Future Technologies Conference (FTC) 2017
29-30 November 2017| Vancouver, Canada

Isolating Bone and Gray Matter in MRI Images using 3D Slicer

Ashley Whiteside
Department of Computer Science,
University of Colorado, Colorado Springs,
CO, USA
amwhiteside1@yahoo.com

Sudhanshu Kumar Semwal
Department of Computer Science,
University of Colorado, Colorado Springs,
CO, USA
ssemwal@uccs.edu

***Abstract*—Slicer has been used in medical community for several years now. This paper describes a Python extension imported in Slicer application. The main contribution of the paper is to outline and explain how to import and test a Python extension which we created to isolate gray matter and bone in MR brain volume images. Our future plans include both qualitative and quantitative analysis, validation, and comparison to other similar techniques, and extensions to 3D surface extractions and interpretations using Slicer.**

Keywords—3D slicer; medical visualization; thresholding

I. INTRODUCTION

The inability of an unskilled and untrained user to identify different parts of the brain, including the bone and gray matter, limits those who can correctly interpret these images to only highly-trained, skilled individuals. However, the number of skilled individuals is much lower than the number of untrained people, and the services of trained individuals is often in high demand. As a result, the number of images that can be interpreted in a reasonable timeframe is severely limited. Using the Python extension functionality in 3D Slicer, we created an extension for the purpose of isolating the gray matter and the bone in an MRI Images as a first step towards understanding both the 3D Slicer and how one could use it to specifically develop their own interfaces using Python extensions.

3D Slicer is an open-source software platform available for Linux, MacOSX, and Windows. It was created for the purpose of image analysis and visualization [1]-[5] and is extensively used for research in image-guided therapy [6]. Its powerful extension capabilities allow for the addition and use of a Python algorithm, discussed in this paper, updates the MRI images to clearly show either the bone or gray matter found in the brain.

The method used to isolate these materials was a combination of averaging and thresholding. For the purpose of isolating the gray matter, the average of each slice was calculated and a user-adjustable standard deviation was applied. All pixel values outside of this range were set to zero. To find the bone, the maximum and average value of each slice was averaged, and the user-defined bone standard deviation was applied. As with the gray matter, all pixel values outside of this range were once again set to zero.

This paper discusses how to create or import an extension in 3D Slicer, testing and running the code used to isolate gray matter and bone, specifics regarding how this code was written, and improvements that can be applied to the code in the future.

II. MOTIVATION

Identifying areas of interest using thresholding, such as the technique described in this paper, can be precursor to obtaining surfaces from medical data, such as 3D volume of CT/MRI scan slices [1]-[5]. Surfaces are extracted by sometimes identifying contours in the 2D slices. Once the contours have been collected, the reconstruction problem becomes one of constructing a surface between them. Most techniques construct the surface with a mesh of polygons [7]-[11]. The most common of these is the triangle. After the polygons are extracted, traditional surface shading techniques are applied. Some details are lost if thresholding is used for identifying contours. Identifying a set of closed contours is a costly process. The *marching cubes* approach [10] is a well-known surface construction model and remains an excellent algorithm to interpret the binary images. Depending upon the intensity values of the eight vertices of the voxel, a surface is generated using a predefined polygonal topology from a look-up table. A major problem with this approach is surface shading due to relatively high number of triangles or polygons generated depending upon the voxels (3D cells) occupied by the object of interest. This also requires a large amount of memory but is simple to implement.

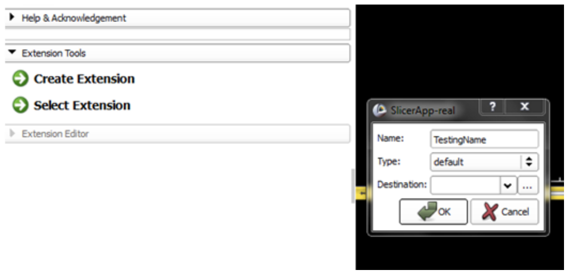
On the other hand, non-thresholding models have been proposed to better understand the entire volume data. Surfaces are not constructed, but rather are displayed implicitly. Voxel-based models correspond closely to the format of the medical data. The entire volume is stored in terms of voxels. Frieder et al. display the surface by showing voxels in a back-to-front approach [8]. Much of the research in volume rendering has been published in the field of medical imaging [12]-[19]. Medical imaging techniques such as computed tomography (CT), magnetic resonance imaging (MRI), and single-photon emission computed tomography (SPECT) use scanners to create volumes by imaging a series of cross-sections, resulting in multiple 2D slices of information [16]. Between two consecutive slices, there are several rectangular parallelepiped regions or voxels. Every voxel is assigned a value, called its density, which represents some object property [17]. Ray tracing is a technique that considers the effect of light by tracing rays in the scene. An excellent reference on this subject is [18]. Often one ray per pixel on the projection plane is traced. In this case, the values encountered along the path of the ray contribute to the pixel's final color. Ray casting [14], [15] is a simpler variation of *ray tracing* where no secondary rays are considered. Only a set of primary rays is traced through the volume. These rays accumulate color, based on the intensities of the points on 2D slices encountered along the path of the ray. When the accumulated opacity reaches a value of 1 or greater, or the ray is outside the boundary of the volume, the ray traversal is terminated [17]. Since no surface construction is required, ray casting models are relatively fast. Schlusselberg et al. [12] and Levoy [13] have proposed models that composite the values along the path of the ray in a back-to-front ordering. Upson and Keeler [14] suggest a front-to-back-volume rendering approach. In their model, the color and opacity for the ray are accumulated until the ray traverses the volume or the accumulated opacity reaches a specified value. Drebin et al. [18] presented a voxel-based technique for rendering images of volumes containing mixtures of materials. In the following sections, we will explain the method we followed so that an extension isolating gray matter and bones can be integrated as a plug-in to 3D Slicer.

