ...
                croppedHead, croppedHead.GetName() + '-label' )
53          selectionNode = slicer.app.applicationLogic().GetSelectionNode()
54          selectionNode.SetReferenceActiveVolumeID( croppedHead.GetID() )
```

```
55          selectionNode.SetReferenceActiveLabelVolumeID( croppedHeadLabel.GetID() )
56          slicer.app.applicationLogic().PropagateVolumeSelection(0)
57
58
59          #
60          # got to the editor and do some drawing
61          #
62          #self.delayDisplay("Paint some things")
63
64          parameterNode = EditUtil.getParameterNode()
65          lm = slicer.app.layoutManager()
66          paintEffect = EditorLib.PaintEffectOptions()
67          paintEffect.setMRMLDefaults()
68          paintEffect.__del__()
69          sliceWidget = lm.sliceWidget('Red')
70          paintTool = EditorLib.PaintEffectTool(sliceWidget)
71          EditUtil.setLabel(1)
72          paintTool.paintAddPoint(1,1)
73          paintTool.paintApply()
74          paintTool.cleanup()
75          paintTool = None
76
77          #define nodes
78          growCutLogic = EditorLib.GrowCutEffectLogic(sliceWidget.sliceLogic())
79          growCutLogic.growCut()
80
81          #
82          # now split the volume, merge it back, and see if it looks right
83          #
84          preArray = slicer.util.array(croppedHeadLabel.GetName())
85          #print preArray
86          slicer.util.selectModule('Editor')
87          slicer.util.findChildren(text='Split Merge Volume')[0].clicked()
88          slicer.util.findChildren(text='Merge All')[0].clicked()
89          postArray = slicer.util.array(croppedHeadLabel.GetName())
90
91          hie_node = slicer.vtkMRMLModelHierarchyNode()
92          roi_node = slicer.util.getNode('MRBrainTumor1_subvolume_scale_1_label')
93          scene = slicer.mrmlScene
94
95          scene.AddNode(hie_node)
96          params = {}
97          params['InputVolume'] = roi_node.GetID()
98          params['ModelSceneFile'] = hie_node.GetID()
99          slicer.cli.run(slicer.modules.modelmaker, None, params)
100         mod_node = hie_node.GetModelNode()
101
102         print("Topic E has finished execution!")
```

Block 5. Automatic testing of module's full functionality

Reload

This section allows the user to reload the modules scripts and user interface.

```
    #method for reloading
    def onReload(self, moduleName = "HelloPython"):
        import imp, sys, os, slicer

        widgetName = moduleName + "Widget"

        # reload the source code
        fPath = eval('slicer.modules.%s.path' % moduleName.lower())
        p = os.path.dirname(fPath)
        if not sys.path.__contains__(p):
            sys.path.insert(0,p)
        fp = open(fPath, "r")
        globals()[moduleName] = imp.load_module(
                moduleName, fp, fPath, ('.py', 'r', imp.PY_SOURCE))
        fp.close()

