LAB 2: SURFACE RENDERING

ANDRÉS MARTÍNEZ MORA

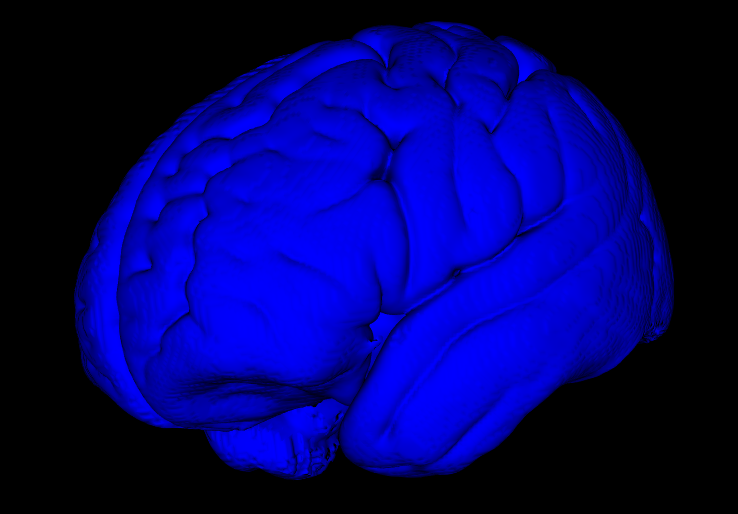
CM2004: VISUALIZATION IN MEDICAL IMAGING

EXERCISE 1

In this exercise we are asked to adapt our code with renderers so to include a typical surface rendering class. The class is called from the *multi\_renderer.py* code, as in the last exercise of Lab 1, so to provide multiple outputs.

In the mentioned class, instead of outputting a port from the given brain image, we will output the image itself. This will be used as input to a cast filter, whose output type is set to *Unsigned Short*, as otherwise there can be errors in the process. The cast filter output is inputted to a Marching Cubes algorithm, including Normal vectors and Gradients computation. The result from the Marching Cubes Algorithm is fed into a poly data mapper, and this mapper is inputted into a VTK actor. The actor is then inputted to a camera and render window with the same parameters as the ones in Lab 1, as we are working with the same image.

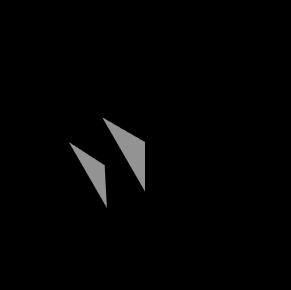
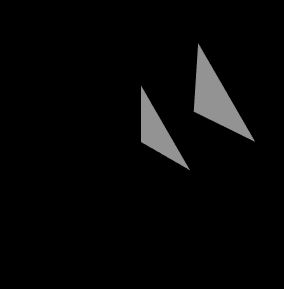
The surface rendering with the built-in Marching Cubes algorithm from VTK results as follows, without the need of downsampling to save memory or reduce computational time:



EXERCISE 2

In this exercise we are asked to visualize our own sources, triangles in this case, as a basis for Exercise 3. We first define a set of VTK points to build 2 triangles, then we build the triangles with those points and we save all the triangles into a VTK cell. Then, these data is inputted to a poly data mapper. After the mapper is built with the given data, it is inputted to a camera and a render window. Camera parameters have to be modified from the ones of the brain image into lower values for the position of the camera, as otherwise the resulting triangles would not be visible.

We modify the point coordinates for the triangle vertices, so to explore different cases, leaving the camera position at (5,5,5) with the focus at (0,0,0).



In the left case, one triangle has as coordinates (1,0,0), (0,0,0) and (0,1,0), while the other triangle has as coordinates (2,1,0), (1,1,0) and (1,2,0).

In the center case, one triangle has as coordinates (-1,0,0), (0,0,0) and (0,-1,0), while the other triangle has as coordinates (-2,-1,0), (-1,-1,0) and (-1,-2,0). It is the same case as in the left, but with opposite sign, that is why the triangles appear inverted with respect to the left case.

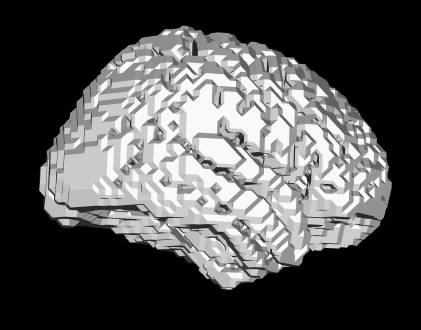
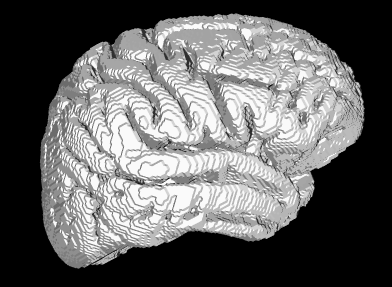
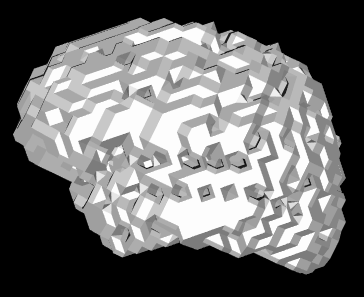
In the right case, one triangle has as coordinates (2,0,0), (0,0,0) and (0,2,0), while the other triangle has as coordinates (4,2,0), (2,2,0) and (2,4,0). These triangles are as twice as big as the ones in the left case, so that is why they do not totally fit into the visualization and they appear in the same orientation as in the left case.

EXERCISE 3

The goal of this exercise is to program the implementation of Marching Cubes and incorporate it to a similar pipeline as the one in Exercise 1. Initially, the volume is converted from VTK to numpy and it is downsampled with a certain scheme. A certain iso-level for defining the surface desired is taken (140 in this case). Next, every 2x2x2 neighborhood of the 3D data is inspected to see if it contains an iso-surface boundary. If so, it computes the triangle (or set of triangles) that is related to that boundary with respect to the involved coordinates. Initially, the triangles are expressed with respect to the local neighborhood and later they are converted to the global coordinates of the volume dataset. After all local 2x2x2 neighborhoods are inspected, the surface is obtained.

To visualize the computed surface, the same pipeline as in Exercise 1 is carried out. The mapper is connected to a render window with a camera with the same parameters as in Exercise 1, as we are still working with the brain image.

Several results are extracted with changing resolutions (from less to more resolution). As more resolution is desired, the computational and memory costs are increased and it takes more time to get the resulting visualization.



In the left case, the volume is downsampled x5, in the center case x3 and in the right case x1 (no downsampling). The right volume is more realistic, having an increased resolution, but it also took around 20s to compute this result in the right, while it just took 2 or 3 seconds to compute the result in the left. It can be seen with this that the built-in functions from VTK are much more optimized, providing quite good surface renderings with not so much memory and computational costs and in less time.

The visualizations could include some look-up-table (LUT) and some lighting profile, so to be more realistic. This would be the next step to follow in the pipeline.