III. Creating or Importing an Extension in 3D Slicer

Code for a boilerplate extension is provided by the 3D Slicer when the extension is first created. Several tutorials on the website (Slicer.org) explain such methods. The code was then adapted to our needs as necessary. To create an extension, the first step is to enable developer mode. First 3D Slicer is opened, and then navigates to Edit -> Application Settings -> Developer, and check the box labeled "Enable developer mode" [6]. Once completed, select Modules -> Developer Tools -> Extension Wizard and click "Create Extension" [6] to create a new extension.

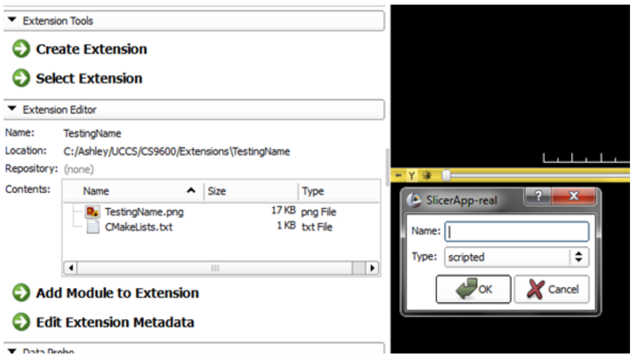
At this point, a new window will open that says "The following module can be loaded. Would you like to load it now?" and the module that was just created will be visible. Ensure it is checked, and check the box labeled "Add selected module to search paths" [6]. A folder will be created in the destination chosen for saving the extension. In this folder will be a python code file (.py). This file can be opened for editing in Notepad++, a Python interface, or any other software that supports reading and editing Python files, then executed in 3D Slicer.

To import an existing extension, open 3D Slicer and navigate to Edit -> Application Settings -> Developer and check the box labeled "Enable developer mode" [6]. Once completed, select Modules -> Developer Tools -> Extension Wizard and click "Select Extension". Navigate to the folder containing the extension to import and choose the folder. The software will automatically find the extension itself and a window will open that says "The following module can be loaded. Would you like to load it now?" Ensure the extension name is checked, and check the box "Add selected module to search path".



Name: TestingName
Type: default
Destination:

Fig. 1. Create extension.



Name: TestingName
Location: C:/Ashley/UCCS/CS9600/Extensions\TestingName
Repository: (none)
Contents:

Name	Size	Type
TestingName.png	17 KB	png File
CMakeLists.txt	1 KB	txt File

Name:
Type: scripted

Fig. 2. Add module to extension.

As shown in Fig. 1, first choose the name for the extension. To update the "Destination" textbox, click on the ellipses button to choose the folder the extension will be saved in, then click "OK". A new window will open labeled "SlicerApp-real". No changes are necessary in this window, click "OK" to continue.

A module must now be added to the extension that has been created. Select "Add Module to Extension" and provide a name for the module, then click "OK" (see Fig. 2).

Once an extension has been loaded or created, find the "Modules" dropdown menu and click on it. Hover over "All Modules" and a menu will open to the right. In this menu, find the name of the extension that has been created or imported and choose it. The GUI on the left side of the 3D Slicer window should update.