        # rebuild the widget
        print "the module name to be reloaded,", moduleName
        # find the Button with a name 'moduleName Reolad', then find its parent (e.g., a ...
            collasp button) and grand parent (moduleNameWidget)
        parent = slicer.util.findChildren(name = '%s Reload' % moduleName)[0].parent().parent()
        for child in parent.children():
            try:
                child.hide()
            except AttributeError:
                pass
        # Remove spacer items
        item = parent.layout().itemAt(0)
        while item:
            parent.layout().removeItem(item)
            item = parent.layout().itemAt(0)
        # create new widget inside existing parent
        globals()[widgetName.lower()] = eval('globals()["%s"].%s(parent)' % (moduleName, ...
            widgetName))
        globals()[widgetName.lower()].setup()
        return
```

Block 6. Default reload object allows user to reload scripts

Appendix

Appendix 1 - User Flow

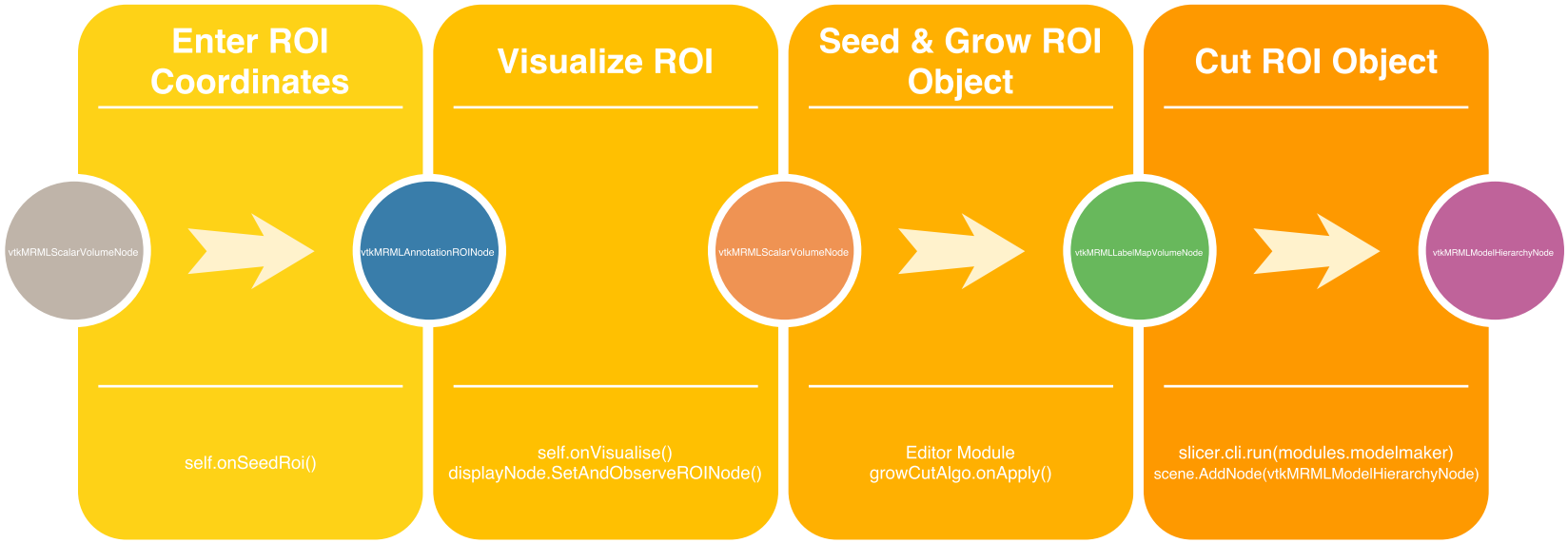

