# Homework 2

Milena Belianovich
Wednesday, February 14

## 1 Problems

### 1. Visualization of 2D images

- What threshold did you use for capturing the riverbed? Experiment with other thresholds and explain what features you may or may not have missed with this approach.

  I used the lower 13 and upper 80 threshold for capturing the riverbed. Using a lower upper threshold removes some features of the riverbed (as seen in Figure 1), and using a higher upper threshold includes unneccessary for this representation parts of land (as seen in Figure 2). Changing the lower threshold in general removes some of the deeper parts of the riverbed (as seen in Figure 3). Therefore, the optimal solution to capture the full riverbed, was to pick the lower 13 and upper 80 threshold (as seen in Figure 4).

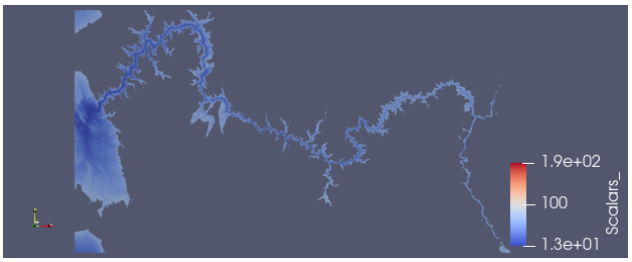


Figure 1: Riverbed representation with lower 13 and upper 70 threshold values

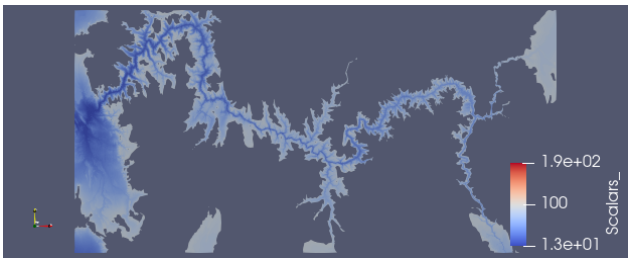


Figure 2: Riverbed representation with lower 13 and upper 90 threshold values

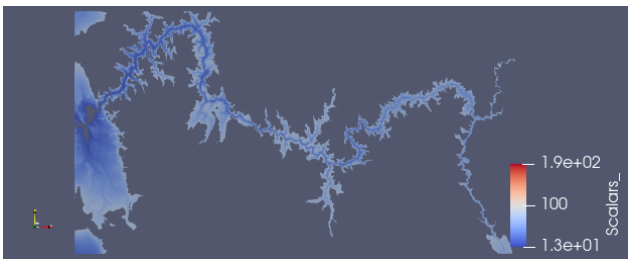


Figure 3: Riverbed representation with lower 20 and upper 80 threshold values

code: https://github.com/milenabel/CS6635-VisforScD

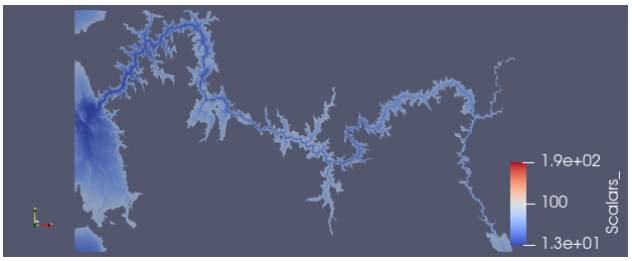


Figure 4: Riverbed representation with lower 13 and upper 80 threshold values (final version as submitted)

- Using the Information panel, report the number of points in the thresholded image. Note that ParaView automatically creates cells from an input image, implicitly forming a structured quad mesh.

  Number of points: $1,289,919$.

  Number of cells: $1,252,447$.

- Create and label a contour plot of the data using multiple isocontour values.

  Such contour plot can be seen in Figure 5.