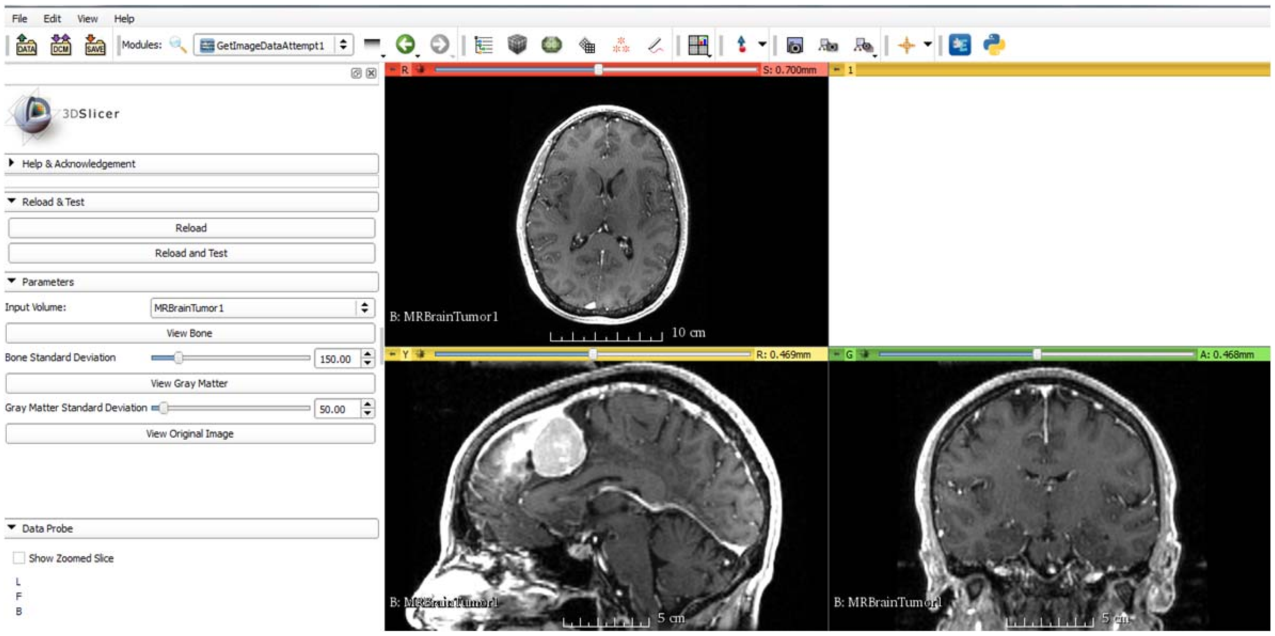
IV. Isolating Grey Matter and Bone

To load the code used for isolating gray matter and bone, follow the steps above to import the extension. Once imported, choose All Modules -> Sample Data from the Modules drop-down menu. Click "Download MRBrainTumor 1". When the volume has been imported and is showing in the images on the right of the GUI, select the loaded extension from the "All Modules" options in the Modules drop-down menu. The window should match Fig. 3.

Click the button "View Bone". Following this, either selects View -> Python Interactor, or push Ctrl+3 to open the Python Interactor window. Errors encountered while executing the code will be shown in red in the Python Interactor window, while a successful execution will show all words in green. Once the Python Interactor window is opened, the images on the right will update and should look similar to Fig. 4.

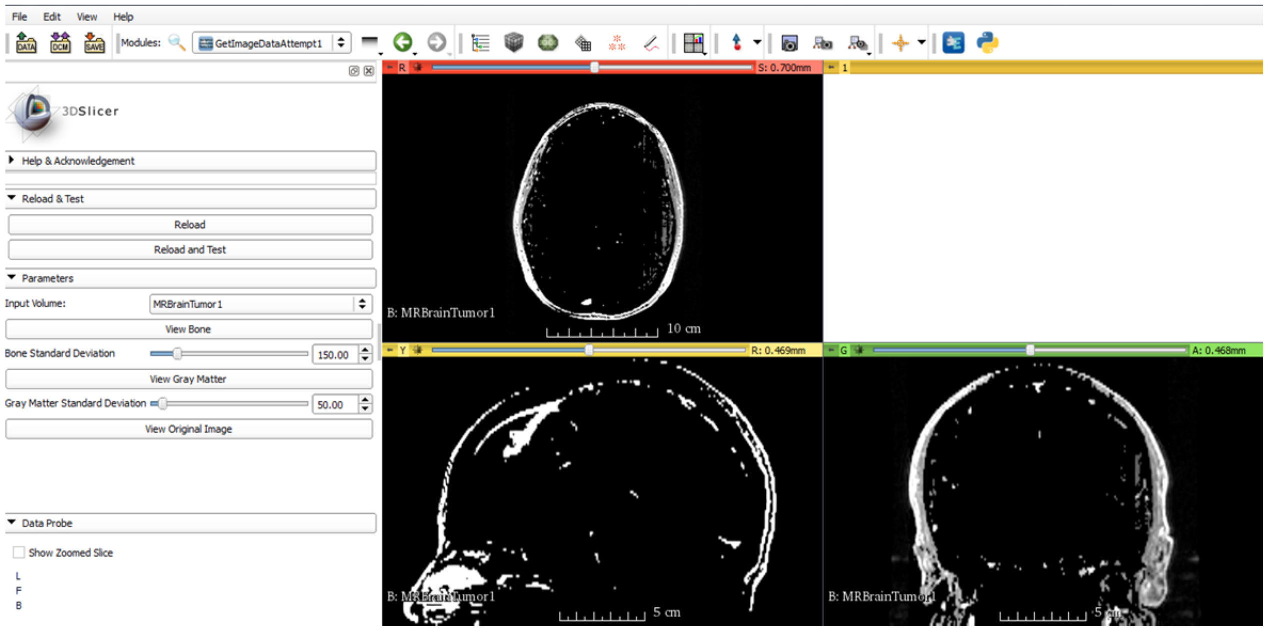
By clicking on and dragging the slider-bars above each image, the individual slices can be viewed. In each slice, the bone alone should be readily visible. To view the gray matter, click "View Gray Matter" and open the Python Interactor Window again. Once again, the images should update and look similar to Fig. 5.

