Fast Slice Requirements Document

System Description

System Purpose:

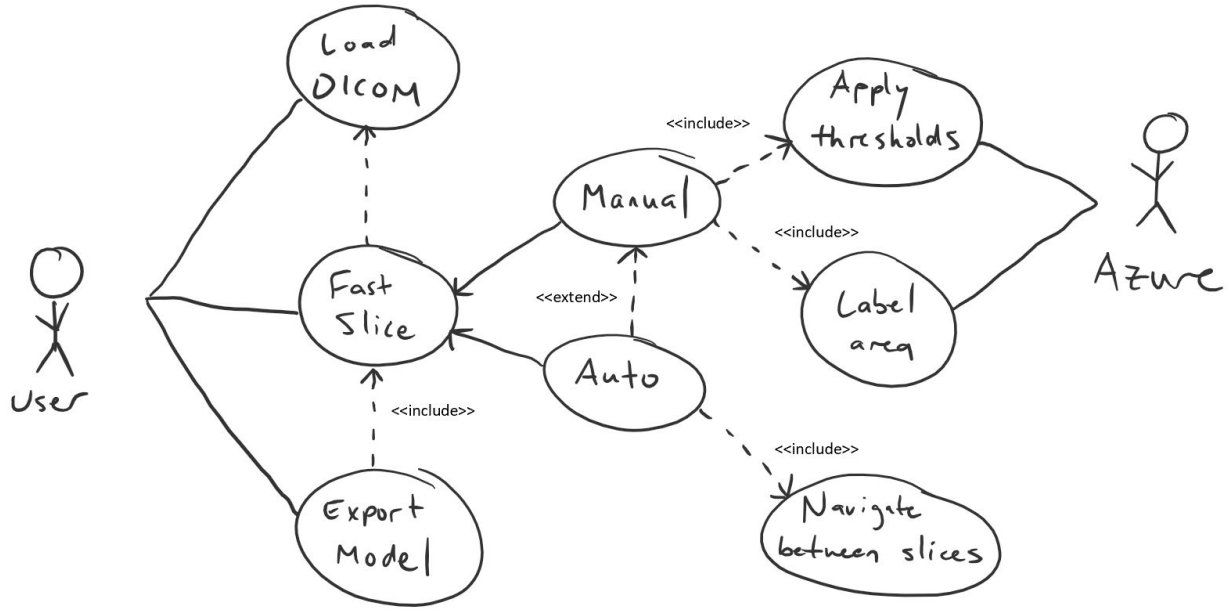
Assist in the process of creating sockets for prosthetics. Currently, a large bottleneck in the process of 3D-modeling the amputation is manually selecting the bone in 3D Slicer (a process that takes multiple hours). Our project aims to automate the process using computer vision.

Project Scope

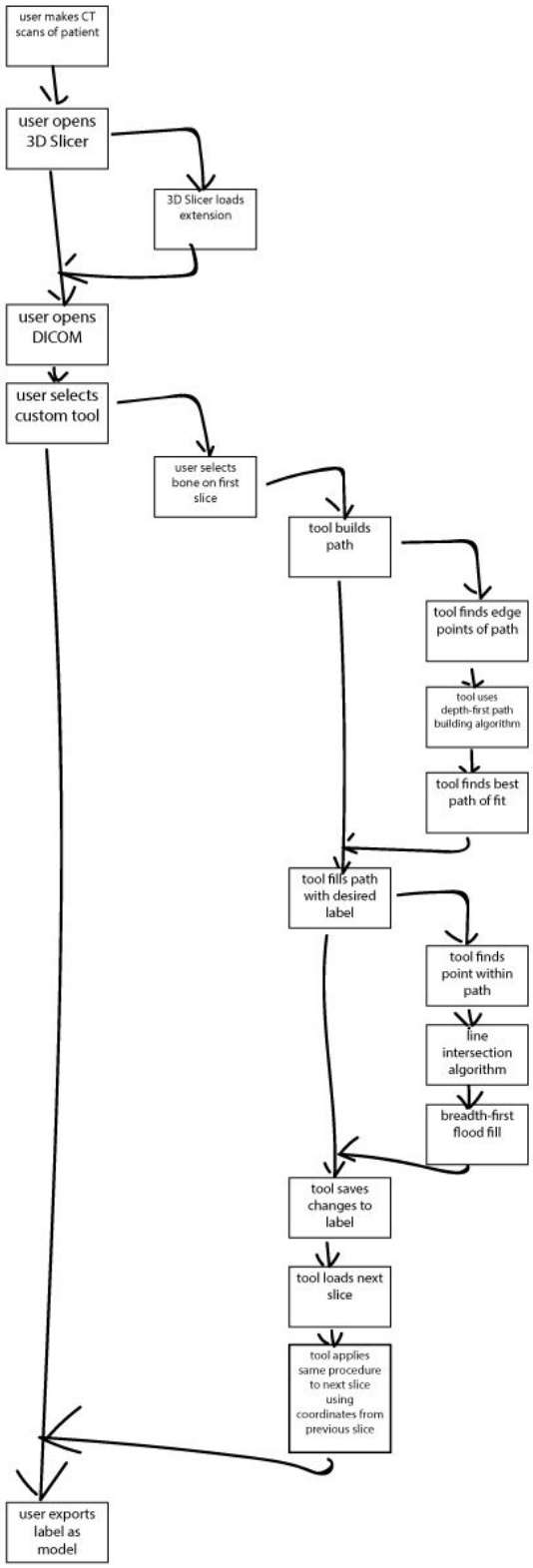
Base: The base functionality of our system should be to identify and flag the leg bones of interest. At a low level, the user can use the tool on an individual image. However, the user will have the option to select and fill a bone through a range of images. The user will select the starting image, the ending image and the bone of interest common through the range. After selection the tool will automatically identify and mark the target bone in each image in the range. The system will offer threshold settings for the bone detection so that the user can fine tune the detection.