Figure 1: User Flow of module, with generated nodes

Key:

- Stage 1: vtkMRMLScalarVolumeNode
- Stage 2: vtkMRMLAnnotationROInode
- Stage 3: vtkMRMLScalarVolumeNode
- Stage 4: vtkMRMLLabelMapVolumeNode
- Stage 5: vtkMRMLmodelHierarchyNode

Appendix 2 - Segmentation Flow

Appendix 2.1 - User Interface

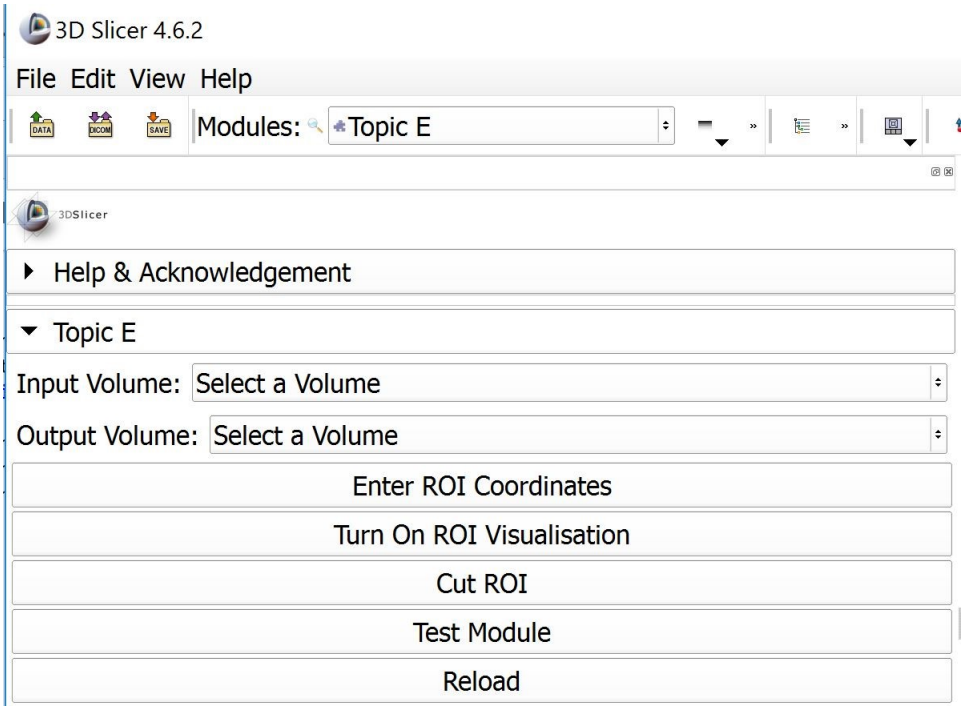

Figure 2: User Interface for Topic E

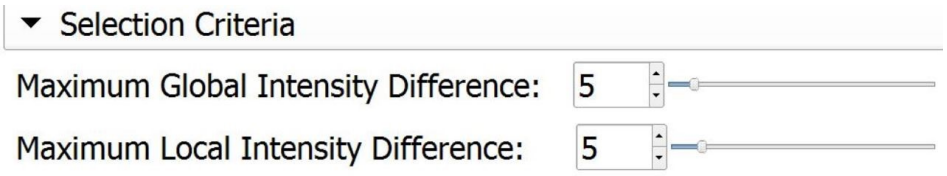

Figure 3: Selection Criteria Interface for Topic E

Appendix 2.2 - ROI Coordinates

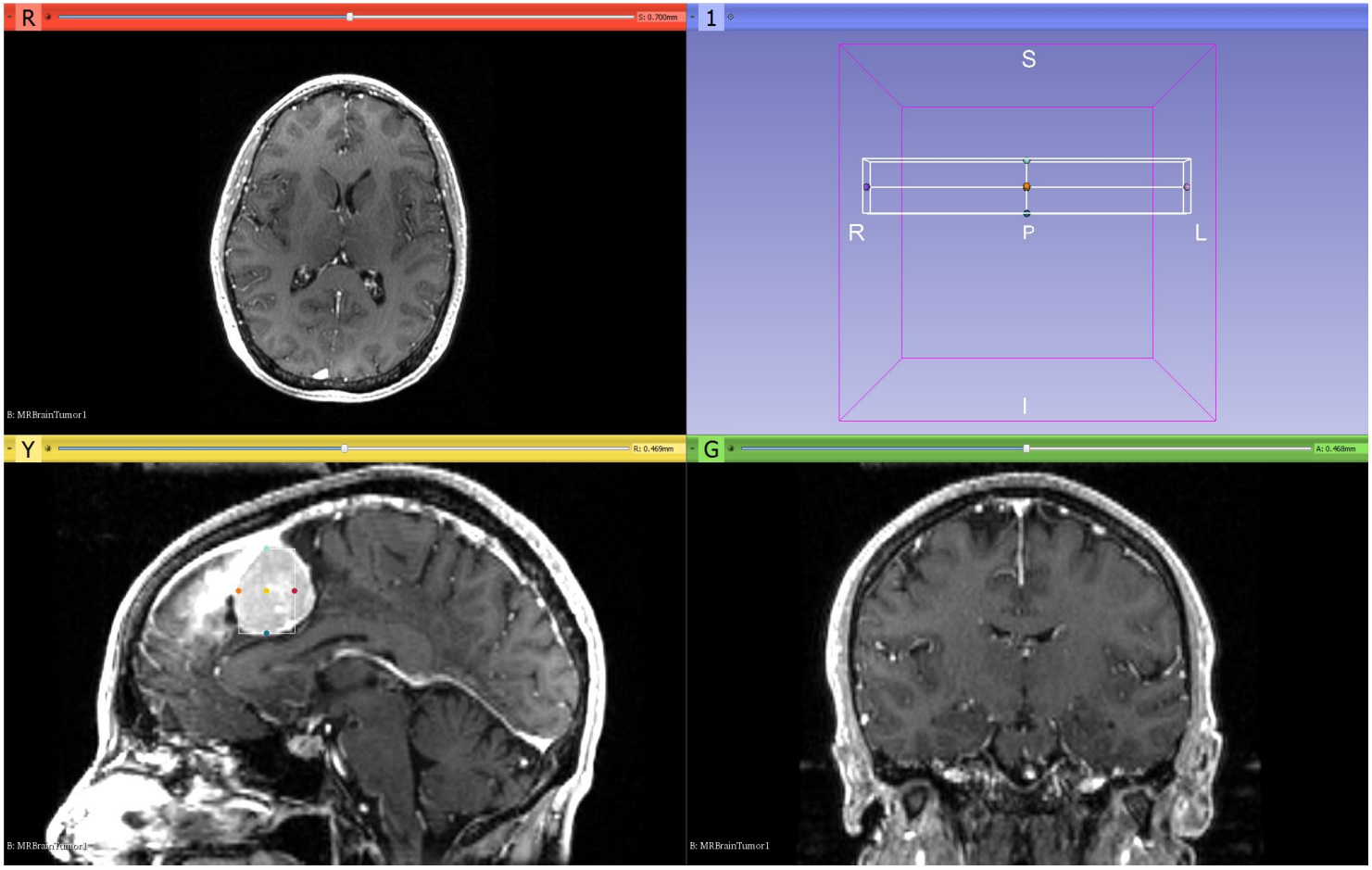

Figure 4: ROI Coordinate Selection

Appendix 2.3 - ROI Segment Visualization

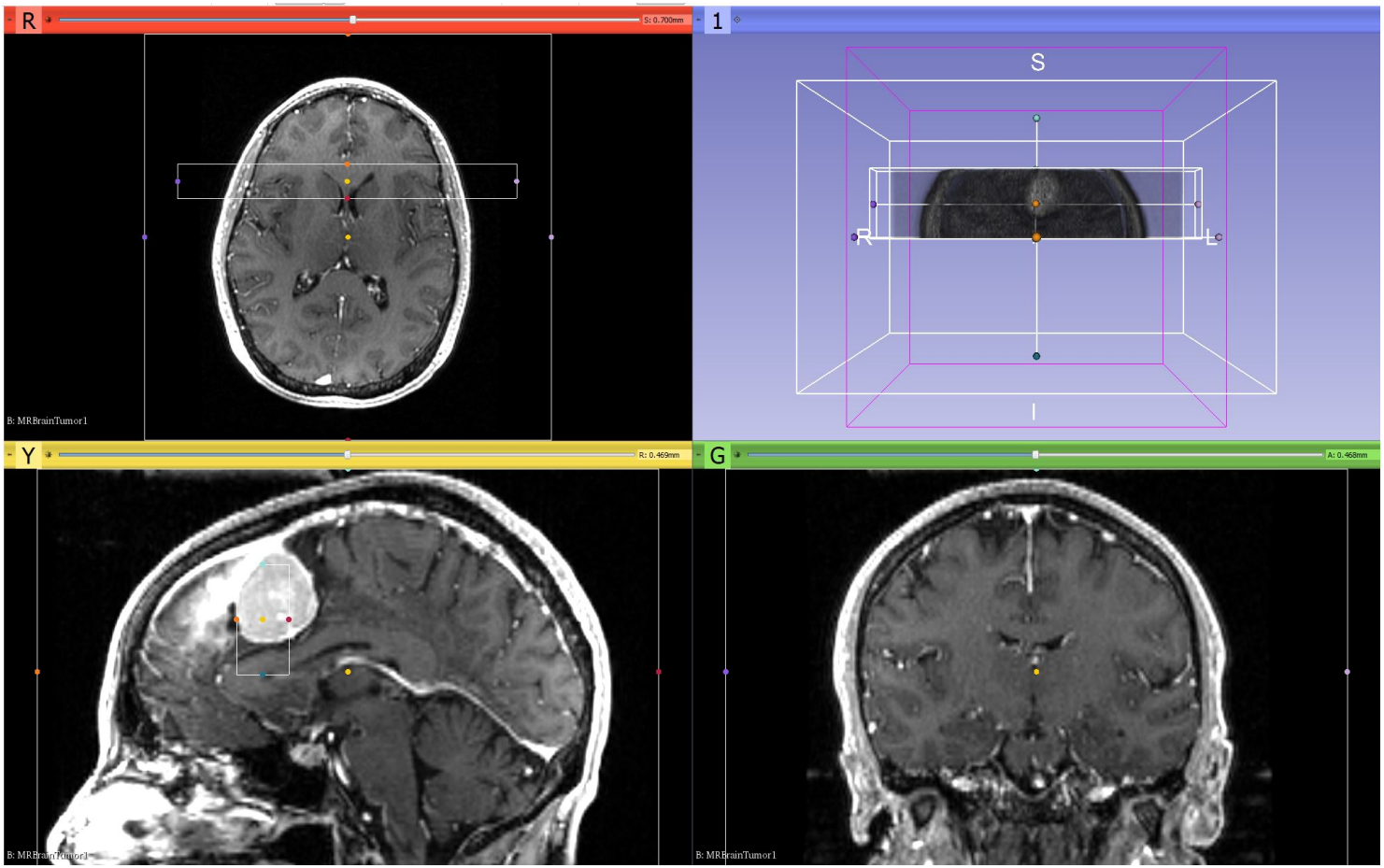

Figure 5: ROI Segmentation Cut

Appendix 2.4 - ROI Object Seeding

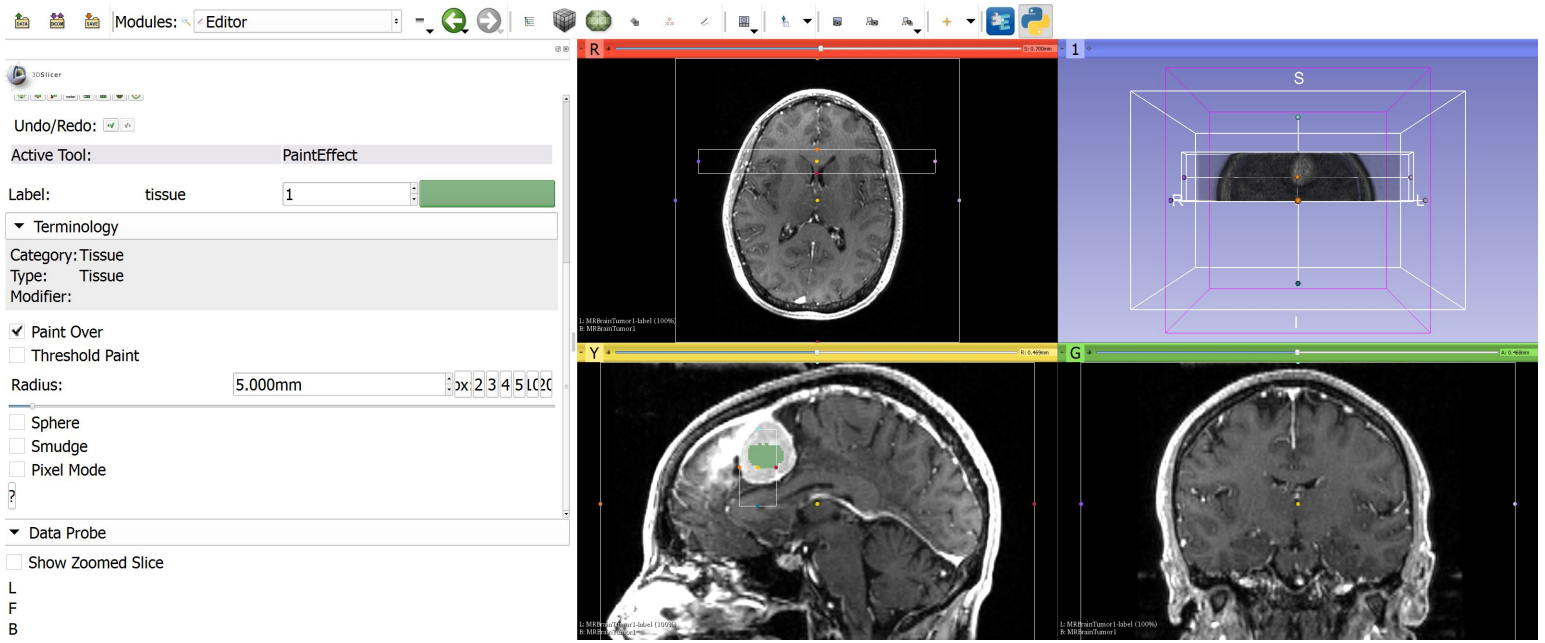

Figure 6: ROI Object Seeding using Editor

Appendix 2.5 - ROI Model Generation

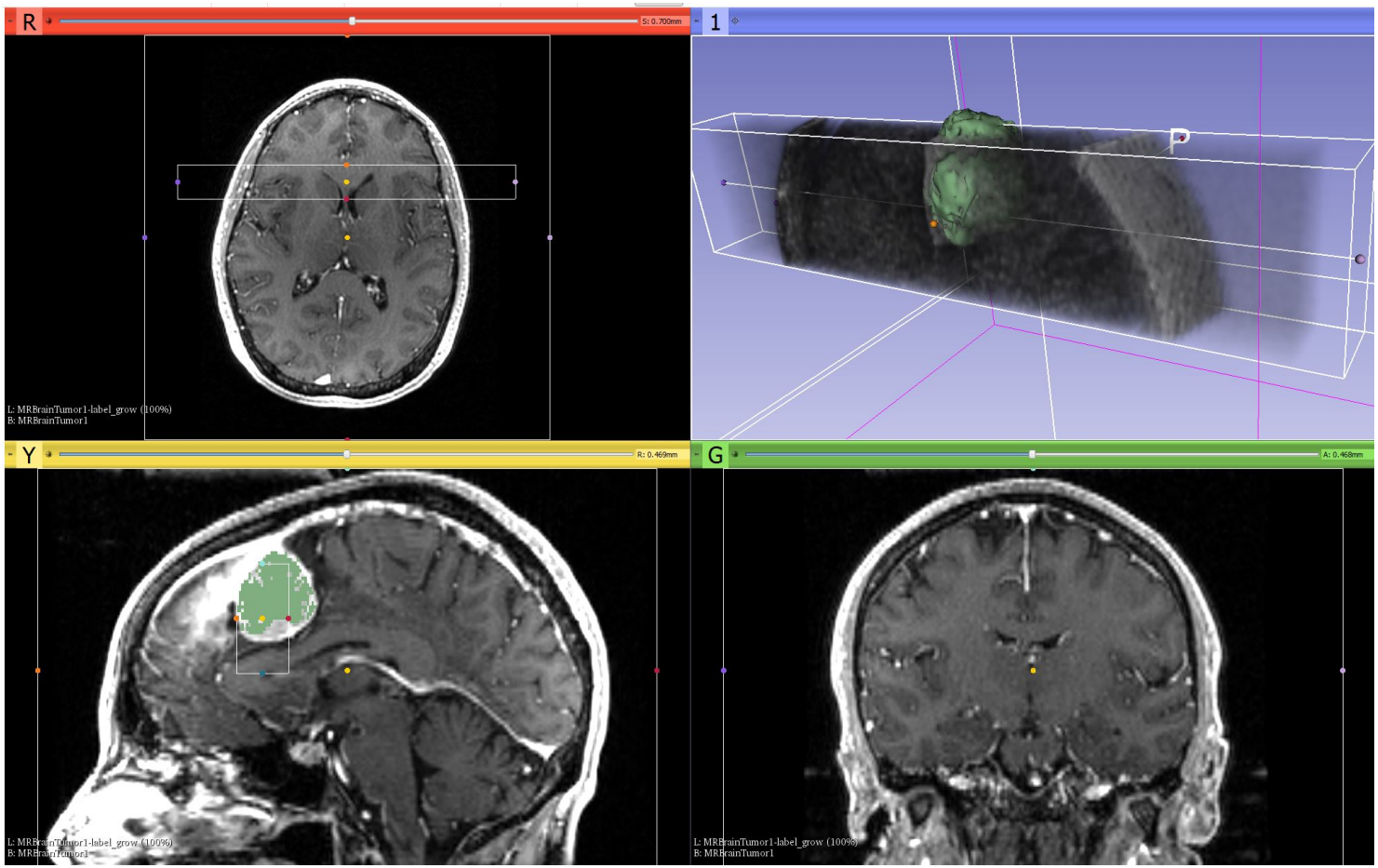

Figure 7: ROI Model Generation with ROI Visualization

Appendix 2.6 - ROI Model Isolated

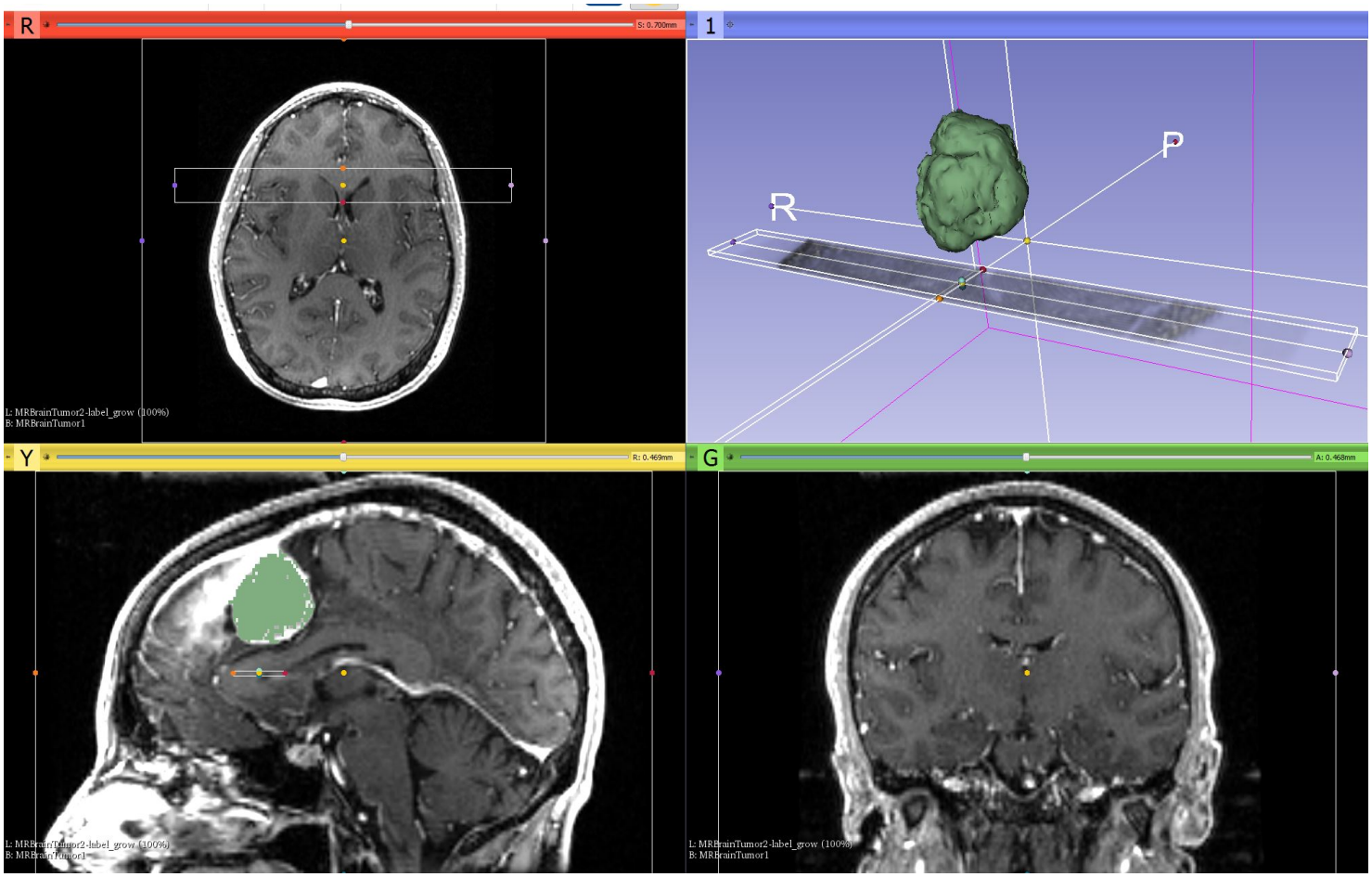

Figure 8: ROI Object Isolated from Visualization