As with the bone, by moving the sliders above each image, different slices can be viewed and the gray matter alone in each slice should be clearly visible. To restore the original image, click the button "View Original Image" and open the Python Interactor. Once completed, the 3D Slicer interface should update to match Fig. 3.



Input Volume: MRBrainTumor1
Bone Standard Deviation 150.00
Gray Matter Standard Deviation 50.00

Fig. 3. Initial window with extension loaded.



Input Volume: MRBrainTumor1
Bone Standard Deviation 150.00
Gray Matter Standard Deviation 50.00

Fig. 4. View bone.

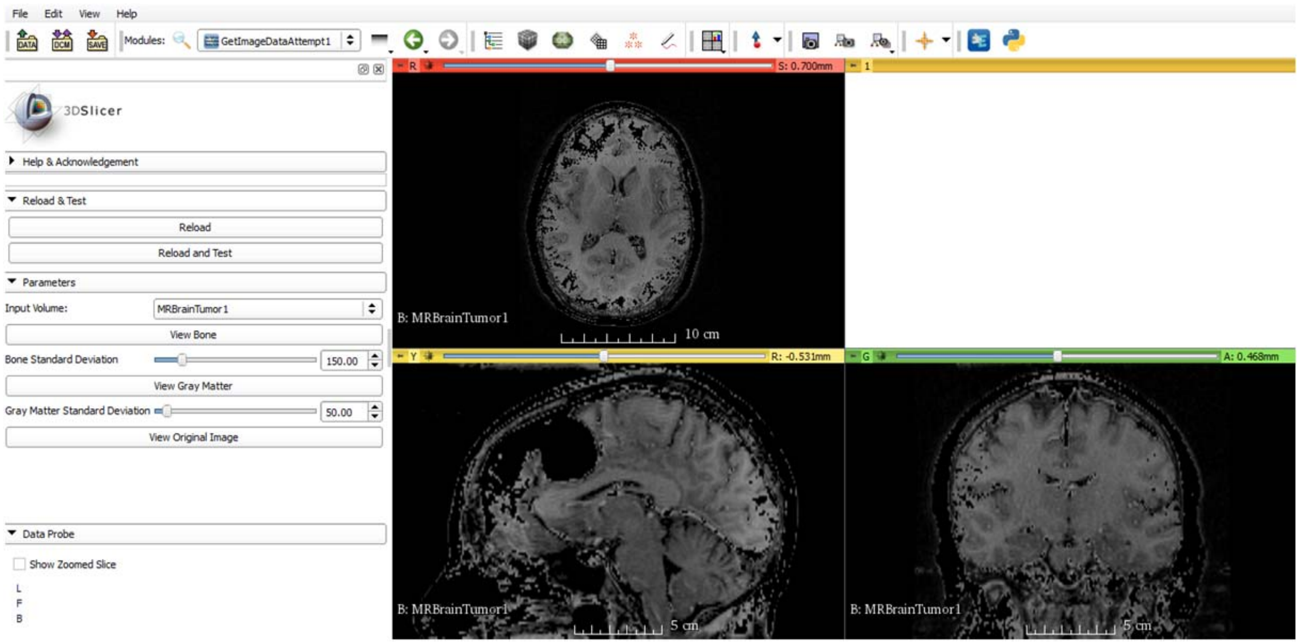


Fig. 5. View gray matter.

To isolate the bone in the MRI volume, the average pixel value of each slice in the image is taken and averaged again with the maximum pixel value of each slice. The value defined on the sliding bar labeled “Bone Standard Deviation” is added and subtracted from the calculated average and all pixel values outside of this range are set to zero. As an example, assume the average of the middle slice in a volume is 100, and the maximum value is 300. The average of 100 and 300 is calculated, returning 200. If the “Bone Standard Deviation” value is 50, all pixel values that are not in the range of 150 and 250 are set to zero and the image is updated.

