Scientific Visualization & Virtual Reality

Visualization of Medical Datasets

Roberto Valenti, 0493198
Felix Hageloh, 0425257

Introduction

Goal of this assignment was to develop a full application that allows physicians to analyze and explore data from CT or MRI scanners. This data consist of scan slices of a patient that can be combined to form a 3D presentation of the insides of a patient. Traditionally, each slice was printed on transparent film and then analyzed separately, which means the physicians had to construct the 3D representation in their mind in order to form a diagnosis.
The application we implemented in this assignment automatically constructs this 3D representation for the physician and thus allowing the physicians to effectively visualize and analyze this data, extracting additional information through the use of computer graphics.

The project comprehends the following tasks:

- Isosurface Visualization
- Interactive Framerates
- Threshold Level Isosurface Extraction
- Segmentation and Quantification
- Slice Extraction
- Volume Rendering

The visualization part of the program was implemented using the VTK open source library. For implementing and designing the graphical user interface we used QT which since recently features a binding for VTK.

In the following sections we will give and overview of our application and look at each part in detail, explain our implementation and mention the VTK classes we used. Furthermore we will give some impression of the final application and show the results of the experiments we conducted with it. Finally we will discuss the results of our experiments, what was achieved in this assignment including its weak points and we will come to a conclusion.

Problem Description

We will now explain in more detail what was required from this application.

- **Isosurface Visualization**
  In general the application has to be able to extract and display isosurfaces from the data set. This means that we need to find all locations in our data set where the value equals to a given threshold

- **Interactive Framerates**
  Rendering a single isosurface can be quite computational expensive and our application will have to render several isosurfaces at the same time. This means we will have to think about possible speedups (three are required by the assignment) to enable our application to run smoothly.

- **Threshold Level Isosurface Extraction**
  Users of our application will be interested to display isosurfaces at different threshold levels. Thus they need to be able to interactively define the threshold of a surface they want to view. Ideally this should happen with immediate feedback, so that users can browse through the isosurface space, searching for the isosurface they are interested in.

- **Segmentation and Quantification**
  Segmentation means isolating structures from the data set to obtain anunobstructed view on a certain anatomical structure. As described in the assignment description we will achieve this by threshold level isosurface extraction that was explained above. However, we have to find away that makes it easier for users to find interesting isosurfaces and add the capability of interactively add and remove isosurfaces. Furthermore we have to think of a clever way to organize and display the selected isosurfaces without being confusing to the user. This could include giving users the ability to

    - look at each segment independently
    - display certain segments together (ie. hide and unhide certain segments)
    - control the color and opacity of each segment

  Furthermore we are required to give extra information about each segment, such as volume and surface area.

- **Slice Extraction**
  Users should also be able to select a slice from the data set in one of the principal axes and view it together with the isosurfaces.

- **Volume Rendering**
  As an optional task we also decided to give users the ability to view the entire data set using volume rendering. Volume rendering has the advantage that it provides better means to view the data set in its entireness. The final image that users can see is a result color mappings of all values in the data set and not just from selected isosurfaces. So our program has to allow users to switch between the viewing modes.

Implementation

Our application was written in C++ using the VTK library to handle all visualization issues. For the interface we used QT, which is actually an application framework for C++ and contains many libraries for building GUIs.
We start by giving a screenshot of our final application, which is segmented in the different sections. For each section we will give an explanation and describe how it solves the problems stated in the previous section.