Extended: At an extended level the tool should be able to automatically identify dense bone in all the images without any user input. All uncertain areas should be marked, and the threshold settings at each image should be adjustable. Changes in this setting being reflected on the image layer in real time. The tool should also allow for manual changes at this level.

Use Case Diagram:



Description: The user can can make use of the extension (named Fast Slice for now), either manually or automatically. In manual mode, Fast Slice--in conjunction with Azure should computations become too costly down the line--applies the default or user-specified thresholds to label an area of bone based on the user's input. This same process is extended in the auto mode to use the detected bone on one slice to identify adjacent bone on adjacent slices. The user can then export the resulting label as a fully-rendered 3D model.



Project Flow

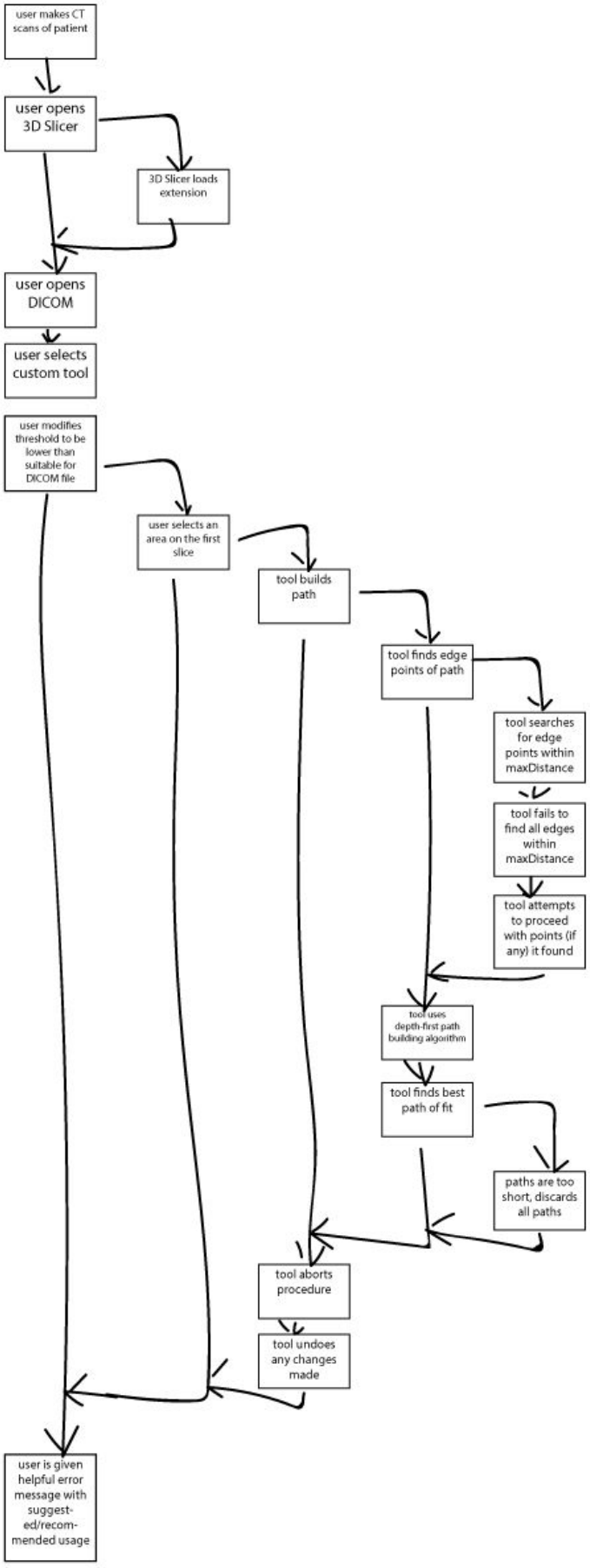
Expected scenario: The user wants to build a 3D model of a patient's lower limbs with which to design a prosthetic socket. The patient has already had a CT scan of the amputation that captures the remaining bones in the lower limbs. The user imports the file to 3D Slicer and loads our custom extension.
The user selects our custom tool and selects the bone on the first slice. The custom tool automatically takes the user's input to determine the shape and boundary of the bone on that slice and label it appropriately, and use the data from that slice to automatically detect and label bones on each subsequent slice, outputting a completed label map. The user then has 3D Slicer generate and export a model from the labels to use for designing a socket to fit it.

