MASTER'S DEGREE THESIS

Environmental Computing

Liv Hovland

Østfold University College

June 2004

Visualization of 3D Structures in Underwater Volumes in Sonar Performance Context

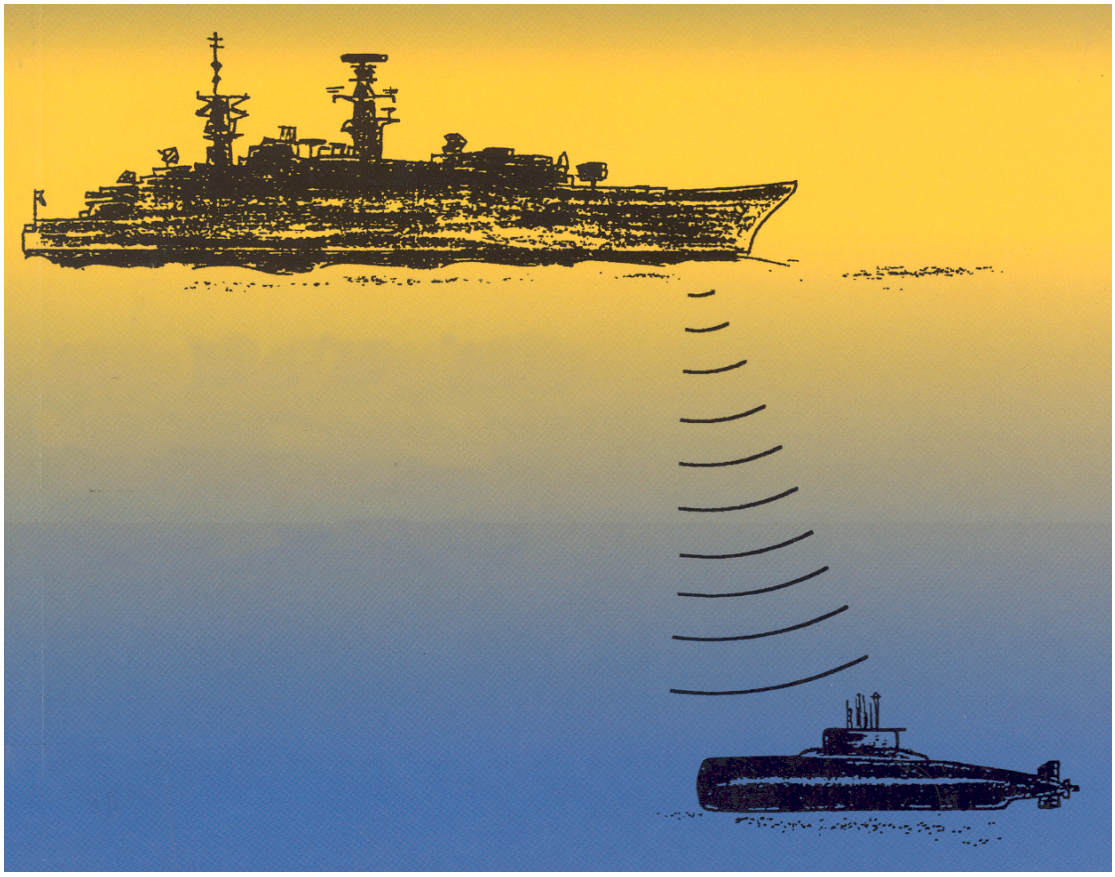


Illustration from [1]

ABSTRACT

The goal of visualization of sonar performance data in the underwater volume is to convey the information effectively to help the sonar operator understand and interpret the data in such a way that decisions can be made to adjust sonar parameters, and discard false alarms.

This paper will explore different visualization techniques, and show that the best techniques for achieving this goal are horizontal and vertical slicing in the sonar range volume. Slicing provide the best result with emphasize on correct information and unambiguous interpretation of the sonar performance data.

Other volume visualization and isosurface visualization techniques provide little additional information on the data compare to slicing, and some techniques such as isosurfaces of different levels of sonar performance data may cause ambiguity because of the richness of detail and scattering of the data.

Abbreviations

SONAR	Sound Navigation and Ranging
FFI	Forsvarets forskningsinstitutt Norwegian Defence Research Establishment
SIMSON	Simulering og evaluering av sonarsystemer og antiubåt operasjoner
GIS	Geographic Information System
PoD	Probability of Detection

1 INTRODUCTION

1.1 Background for Master project

The Royal Norwegian Navy is acquiring five new frigates - called the Fridtjof Nansen class. The first one will be delivered in the autumn of 2005. The new frigates will be multipurpose vessels, equipped with new technology and modern defensive features.