To isolate the gray matter in the MRI volume, the average pixel value of each slice in the image is taken and the value defined on the sliding bar labeled “Gray Matter Standard Deviation” is added to and subtracted from this average. All pixel values outside of this range are set to zero. As an example, assume the average pixel value of the middle slice in a volume is calculated as 100 and the “Gray Matter Standard Deviation” is set to 25. All pixel values outside of the range 75-125 are set to zero, and the image is updated, visually isolating only the pixels showing the gray matter.

V. Implementation

When an extension is first created and boilerplate code is generated, there is a class generated with a function named “setup” within it. The setup function is used to create all of the buttons, slider bars, drop-down menus, etc. shown in the left panel of the 3D Slicer when running an extension. In this function, the object “self” is used to refer to the created elements. For example, a slider is created with the following code:

```
self.slider = ctk.ctkSliderWidget()

self.slider.singleStep = 1

self.slider.minimum = 0

self.slider.maximum = 100

self.slider.value = 50

parametersFormLayout.addRow(“Slider”, self.slider)
```

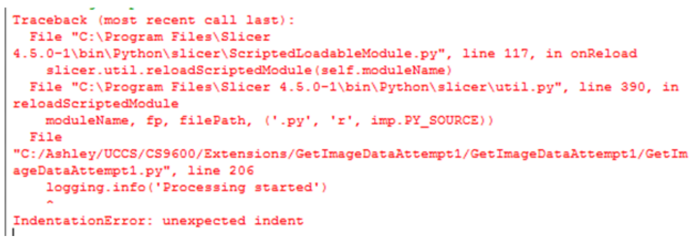
This code will create a slider bar with a default value of 50, increments of 1, a minimum of 0, and a maximum of 100. If the value of the slider bar is needed, it can be accessed with the command self.slider.value and applied to any necessary calculations.

To create a pushbutton, the following code is used under the “setup” function.

```
self.buttonEx = qt.QPushButton(“Pushbutton Testing”)

self.buttonEx.toolTip = “Testing”

self.buttonEx.enabled = True

parametersFormLayout.addRow(self.buttonEx)

self.buttonEx.connect(‘clicked(bool)’, self.onButtonEx)
```

This code creates a pushbutton labeled “Pushbutton Testing” that is enabled (so it can be pushed), and with a tooltip (shown when the button is hovered over) saying “Testing”. Once this button is clicked, the function “onButtonEx” will be accessed. This function must be created under the same class as setup and have an input of self.

A common obstacle encountered during the coding process was using tabs instead of spaces. The code in this case was edited using Notepad++, and by default, when the “Enter” key was pushed, the next line was tabbed in from the edge. However, with the Python Interactor through 3D Slicer, tabs are not acceptable. Instead spaces need to be used. If the code is executed with tabs included instead of spaces, the following message, or a very similar one, will be shown in the Python Interactor:



```
Traceback (most recent call last):
  File "C:\Program Files\Slicer
4.5.0-1\bin\Python\slicer\ScriptedLoadableModule.py", line 117, in onReload
    slicer.util.reloadScriptedModule(self.moduleName)
  File "C:\Program Files\Slicer 4.5.0-1\bin\Python\slicer\util.py", line 390, in
reloadScriptedModule
    moduleName, fp, filePath, ('.py', 'r', imp.PY_SOURCE))
  File
"C:/Ashley/UCCS/CS9600/Extensions/GetImageDataAttempt1/GetImageDataAttempt1/GetIm
ageDataAttempt1.py", line 206
    logging.info('Processing started')
    ^
IndentationError: unexpected indent
```

Fig. 6. Python interactor indentation error.

Assuming no errors are shown in the Python Interactor when the code is loaded, once the Pushbutton Testing button is pushed, the function onButtonEx is executed. In this function, a new object is created that accesses another class. Within this class is a "run" function. The run function is where the image calculations isolating the gray matter and bone are completed and applied. Once the software has completed execution, the Python Interactor will be updated and will display any messages that were defined in the code. These messages can be displayed two ways. First, by using the "logging.info" command (logging.info('This message will be displayed')) Second, by using "print". In the code relevant to this paper, print was used to display variable information. For example, print(inputVolume) will display all the information in the inputVolume variable. When either of these two commands attempted to combine a string and a variable value, an error was given and the execution couldn't complete. Also included in the setup section of code was the use of functionality from the "NumPy" library.