Appendix 2.7 - Selection Control

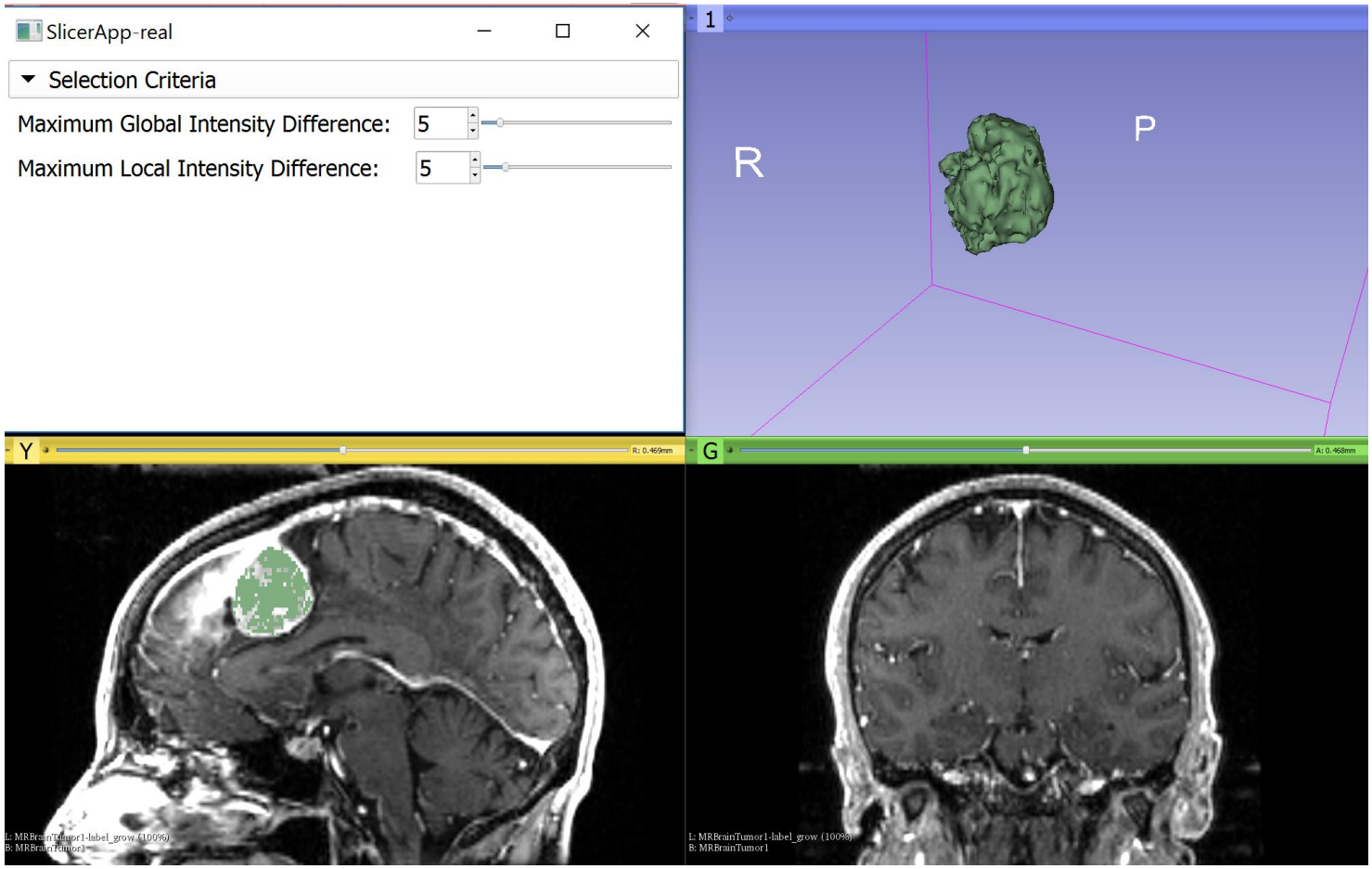

Figure 9: ROI Selection Criteria for Grey-scale Control

Appendix 3 - Code Development

Full git repository may be found here: https://github.com/Schlutz1/COMP5424
Full history of code development may be found here: https://github.com/Schlutz1/COMP5424/commits/master

Significant features from development include making quite significant changes to the code at two stages:

- The code first developed on May 6th initially placed seeding points using fiducials, and the bulk of the code was built around this premise. This was removed entirely in subsequent revisions, as better methods for seeding points were developed.
- The code initially attempted to implement a growCut algorithm within the main file. Due to the complexity of this algorithm this was eventually chosen to be developed in another file (*ITBRegionGrow.py*), and called in a more modular fashion.

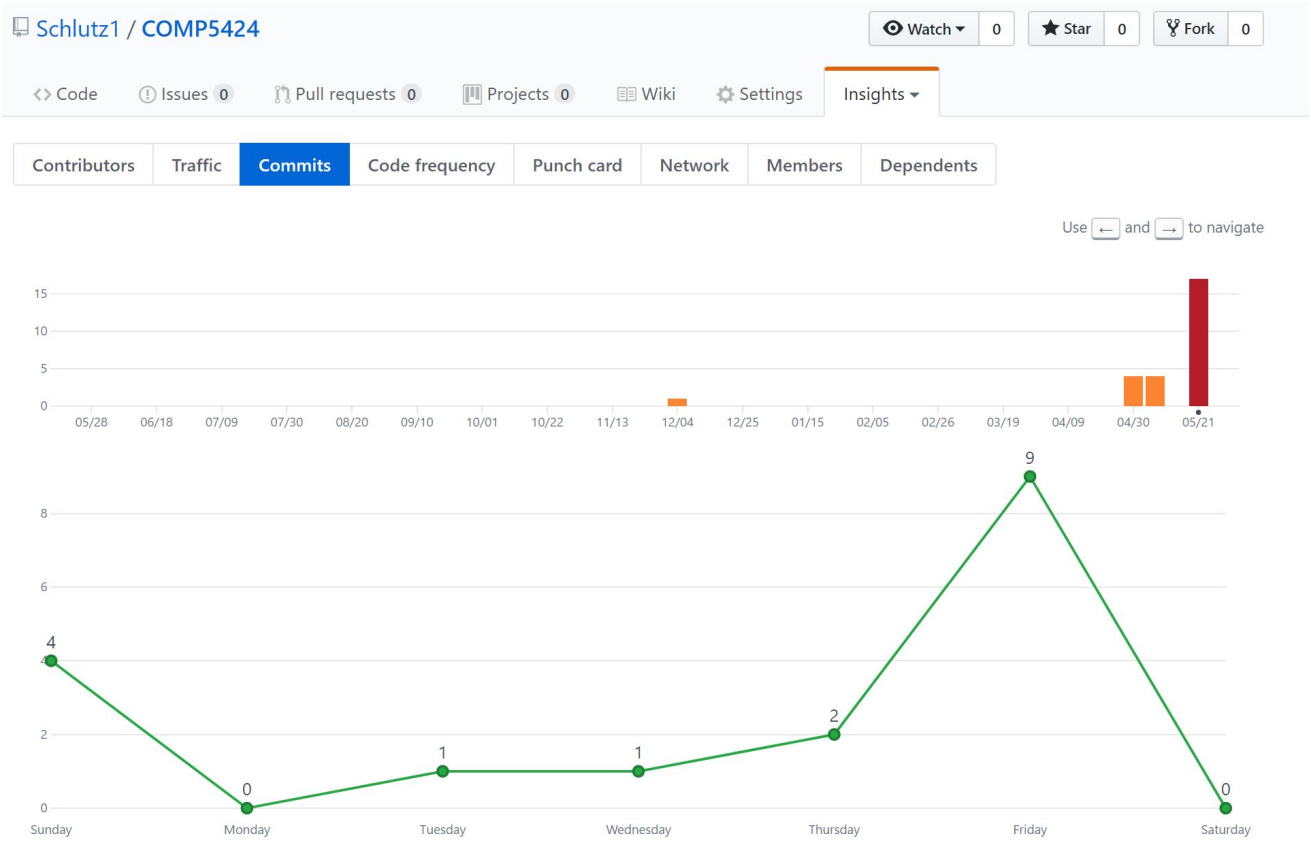

Figure 10: Chart of git commits and activity

Appendix 4 - Main Script