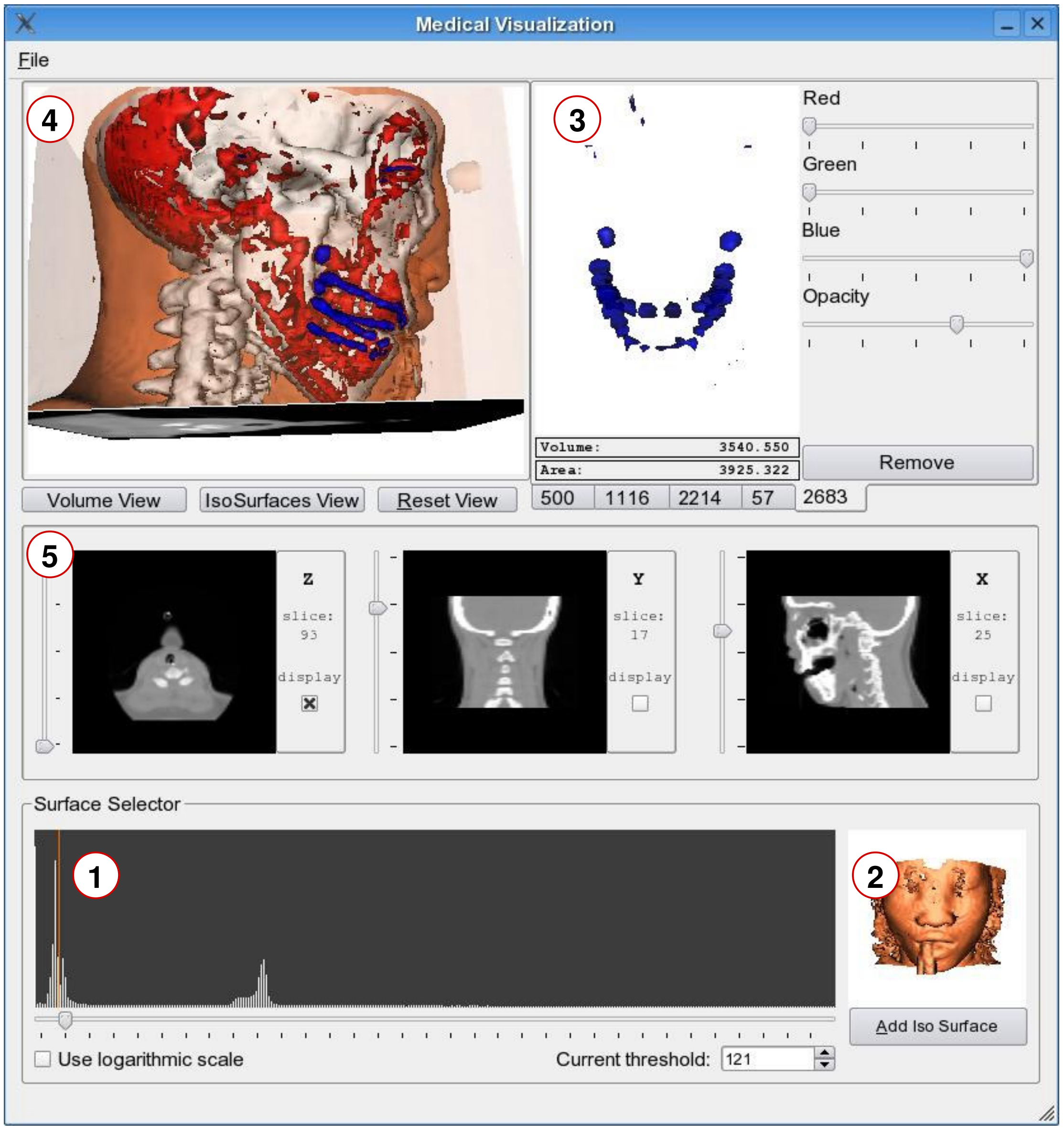


Figure 1 - overview of the final application

The numbered parts can be described as follows

1.) Histogram panel and isosurface selector
2.) Preview window
3.) Segments tab area
4.) Main window
5.) Slice extraction area

We will continue by looking at each part in more detail.

Histogram panel and isosurface selector

This area is the starting point for exploring the data set and selecting interesting isosurfaces/segments. Normally, extracting the correct isosurfaces from the data is a trial-error task. However, knowing that many of the pixels in a CT dataset are either part of the patient's skin or bone, we can imagine that if we grouped pixels of equal value together in a histogram we can get some better insight to the data. Big peaks in the histogram would then correspond to the main segments of the data set, such as skin and bones. Smaller peaks might correspond to smaller segments such as teeth.

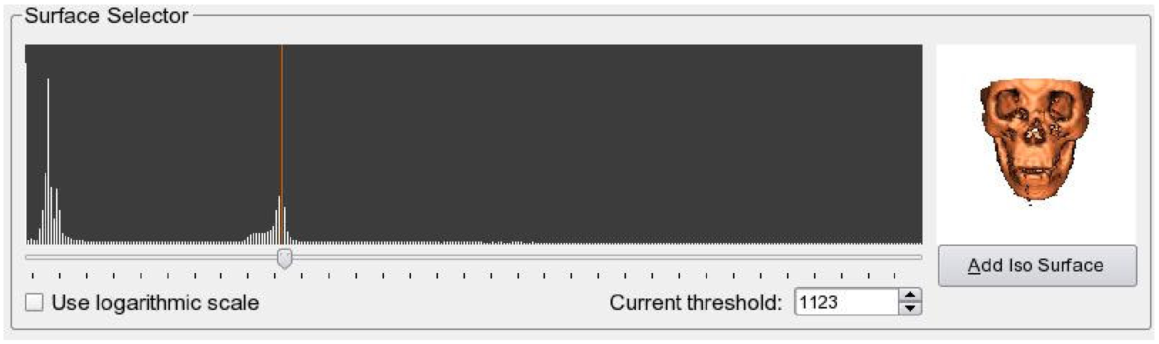


Figure 2 - histogram panel

We construct the histogram using the vtkAccumulate class and display it using a Canvas2D source that is connected to a vtkImageViewerPanel. For each bin in the histogram we draw a line segment with height corresponding to the number of pixels in that bin. The number of bins and the height of the line segments are correctly scaled to the size of the window, so that the number of bins fills the entire width of the window and the maximum height corresponds to 90% of the height of the window. The render window of the vtkImageViewerPanel is connected to Qt using the QVTK widget class.

Underneath the panel is a slider that allows users to browse through different threshold levels. The position of the slider is indicated inside the histogram as well by an orange line segment. Alternately, users can also just type in desired threshold levels in the textbox below, or increase/decrease the level by one unit using the up/down arrows of the textbox.

