1. Introduction

This project uses Python and the Visualization Toolkit (VTK) for medical data visualization, specifically leveraging the *SPL Head and Neck Atlas*. The atlas, chosen for its detailed and high-resolution CT scans, provides a rich resource for visualizing complex anatomical structures. The goal is to develop Python scripts that can effectively render both 2D slices and 3D models from this data. This approach aims to enhance understanding and analysis of medical imaging, benefiting education, research, and clinical practice. The project emphasizes practical application, algorithmic learning, and visualization techniques in medical imaging.

2. Dataset

The chosen input data for this visualization project is the *SPL Head. The Neck Atlas* is a CT-based atlas of the skull, mandible, upper ribs, spine, and associated neck muscles. The atlas includes Cartilage, blood vessels, and glands. The data set consists of a reduced resolution (256x256) of the MANIX data set from the OSIRIX data sets, detailed label maps, and three-dimensional models of the labelled anatomical structures. The data was created with the association of the SPL: Surgical Planning Laboratory, Department of Radiology, Brigham and Women's Hospital, Harvard Medical School, Boston, MA, USA by Marianna Jakab and Ron Kikinis.

3. Preliminary information

a) The dataset could be downloaded by following the link: https://www.openanatomy.org/ [last access: 15.12.2023r.] or manually entering the Open Atlas Project page.

b) The project contains 5 functionalities with each separated into a separate program. Therefore, you will need a set of the following files:

- Only_Slices.py

- Colour_Slices.py

- 3D_head.py

- 3D_Full_w_slices

- 3D_From_Slices.

Fig. 1. The contents of the folder containing the project

.venv	14.12.2023 10:18	Folder plików	
head-neck-2016-09	14.12.2023 21:58	Folder plików	
3D_From_Slices	15.12.2023 00:41	Python Source File	19 KB
3D_Full_w_slices	15.12.2023 00:53	Python Source File	7 KB
3D_head	15.12.2023 01:09	Python Source File	16 KB
Colour_Slices	15.12.2023 18:39	Python Source File	18 KB
Only_Slices	15.12.2023 01:09	Python Source File	4 KB

c) For the process of running the programs successfully, it is necessary that they are in the same folder as the downloaded and unzipped SPL Head and Neck Atlas, as shown in the picture above (*Fig. 1.*).

d) The programs were tested on Windows 11 computer, in the following configuration:

- Python 3.12.1, VTK 9.3.0, VSC 1.85.1 (user setup) with configurated .venv (Python Virtual Environment)

Important note: For correct data loading, you must have a VTK version other than 9.0.x (distributions from this particular line contain a glitch in handling .nrrd files using the vtkNrrdReader class; also, older version than 9.3.x could not work with newer version of Python).

4. Script overview – methods, VTK class libraries

The development used scalar colour mapping, also known as the *Lookup Table* method, which involves assigning an index to an n-element array of a given scalar value (containing n component colours, the so-called colour palette). An extension of this functionality is contouring, the algorithm used to generate the 3D boundary. Above that, the following classes (and their methods) from the VTK library used in the project are shown in the *Additament 1.*