```
'''
Max Schultz
SID: 440176697

MAIN FILE FOR TOPIC E – COMP5424
Handles:
    – Setup of interface
    – ROI selection
    – ROI visualisation
    – Model Creation
Imports:
    – ITBRegionGrow for growCut algorithim
'''

from __main__ import vtk, qt, ctk, slicer
from vtk.util import numpy_support

import numpy as np
import vtkSegmentationCorePython as vtkSegmentationCore
import vtkSlicerSegmentationsModuleLogicPython as vtkSlicerSegmentationsModuleLogic
import SampleData

import unittest
import qt
import slicer
import EditorLib
from EditorLib.EditUtil import EditUtil

from ITBRegionGrow import ITBRegionGrowWidget
growCutAlgo = ITBRegionGrowWidget()

#from ThresholdThreadingTest import

class HelloPython:
    def __init__(self, parent):
        parent.title = "Topic E"
        parent.categories = ["COMP5424"]
        parent.dependencies = []

        parent.contributors = ["Max Schultz"] # replace with "Firstname Lastname (Org)"
        parent.helpText = """
        Module for visualising 3D segments, and then constructing 3D modules of regions of ...
            interest
        """
        parent.acknowledgementText = """
        I'd like to acknowledge Sidong Liu (USYD) for his w in the development of this module.
        """
        self.parent = parent

class HelloPythonWidget:
    def __init__(self, parent = None):
        if not parent:
            self.parent = slicer.qMRMLWidget()
            self.parent.setLayout(qt.QVBoxLayout())
            self.parent.setMRMLScene(slicer.mrmlScene)
        else:
            self.parent = parent
        self.layout = self.parent.layout()
        if not parent:
            self.setup()
            self.parent.show()
```

```
61
62          ##############################VARIABLE SETUP#################################
63
64          #method counters
65          self.roiMethodCounter = 0 #increments every time generate ROI or visualise is called
66          self.roiNodeIncrement = 0 #takes value of roiMethodCounter on Generate ROI
67          self.modelMethodCounter = 0 #increments every time new model is generated for id String
68
69          #strings for node IDs
70          self.roiIDString = ""
71          self.modelIDString = ""
72
73      def setup(self):
74          # Instantiate and connect widgets ...
75
76          ##############################TEMPLATE SET UP################################
77          # Collapsible button
78          self.sampleCollapsibleButton = ctk.ctkCollapsibleButton()
79          self.sampleCollapsibleButton.text = "Topic E"
80          self.layout.addWidget(self.sampleCollapsibleButton)
81
82          # Layout within the sample collapsible button
83          self.sampleFormLayout = qt.QFormLayout(self.sampleCollapsibleButton)
84
85          ##############################INPUT FRAMES###################################
86
87          self.inputFrame = qt.QFrame(self.sampleCollapsibleButton)
88          self.inputFrame.setLayout(qt.QHBoxLayout())
89          self.sampleFormLayout.addWidget(self.inputFrame)
90          self.inputSelector = qt.QLabel("Input Volume: ", self.inputFrame)
91          self.inputFrame.layout().addWidget(self.inputSelector)
92          self.inputSelector = slicer.qMRMLNodeComboBox(self.inputFrame)
93          self.inputSelector.nodeTypes = ( ("vtkMRMLScalarVolumeNode"), "" )
94          self.inputSelector.addEnabled = False
95          self.inputSelector.removeEnabled = False
96          self.inputSelector.setMRMLScene( slicer.mrmlScene )
97          self.inputFrame.layout().addWidget(self.inputSelector)
98
99          ##############################OUTPUT FRAMES##################################
100
101         self.outputFrame = qt.QFrame(self.sampleCollapsibleButton)
102         self.outputFrame.setLayout(qt.QHBoxLayout())
103         self.sampleFormLayout.addWidget(self.outputFrame)
104         self.outputSelector = qt.QLabel("Output Volume: ", self.outputFrame)
105         self.outputFrame.layout().addWidget(self.outputSelector)
106         self.outputSelector = slicer.qMRMLNodeComboBox(self.outputFrame)
107         self.outputSelector.nodeTypes = ( ("vtkMRMLScalarVolumeNode"), "" )
108         self.outputSelector.setMRMLScene( slicer.mrmlScene )
109         self.outputFrame.layout().addWidget(self.outputSelector)
110
111
112         ##############################wIDEGET SET UP#################################
113
114         # Add a seed button ROI
115         roiButton = qt.QPushButton("Enter ROI Coordinates")
116         roiButton.toolTip = "This allows you to select an ROI for visualisation"
117         self.sampleFormLayout.addWidget(roiButton)
118         roiButton.connect('clicked()', self.onSeedROI)
119
120         # Add a visualisation button ROI
121         visButton = qt.QPushButton("Turn On ROI Visualisation")
122         visButton.toolTip = "This visualises the ROI selected using Enter ROI Coordinates"
123         self.sampleFormLayout.addWidget(visButton)
124         visButton.connect('clicked()', self.onVisualise)
125
```