In both cases users will get immediate feedback of the isosurface they selected in the preview window 2) next to the histogram (described next). All these correlated events, i.e. moving the slider, typing a value and rendering the preview are synchronized by the use of QT signals.
Since one bin with a high number of pixels will cause other bins to be scaled down it might be difficult to see smaller peaks in the histogram. Thus we implemented the option to view the histogram in logarithmic scale. Enabling and disabling histogram viewing is done via a checkbox underneath the slider (figure 3)

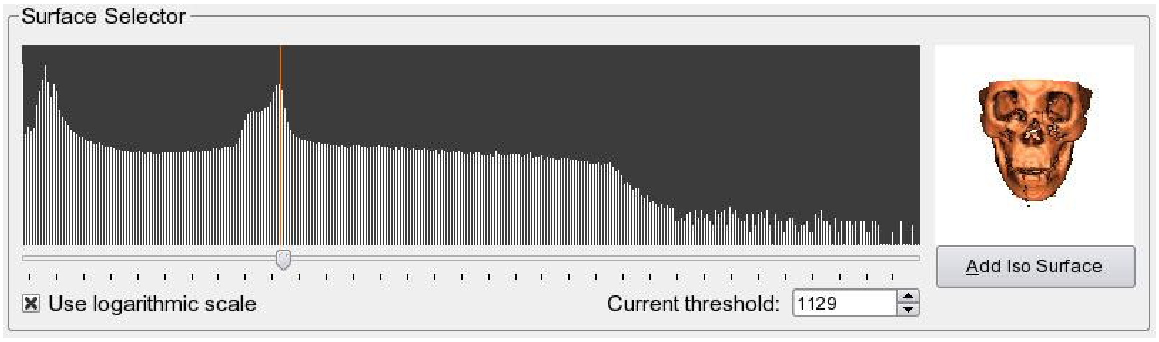


Figure 3 - histogram panel with log scale

Preview window

As described in the previous section, the preview window displays the isosurface of the currently selected threshold level. Surfaces are displayed using the steps described in the first task of the assignment. During the initialization phase of the program the data is read from disk using a volume16 reader. Isosurfaces extracted using the marching cubes algorithm to create a polygonal representation of the surface. The marching cubes algorithm is implemented by VTK in the vtkMarchingCubes class. Alternatively we can also use the more general class vtkContourFilter which will then use vtkMarchingCubes for volume isosurface extraction.

To set up the marching cubes algorithm we simple have to specify the level of the isosurface that we want to extract. This is done every time the value of the threshold level is changed, either via the slider or the textbox. The output of the marching cubes class is then piped to a poly data mapper and finally to a VTK actor. The preview window itself is again a QVTK widget and interaction is disabled.

Since during preview new isosurfaces have to be rendered very often and quickly it was necessary to explore possible speedups for this task. For the preview the quality of the image is not very important and it is quite small anyways, so we decided an appropriate speedup would be to down sample the image. This means that instead of using 64x64 pixels sampling resolution on the data set we use a lower resolution. This can be done by using the vtkImageShrink3D class and setting the shrink factor for each dimension to a value greater than 1 (2 for all dimensions in our case).
While we loose quite a bit of accuracy in the resulting image, extracting isosurfaces and rendering is sped up a lot. Also this method is more appropriate than other speedups for this task. For example decimation (explained later) actually slows down the preview, since every time we extract a new isosurface extra calculations have to be performed. In figure 4 we can see an image of the preview window with and without down-sampling.

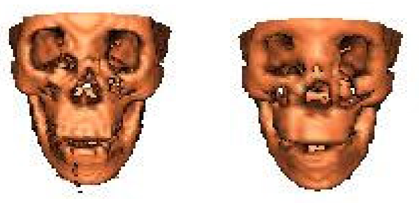


Figure 4 - effect of down-sampling with shrinking factor 2

After the users have found an isosurface they are interested in, they simply have to click on the "Add Isosurface" button below the preview window to select the current segment and view it in more detail.

Segments tab area

After having and isosurface/segment from the data set by clicking on the "Add Isosurface" button the segment will be added to the segments tab area. The tab area is a QT widget which allows stacking widgets in a tab fashion. Tabs are added on the fly as the user selects more segments to display, each containing a detailed and interactive view of the single selected segment (figure 5).

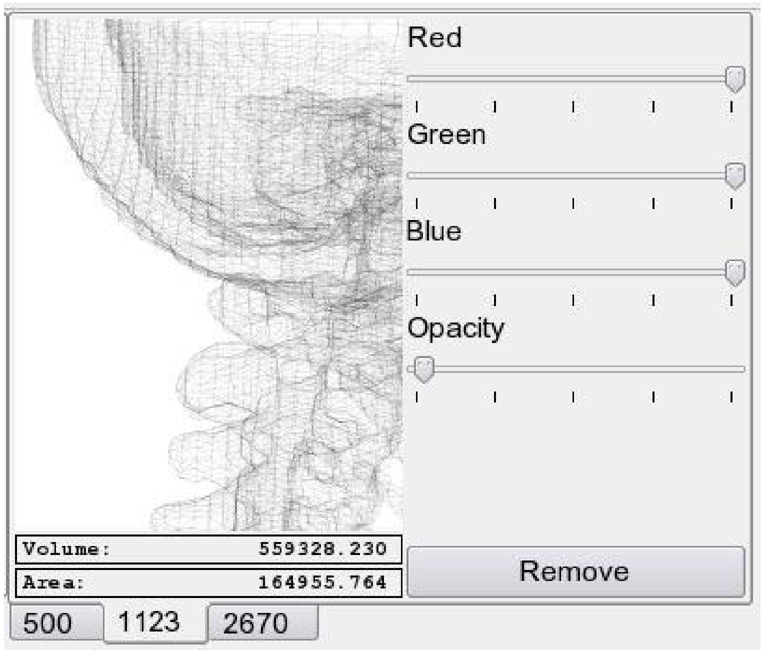


Figure 5 - Segments Tab Area

Each tab consists of a custom QT widget that contains a QVTK render window and several sliders and buttons. The label of the tab is the threshold level of the segment it displays. The sliders next to the render window allow users to adjust the RGB color values of the displayed segment as well as the opacity. Zero opacity will hide the segment completely. All the changes performed to the segment inside the tab (interactions excluded) will also have effect on the corresponding segment displayed in the main render window (described later). If users decide to completely remove a segment they can click on the "Remove" button below the sliders. This will destroy the current tab and remove the corresponding segment from all views.
An arbitrary maximum of 20 tabs was hard-coded for simplicity and it seems not likely that more are required.