5. Execution and usage

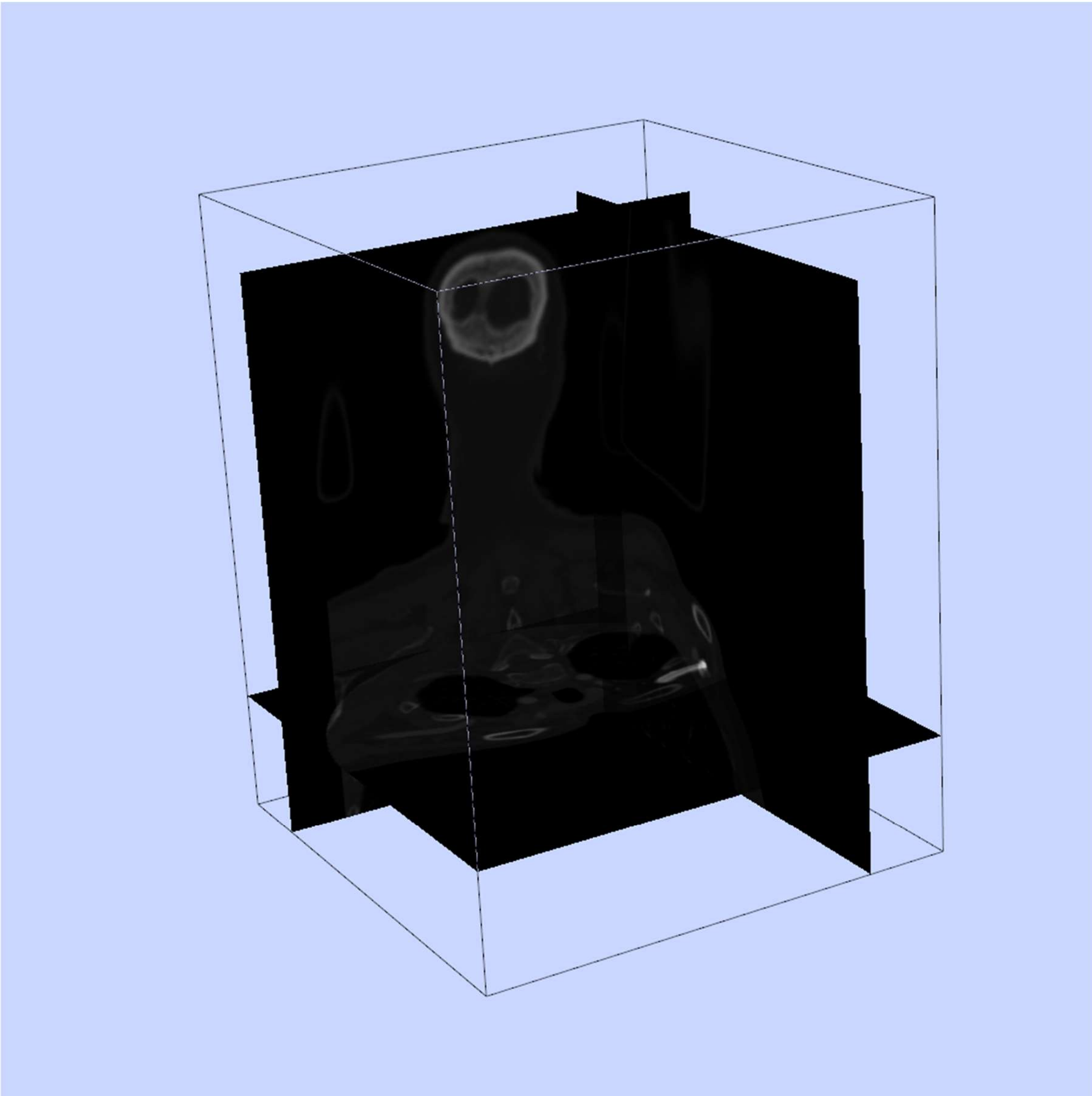
a) Only_Slices.py

Once started, the program will ask the user to enter the section numbers for each of the three main planes he would like to display from the keyboard. Based on the received numbers scans, it will display the raw data in three orthogonal planes with monochromatic colour mapping, implemented using the *Lookup Table* method. The data, as before, is taken from the *Osirix-Manix-255-res.nrrd* file. An example of the program's result after loading the same values (slice 43) is placed below (*Fig. 3.*)

Fig. 2. Program queries for slice numbers with sample values

```
Input slice number for axial plane from range 0-206: 43
Input slice number for sagittal plane from range 0-255: 43
Input slice number for coronal plane from range 0-255: 43
```

Fig. 3. Execution of the Only_Slices.py code.