Description: Once the user has loaded the DICOM file and has made the first selection, the tool uses pre-set threshold values for typical bone densities to identify and label them. This process is broken up into four stages: identifying edge points, building paths from the edge points, selecting the largest path, and finally, labeling the voxels enclosed in the path. Once this is complete for the given slice, the tool automatically switches the next slice and uses data from the previous slice to find edge points for the next, and repeating this process for the entire DICOM file. Once the entire file has been

labeled, the user can then output the label as a model and export it into a modeling software for designing the socket.

Unexpected scenario: The user has loaded a DICOM file and our extension. The user goes to select our tool, but either sets the threshold level too low, or clicks somewhere way outside the desired area.
The tool attempts to proceed as usual, but upon being unable to find edges within a max distance, finding all resulting paths unsuitable, or hitting a max pixel count prematurely, throws an exception and aborts the labeling process. In doing so, any changes made to the file are undone (so as to prevent any previous work from being lost), and the user is given a helpful error message with suggestions on how to fix it.

Unexpected sequence diagram:



Description: Once the user has loaded the DICOM file and our extension, the user either attempts to manually set the threshold values from their pre-sets and mistakenly sets lower threshold too low, or uses the tool on the wrong part of the DICOM file. The tool attempts to proceed as normal, but ends up unable to find edge points within a built-in max_distance value from the starting point, which then leads to either no paths being built or the paths that are built being shorter than the minimum required path length. To prevent any strange labeling, the tool aborts the process before attempting to fill, and gives the user an error message with suggested correct usage.

Enumeration of Requirements:

SLICE-1	3DSlicer	non	α
The customer knows how to use 3DSlicer already, and wants our team to improve aspects of 3DSlicer, rather than create a whole new application. So we must create something that builds upon 3DSlicer.			
Customer			

CROSS-1	Cross-Platform Support	non	α
3DSlicer is a cross-platform program, thus we must take care to develop our tool to work on all supported platforms.			
3DSlicer Project			
This will also position our tool to be used by a wider audience.			

SMALL-1	Sparsity of DICOM Files	non	α
There is only a small set of DICOM files available to us -- and fewer labeled, which rules out using machine learning techniques to try and solve the problem. We must devise clever algorithms to solve marking the dense bone.			
Internal			

EXT-1	Extensibility	non	¥
For the customer's purpose, the extension only needs to work well for leg bones. After finding optimal settings for identifying typical leg bones, the extension could be expanded to accurately identify other bones or other anatomical features.			
Customer			

SET-1	Threshold Slider	func	α
Slider that sets the threshold settings that are then used by the bone finding algorithm.			
Customer			

SET-2	Image Range Selector	func	α
Two buttons, one to flag the bone in the start image and one to flag the end image bone.			
Customer			

SET-3	Max Pixel Selection	func	α
Max number of pixels that can be filled for an activation of the tool.			
Internal			
Mainly to be used for a failsafe			

TUTOR-1	Guided User Tutorial	func	Ω
Inclusion of a guided user tutorial built into the existing GUI to walk the user through the tool's use and function of the settings.			
Customer			

ICON-1	FastSlice Tool Icon	func	α
Icon that can be selected to use the tool on the images.			
Internal			

ERR-1	Error Feedback	func	β
Feedback from failed processes, and logging output			
Internal			
Will be more useful for us than user perhaps			

FAIL-1	Module, not System Failure	non	β
A failure in the tool should not crash 3DSlicer as a whole.			
Internal			
This should decide how we handle our errors			

LANG-1	Programming Language	non	α
We must develop all solutions in Python (2.7) and C++, as that is what Slicer is written in.			
3DSlicer Project			
3DSlicer is written only in C++ / Python so we don't have a choice.			