Underneath the render window are two labels that indicate the volume and surface area of the isosurface/segment. They are calculated using the vtkMassProperties class, which use the discrete form of the divergence theorem to calculate these properties.
The values given have to be treated with caution. The algorithm is known to work decently for simple shapes closed surfaces, but since most segments have a much more complex structure and might not be closed surfaces, we can expect much a worse accuracy than that.

Since the window inside the tab is supposed to give a detailed view of a segment we decided not to use any speed up method that would reduce the quality of the image. However, we do calculate the normals of the isosurface and pipe them to a vtk stripper which slightly speeds up rendering on most systems.

Main Render Window

This window serves as an overview of all the segments selected. All isosurfaces are displayed together here using the color and opacity attributes set in the corresponding tabs. A good example of this can be seen in figure 11 in the experiments section.
To speed up the rendering of several isosurfaces together we had to implement another speed up method. In this case using decimation was appropriate since we mostly care about the rendering

speed during interaction and want to maintain a certain level of image quality. Decimation is implemented in vtkDecimate and vtkDecimatePro. Both methods classify every vertex using the plane-point distance measure. This measure gives an indication of how much the mesh will change if the vertex is removed. The user has to specify how large this change can be when setting up the algorithm. This threshold is called the decimation criterion. If the decimation criterion is satisfied, the vertex is removed and the resulting hole is triangulated. In vtkDecimatePro, the substantial improvement over vtkDecimate is the generation of progressive meshes during the decimation process. This means process can be applied recursively on resulting sub meshes leading to better decimation results.
In our application we used decimate pro with a target reduction of 0.5. The effect of decimation can be seen in figure 6.

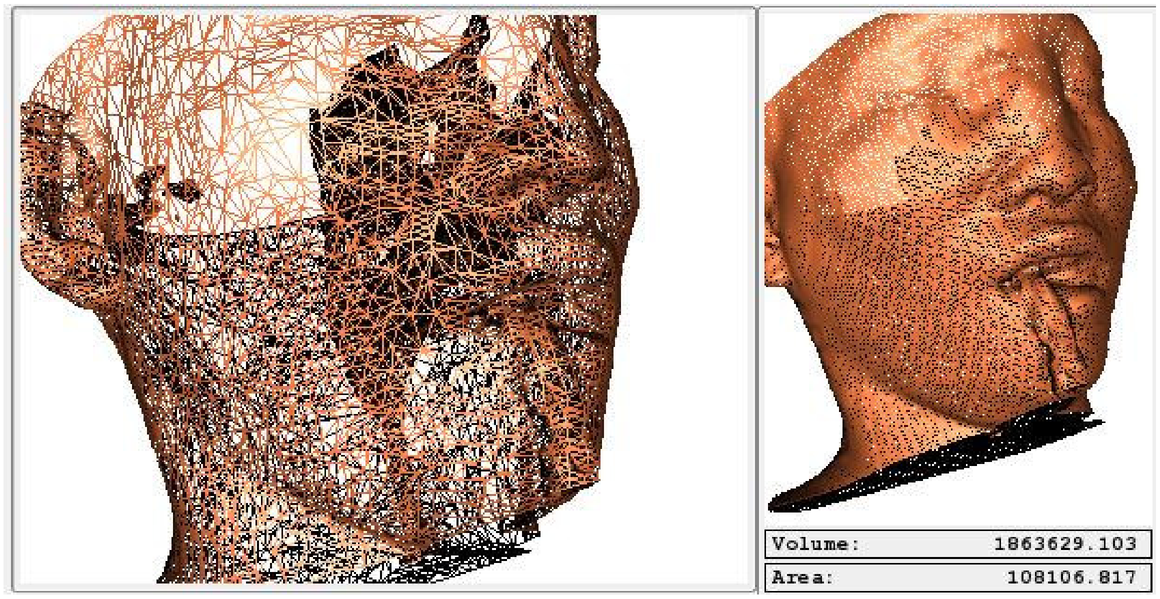


Figure 6 - effect of decimation with target reduction 0.5

Since during interaction users might want to quickly center the view of the main render window again we added a "reset" button beneath to automate this. Also there are two buttons to switch between volume rendering and isosurface view.

As mentioned earlier, volume rendering has the advantage that it gives a holistic view of the data and we decided to add a basic implementation of this feature to our application. It can be achieved using VTK in 3 steps:

- define a opacity mapping function using a vtkPiecewiseFunction
- define a color mapping function using a vtkColorTransferFunction
- define ray cast function using vtkVolumeRayCastCompositeFunction which will be used by a vtkVolumeRayCastMapper to map the correct color to each displayed pixel

The volume itself is represented by a vtkVolume class who has as a property the first two functions above. The points in these function where defined by trial and error. For example we knew that a value of 500 in the dataset corresponds to the skin of the patient, so in the color mapping function we define an RGB point after 500 (at value 600 for us) using a skin color. Moreover we want the skin to be quite transparent so we can add a point to the opacity function defining high opacity from 0 to the same value (600). The results can be seen in figure 7.