The NumPy library was imported using the following line of code: "import numpy as np". Once imported, the following code was executed:

curVolDataOrig = slicer.util.array(self.inputSelector.currentNode().GetName())

self.curVolData = np.copy(curVolDataOrig)

The first line saves the array containing the image data in the variable curVolDataOrig. The array returned is 3-dimensional, containing all the pixel values for each slice of the imported volume. The second line creates a copy of the array data. This was necessary because any changes made to the array data in corresponding functions were made to the original image itself. As a result, if the original image needed to be displayed or updated again, the original image was replaced with the updated one. The copy of the original image data is stored in a separate memory location that preserves the original image pixel values, allowing the original image data to be restored as/if needed.

Both gray matter and bone were isolated in their corresponding run functions by using the functionality provided by the numpy library. The inputs to the run function are the "self" object created in the setup function, the current array data information, and the original volume data. The np.copy function is utilized first in both of the "run" functions, for the purpose of making a copy of the original image data. Once again, this was done to prevent changes made in to the image data from affecting the original image. The dimensions of the volume data are then found using the following line of code:

Dim =inputVolumeBone.GetImageData().GetDimensions()

An array is stored in the Dim variable, and the first element is isolated. Initially, working under the assumption that the values in the array correspond to the x, y, and z dimensions in that order, the third element was isolated and saved in a variable (numSlices = Dim[2]). However, when the next coding statement, a "for" loop, was executed (this loop runs through each slice and updates the pixel values), only half the image was updated. However, when the first element was saved in numSlices (numSlices = Dim[0]), the entire volume image was updated. While running through this loop, the code to isolate bone and gray matter diverges. To isolate bone, the mean and the maximum values of the current slice are averaged using the code below:

maxValCur = np.amax(curSlice)

meanImgCur = np.mean(curSlice)

arrayValues = [maxValCur,meanImgCur]

meanMaxMean = np.mean(arrayValues)

and the value defined on the "Bone Standard Deviation" slider bar (saved in the variable boneThreshold) is both added and subtracted from meanMaxMean. All pixel values outside of this range are set to 0 with the following code:

threshBone1 = meanMaxMean + boneThreshold

threshBone2 = meanMaxMean - boneThreshold

curSlice[curSlice > threshBone1] = 0

curSlice[curSlice < threshBone2] = 0

To isolate gray matter, the mean value of the current slice is calculated, and the value defined in the "Gray Matter Standard Deviation" slider bar (saved in the variable gmThreshold) is added and subtracted from the calculated mean. All pixel values outside of this range are set to 0. The code to execute this is:

meanValCur = np.mean(curSlice)

threshGm1 = meanValCur + gmThreshold

threshGm2 = meanValCur – gmThreshold

curSlice[curSlice > threshGm1] = 0

curSlice[curSlice < threshGm2] = 0

While the code above for both the bone and the gray matter certainly sets the pixel values of the image to 0 if they are outside the designated range, the volume image itself is not updated until the following is executed:

curArrayGm[:,:,index] = curSlice[:,:]

curNodeGm.GetImageData().Modified()

Initially, instead of using the code curArrayGm[:,:,index] = curSlice[:,:] to update the slice values, curArrayGm[:,:,index] = curSlice[:] was attempted. Unfortunately, as a result, the entire volume was shown as black when it was updated.

VI. User Manual Instructions

Instructions for Use: Isolating Bone and Gray Matter in MRI Images Using 3D Slicer.

1) Open the 3D Slicer Interface.

2) Click the button "Download Sample Data".

3) Choose "Download MRBrainTumor1".

4) Click on the drop-down menu with a default label of "Sample Data".

- Choose Developer Tools -> Extension Wizard.

5) Push the button labeled "Select Extension".

6) Navigate to the folder used to store the folder "GetImageDataAttempt1" and click "Select Folder".

7) Select "Add Module to Extension" and give it a name of your choice.

- This name should be relevant to what the code is being used for – it is what will be seen from this point forward when trying to run the code.

8) Click "OK".

9) A window will open saying "The following module can be loaded. Would you like to load it now?"

- Check the box labeled "Add selected module to search paths".
- Click "Yes".

*10)*The drop-down menu that had "Sample Data" chosen as the default will now read "Extension Wizard".

- From this drop-down, select "All Modules" and choose the name of the module added in Step 7.

*11)*The left-side part of the GUI should update. Buttons labeled "Reload", "Reload and Test", "View Bone", "View Gray Matter", and "View Original Image" should be visible.

- Also visible should be a drop-down menu labeled "Input Volume" that defaults to MRBrainTumor1.
- There should be two slider bars as well, one labeled "Bone Standard Deviation", with a default value of 150, and one labeled "Gray Matter Standard Deviation" with a default value of 50.

*12)*Push "View Bone", then open the Python Interactor.

- The Python Interactor can be opened in two ways.
    - o First, by using a shortcut of Ctrl+F3 (essentially, just push the Ctrl button, followed by the F3 button).
    - o Second, by choosing View -> Python Interactor.