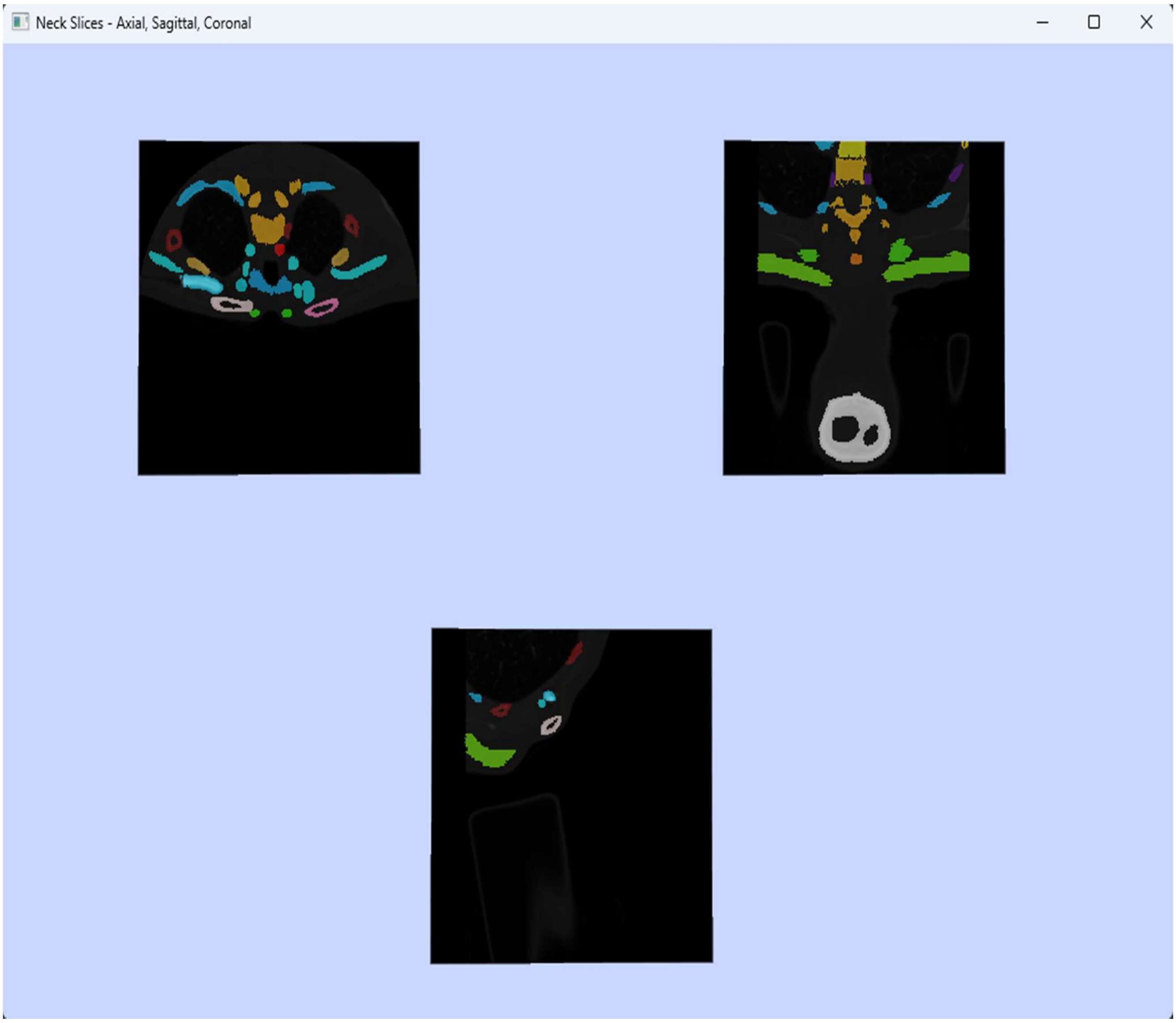
b) Colour_Slices.py

This program colours the raw data with the segmentation results. It first retrieves the raw data from the *Osirix-Manix-255-res.nrrd* file, asking the user to provide section numbers for each of the three planes (axial, transverse and sagittal), then displayed in grayscale. Then, the segmentation results, located in the *HN-Atlas-labels.nrrd* file, are read and coloured using the *Lookup Table* algorithm based on a defined colour palette and applied to the raw data. An example of the program's operation is illustrated below (*Fig. 5.*)

Fig. 4. Program queries for slice numbers with sample values

```
Input slice number for axial plane from range 0-206: 43
Input slice number for sagittal plane from range 0-255: 43
Input slice number for coronal plane from range 0-255: 43
```

Fig. 5. Program queries for slice numbers with sample values (color segmentation result on raw data)



c) 3D_From_Slices.py

The program determines 3D solids representing the anatomical structures of the ear based on the segmentation results contained in the *HN-Atlas-labels. nrrd* file. Since the contouring algorithm yields somewhat "*angular*" solids, a *Gaussian filter* was used, thus levelling out the jagged effect. Again, based on the *Lookup Table* algorithm and the predefined colour palette, coloured the structures with the appropriate colours. The result of the program is illustrated below (*Fig. 8.*). We need to label and choose each tissue manually.