```
126         # Add a growCut algo button
127         cutButton = qt.QPushButton("Cut ROI")
128         cutButton.toolTip = "After placing a seeding point in Editor, this button allows you ...
                to cut out that POI"
129         self.sampleFormLayout.addWidget(cutButton)
130         cutButton.connect('clicked()', self.onCut)
131
132         # Add a test button
133         testButton = qt.QPushButton("Test Module")
134         testButton.toolTip = "Test all modules features"
135         self.sampleFormLayout.addWidget(testButton)
136         testButton.connect('clicked()', self.onTest)
137
138         # Add a reload button for debug
139         reloadButton = qt.QPushButton("Reload")
140         reloadButton.toolTip = "Reload this Module"
141         self.sampleFormLayout.addWidget(reloadButton)
142         reloadButton.connect('clicked()', self.onReload)
143
144         # Set local var as instance attribute
145         self.reloadButton = reloadButton
146         self.roiButton = roiButton
147         self.visButton = visButton
148         self.testButton = testButton
149         self.cutButton = cutButton
150
151         # Add vertical spacer
152         self.layout.addStretch(1)
153
154
155     ##############################METHODS FOR WIDGETS#################################
156
157     #method to get ROI
158     def onSeedROI(self):
159
160         selectionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLSelectionNodeSingleton")
161         # place rulers
162         selectionNode.SetReferenceActivePlaceNodeClassName("vtkMRMLAnnotationROINode")
163
164         interactionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLInteractionNodeSingleton")
165         placeModePersistence = 1
166         interactionNode.SetPlaceModePersistence(placeModePersistence)
167         # mode 1 is Place, can also be accessed via slicer.vtkMRMLInteractionNode().Place
168         interactionNode.SetCurrentInteractionMode(1)
169
170         self.roiMethodCounter += 1
171         self.roiNodeIncrement = self.roiMethodCounter #counter only increments when region is ...
                segmented, means visalisation only occurs for this node
172
173     #method to visualise segment
174     def onVisualise(self) :
175
176         logic = slicer.modules.volumerendering.logic()
177         volumeNode = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')
178         displayNode = logic.CreateVolumeRenderingDisplayNode()
179         self.roiIDString = 'vtkMRMLAnnotationROINode' + str(self.roiNodeIncrement) ...
                #visualises correct node based on methods
180
181         #crop volumeNode for ROINode and visualise
182         displayNode.CroppingEnabledOn()
183         slicer.mrmlScene.AddNode(displayNode)
184         displayNode.UnRegister(logic)
185         logic.UpdateDisplayNodeFromVolumeNode(displayNode, volumeNode)
186         volumeNode.AddAndObserveDisplayNodeID(displayNode.GetID())
187
```

```
188         #visualise ROI node
189         displayNode.SetAndObserveROINodeID(self.roiIDString)
190 
191         self.roiMethodCounter += 1
192 
193 
194 
195         return
196 
197 
198     #method to segment ROI
199     def onCut(self) :
200 
201 
202         inputVolume = self.inputSelector.currentNode()
203         outputVolume = self.outputSelector.currentNode()
204 
205         growCutAlgo.onApply() #apply growCut Algo
206         print("Iteration Complete! That was iteration number: %i" % self.modelMethodCounter)
207 
208         #define ROI vairables for cutting
209         scene = slicer.mrmlScene
210         self.modelMethodCounter += 1
211         self.modelIDString = "MRBrainTumor" + str(self.modelMethodCounter) + "-label_grow"
212 
213         #define nodes
214         hie_node = slicer.vtkMRMLModelHierarchyNode()
215         roi_node = slicer.util.getNode(self.modelIDString)
216 
217         #generate model
218         scene.AddNode(hie_node)
219         params = {}
220         params['InputVolume'] = roi_node.GetID()
221         params['ModelSceneFile'] = hie_node.GetID()
222         slicer.cli.run(slicer.modules.modelmaker, None, params)
223         mod_node = hie_node.GetModelNode()
224 
225         return
226 
227     #Add button to test all features of Topic E module
228     '''
229     Full reference for slicer unit test scripts found here:
230     https://github.com/Slicer/Slicer/blob/master/Modules/Scripted/EditorLib/Testing/
231 
232     full reference for threshold testing found here:
233     https://github.com/Slicer/Slicer/blob/master/Modules/Scripted/EditorLib/Testing/ThresholdThreadingTest.py
234 
235     Above git has been used in the development of this metho'd
236     '''
237     def onTest(self) :
238         #Import Sample Data
239         import SampleData
240         sampleDataLogic = SampleData.SampleDataLogic()
241         MRBrainTumor1 = sampleDataLogic.downloadMRBrainTumor1()
242 
243         #Define ROI annotation Node
244         roi = slicer.vtkMRMLAnnotationROINode()
245         slicer.mrmlScene.AddNode(roi)
246 
247         #ENTER COORDINATES FOR ROI SEGMENT HERE
248         roi.SetXYZ(-17.5, 26, 31)
249         roi.SetRadiusXYZ(20, 20, 20)
250 
251         #Apply ROI cropping to node
252         cropLogic = slicer.modules.cropvolume.logic()
```

```
253         cvpn = slicer.vtkMRMLCropVolumeParametersNode()
254         cvpn.SetROINodeID( roi.GetID() )
255         cvpn.SetInputVolumeNodeID( MRBrainTumor1.GetID() )
256         cropLogic.Apply( cvpn )
257         croppedHead = slicer.mrmlScene.GetNodeByID( cvpn.GetOutputVolumeNodeID() )
258 
259         #Visualise ROI
260         logic = slicer.modules.volumerendering.logic()
261         volumeNode = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')
262         displayNode = logic.CreateVolumeRenderingDisplayNode()
263 
264         #crop volumeNode for ROINode and visualise
265         displayNode.CroppingEnabledOn()
266         slicer.mrmlScene.AddNode(displayNode)
267         displayNode.UnRegister(logic)
268         logic.UpdateDisplayNodeFromVolumeNode(displayNode, volumeNode)
269         volumeNode.AddAndObserveDisplayNodeID(displayNode.GetID())
270 
271         #visualise ROI node
272         displayNode.SetAndObserveROINodeID(roi.GetID())
273 
274         #
275         # create a label map and set it for editing
276         #
277         volumesLogic = slicer.modules.volumes.logic()
278         croppedHeadLabel = volumesLogic.CreateAndAddLabelVolume( slicer.mrmlScene, ...
                croppedHead, croppedHead.GetName() + '-label' )
279         selectionNode = slicer.app.applicationLogic().GetSelectionNode()
280         selectionNode.SetReferenceActiveVolumeID( croppedHead.GetID() )
281         selectionNode.SetReferenceActiveLabelVolumeID( croppedHeadLabel.GetID() )
282         slicer.app.applicationLogic().PropagateVolumeSelection(0)
283 
284 
285         #
286         # got to the editor and do some drawing
287         #
288         #self.delayDisplay("Paint some things")
289 
290         parameterNode = EditUtil.getParameterNode()
291         lm = slicer.app.layoutManager()
292         paintEffect = EditorLib.PaintEffectOptions()
293         paintEffect.setMRMLDefaults()
294         paintEffect.__del__()
295         sliceWidget = lm.sliceWidget('Red')
296         paintTool = EditorLib.PaintEffectTool(sliceWidget)
297         EditUtil.setLabel(1)
298         paintTool.paintAddPoint(1,1)
299         paintTool.paintApply()
300         paintTool.cleanup()
301         paintTool = None
302 
303         #define nodes
304         growCutLogic = EditorLib.GrowCutEffectLogic(sliceWidget.sliceLogic())
305         growCutLogic.growCut()
306 
307         #
308         # now split the volume, merge it back, and see if it looks right
309         #
310         preArray = slicer.util.array(croppedHeadLabel.GetName())
311         #print preArray
312         slicer.util.selectModule('Editor')
313         slicer.util.findChildren(text='Split Merge Volume')[0].clicked()
314         slicer.util.findChildren(text='Merge All')[0].clicked()
315         postArray = slicer.util.array(croppedHeadLabel.GetName())
316 
```