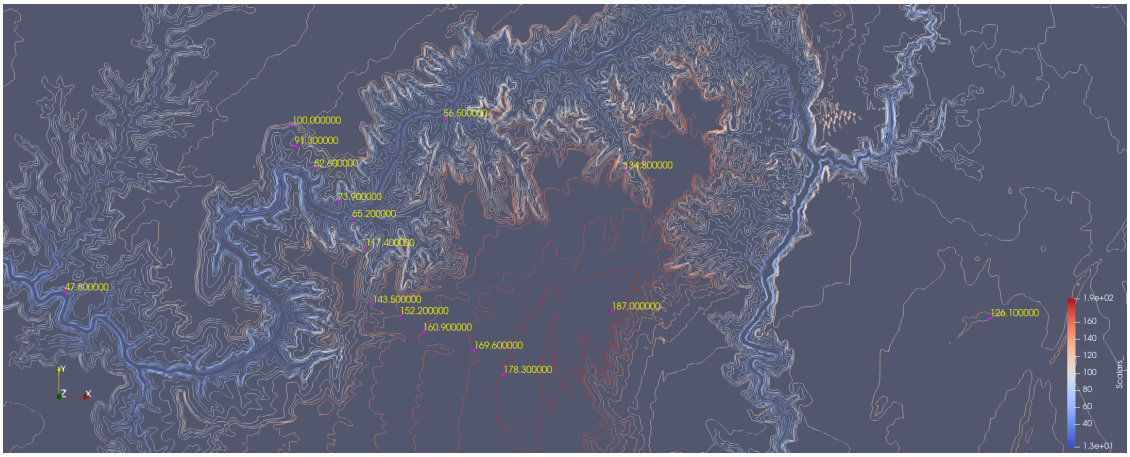


Figure 5: Contour plot of the data using multiple isocontour values

**2. Exploring Data on Polygonal Meshes.**

- What were the minimum and maximum values that best captured the single cylinder associated with the bolt's cylinder?

  The minimum value for capturing the cylinder is 27 for higher threshold, and the maximum value for capturing the cylinder is 43 for higher threshold. Figures 6, 7, and 8 show the differences in changing these values.

- How many ventilation slots are there?

  There 48 ventilation slots as seen on Figure 9.

code: https://github.com/milenabel/CS6635-VisforScD

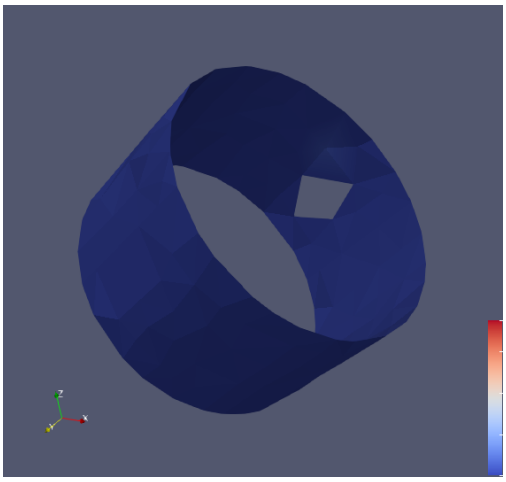

Figure 6: Cylinder representation with higher threshold 26

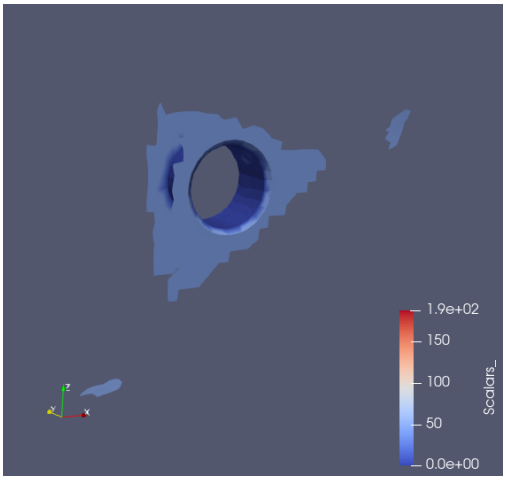


Figure 7: Cylinder representation with higher threshold 44

**3. Visualization of 3D images**

In order to achieve the desired result for this task, the following steps were followed:

- Loading 3d.vti, we see only an outline of the bounding box at this point as seen in Figure 10.
- Creating slices that are aligned with the X normal of the image, Y normal of the image, and Z normal of the image, as seen in Figures 11, 12, and 13.
- Creating a linked view in ParaView as seen in Figure 14.
- Editing the multi-slice view using the colormap editor, such that the background is thresholded out and the remaining data ranges from slightly transparent to fully opaque at the highest value, as seen in Figure 15.

code: https://github.com/milenabel/CS6635-VisforScD

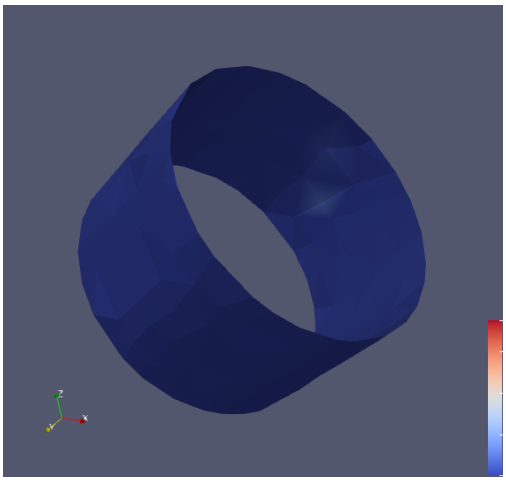

Figure 8: Cylinder representation with higher threshold 27

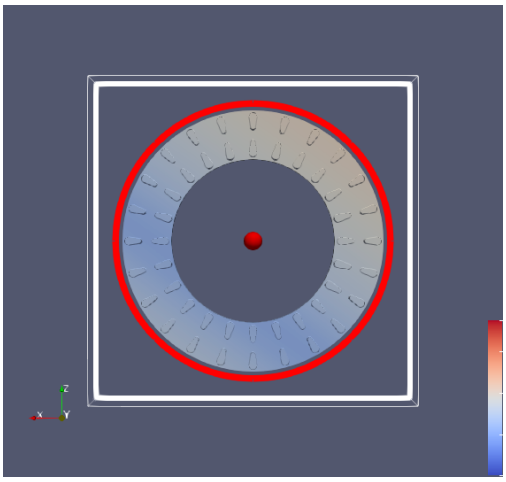

Figure 9: Clipped cylinder representation

**4. Marching Squares using Python.**

- Draw a dot at the center of each cell. Color the dot red if it is above the isovalue and black if it is below. The isovalue is set to 50 by default. You are given a helper function, "draw dot," to add a point to the plot. Show an image of your plot.

  The plot image can be seen in Figure 16.

- Create a naive implementation of marching squares where lines start and end at the center of cell edges. You will need to loop through groups of 4 cells and use the lookup table to determine where the contour line will be drawn. You are given a helper function, "draw line," to add the contour line segment to the plot. Show an image of your plot.

  The plot image can be seen in Figure 17.

- Modify your marching squares algorithm to use linear interpolation for placement of line vertices. Show an image of your plot.

  The plot image can be seen in Figure 18.

---

code: https://github.com/milenabel/CS6635-VisforScD

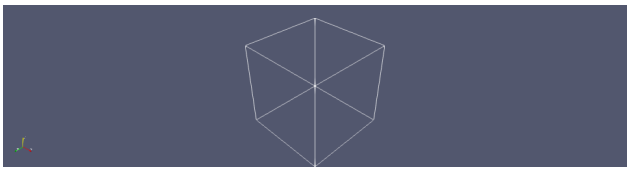

Figure 10: 3D Visualization: loading 3d.vti

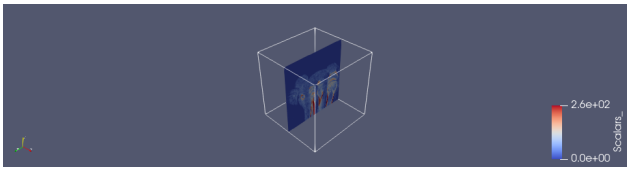


Figure 11: 3D Visualization: creating a slices aligned with the X normal

**Isosurface Animation.**
Attach screenshots with your answers.

- Find the approximate range of isovalues for the transition between skin and skull.

