Assignment: point cloud

1 Introduction

During this assignment, you will extract a point cloud from your collection of DICOM files. This is a preliminary step to perform reconstruction[1] with the library `CGAL`.

In order to extract useful data, you will start by performing manual thresholding to partition the voxels in different sets. Then, you will remove points which have the same value as their neighbors to keep only the contour of objects. Finally, you will export them in a commonly used format.

2 Replacing setWindow

In the previous assignments, you relied on the function `setWindow` of the class `DicomImage` to convert from Hounsfield units to pixel values. It is preferable to have access to the raw data in order to perform the segmentation task because it might contain more information.

The method `setNoVoiTransform` of `DicomImage` will allow you to make sure you are accessing the raw data. In order to cover the full range of the data, `unsigned char` will not be sufficient. If your class storing the volumic data is using an 8-bit representation, you will need to move to a 16-bit representation (e.g. `uint16_t` or `int16_t` in `cstdint` module).

Be careful when importing your data with a 16 bits representation, you might need to use an offset in order to get values in the same range as the window you are choosing through sliders. It is possible that the range you obtain is significantly different and you might need to use an offset when importing the content.

In order to keep the 3d view working properly, you can implement manually a conversion that replaces the `setWindow` and `getOutputData` based on equation (1). In order to significantly save computation time, make sure that `getOutputData` is not called at every change of the window.

$$c = \begin{cases} 0 & v < w_{\min} \\ 1 & v > w_{\max} \\ \frac{v - w_{\min}}{w_{\max} - w_{\min}} & \text{otherwise} \end{cases} \tag{1}$$

3 Binary segmentation

The next step is to use the values of the sliders `window_width` and `window_height` to impact the OpenGL visualization. For every voxel with a value of w, you can compute if it belongs to partition 1 (its value is inside the window range) or 0 (its value is outside the window range) based on equation (2). Change your code to make sure you only display the points that belong to partition 1.

$$c = \begin{cases} 1 & w_{\min} \leq v \leq w_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

[1] `https://doc.cgal.org/latest/Manual/tuto_reconstruction.html#TutorialsReconstruction_overview`

4 Obtaining contours

The representation obtained after the binary thresholding contains many points which carry very little information because all of their neighbors have the same value. In order to keep only the contours of the partition of voxels you must filter out the points that belong to partition 1 if all of their neighbors also belong to partition 1. The notion of neighbors in this case can include 6-connectivity (face neighbors), 18-connectivity (face and edge neighbors) or 26-connectivity (face, edge and corner neighbors)[2]. Using this filter should allow to greatly reduce the number of points drawn in your `QOpenGLWidget`, thus leading to better performances when moving the camera while using complex models. It should be possible to enable or disable the use of this feature.

5 Exporting point clouds

Implement a `Save to XYZ` functionality which allows to export the point cloud you are currently displaying to a `XYZ` file[3]. It is a pretty simple format to describe point clouds in which each line has the following format `X.XX Y.YY Z.ZZ`, note that you do not need to export the normals. You can test the validity of the file produced by importing it with `meshlab`.

6 Multiple partitions by manual thresholding

When using Hounsfield units, different ranges of values for voxels corresponds to different kind of content[4]. It is valuable to be able to partition the voxels in more than 2 different sets for visualization purposes.

Add the option to load a json dictionary containing a named list of ranges with colors associated, such as:

```
{
  "bone" : {
    "range" : [200, 1000],
    "color" : [0, 0, 0]
  },
  "lipidic_structure" : {
    "range" : [-150, -10],
    "color" : [1.0, 0, 0]
  }
}
```

Implement a different mode for the openGL interface where `window_center` and `window_width` are ignored and where color is attributed based on the provided dictionary. Note the following elements:

- If the value of a voxel does not lie in the range of any of the elements in the dictionary, it should not be shown.
- If two ranges are overlapping, an error message should be shown and the dictionary should not be loaded.
- When obtaining contours in this mode (removing inner points), the partition should be based on the values in the dictionary.

[2] `https://en.wikipedia.org/wiki/Pixel_connectivity#3-dimensional`
[3] `https://doc.cgal.org/latest/Stream_support/IOStreamSupportedFileFormats.html#IOStreamXYZ`
[4] See `https://dept-info.labri.fr/~desbarat/IMF/IMF_RX.pdf` (p19)

7 A more coherent display of options (optional)

Some of the options you have implemented are not compatible or should not be displayed depending on others. A simple example is the `hide_empty_points` option which removes from the points to be drawn all the points with a display color $c \leq 0$, it only makes sense if segmentation is not used.

Make sure that it is only possible to interact with widgets relevant to the current set of parameters. You can achieve this using `setVisibility` or `setEnable`.