```
317         hie_node = slicer.vtkMRMLModelHierarchyNode()
318         roi_node = slicer.util.getNode('MRBrainTumor1-subvolume-scale_1-label')
319         scene = slicer.mrmlScene
320 
321         scene.AddNode(hie_node)
322         params = {}
323         params['InputVolume'] = roi_node.GetID()
324         params['ModelSceneFile'] = hie_node.GetID()
325         slicer.cli.run(slicer.modules.modelmaker, None, params)
326         mod_node = hie_node.GetModelNode()
327 
328         print("Topic E has finished execution!")
329 
330     #method for reloading
331     def onReload(self, moduleName = "HelloPython"):
332         import imp, sys, os, slicer
333 
334         widgetName = moduleName + "Widget"
335 
336         # reload the source code
337         fPath = eval('slicer.modules.%s.path' % moduleName.lower())
338         p = os.path.dirname(fPath)
339         if not sys.path.__contains__(p):
340             sys.path.insert(0,p)
341         fp = open(fPath, "r")
342         globals()[moduleName] = imp.load_module(
343                 moduleName, fp, fPath, ('.py', 'r', imp.PY_SOURCE))
344         fp.close()
345 
346         # rebuild the widget
347         print "the module name to be reloaded,", moduleName
348         # find the Button with a name 'moduleName Reolad', then find its parent (e.g., a ...
              collasp button) and grand parent (moduleNameWidget)
349         parent = slicer.util.findChildren(name = '%s Reload' % moduleName)[0].parent().parent()
350         for child in parent.children():
351             try:
352                 child.hide()
353             except AttributeError:
354                 pass
355         # Remove spacer items
356         item = parent.layout().itemAt(0)
357         while item:
358             parent.layout().removeItem(item)
359             item = parent.layout().itemAt(0)
360         # create new widget inside existing parent
361         globals()[widgetName.lower()] = eval('globals()["%s"].%s(parent)' % (moduleName, ...
              widgetName))
362         globals()[widgetName.lower()].setup()
363         return
```

Appendix 5 - Grow Cut Algorithim

```
from __main__ import vtk, qt, ctk, slicer
from vtk.util import numpy_support
import numpy as np

#
# ITBRegionGrow
#
class ITBRegionGrow:
  def __init__(self, parent):
    parent.title = "ITB RegionGrow Segmentation"
    parent.categories = ["COMP5424"]
    parent.dependencies = []
    parent.contributors = [ """
    Sidong Liu (USYD)
    Siqi Liu (Simense)
    """]
    parent.helpText = """
    Example of scripted loadable extension for the ITB Medical Image Segmentation lab.
    """
    parent.acknowledgementText = """
    This python program shows a simple implementation of the 3D region growing algorithm for
    the ITB LabW5.
    """
    self.parent = parent

#
# The main widget
#
class ITBRegionGrowWidget:
  def __init__(self, parent = None):
    if not parent:
      self.parent = slicer.qMRMLWidget()
      self.parent.setLayout(qt.QVBoxLayout())
      self.parent.setMRMLScene(slicer.mrmlScene)
    else:
      self.parent = parent
    self.layout = self.parent.layout()
    if not parent:
      self.setup()
      self.parent.show()

    self.roiCounter = 0
    self.roiID = ""

  #  Setup the layout
  def setup(self):

    # Collapsible button
    self.laplaceCollapsibleButton = ctk.ctkCollapsibleButton()
    self.laplaceCollapsibleButton.text = "3D Region Grow Inputs"

    # Layout within the laplace collapsible button
    self.segmentationFormLayout = qt.QFormLayout(self.laplaceCollapsibleButton)
    self.inputSelector = slicer.qMRMLNodeComboBox()
    self.inputSelector.nodeTypes = ( ("vtkMRMLScalarVolumeNode"), "" )
    self.inputSelector.addEnabled = True
    self.inputSelector.removeEnabled = True
    self.inputSelector.setMRMLScene( slicer.mrmlScene )
    self.seedingSelector = slicer.qMRMLNodeComboBox()
    self.seedingSelector.nodeTypes = ( ("vtkMRMLLabelMapVolumeNode"), "" )
    self.seedingSelector.addEnabled = True
```

```
62      self.seedingSelector.removeEnabled = True
63      self.seedingSelector.setMRMLScene( slicer.mrmlScene )
64
65      # Change the parameters
66      #selCriteria = updateParameterCollapsibleButtion
67      updateParameterCollapsibleButtion      = ctk.ctkCollapsibleButton()
68      updateParameterCollapsibleButtion.text = "Selection Criteria"
69      self.layout.addWidget(updateParameterCollapsibleButtion)
70      updateParameterFormLayout              = qt.QFormLayout(updateParameterCollapsibleButtion)
71
72
73      chooseGlobalFrame, chooseGlobalSlider, chooseGlobalSliderSpinBox = ...
            numericInputFrame(self.parent, \
74                                                          "Maximum Global Intensity ...
                                                                Difference:   ", \
75                                                          "Determin the global range of ...
                                                                intensity values", \
76                                                          1, 50, 1, 0)
77      updateParameterFormLayout.addWidget(chooseGlobalFrame)
78
79      chooseLocalFrame, chooseLocalSlider, chooseLocalSliderSpinBox = ...
            numericInputFrame(self.parent, \
80                                                          "Maximum Local Intensity ...
                                                                Difference:    ", \
81                                                          "Determine the local range of ...
                                                                of intensity values", 0, ...
                                                                50, 1, 0)
82      updateParameterFormLayout.addWidget(chooseLocalFrame)
83
84
85      class state(object):
86        maxGlobalDiff    = 5
87        maxLocalDiff     = 5
88
89      scopeLocals    = locals()
90
91      def connect(obj, evt, cmd):
92        def callback(*args):
93          currentLocals = scopeLocals.copy()
94          currentLocals.update({'args':args})
95          exec cmd in globals(), currentLocals
96          updateGUI()
97        obj.connect(evt, callback)
98
99      def updateGUI():
100       chooseGlobalSlider.value          = state.maxGlobalDiff
101       chooseGlobalSliderSpinBox.value   = state.maxGlobalDiff
102       chooseLocalSlider.value           = state.maxLocalDiff
103       chooseLocalSliderSpinBox.value    = state.maxLocalDiff
104
105     connect(chooseGlobalSlider, 'valueChanged(double)', 'state.maxGlobalDiff = args[0]')
106     connect(chooseGlobalSliderSpinBox, 'valueChanged(double)', 'state.maxGlobalDiff = args[0]')
107     connect(chooseLocalSlider, 'valueChanged(double)', 'state.maxLocalDiff = args[0]')
108     connect(chooseLocalSliderSpinBox, 'valueChanged(double)', 'state.maxLocalDiff = args[0]')
109
110     updateGUI()
111     self.updateGUI  = updateGUI
112     self.state      = state
113     self.layout.addStretch(1)
114
115
116   # When the apply button is clicked
117   def onApply(self):
118
119     # Read in the input volume
```