  The approximate range of isovalues for the transition between skin and skull is $276 - 1162$, as seen in Figures 19 and 20.

- At what isovalue the spine vanishes?

  Spine vanishes at isovalue $2490$ as seen in Figure 21.

- What's the approximate isovalue for the teeth.

  The approximate isovalue for teeth is $2932$ as seen in Figure 22.

**Reading Questions.**

- What is time-varying data? What challenges do we face when extracting isosurfaces from time-varying data.

  Time-varying data refers to datasets that change over time, often seen in simulations or data collection over periods. Extracting isosurfaces from such data is challenging due to the dynamic nature of the data, requiring efficient and accurate methods to capture the evolving shapes and features of the isosurfaces over time. This task involves handling large volumes of data with potentially complex changes, necessitating advanced computational techniques for effective extraction and visualization.

- Briefly explain the need for temporal hierarchical index tree data structure for isosurfaces extraction in time-varying data.

  A temporal hierarchical index tree data structure is crucial for efficiently managing and accessing time-varying data for isosurface extraction. This data structure enables the organization and indexing of data in a hierarchical manner, facilitating quick and efficient retrieval of relevant data points for isosurface extraction. It helps in handling the complexity and volume of time-varying data by allowing for quick searches and updates, which is vital for tracking and visualizing the evolution of isosurfaces over time.

---

code: https://github.com/milenabel/CS6635-VisforScD

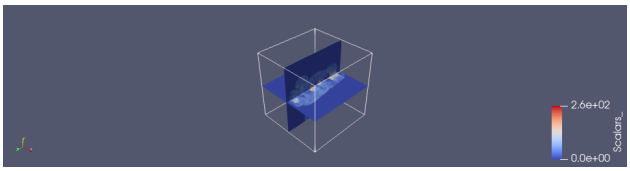


Figure 12: 3D Visualization: creating a slices aligned with the Y normal

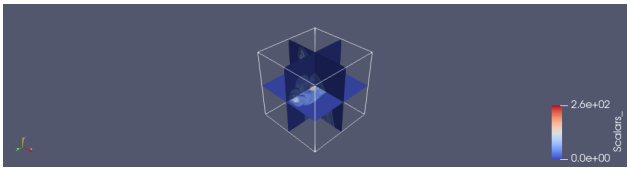


Figure 13: 3D Visualization: creating a slices aligned with the Z normal

- Given the temporal hierarchical index tree, briefly describe isosurfaces extraction algorithm in time-varying data in your own words.

  The isosurface extraction algorithm utilizing a temporal hierarchical index tree involves several key steps. First, the algorithm organizes the time-varying data into the index tree, allowing for efficient data management. Then, it retrieves the relevant data points from the tree based on the desired iso-value (the value defining the isosurface). The algorithm processes these data points to construct the isosurface, adjusting to changes in the data over time. This process involves interpolating values between data points and constructing a mesh that represents the isosurface. The hierarchical nature of the index tree enables quick adjustments to the isosurface as the underlying data evolves, ensuring that the extracted isosurface accurately represents the current state of the data.

## 2 Conclusion

In this assignment, I undertook a detailed study of visualization using both Python and ParaView, focusing on the Marching Squares algorithm and exploring various visualization techniques. The key aspects of the work include:

- Thresholding for Riverbed Visualization: Experimented with different thresholds to capture the riverbed in ParaView, leading to the selection of optimal threshold values for an accurate representation. This task underscored the importance of parameter tuning in visualizing specific features within a dataset.