Fig. 6. Some tissue configuration

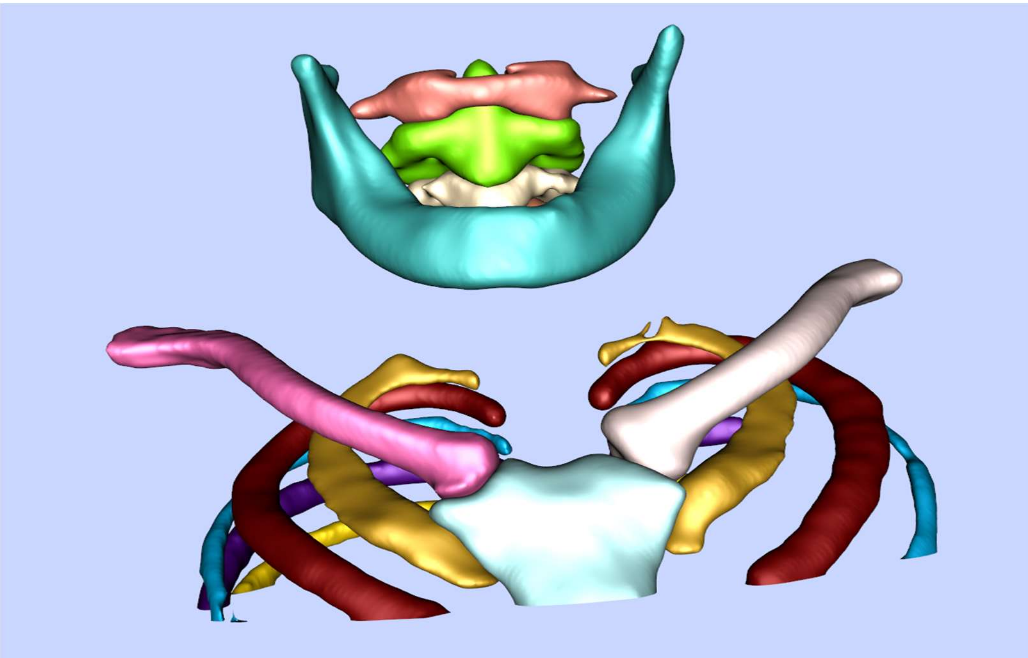
```
def hyoid():
    p = head()
    p['NAME'] = 'Hyoid'
    p['TISSUE'] = 9
    p['VALUE'] = 27.5
    return p

def atlas():
    p = head()
    p['NAME'] = 'Atlas'
    p['TISSUE'] = 11
    p['VALUE'] = 90
    p['GAUSSIAN_STANDARD_DEVIATION'] = [1, 1, 1]
    return p
```

Fig. 7. Program shows the chosen tissue number and its given label

```
Tissue:      Hyoid, label:  9
Tissue:      Atlas, label: 11
Tissue:       Axis, label: 12
Tissue: Cervical3, label: 13
Tissue: Cervical4, label: 14
Tissue:   Mandible, label: 25
Tissue: Right_Clavicle, label: 26
Tissue: Left_Clavicle, label: 27
Tissue:    Sternum, label: 28
Tissue:       Rib1, label: 31
Tissue:       Rib2, label: 32
Tissue:       Rib3, label: 33
Tissue:       Rib4, label: 34
Tissue:       Rib5, label: 35
```