Each frigate will have two active sonars for detection of enemy submarines; one hull mounted sonar and a towed array sonar, which can be adjusted to various depths. Active sonar uses a projector to generate a pulse of sound, which travels through the water to a target and is returned as an echo to a hydrophone (underwater microphone), often the same device as the projector [1]. The sonars have different characteristics, including coverage. The sonars are capable of transmitting several types of acoustic signals in all directions horizontally or in selected sectors.

The platform (a term to describe any vessel or site possessing a sonar system [1]) also consists of a helicopter with a dipping sonar capable of deploying and retrieving the sonar far away from the frigate. A model of the platform is shown in figure 1.1.

Figure 1.1 Fridtjof Nansen class

Illustration from http://www.mil.no/fregatter

Acoustic signals loose energy as they spread in the underwater volume, this is called transmission loss. When sound is transmitted underwater it is scattered by marine life, inanimate matter distributed in the sea and the inhomogeneous structure of the sea itself, as well as by reflection from the surface and the sea bed. The component of the incident sound energy reflected back to the source is known as backscattering. This backscattered energy is reverberation [1].

The received signal is a function of surface reverberation, bottom reverberation, volume reverberation, noise from rain, wind, waves, towing vessel, others vessels etc., water temperature on different depths, and water salinity. The bottom reverberation varies with seabed topography and bottom type.

Signal excess can be used as a measure for sonar performance. Signal excess varies with range, bearing and target depth.

In modern sonar systems, computer models will be used to predict the sonar performance. The prediction data should be presented for the sonar operator in such a way that decisions can be made to adjust sonar parameters (e.g. sonar depth and tilt), and eventually the speed and course of the vessel [2].

The FFI Simson project is developing a simulation and evaluation tool for the Fridtjof Nansen class' anti-submarine sonars. The simulation tool will be used for tactics development at the naval base Haakonsvern in Bergen, e.g.:

- Simulate how manoeuvres of the vessels and changing of sonar parameters affect the sonar performance.
- Route planning: Find the optimal route for the frigate from position A to position B, which provide optimal observation conditions.
- The visualized sonar performance, together with electronic topography charts for large parts of the Norwegian coast made by FFI, will constitute a helpful tool in separating signals from enemy submarines from false alarms. A main contributor to false alarms along the cost is bottom reverberation.

The Royal Norwegian Navy acoustic model LYBIN uses incoherent ray theory to calculate transmission loss, noise, reverberation and detection probabilities for active sonar systems using a given set of environmental conditions and sonar parameters. In LYBIN the prediction data are visualized in 2D structures, as vertical or horizontal slices.

The data visualized in 2D in LYBIN are saved in binary files as numerical values in a matrix, with one dimension representing different ranges from the sonar (x-coordinate) and the other representing sea depths (y-coordinate). Each cell of the matrix contains a value representing the sonar performance in this particular point (range/depth). These data will be provided for each bearing around the sonar, and will constitute the basis for a 3D view of the sonar performance at one point of the route of the frigate.

The depths need to be corrected against seabed profiles, provided in separate files. The values of signal excess, transmission loss, reverberation etc. are computed in terms of sound pressure intensity in the acoustic model, and each value will be converted to decibel before visualization.

1.2 Problem to be addressed

The assignment of the Master project will be:

- Visualize the prediction data for sonar performance in 3D structures in the underwater volume.
    - 3D surface visualization.
    - Volume visualization of different levels of signal excess, transmission loss, probability of detection etc. on various ranges as well as depths, e.g. with use of different colours and degrees of transparency.
        - Visualize in a way where as little ambiguity as possible arises.
        - Avoid too many details (input of modelled data).
- Evaluate alternative methods for visualization.
    - The evaluation will emphasize on:
        - User friendliness
        - As much information as possible
        - Correct information
- Provide facilities for:
    - Slicing - vertical and horizontal.
        - Provide possibility to move the incision dynamically to search for the image that provides the information wanted.
    - Rotation and zoom.
    - Dynamic updating of the visualization as the frigate moves.
        - Sonar performance data are generated for each ping
        - No demand for visualization in real time.

