PARSE INPUT

In this section, all the inputs that the programmed is invoked with are parsed. This section reads in any file name given, assuming its a DICOM file, unless the input has the format "-makeMasks", "-outputImages", “-nominalDensity”, or "tag=string", then those appropriate options are enabled on instead. If a file fails to be read in as DICOM, the code terminates.

tag=string changes the lookup of the priority and TAS files from Default to string, ie,

Default\_priority.txt -> string\_priority.txt

Default\_all\_TAS.txt -> string\_all\_TAS.txt

Default\_mediaDensity.txt -> string\_mediaDensity.txt

makeMasks outputs begsphant files for each structure with uniform density and two possible media, NONTARGET and TARGET, used to identify structures later for post-processing apps such as 3ddose\_tools.

outputImages outputs PNGs of each slice of the output phantoms for both media and density, as well as outlines of structures over the media images. It is very time intensive, so best to be used to check TAS and registration.

nominalDensity uses the file Default\_mediaDensity.txt (where the file lookup would change for the appropriate tag) to assign densities to all media.

INITIAL DICOM INFORMATION FETCHING

In this section, we define all the variables that are used to store the DICOM data. Then, we go through every DICOM file to check their tag[0008,0008] field to ensure that it contains AXIAL, as a quick and easy way to see if we have an image file or not. Then we separate the DICOM files that don't have the field into dicomExtra, and now the QVector dicom, holds all the ct slice info.

As it is not safe to assume that the CT data is given in proper ascending order, all the dicom data is sorted by the z value extracted as the code was parsed. The z value is given by the tag[0020,1041] which provides slice height. A classic, non-recursive merge sort is used to sort them.

FETCH CT DATA FROM DICOM FILES

In this section, we extract all the data we require from the dicom data. Here are the tags and the relevant data we fetch from each of them:

|  |  |  |
| --- | --- | --- |
| **Tag** | **VR** | **Representation** |
| 0028,0030 | DS | Pixel Spacing |
| 0018,0050 | DS | Slice Thickness |
| 0020,0032 | DS | Image Position |
| 0028,0010 | US | Rows |
| 0028,0011 | US | Columns |
| 0028,1053 | DS | Rescale Slope Value |
| 0028,1052 | DS | Rescale Intercept Value |
| 7fe0,0010 | OB | HU Pixel Data |

Careful in that most DICOM data is in mm, and we are converting to cm for the egsphant. The HU data starts at the most -x and +y voxel and iterates first by x, and then by y.

The number of voxels in z is then therefore simply the number of files we have iterated through. It is important to note that the z index is now the most top-level iterator for all the above arrays.

FETCH RS FILE STRUCTURE DATA

In this section, we look for structure data in the dicomExtra array. Like above, here are the tags and the relevant data we fetch from each of them:

|  |  |  |
| --- | --- | --- |
| **Tag** | **VR** | **Representation** |
| 3006,0020 | SQ | Structure Set Sequence |
| →3006,0026 | LO | ROI Name |
| →3006,0022 | IS | ROI Number |
| 3006,0039 | SQ | ROI Contour Sequence |
| →3006,0040 | SQ | Contour Sequence |
| →→3006,0050 | DS | Contour Data |
| →3006,0084 | IS | Referenced ROI Number |

→nested sequence, →→ double-nested sequence

As the structure data that we are interested in (name and contours) are in seperate tag groups, we need to extract the structure number twice, to match structure name to the contour data.

This section involves reading SQ at one and two layers deep, so the parseSequence function is invoked several times to parse the appropriate data.

The structures (which are many sets of [x,y,z] positions) into an array of polygons and an array of z positions, for each structure.

GENERATE EGSPHANT AND HU CONVERSION MAPS/THRESHOLDS

In this section, we fetch the density curve, the priorities, and all of the TAS files associated with the struct names. Though there is a lot of code, it is all very simple and similar.

The latter half of this section defines an empty egsphant of the correct dimensions based on the previously retrieved CT data. It also adds all the different media in the given TAS files to the egsphant.

The z dimensions are taken as the average of the z bounds of two neighbouring slices (as z values don't always line up).

CONVERTING HU TO APPROPRIATE DENSITY AND MEDIUM

In this section, we define a list of bounding rectangles for each structure, which will be used to determine whether we need to check against structures when going through different slices and columns for efficiency.

If masks are requested, they are generated based on the egsphant dimensions.

Here is the main loop, that iterates through z, y, and x HU data. For every new z value, a list of structures that could occur on this slice is made. Then, for each new y, the list is checked again to see if a struct occurs in this column.

Then, the HU data for the [i,j,k] voxel is used to assign media and density to the [i,ny-1-j,k] voxel of the egsphant. This inverts the y-axis, so that it counts up from -y rather than from +y.

Finally, the egsphant file and masks if requested are output.

OUTPUT IMAGES

In this section, we simply output images using the prebuilt image functions in the egsphant class. There is also an additional step in drawing the structure points overtop of the media images.