TOOL-1	Speed of Algorithm	non	α
Algorithm to identify dense bone structures must be quick enough to run in near instantaneous (to the user).			
Internal			
We're trying to speed up the process, if the algorithm is too slow, we could end up taking more time than manually selecting bones would.			

TOOL-2	No Outside Libraries	non	α
It's difficult to use outside libraries in our extension because they may not play nicely with 3DSlicer's customized code base. We must implement all the methods not offered in the 3DSlicer's code base we wish to use ourselves.			
3DSlicer Project			
We can, and should look at other libraries to see how some of the algorithms are implemented.			

TOOL-3	Identify Bone Boundary	func	β
A click anywhere on or in the bone should fill the full bone to the boundary.			
Customer			

EXT-1	3DSlicer Extension	non	α
The tool we create should be packaged as an extension. Having our tool as a custom build would be limiting, as the user might miss on new updates and features of the main project.			
Internal			

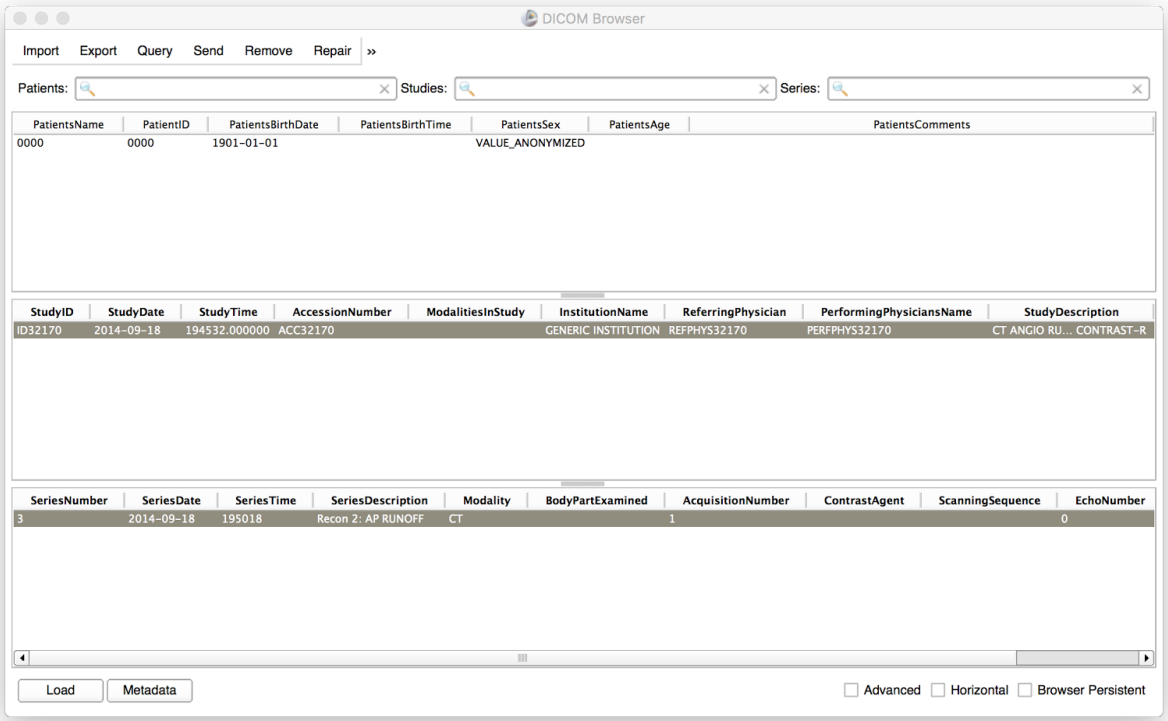
FILE-1	DICOM Files	non	α
The customer uses the DICOM file format for their medical images, so the project must be able to work on DICOM files.			
Customer			

TUN-1	Manual Fine Tuning	func	Ω
There should be a tool that allows the customer to fine tune the results of our tool with ease. Ideally the customer would click on the area that was / was not filled, and would adjust the threshold settings automatically to try and include that area.			
Internal			