If time allows, some additional features may be implemented:

- Combine different sets of simulation-data from LYBIN; e.g. visualize sonar performance data as a function of sonar depth or the speed of the frigate. Dynamic visualization of the sonar's range if it is lowered or raised to various depths, or if the speed of the vessel is altered.
- Update the visualization with a scrollbar / timeline. If necessary reduce the resolution of range and depth, and reduce the number of pings. Provide possibility to stop at a point in time and zoom into a higher resolution (if reduced).
- Generate a picture (gif, jpg) from each update, to be used to create an animation with software provided by FFI.
- 3D visualization of seabed topography with surface projection of data, e.g. bottom reverberation. The seabed will then constitute a part of the total structure, and will be revealed as slices are cut out. The sonar data structure cannot be rotated individually, but can be studied from different angles with pan.

1.3 Tools and technology

The FFI Simson sonar simulation and evaluation tool is developed using .NET [3] and C# [4]. My visualization tool will be integrated into the Simson tool at a later point, and therefore the same technology was chosen. As graphic libraries both OpenGL [5] and DirectX [6] were considered. FFI had little experience with either. OpenGL was chosen, i.a. since this is being taught at Østfold University College.

In order to be able to use OpenGL in C# and .NET, an OpenGL module for this purpose is needed. There are a number of such modules. CSOpenGL [7] was chosen, because of its compatibility with VisualStudio.NET. After setting up the OpenGL context, which requires very little code, OpenGL functions can be used without any need for class- and object-names. The CSOpenGL module contains all the gl-, glu- and wgl-functions OpenGL programmers are familiar with (see Appendix for example code).

However as I proceeded with my work, I experienced that this module has been very little used. There is hardly any example code to be found. I have experienced some problems with textures using C# and CSOpenGL. The images were rendered upside down, and it required additional code to turn the byte array the right way. Textures are used for labels showing the sonar data structure's orientation in space (see figures 3.9, 3.11 and 4.10).

1.4 Chapter overview

Chapter 2 describes some volume visualization techniques, and provides some examples from sciences where volume visualization techniques have been applied.

Chapter 3 describes the design choices made for the applications:

- Which visualization techniques have been evaluated, which have been chosen and why.
- Software architecture
- Graphic User Interface

Chapter 4 describes how the chosen techniques have been implemented, in addition to some experiments done on different datasets and additional visualization techniques. The chapter ends with user feedback from test users at FFI.

Chapters 3 and 4 provide a number of figures that show the architecture, GUI and visualization of the sonar performance data.

Chapter 5 sums up the experiences made from the work done – which visualization techniques give the best basis for interpretation of sonar performance data, and which improvements can be done.

2 RELATED WORK

2.1 Volume visualization

Visualization is the study of data presentation into visual form. Its ultimate goal is to convey the information effectively to help the viewer understand the data [8].

3D Surface Graphics define objects explicitly by a surface or boundary representation, discarding the interior of the object and just maintaining the object's shell.

Volume Graphics is a powerful technique to support the visualization of inner structures, and for the exploration of datasets. Different aspects of the dataset can be emphasized via changes in the functions that translate raw densities into colours and transparencies. Volume Graphics also facilitates real-world operations such as cutting, slicing and dissection [9].

Typically, the volumetric dataset is represented as a 3D discrete regular grid of volume elements (called voxels), stored in a discrete regular volume buffer V(x,y,z). A collection of all the values (attributes) associated with each point in a volume is called a scalar field on the volume [8].

3D matrix data representation of a volume grid and interpolation of data into a volume grid are techniques used in this Master project, and they are described in chapters 4.1 and 4.2. The volume grid has been used as a tool for horizontal slicing, described in chapter 4.2, and for visualization of isosurfaces with the Marching Cubes algrorithm, described later in this chapter and in chapter 4.4.2.

Colour mapping is a common scalar visualization technique that maps scalar data to colours, and displays the colours on the computer system. The scalar mapping is implemented by indexing into a colour lockup table. Scalar values serve as indices into the lookup table [10].

Various volume visualization methods have been developed. Generally there are two major categories: (direct) volume rendering, and level surface (isosurface) rendering. The direct volume rendering methods can be further divided into two sub-categories: image-space based (like ray tracing) and object-space based (like splatting) [8].

Volume visualization is a useful tool for research and computer simulations in professions and sciences such as medical imaging, oil reservoir characterization and nuclear research. Figures 2.1 and 2.2 show some examples of medical imaging using both 3D surface graphics and volume visualization (from [12]).