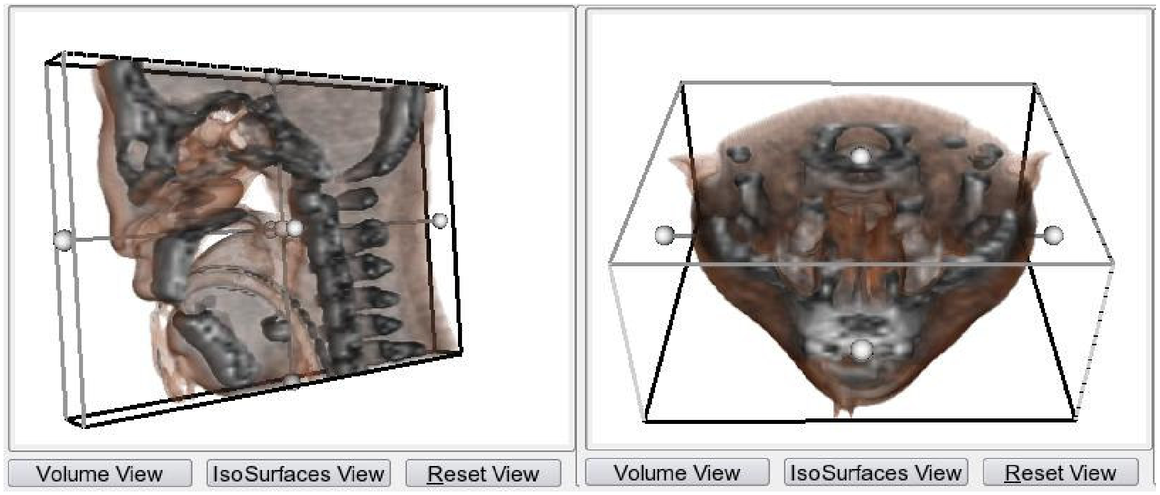


Figure 7 - volume rendering with box widget

The image also shows the VTK box clipping widget that we added for interactive exploration. The path of the tube and the nose structures are clearly visible to the left, while an overview of the lower half part of the skull is displayed on the right.

Slice extraction area

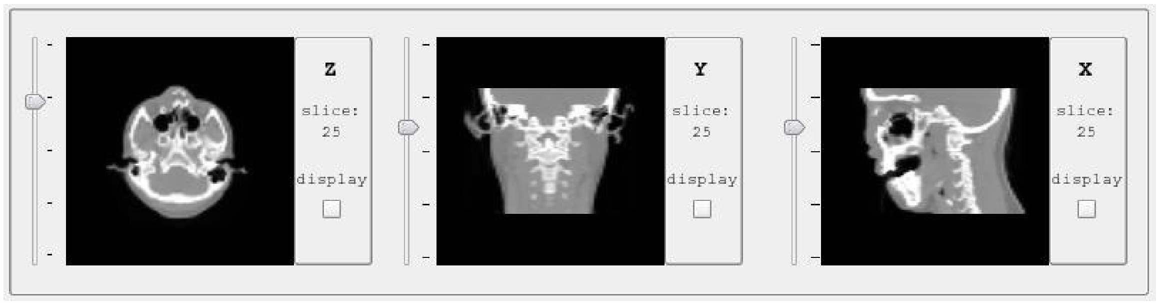


Figure 8 - Slice Extraction Area

To enable the user to extract slices in all principal axes we added an extra panel to our application. The panel contains three render windows displaying a slice in each principal direction. The position of each slice can be adjusted using the slider next to it. This way, users can browse through the data set slice by slice. By default slices are only displayed in their corresponding window but the user can also decide to display them together with the isosurfaces in the main render window by toggling the "display" checkbox.

In figure 9 is an example of the power of this additional tool, displaying the slices in the three main dimensions together with an isosurface. Through this procedure, it is possible to have better understanding of the data and to locate the 3D position of an area seen on the slices.