- Multi-Isocontour Visualization: Created and labeled a contour plot using multiple isocontour values. This technique showcased the utility of contour plots in revealing different data layers and features within a single visualization.

- Exploring Data on Polygonal Meshes: Investigated the best values for capturing specific features, such as a single cylinder in a bolt's cylinder and the number of ventilation slots. This exercise demonstrated the application of visualization techniques in engineering and mechanical design contexts.

- 3D Image Visualization in ParaView: Loaded and manipulated 3D images, including creating slices aligned with different axes and editing multi-slice views. This part of the assignment highlighted the capabilities of ParaView in handling and visualizing complex 3D datasets.

- Marching Squares Implementation in Python: This involved drawing dots and lines, analyzing cell values, implementing midpoint and linear interpolation methods, and visualizing the contours. It was an essential practice for understanding the underlying mechanics of contouring algorithms and their practical applications.

---

code: https://github.com/milenabel/CS6635-VisforScD

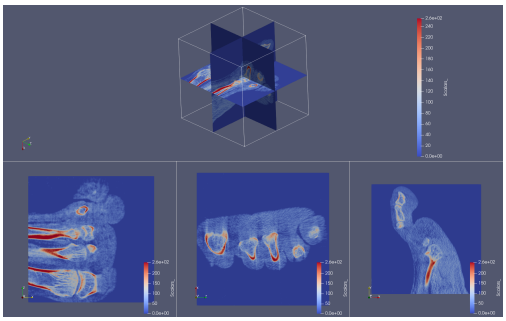

Figure 14: 3D Visualization: creating a linked view

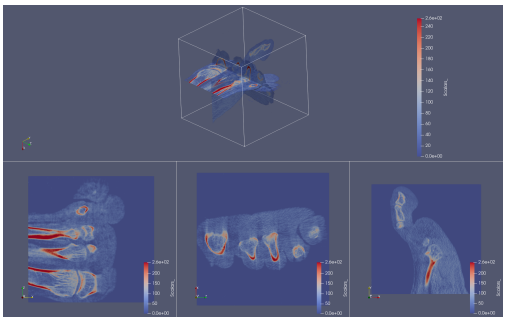

Figure 15: 3D Visualization: multi-slice view edit using colormap

- Isosurface Animation: Determined the approximate range of isovalues for different transitions in the data, such as between skin and skull, and the disappearance of the spine. This part emphasized the relevance of isosurface animations in medical imaging and anatomical studies.

Throughout this assignment, I gained an understanding of the principles and applications of scalar field visualization. The tasks in ParaView complemented the programming-based approach in Python, offering a holistic view of visualization techniques.