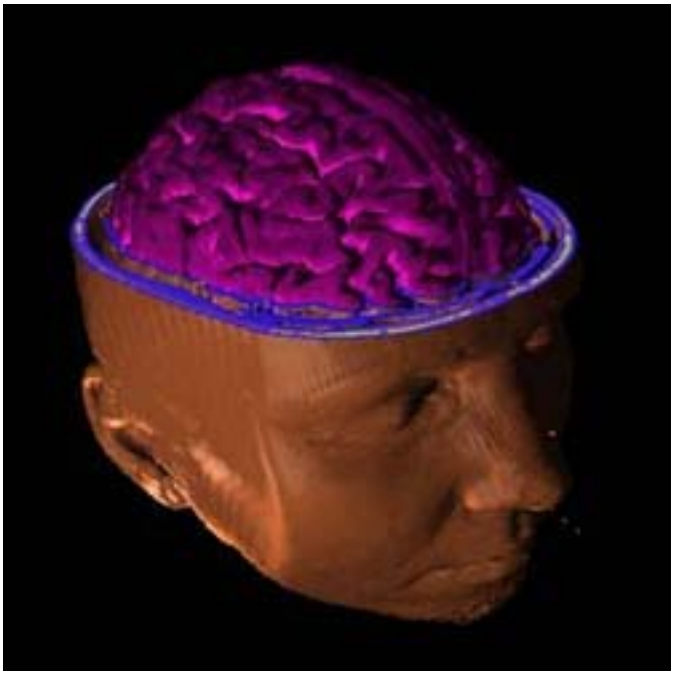


Figure 2.1 Surface of human head with brain cut

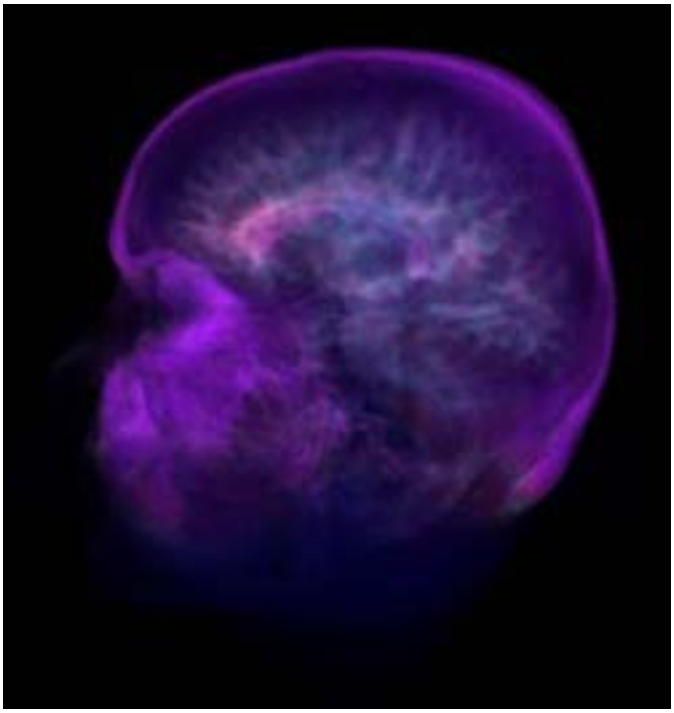


Figure 2.2 Volume visualization of human head

Figure 2.3 shows computational fluid simulation (from [13]).