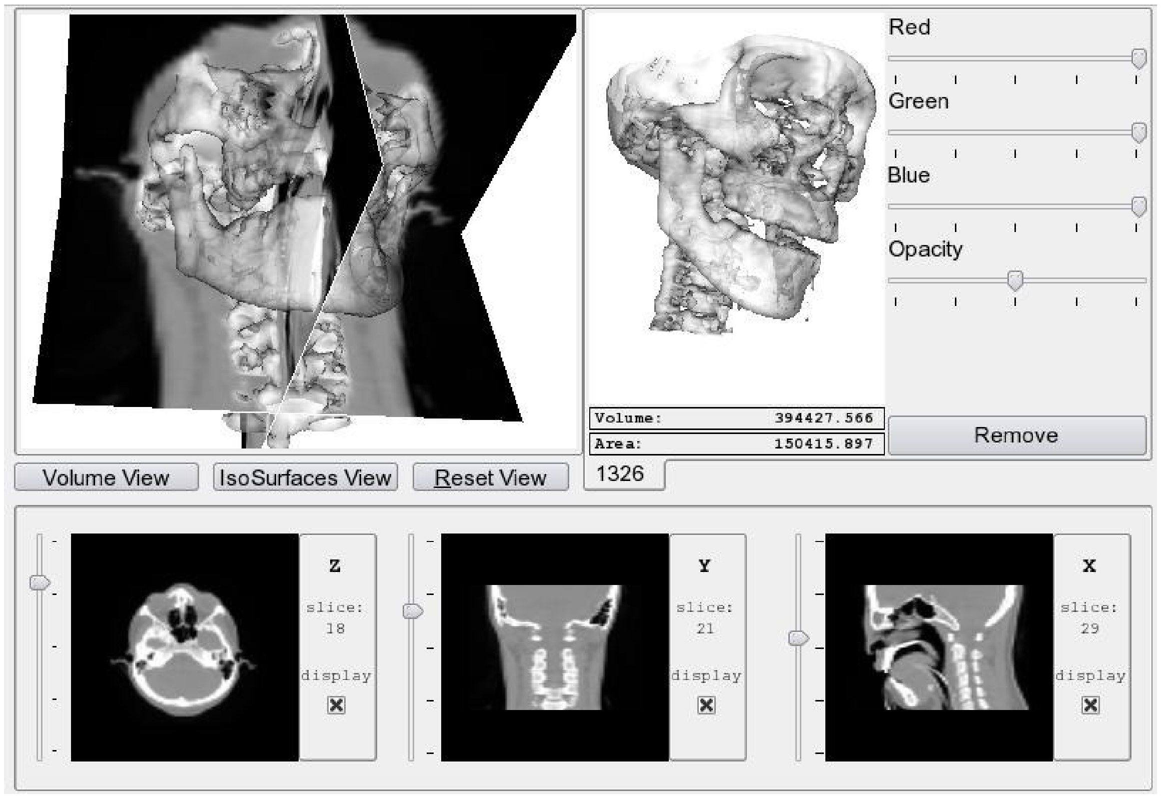


Figure 9 - slice extraction together with isosurface view

This was easily implemented because we use a vtkImagePlaneWidget to extract the slices to begin with. While "display" is unchecked we simply disable the plane widget and only use its output to pipe to a vtkImageMapToColors object. This class uses the output of the plane widget and a color lookup table to create an image of a slice, which is then piped to a vtkImageActor to display in the corresponding window. We defined a simple black and white lookup table for this.

When display is enabled the same lookup table is used to display the image on the plane widget itself, which is done by VTK automatically.
The plane widgets are moved using the sliders mentioned above. A change in value of a slider will cause a QT signal to be fired which is used to update the position of the widget. For simplicity interaction with the plane widgets is disabled so that they can only be moved via the sliders.
Since we opted for this approach it was much more convenient to use plane widgets instead of vtkExtractVOI as was required in the assignment.

Experiments and Results

For the first task we had to find the bone structure of the patient. We found that a value of 1200 works well as can be seen in figure 10.

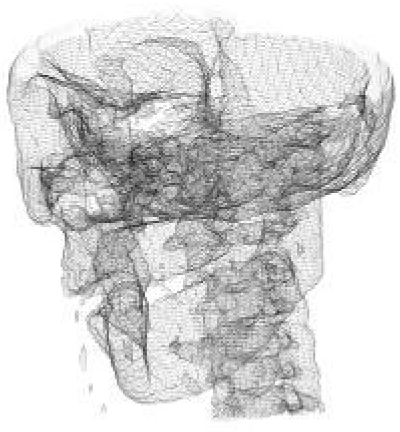
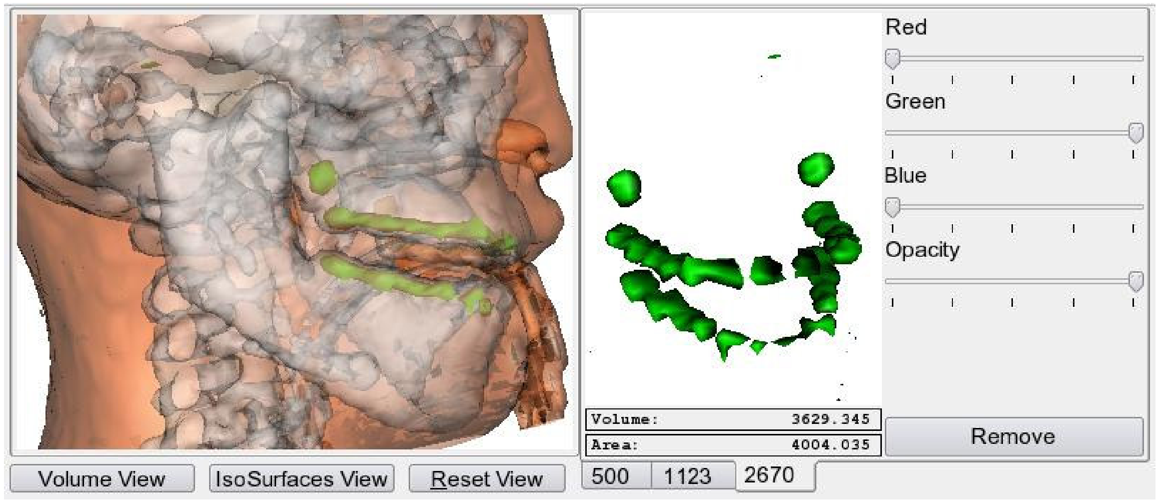


Figure 10 - patients bone structure at threshold 1200

Further experiments where mostly conducted to test and demonstrate the workings of our application. Since we have no medical experience it would be difficult for us to spot any anomalies in the data set even if they are clearly visible and obvious to a physician.