code: https://github.com/milenabel/CS6635-VisforScD

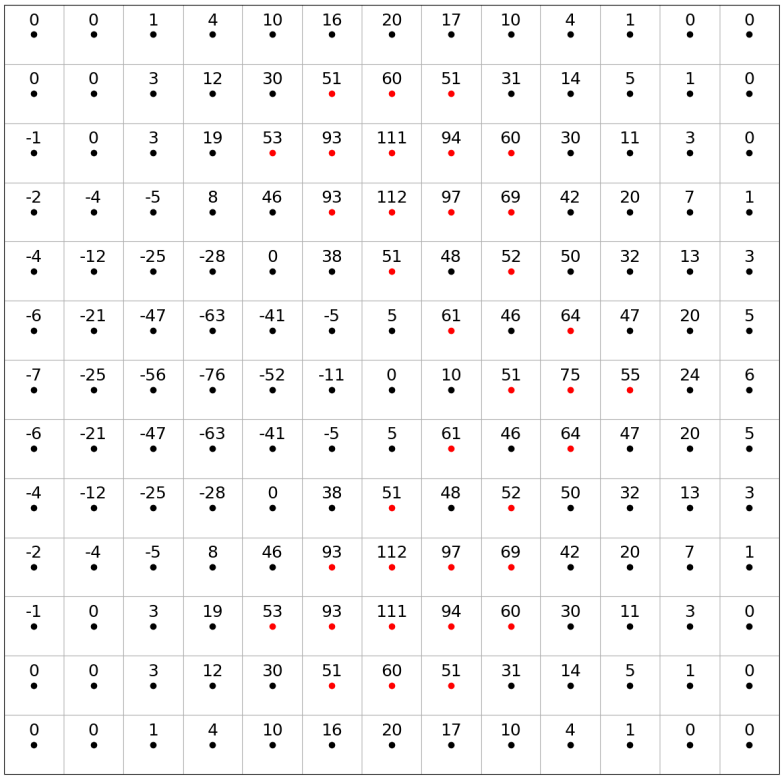


Figure 16: Scalar field square grid with added red and black colored dots

code: https://github.com/milenabel/CS6635-VisforScD

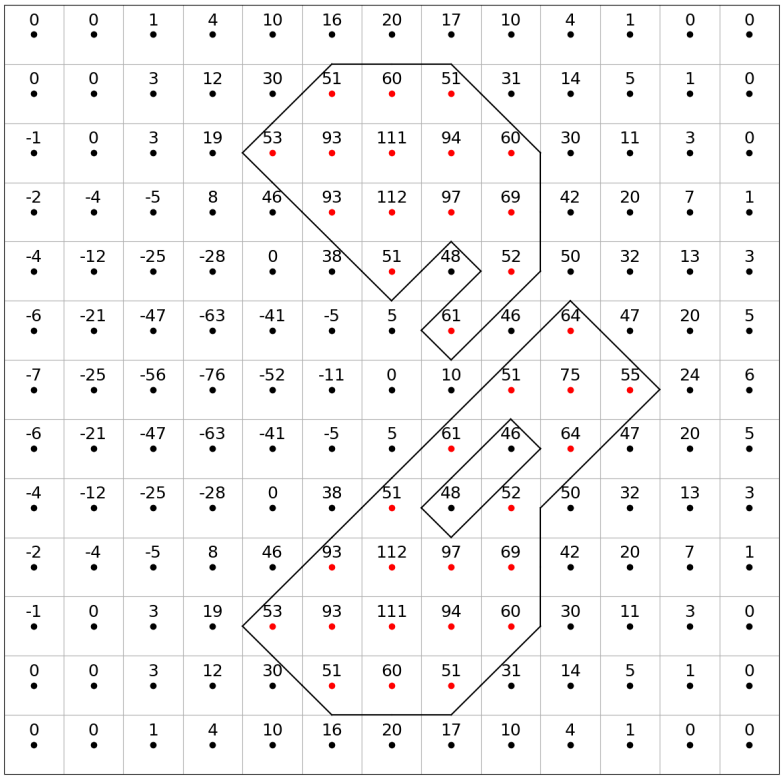


Figure 17: Scalar field square grid with added naive implementation of marching squares