Fig. 8. Program queries for slice numbers with sample values



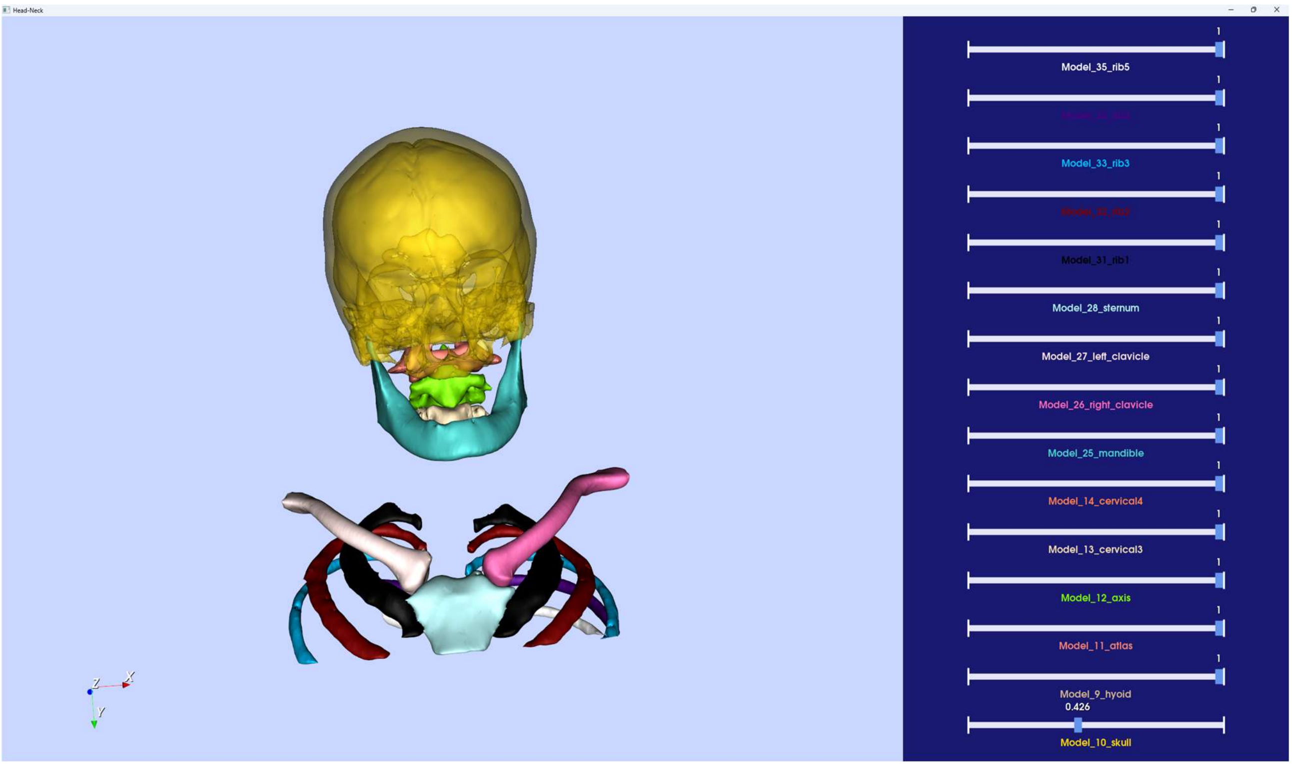
d) 3D_head.py

This simple VTK–based program allows users to interact with the displayed tissues by operating the sliders. They are responsible for changing the transparency of 14 anatomical structures of the ear. Each has the name of the structure it is responsible for adjusting at the bottom. The labels of the sliders have also been coloured according to the colour palette defined for the *Lookup Table* algorithm, which allows the program to operate intuitively and quickly. The data for the program are taken from VTK files located in the model's subfolder containing ready-made segmentations. An example of the program's operation is shown below (*Fig. 11.*).

Fig. 9. Program displays in terminal currently used tissues

```
Using the following tissues:
Model_10_skull, label: 10
Model_9_hyoid, label:  9
Model_11_atlas, label: 11
Model_12_axis, label: 12
Model_13_cervical3, label: 13
Model_14_cervical4, label: 14
Model_25_mandible, label: 25
Model_26_right_clavicle, label: 26
Model_27_left_clavicle, label: 27
Model_28_sternum, label: 28
Model_31_rib1, label: 31
Model_32_rib2, label: 32
Model_33_rib3, label: 33
Model_34_rib4, label: 34
Model_35_rib5, label: 35
```