In figure 11 there is an example visualizing several segments together in the main window three isosurface were added to the main view. Looking on the tab panel it is possible to understand the histogram value of each of the surfaces directly from the title of each tab. The color of the isosurface at value 500 (skin) was left as default, the color of the isosurface at 1123 (bones) was changed to white and the color of the isosurface at 2670 (teeth) was changed to green. The opacity of both the skin and the bones was slightly reduced to clearly display the actual position of the last segmented surface in the skull.



Volume: 3629.345
Area: 4004.035

Figure 11 - visualization of several isosurfaces together

Figure 11 clearly shows the teeth of the patient. There are two other objects above the teeth with the same density, which are probably wisdom teeth. However there is another tube like segment far above the teeth, which appears to almost pierce the patient right eye when visualized together with the skin and bones as can be seen in figure 12 in blue.

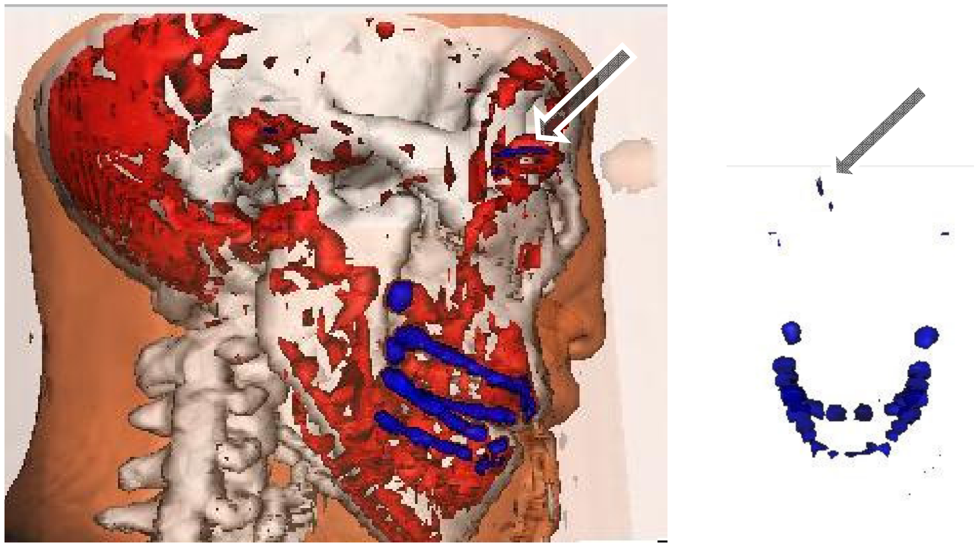
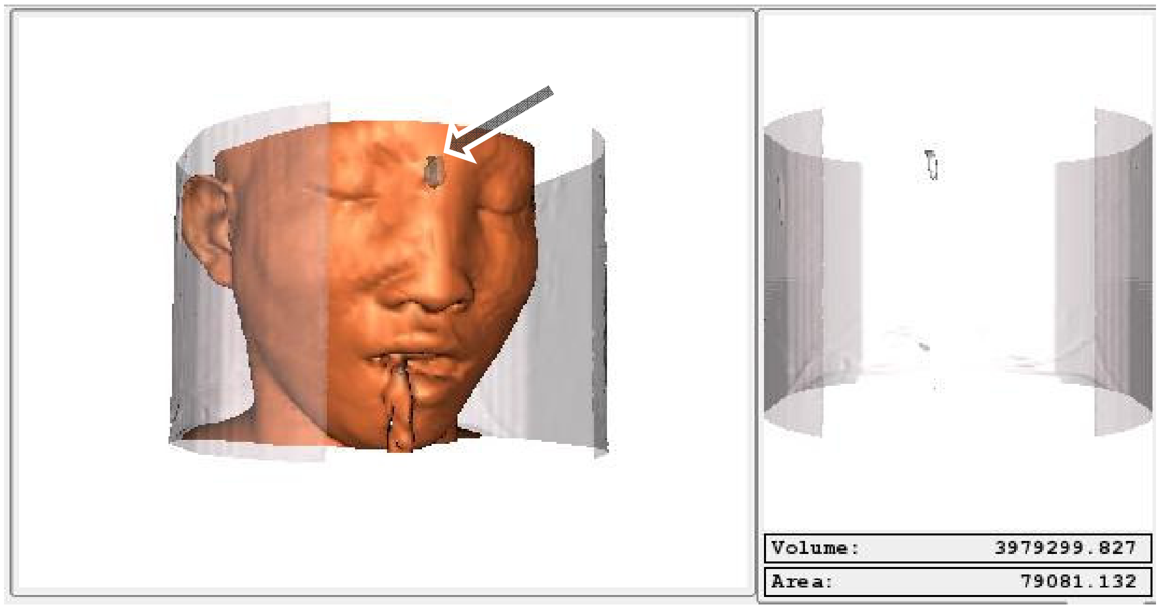


Figure 12 - interesting segment next to patients right eye

When selecting an isosurface with approximately the same density as the wall of the CT scanner a similar tube like object appears outside the patients head, which seems to be a continuation of the object inside the head (figure 13).



```
Volume:        3979299.827
Area:            79081.132
```

Figure 13 - a similar segment outside the patients head

The experiment depicted in figure 14 is a demonstration that slice extraction also works with volume rendering.