*13)*Once the Python Interactor has opened, the images should visibly update, isolating parts of the image that show bone. At this point, the Python Interactor can be closed and the different slices for each image can be viewed as desired.

*14)*Click the button "View Gray Matter" and open the Python Interactor again (as discussed in Step 12)

*15)*Close the Python Interactor once the images have updated. Areas containing gray matter should be clearly visible and the different slices for each image can be viewed as desired.

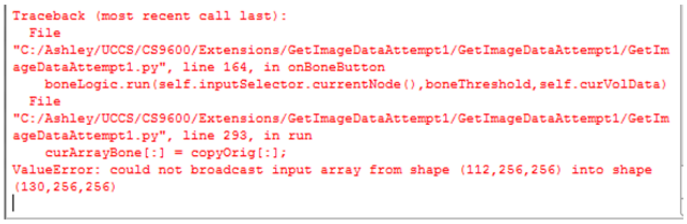
*16)*Click the button "View Original Image" and open the Python Interactor again to restore the original demo image that was loaded.

Tips and Tricks

- The three buttons – "View Bone", "View Gray Matter", and "View Original Image" can be pressed in any order.
- To update the image so it shows the bone only, the average pixel values in each slice and the maximum pixel values in each slice are averaged together. Following this, all pixel values outside of the average + the Bone Standard Deviation (defined by the user), and the average – the Bone Standard Deviation (defined by the user) are set to 0.
    - o For example, if the average of an image is 100 and the maximum value is 300, the average of 100 and 300 is calculated (200). If the Bone Standard Deviation is set to 150, all pixel values that have a value between 50 and 350 are preserved, and all values outside of this range are set to 0.
- To update the image so it shows the gray matter only, the average pixel values in each slice are calculated, then all pixel values outside of the average + Gray Matter Standard Deviation and the average – Gray Matter Standard Deviation are set to 0
    - o For example, if the average of an image is 100 and the Gray Matter Standard Deviation is 50, all pixel values in the range of 50-150 are preserved, while all outside of that range are set to 0.
- To test additional demo images (for example, the MRHead Sample Data), currently the most efficient way is to close out of 3D Slicer, then reopen it and load the sample data from MRHead instead of MRBrainTumor1 in Step 3. If Steps 4-9 above have already been executed, they can be skipped if continued testing of the GetImageDataAttempt1 is to be done.
    - o We have tried loading different Sample Data without closing 3D Slicer. While the drop-down menu does update to include the new data, if any of the buttons are pushed, it is not executed properly. Furthermore, while the drop-down menu does have the name of the selected file, the images to the right do not update when a new file name is selected.
- Once you have completed Step 10, as long as 3D Slicer is open, the name you created in Step 7 (the module tied to GetImageDataAttempt1), will be visible under the drop-down menu that shows different shades of gray from the top to the bottom. When hovered over, it reads "Modules History". This is a shortcut to finding the module under the list of all modules (the drop-down menu that by default has "Sample Data" selected).

VII. Results and Future Extensions

To test additional demo images (for example, to run the code on both the MRHead and MRBrainTumor1 sample data), 3D Slicer currently needs to be closed and reopened. To continue the example, assume 3D Slicer is opened and the MRBrainTumor1 sample data is downloaded. The extension to isolate bone and gray matter is loaded and run, and there are no problems. If the code then needs to be run on the MRHead sample data, 3D Slicer needs to be closed, then opened again. This time, select the MRHead sample data to download, and then load the extension to isolate bone and gray matter. Once run, there should be no errors. If the MRBrainTumor1 sample data is downloaded initially, then File -> Download Sample Data is selected and the MRHead sample data is downloaded, the drop-down menu labeled "Input Volume" will update to include both of the sample data names. However, if any of the buttons are pushed while "MRHead" is selected from the drop-down, errors similar to Fig. 7 will be shown in the Python Interactor. Note that Fig. 3 to 5 shows the working of our plugin extension.



```
Traceback (most recent call last):
  File
"C:/Ashley/UCCS/CS9600/Extensions/GetImageDataAttempt1/GetImageDataAttempt1/GetIm
ageDataAttempt1.py", line 164, in onBoneButton
    boneLogic.run(self.inputSelector.currentNode(),boneThreshold,self.curVolData)
  File
"C:/Ashley/UCCS/CS9600/Extensions/GetImageDataAttempt1/GetImageDataAttempt1/GetIm
ageDataAttempt1.py", line 293, in run
    curArrayBone[:] = copyOrig[:];
ValueError: could not broadcast input array from shape (112,256,256) into shape
(130,256,256)
```

Fig. 7. Python interactor error.

