Readme for the assignment Image Processing library

Iraklis Spyrou, 9285091

1st exercise review

In the first exercise we had to implement 4 IO functions in order to read and write 2 specific data types of medical images, .pip and .mhd. The 4 functions were basically 2. A read function for .pip and .mhd files and a write function for .pip and .mhd files.

.pip:

In the read function for .pip we should first read the header and then the data (pixel values). In order to read the header, we first read the unsigned character short with an ifstream, the read 5 integers (which are the dimensions) and then we read all the data. The read function returns a vector with the data of the pip image.

In the write function for .pip we should first write with an ofstream the unsigned character short, then the 5 integers (dimensions) and then the data.

.mhd:

In the read function for .mhd we should again first read the header and then the data. The difference for mhd files is that there are two files, the header (.MHD) and the data (.RAW). So, we read the .MHD (header) as a text file. The last element of the header is the name of the .RAW, so when we read the name of the file of the data, we open a new ifstream and read the data as binary. The read function again returns a vector of the data stored.

In the write function we hard copy the header in a .MHD output file and then write all the data in a .RAW output file.

In the first exercise we also had to handle errors for some corrupting files for both .pip and .mhd images.

2nd exercise review

In the second exercise we should implement the following classes in header files:

1) ImageIOBase class: a base class that includes the read and write function as pure virtuals in order for the functions to be overridden in the derived classes and for pure in order to make the base class abstract. ImageIOBase also includes constructors and a destructor.
2) PipImageIO class: is a derived class of ImageIOBase. In this class we override the read and write functions. We also use the constructor of the ImageIOBase class and make it public. We implemented the 2 functions outside of the class, in the same way as we did in exercise 1. The only difference is that now the read function doesn't take an argument as we pass the filename inside the function scope. The filename is derived from the ImageIOBase class.
3) MhdImageIO class: is a derived class of ImageIObase. In this class we override again the read and write functions and make the constructor of ImageIOBase

class public. We again implemented the functions outside of the class, again as we did in exercise 1.

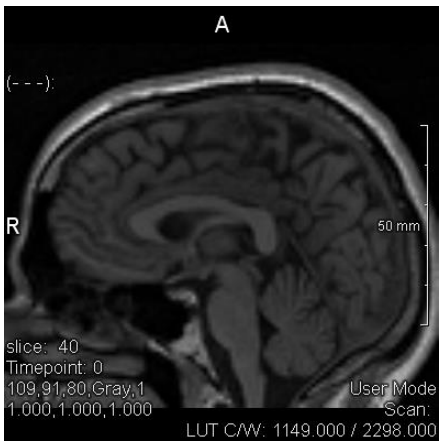
4) ImageIOFactory class: is a class that has only one function as a member. This function is used to return a pointer in an object of either PipImageIO class or MhdImageIO class, depending on the extension that the filename has (pip or mhd). This is done in order for the program to choose automatically which read and write function should use.
5) ImageFilter class: is a base class where we implement every function needed in order to implement later filter classes, which will be derived classes of the ImageFilter.
6) ThresholdFilter class: a class that implements a threshold filter. The threshold filter returns an output image which has only binary values depending on the threshold that we have defined each time.
7) ConvolutionFilter class: a convolution filter that returns a blurred output image.
8) StatisticsFilter class: a filter that returns statistics about an image. The statistics that this filter returns are the minimum, maximum and mean values of the image.
9) MaskFilter class: a filter that takes 2 input images, one input original .pip or .mhd image and one mask image. The mask image could be the output of the threshold filter. The mask filter combines the 2 images into 1 in such a way that set the input image voxels to 0 where the mask is 0 and lets all the other voxels unchanged.
10) NegativeFilter class: a filter that takes 1 input image and multiplies all the voxel values with -1. In this way, the voxels that were white, now become black and vice versa.
11) GammaFilter class: a filter that adjusts the brightness in the midtones of the image, without changing the black and white voxels.

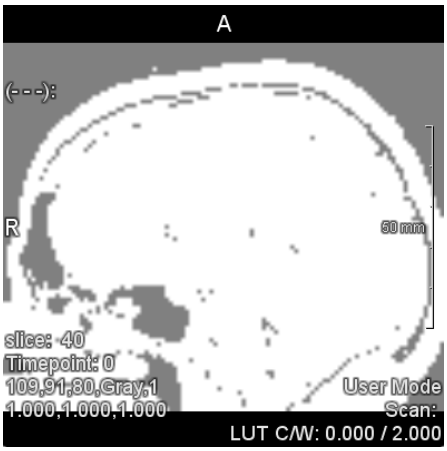
3rd exercise review

In the third exercise we had to make our own Image container. The idea of this exercise was to create a new data type that can hold images, so that we don't have to use vectors and hard copy the dimensions in the write functions. The Image class, contains various members such as constructors, operators, iterators, destructor, functions and 2 private data members, the dimensions as an array of 5 ints and the data of the image. After finishing with the implementation of the Image class, we then changed everything in the previous classes to use Image data type and not vectors. Now the write function takes only one argument, the input image which is going to be written to an output file.

Some image examples, that we visualized with the Mevislab, follow.