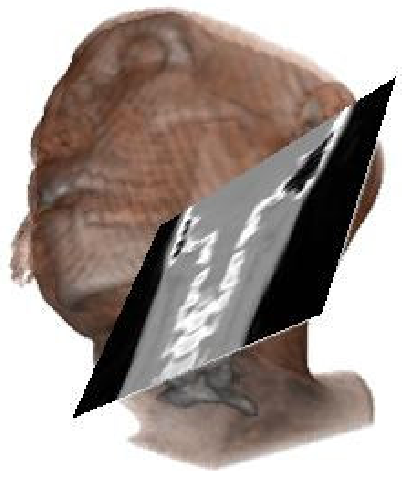


Figure 14 - slice extraction with volume rendering

Finally we also experimented using a different color mapping for volume rendering. Instead of using the actual data value in each voxel we use the gradient magnitude at that voxel as input to the color mapping function. These gradients magnitudes can be obtained using the vtkImageGradientMagnitude class. Only one RGB point (red) was specified at the maximum value of the gradient. The results can be seen in figure 15.

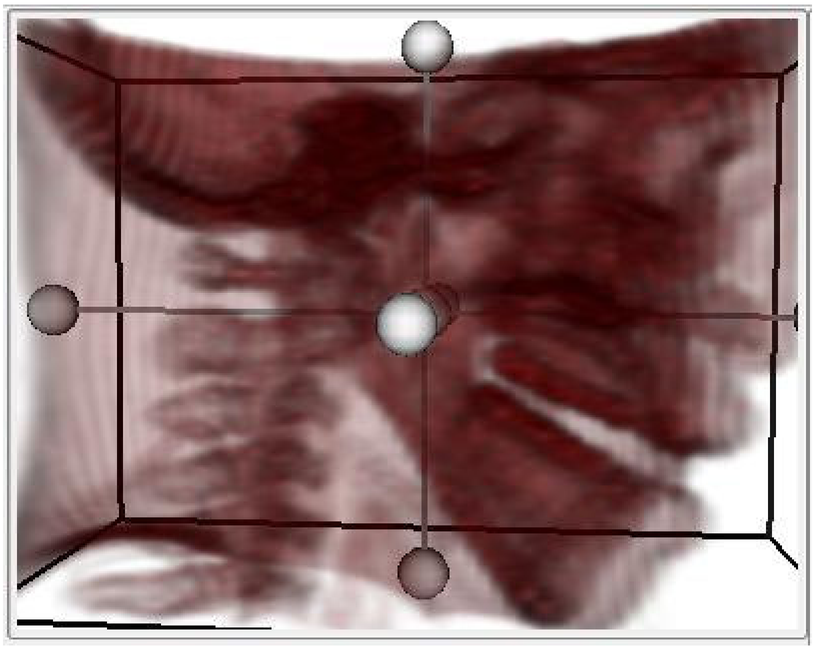


Figure 15 - volume rendering with gradient coloring

Discussion

As mentioned before, most experiments were aimed to demonstrate the proper workings of our application and we think we have succeeded in that. The interesting shape we discovered inside and outside the patients head might be a good example of how this application can help to discover anomalies from the scan. However, we do not have the necessary background to verify this.

The experiment using gradient magnitude for color mapping in volume rendering is proof of concept that this method also works. In fact it might even give better insight to the data than the default coloring using the values of the data set itself. This is because a high density gradient should indicate the transition from one type of tissue to another. Unfortunately the resulting image is quite blurry, but we believe this is due to a bad color and opacity mapping function. We didn't have enough time to experiment with different color mapping functions so we decide to exclude this feature from the final version of the code.

About the GUI in general we can say that we tried to keep it as user friendly as possible. The number of buttons was limited to a minimum necessary. When the application starts, a demonstrative isosurface (skin segmentation at value 500) is automatically added. When in Volume View mode the access to the Surface Selector and the isosurfaces is interdicted to allow users to focus only on the relevant parts of the application in its current state, which avoids useless/erroneous operation. In case the users gets lost during the interactive exploration of the data, they can easily press the reset view button and the default frontal view will be set, both in "Volume View" and "IsoSurfaces View" mode. Furthermore, the application was tested to be user-error-safe: the user can click twice on each button, switch views, try to add over twenty and remove more than zero isosurfaces without generating a segmentation fault or creating unexpected application behaviors.

The two speedups mentioned in the report (down sampling and decimation) constitute only two of the required three speedups. A third speed up could be using dot surfaces instead of polygonal representations for isosurfaces. In theory this should render faster than polygonal representation, especially for smaller resolutions. However, since most hardware nowadays is optimized to render polygons this speed up is questionable. Again we didn't have enough time to fully investigate the effect of this type of rendering, so that it was excluded from the code.

Conclusion

We successfully manage to create an application that allows the visualization and exploration of CT or MRI scans and fulfills all the requirements of the assignment. By doing so we can enable physicians to spot anomalies that would be difficult or impossible to find by looking at the data slice by slice, as it was done traditionally.

Overall, we feel that we got good hands-on experience on scientific visualization in general and also on creating GUI applications. Specifically we learned a lot about using VTK and QT, both of which were completely new to us.