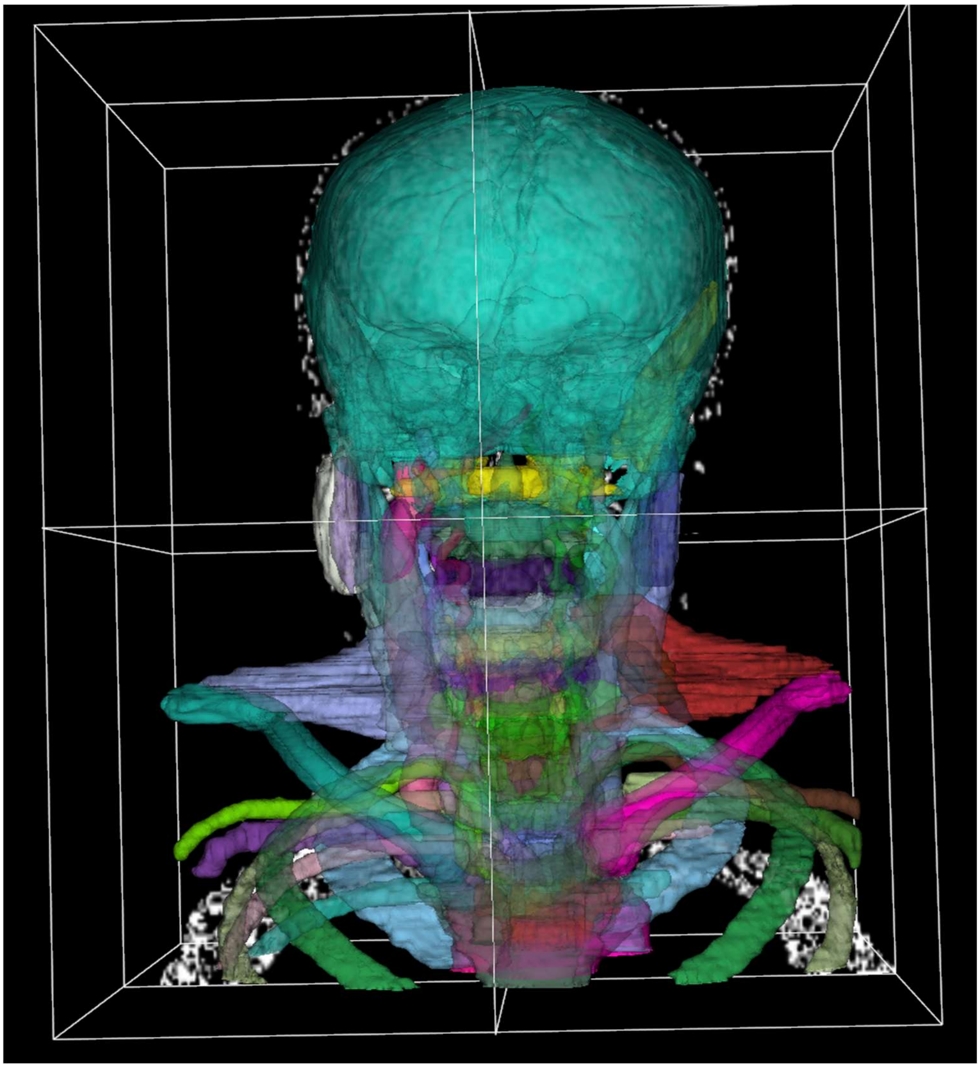
Fig. 10. Working tissue viewer program



e) 3D_Full_w_slices.py

This program allowed to see the entire model with all structures colored. It also applied planes with CT slices (*but for some reason, after coloring all the structures, they deteriorated significantly*).

Fig. 11. Program showing all structures colored



6. Results

In essence, the VTK library allows the development of a program for visualizing various anatomical structures quite simply and efficiently. It works not only with the Python language, in which the above project was performed but also with C++ (in combination with, for example, Qt) or Java, among others—successfully loaded data from the selected *SPL Head and Neck Atlas*, which was confirmed by displaying its outline. In addition, it was possible to develop a display of the data in three orthogonal planes with monochromatic colour mapping based on user-entered section numbers and to apply colour segmentation results. Using these segmentation results, a 3D solid reconstruction of individual structures was also realized and then coloured and treated with a Gauss filter, levelling out the jagged effect. Finally, the possibility of user interaction with the program was added using sliders that support changing the transparency of the selected structure.

7. Conclusion

In conclusion, this project highlights the potent synergy between Python programming and the Visualization Toolkit (VTK) in medical imaging. By harnessing the detailed datasets from the *SPL Head and Neck Atlas*, the developed scripts demonstrate the capability to transform complex medical data into insightful visual representations and underscore the importance of visualization in enhancing our understanding of anatomical structures. This endeavour not only contributes to the fields of medical education and research but also opens avenues for future advancements in medical imaging technology. The challenges faced and overcome during this project serve as valuable learning experiences, paving the way for further refinements and innovations in applying VTK in medical data visualization.