```
120     inputVolume = slicer.mrmlScene.GetNodeByID('vtkMRMLScalarVolumeNode1')
121     inputVolumeData = slicer.util.array(inputVolume.GetID())
122
123     # Read in the seeding ROI
124     idString = ""
125     if slicer.util.getNode('vtkMRMLLabelMapVolumeNode1') is None:
126       idString = 'vtkMRMLLabelMapVolumeNode2'
127     else:
128         idString = 'vtkMRMLLabelMapVolumeNode1'
129     seedingROI = slicer.mrmlScene.GetNodeByID(idString)
130     seedingROIData  = slicer.util.array(seedingROI.GetID())
131
132
133     # Copy image node, create a new volume node
134     self.roiCounter += 1
135     outputROI_name = "MRBrainTumor" + str(self.roiCounter) + "_label_grow"
136     outputROI       = slicer.modules.volumes.logic().CloneVolume(slicer.mrmlScene, ...
            seedingROI, outputROI_name)
137     outputROIData   = slicer.util.array(outputROI.GetID())
138
139     # Get the mean of the seeding ROI
140     seedingROI_coords    = np.where(seedingROIData > 0)
141     seedingROI_values    = inputVolumeData[seedingROI_coords]
142
143     # # the location of the seeding voxel
144     sx = seedingROI_coords[0][seedingROI_values.argmax()]
145     sy = seedingROI_coords[1][seedingROI_values.argmax()]
146     sz = seedingROI_coords[2][seedingROI_values.argmax()]
147
148     # The global parameter is used to select the voxels within a range
149     ROI_min = seedingROI_values.min() - self.state.maxGlobalDiff
150     ROI_max = seedingROI_values.max() + self.state.maxGlobalDiff
151
152     # Dimension of the input volume
153     dx, dy, dz = inputVolumeData.shape
154
155     iteration = 0
156     # the local searching radius
157     radius = 1
158
159     while True:
160
161         iteration = iteration + 1
162         #print 'INFO Current Iteration: ', iteration
163
164         # First stop criterion: reach the boundary of the image
165         searching_extend    = np.array([iteration+radius-sx, sx+iteration+radius+1-dx, \
166                                         iteration+radius-sy, sy+iteration+radius+1-dy, \
167                                         iteration+radius-sz, sz+iteration+radius+1-dz])
168         if (searching_extend >= 0).any():
169             break
170
171         # Second stop criterion: there is no new voxel with in the global value range
172         new_voxel_coords    = find_new_voxels(sx, sy, sz, iteration)
173         new_voxel_values    = inputVolumeData[new_voxel_coords[:, 0], new_voxel_coords[:, 1], ...
              new_voxel_coords[:, 2]]
174         glb_voxel_indices   = np.where(np.logical_and(new_voxel_values < ROI_max, ...
              new_voxel_values > ROI_min))
175
176         if not glb_voxel_indices:
177             break
178
179         else:
180             for i in glb_voxel_indices[0]:
181                 lx, ly, lz          = new_voxel_coords[i, :]
```

```
182                 patch_boolen        = outputROIData[lx - 1 : lx + 2, ly - 1 : ly + 2, lz - 1 ...
                        : lz + 2]
183
184                 if patch_boolen.sum() > 1:
185                     local_value     = inputVolumeData[lx, ly, lz]
186                     patch_values    = inputVolumeData[lx - 1 : lx + 2, ly - 1 : ly + 2, lz - ...
                            1 : lz + 2]
187                     boolen_values   = patch_values[:] * patch_boolen[:]
188
189                     existing_values = boolen_values[np.where(boolen_values > 0)]
190                     local_min       = existing_values.min() - self.state.maxLocalDiff
191                     local_max       = existing_values.max() + self.state.maxLocalDiff
192
193                     # Third stop criterion: the voxel value is beyond the range of local ...
                            existing neighbors
194                     if local_value < local_max and local_value > local_min:
195                         outputROIData[lx, ly, lz] = 1
196
197     ########################################################
198
199     outputROI.GetImageData().Modified()
200
201     # make the output volume appear in all the slice views
202     selectionNode = slicer.app.applicationLogic().GetSelectionNode()
203     selectionNode.SetReferenceActiveLabelVolumeID(outputROI.GetID())
204     slicer.app.applicationLogic().PropagateVolumeSelection(0)
205
206   #
207   # Supporting Functions
208   #
209   # Reload the Module
210   def onReload(self, moduleName = "ITBRegionGrow"):
211     import imp, sys, os, slicer
212     widgetName = moduleName + "Widget"
213     fPath = eval('slicer.modules.%s.path' % moduleName.lower())
214     p = os.path.dirname(fPath)
215     if not sys.path.__contains__(p):
216       sys.path.insert(0,p)
217     fp = open(fPath, "r")
218     globals()[moduleName] = imp.load_module(
219         moduleName, fp, fPath, ('.py', 'r', imp.PY_SOURCE))
220     fp.close()
221     print "the module name to be reloaded,", moduleName
222     # find the Button with a name 'moduleName Reolad', then find its parent (e.g., a collasp ...
          button) and grand parent (moduleNameWidget)
223     parent = slicer.util.findChildren(name = '%s Reload' % moduleName)[0].parent().parent()
224     for child in parent.children():
225       try:
226         child.hide()
227       except AttributeError:
228         pass
229     item = parent.layout().itemAt(0)
230     while item:
231       parent.layout().removeItem(item)
232       item = parent.layout().itemAt(0)
233     globals()[widgetName.lower()] = eval('globals()["%s"].%s(parent)' % (moduleName, widgetName))
234     globals()[widgetName.lower()].setup()
235
236
237 # Numeric parameter input
238 def numericInputFrame(parent, label, tooltip, minimum, maximum, step, decimals):
239   inputFrame              = qt.QFrame(parent)
240   inputFrame.setLayout(qt.QHBoxLayout())
241   inputLabel              = qt.QLabel(label, inputFrame)
242   inputLabel.setToolTip(tooltip)
```

```
243    inputFrame.layout().addWidget(inputLabel)
244    inputSpinBox            = qt.QDoubleSpinBox(inputFrame)
245    inputSpinBox.setToolTip(tooltip)
246    inputSpinBox.minimum    = minimum
247    inputSpinBox.maximum    = maximum
248    inputSpinBox.singleStep = step
249    inputSpinBox.decimals   = decimals
250    inputFrame.layout().addWidget(inputSpinBox)
251    inputSlider             = ctk.ctkDoubleSlider(inputFrame)
252    inputSlider.minimum     = minimum
253    inputSlider.maximum     = maximum
254    inputSlider.orientation = 1
255    inputSlider.singleStep  = step
256    inputSlider.setToolTip(tooltip)
257    inputFrame.layout().addWidget(inputSlider)
258    return inputFrame, inputSlider, inputSpinBox
259
260  # define the cartesian function
261  def cartesian(arrays, out = None):
262      arrays = [np.asarray(x) for x in arrays]
263      dtype = arrays[0].dtype
264      n = np.prod([x.size for x in arrays])
265      if out is None:
266          out = np.zeros([n, len(arrays)], dtype = dtype)
267      m = n / arrays[0].size
268      out[:,0] = np.repeat(arrays[0], m)
269      if arrays[1:]:
270          cartesian(arrays[1:], out = out[0:m, 1:])
271          for j in xrange(1, arrays[0].size):
272              out[j*m:(j+1)*m, 1:] = out[0:m, 1:]
273      return out
274
275  # find the coordinates of new voxels
276  def find_new_voxels(sx, sy, sz, iteration, out = None):
277
278      new_voxel_coordinates_yz = cartesian((np.array([sx-iteration, sx+iteration]), \
279                                            np.arange(sy-iteration, sy+iteration+1), \
280                                            np.arange(sz-iteration, sz+iteration+1)))
281
282      new_voxel_coordinates_xz = cartesian((np.arange(sx-iteration+1, sx+iteration), \
283                                            np.array([sy-iteration, sy+iteration]), \
284                                            np.arange(sz-iteration, sz+iteration+1)))
285
286      new_voxel_coordinates_xy = cartesian((np.arange(sx-iteration+1, sx+iteration), \
287                                            np.arange(sy-iteration+1, sy+iteration), \
288                                            np.array([sz-iteration, sz+iteration])))
289
290      new_voxel_coordinates = np.concatenate((new_voxel_coordinates_yz, ...
           np.concatenate((new_voxel_coordinates_xz, new_voxel_coordinates_xy))))
291      return new_voxel_coordinates
```