VIII. Conclusions and Future Work

The isolation of gray matter and bone in MR brain volume images was completed through the application of pixel averaging and a user-defined standard deviation. Combined, these values created upper and lower thresholds for each slice and all pixels outside of these bounds were set to 0. However, the current code limits the number of images that can be analyzed to *one* at a time, and any new image needs to be loaded after 3D Slicer has been closed and reopened. Furthermore, the code was tested on a limited sample size. To determine its effectiveness for a wide range of volume images, many more need to be tested so that both qualitattive and quantitative analysis can be further prerformed. Visually isolating the bone and gray matter in MR Images via 3D Slicer has the potential to dramatically increase the number of images that can be processed and interpreted by removing the need for highly-trained, skilled individuals to interpret each image themselves. Our system was tested by two other graduate students who are working in this area. Our future plans include both qualitative and quantitative analysis of thresholding itself, validation of false positives and false negatives, and comparison to other similar techniques. Further extensions to 3D surface extractions and interpretations using Slicer are also planned.

Acknowledgment

This work and the corresponding code is only valid for research purposes. We are grateful for 3D Slicer community for their open source software, as that allowed us to perform this Independent Study at University of Colorado, Colorado Springs, Summer 2016.

References

[1] T. S. Community, "What is NumPy?," Scipy, 29 May 2016. [Online]. Available: http://docs.scipy.org/doc/numpy/user/whatisnumpy.html. [Accessed 23 July 2016].

[2] X. He, X. Li, G. Yang, J. Xu and Q. Jin, "Adaptive Tag Selection for Image Annotation," Springer International Publishing, Switzerland, 2014.

[3] W. T. Ooi, C. G. Snoek, H. K. Tan, C.-K. Ho, B. Huet and C.-W. Ngo, "Advances in Multimedia Information Processing - PCM 2014," 15th Pacific-Rim Conference on Multimedia, Kuching, 2014.

[4] V. E. Balas, L. C. Jain and B. Kovacevic, "Soft Computing Applications," Springer, 2016, 2014.

[5] Medical Perspective by SK Semwal, pp. 1-17, Internal Document, GMI Lab, Department of Computer Science, University of Colorado, Colorado Springs, 2016.

[6] "3D Slicer - Documentation/4.5/Announcements," SlicerWiki, 15 December 2015. [Online]. Available: http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.5/Announcements. [Accessed 15 July 2016].

[7] A. Wallin, Constructing isosurfaces from CT data, IEEE Compuf. Graphics Applic. 11, 28-33 (1991).

[8] G. Frieder, D. Gordon and R. A. Reynolds, Back-to-front display of voxel-based objects, IEEE Comput. Graphics Appfic. 5, 52-60 (1985).

[9] L. Chen, G. T. Herman, R. A. Reynolds and J. K. Udupa, Surface shading in the cuberille environment, IEEE Comput. Graphics Applic. 5, 33-43 (1985).

[10] W. E. Lorenson and H. E. Cline, Marching cubes: a high resolution 3D surface reconstruction algorithm, Comput. Graphics (SIGGRAPH '87 Proc.), 21, 163-169 (1987).

[11] L. Axel, P. H. Arger and R. A. Zimmerman, Applications of computerized tomography to diagnostic radiology, Proc. IEEE 71, 293-372 (1980).

[12] D. S. Schlusselberg, W. K. Smith and D. J. Woodward, Three-dimensional display of medical image volumes, Proc. NCGA 3, 114-123 (1986).

[13] M. Levoy, Display of surfaces from volume data, IEEE Comput. Graphics Applic. 8, 29-37 (1988).

[14] C. Upson and M. Keeler, V-BUFFER: visible volume rendering, Comput. Graphics (SIGGRAPH'88 Proc.) 22, 59-64 (1988).

[15] R. Drebin, L. Carpenter and P. Hanrahan, Volume rendering, Cdmput. Graphics (SIGGRAPH '88Proc.) 22, 65-74 (1988).

[16] V. M. Spitzer and D. G. Whitlock, High resolution electronic imaging of the human body, Biol. Photogr. 60, 167-172 (1992).

[17] Darin Buchanan and Sudhanshu Kumar Semwal, A Front to Back Technique for Volume Rendering, Computer Graphics International, Computer Graphics Around the World, Singapore, pp. 149-174. Springer-Verlag (1990).

[18] Sudhanshu Kumar Semwal and Mark Freiheit, Mesh Splitting for the Enclosing Net Algorithm, Proceedings of the International Conference on Imaging Science, Systems and Technology, Las Vegas, Nevada, USA, pp. 375-382, (July 1998).

[19] P. Swann and S. K. Semwal, Flow visualization of point data, IEEE Visualization 91 Conference, IEEE Computer Societv Press. San Diego, California. PP. 25-32 (1991).