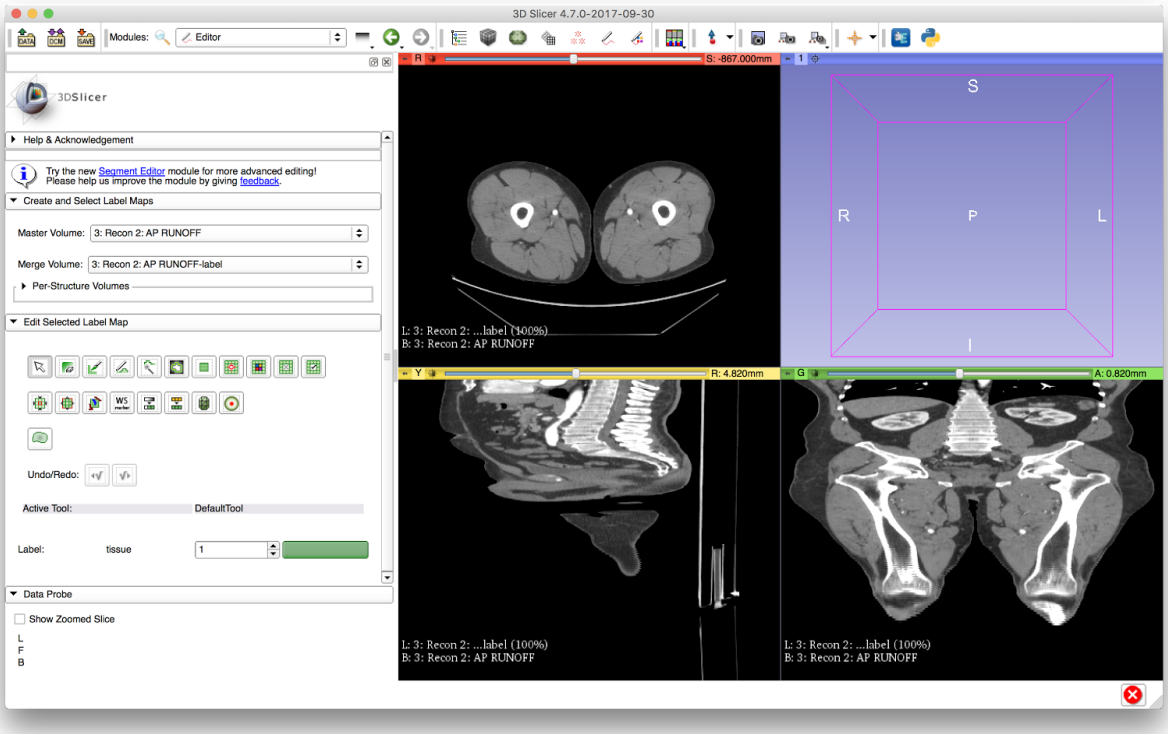
TUN-2	Automatic Fine Tuning	func	¥
A great extended functionality would be to automatically try different threshold settings, then find the one that best marks the bone.			
Internal			
This is probably pretty difficult to achieve, but we should still strive for it.			

AZU-1	Azure Transfer	func	¥
Create an easy method to (optionally) transfer the current process of the tool to Azure for quicker computation.			
Internal			
This would be definitely speed up any computation costs, and would allow us to explore better, but more computationally intensive algorithms.			

Prototype User Interface (UI)



This is the window after you load the DICOM file into the application. This DICOM Browser shows entries that contain the 2D slices. In our use case these slices will contain scans of a patient's legs. After loading this file the user chooses the entry they wish to load into 3DSlicer and then clicks load.



This is the main 3DSlicer window. The DICOM file has already been loaded and the patient entry selected in the previous DICOM Browser window. Our tool is selectable in the third (bottom) row of the icons on the left. After selecting the tool, the user clicks inside of the top left image within the area of bone to automatically select the bone in the slice. This works whether the user clicks in the white area or the black area. After selecting a piece of bone, the user can drag the slider to switch to a different slice.

Environment Description

Development Environment: *What platform(s) will be used for the development effort?*

Thus far, a combination of Windows and Linux has been used for the development effort.

Target Environment: *What platform is anticipated as the target environment?*

Given the cross-platform nature of 3DSlicer, we don't need to target any specific platform. The Qt framework that 3DSlicer is written in allows our project to be used with equal success on Mac, Windows, and Linux.

Software Packages and Tools: *What software packages and tools will be used, including any code from previous efforts or courses?*

Programming languages:

- Python (for creating an extension)
- C++ (for making use of 3DSlicer internal libraries)

Tools:

- Microsoft Azure (for creating a replicable build environment, as well as use of cloud services)
- Visual Studio 2013 (for building solution files)
- CMake (for generating build files)

Code from previous efforts has been utilized in parts of our project. 3DSlicer is open-source; as such, we are building off of the work of others. Other aspects of this project, like our extension, are completely greenfield.

References

None.