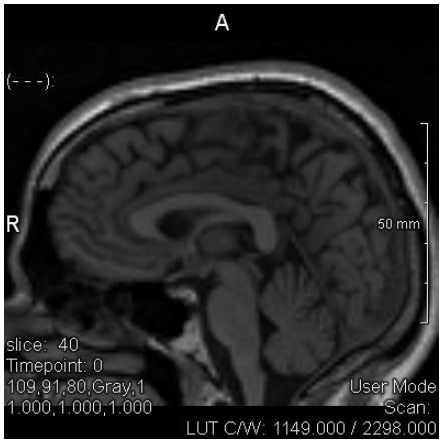
brain.pip (left) vs brain_out_threshold.pip (right)

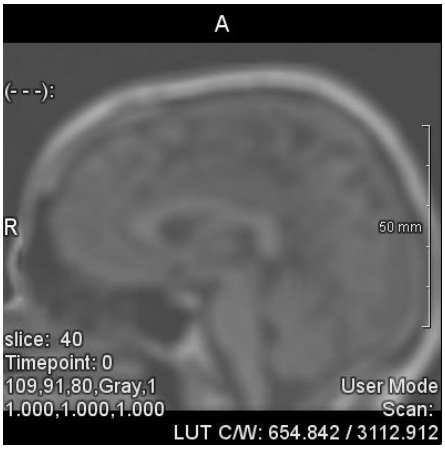




The threshold in this example was 90.

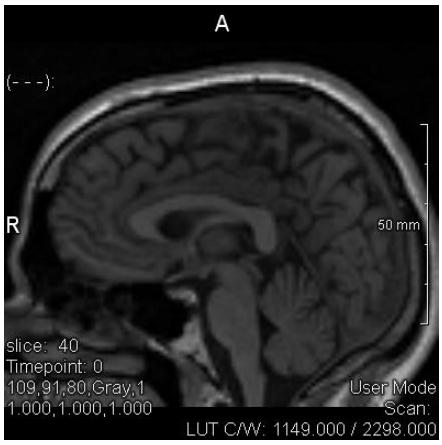
brain.pip (left) vs brain_out_conv.pip (right)

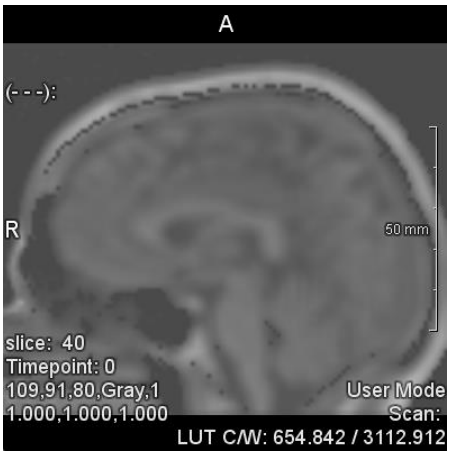




The kernel for this convolution filter was 4.

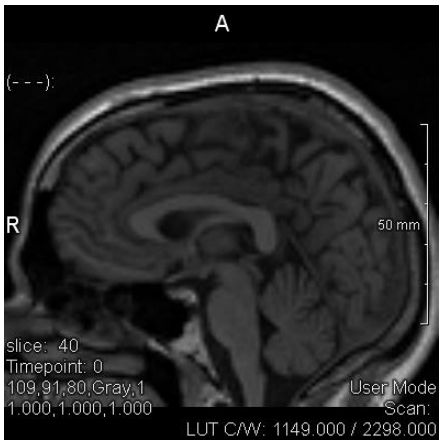
brain.pip (left) vs brain_out_mask.pip (right)

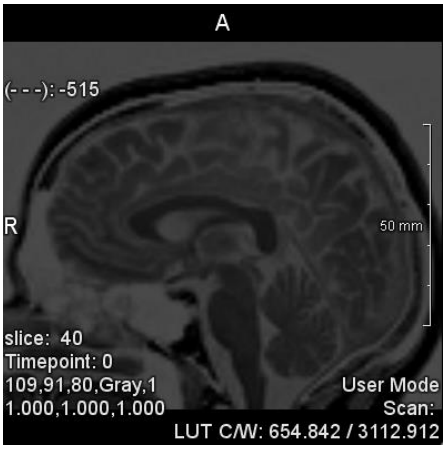




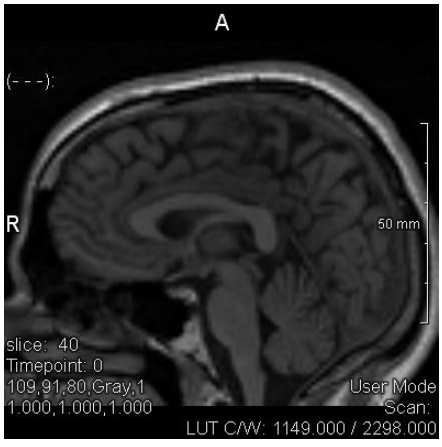
The mask used for this image was the output image of the threshold filter. The input image was brain.pip.

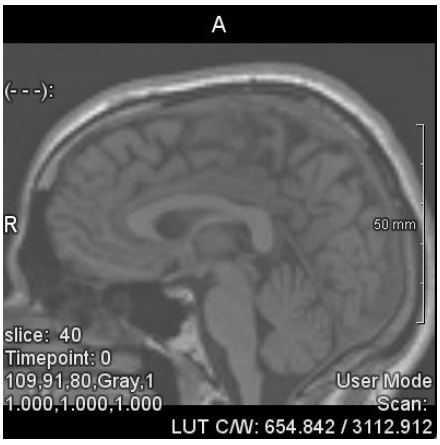
brain.pip (left) vs brain_out_neg.pip (right)





brain.pip (left) vs brain_out_gamma.pip (right)





The gamma value used for this example was 1.

P.s.: the source code that I will hand in contains only output examples for .pip images. In order to create .mhd files we should change .pip to .mhd in the ImageIOFactory::getIO(brain_out.pip)