code: https://github.com/milenabel/CS6635-VisforScD

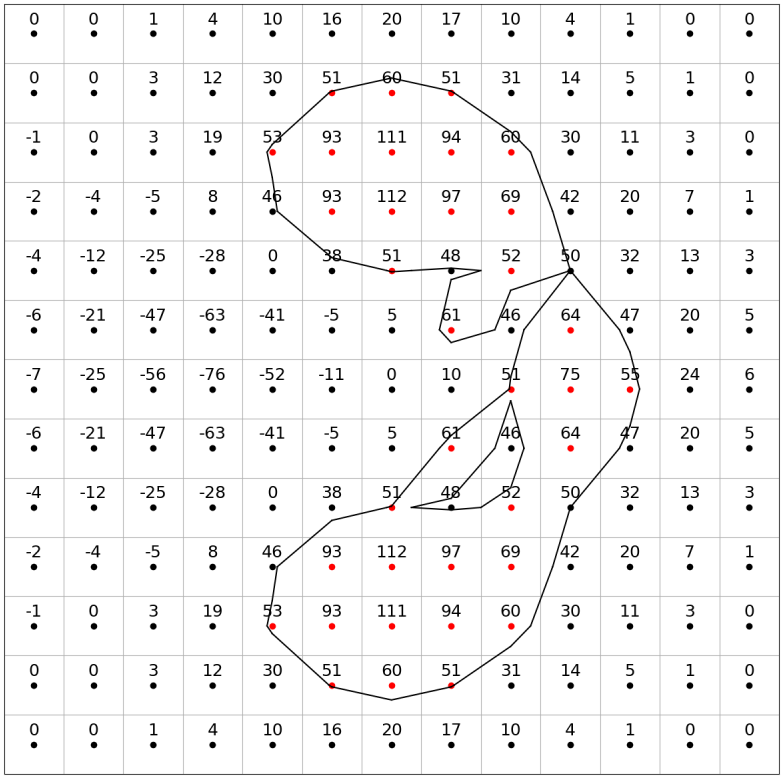


Figure 18: Scalar field square grid with added naive implementation of marching squares

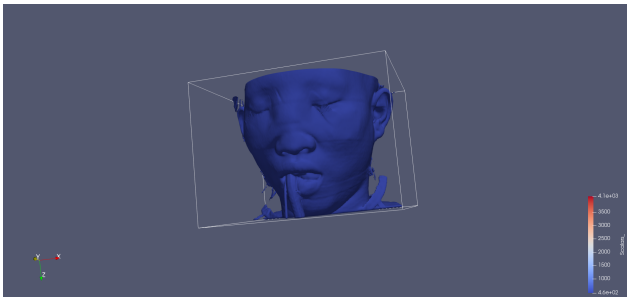

Figure 19: Isosurface Animation: skin isovalue capture

code: https://github.com/milenabel/CS6635-VisforScD

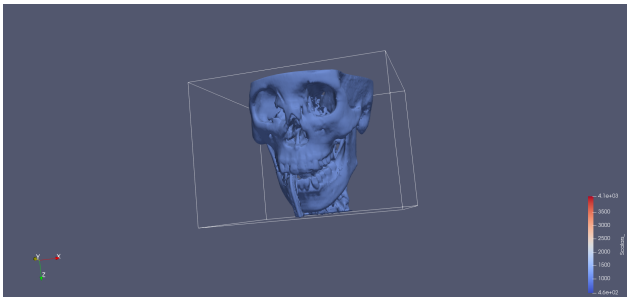

Figure 20: Isosurface Animation: skull isovalue capture

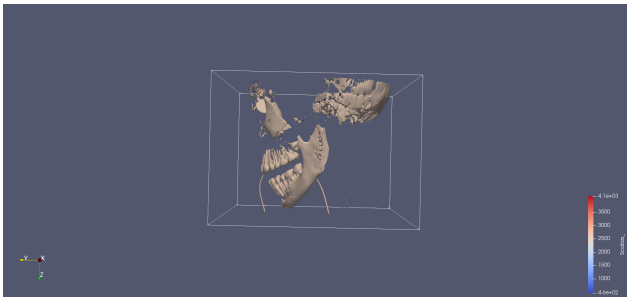

Figure 21: Isosurface Animation: spine vanishing isovalue capture

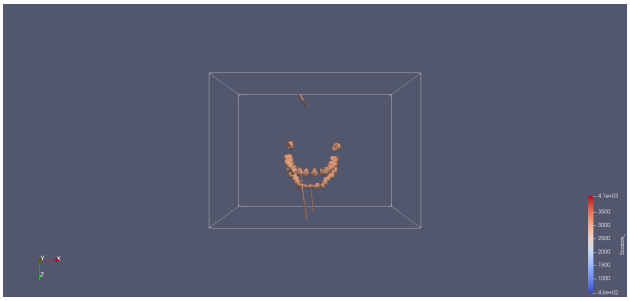

Figure 22: Isosurface Animation: teeth isovalue capture

code: https://github.com/milenabel/CS6635-VisforScD