8. Bibliography [Last access: 14/15.12.2023r.]

[1] https://www.openanatomy.org/

[2] https://examples.vtk.org/site/VTKBook/12Chapter12/

[3] https://vtk.org/

[4] https://examples.vtk.org/site/Python/Visualization/FrogBrain/

[5] https://examples.vtk.org/site/Cxx/Widgets/Slider/

[6] https://youtu.be/QDJgbSQnhjc?si=KYkSGPnbThrB8gsp

[7] https://youtu.be/NA8_Yi_q7X4?si=cmN0GGevDFCH1Az-

[8] https://stackoverflow.com/questions/66834030/how-to-use-vtk-python-to-visualize-a-3d-ct-scan

[9] https://www.kaggle.com/code/wrrosa/advanced-dicom-ct-3d-visualizations-with-vtk

Additament 1.

a) vtkNamedColors - sets colors

b) vtkRenderer - creates an object responsible for rendering the image

- SetBackground - set the background (e.g., colour)

- AddActor - add an object for rendering

- SetActiveCamera - set the camera to "follow" the renderer

- ResetCamera, ResetCameraClippingRange

c) vtkRenderWindow - creates a rendering window

- AddRenderer - add a renderer to the render window

- SetSize - set the size of the program window (in pixels)

- SetWindowName - name of the program window

- Render - generate the image

d) vtkRenderWindowInteractor - is responsible for user interaction

- SetRenderWindow - sets the view for the given window

e) vtkNrrdReader - class responsible for proper loading of data from .nrrd files

- SetFileName - setting the name of the file from which data is to be loaded

- Update - update the algorithm after receiving a request to the output port

f) vtkOutlineFilter - outline of data, the so-called "cage" or cube

- SetInputConnection - setting the input connection

g) vtkPolyDataMapper - mapping data for later display

h) tractor - actor, i.e., the object that is displayed

- SetMapper - plugging in a given mapper

- GetProperty - "exposing" the given actor property

i) vtkCamera - the camera, responsible for the view

- SetViewUp - setting the view "from above"

- SetPosition - setting the position of the camera

- SetFocalPoint - setting the focal point

- ComputeViewPlaneNormal - view in the normal plane

- Azimuth - setting azimuth

- Elevation - setting the elevation of the view

- Roll - rotation of the camera

j) vtkImageConstantPad - margin

- SetOutputWholeExtent - set the extent of the output data

k) vtkPlaneSource - an array of quadrilaterals regularly arranged on a plane

l) vtkTransformPolyDataFilter - a filter that transforms points, associated normals and vectors for spatial data

m) vtkPolyDataNormals - normals for spatial data

n) vtkTexture - image algorithm responsible for generating textures and their properties

- SetLookupTable - setting the colour palette

- SetColorModeToMapScalars - colour scalar mapping

o) vtkMatrix4x4 - represents a 4x4 matrix

p) vtkTransform - represents linear transformations of 4x4 matrices

q) vtkAxesActor - a hybrid of 2D/3D actor that allows displaying 3D axes (mainly for the vtkOrientationMarkerWidget class)

r) vtkOrientationMarkerWidget - displays the x,y,z coordinate system for reference

- SetOrientationMarker - set the marker in vtkAxesActor space

- SetEnabled - enable the marker

- InteractiveOn - enables user interaction with solid rotation in real time

s) vtkImageThreshold - thresholding the data

- ThresholdBetween - setting the threshold

t) vtkImageShrink3D - reduces the image by sampling evenly

u) vtkImageGaussianSmooth - Gaussian smoothing of the input image by convolution method

v) vtkMarchingCubes - determines isosurfaces of data

x) vtkFlyingEdges3D - scalable, high-performance is contouring algorithm for 3D

y) vtkDecimatePro - filter that reduces triangles in the mesh, approximating the original solid geometry

z) vtkWindowedSincPolyDataFilter - improving the position of data points using windowed sine

aa) vtkStripper - a filter that generates triangular stripes or multiline

bb) vtkContourFilter - contouring algorithm

cc) vtkSliderWidget - a slider widget that allows you to change a parameter at runtime

dd) vtkSliderRepresentation2D - renders and represents the vtkSliderWidget