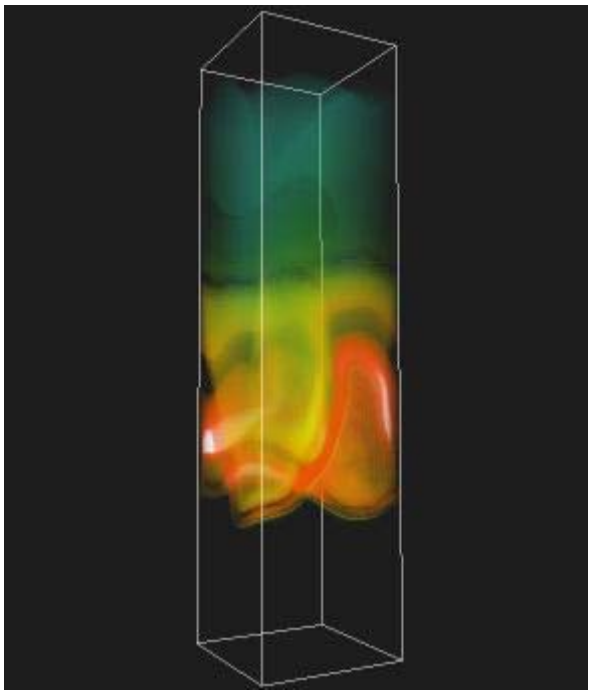


Figure 2.3 Computational fluid simulation

Three-dimensional visualization is important when the sonar performance varies for different bearings, and when environmental variables (i.e. sea bottom depth and sound speed profile) vary for different ranges and bearings. Sonar performance can be visualized as isosurfaces with constant signal excess value. Isosurfaces can be generated for different choices of signal excess levels, and visualized as white surfaces. A sector of the surface can be removed to expose internal details. There are several methods to generate isosurfaces, and many of them are based on the Marching Cubes-algorithm [2].

In this algorithm, scalar values are given at each point of a lattice in 3-space. A particular level surface can be approximated by determining all intersections of the level surface with edges of a lattice. We look for pairs of adjacent lattice points whose field values surround the desired value (i.e. the value of one vertex is greater than the chosen level, the value of the other is less). The location of an intersection of the level surface with the edge is then estimated by linear interpolation. Each cube in the lattice now has some number of edges marked with intersection points. The arrangement of the intersection points on the edges can be classified into 256 cases. For each case, a choice is made of how to fill in the surface within the cube. The collection of all the surface pieces just defined constitutes a surface [11].

Samples of isosurfaces of different levels of sonar performance data are shown in chapter 4.4.2 - figures 4.23 through 4.28.

To visualize the sonar performance data only in 3D surface structures would mean a loss of information, as only the data far away from the sonar would be visible. One way to explore the data inside the 3D structure is to cut slices. Volume visualization can be used to display all details of the dataset. This can be achieved by applying different degrees of transparencies on the data visualized.

One method is described in a FFI report on visualization of sonar performance [2]: First the data are converted to a regular grid. Then we use tables to convert signal excess values to red, green and blue colour components, together with a transparency value. Drawing many overlapping partial transparent images then creates the final image. The transparency can be adjusted to form compact or blurred objects.

In OpenGL transparency can be achieved by using colour blending. The colours are specified in RGBA-mode, with A representing the alpha-value. This value is used to combine the colour value of a new fragment with that of the pixel already stored in the frame buffer. Thus, one can use alpha blending to create a translucent fragment, one that lets some of the previously stored colour value "show through" [14].

Figure 2.4 shows an example of colour blending using different alpha-values on the front facing and back facing sides of a cube. The back, bottom and right faces are made opaque, while the front, top and left faces are made translucent with an alpha value of 0.75. This provides an effect of transparency. The image has been generated using .NET, C# and CSOpenGL.

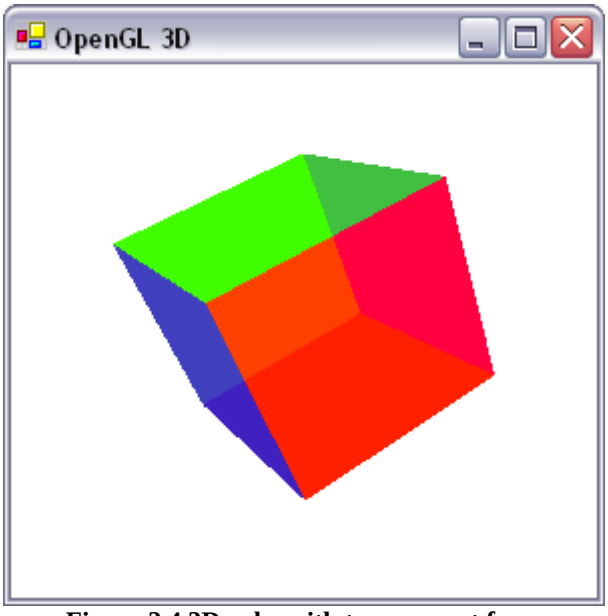


Figure 2.4 3D cube with transparent faces

3 DESIGN

3.1 Main choices and evaluation

Analysis of Algorithm:

T	computation time / complexity
N	input data
T(N)	worst case time complexity: Time needed by an algorithm to complete execution as a function of size of input N

Rules for analysing algorithms:

- The running time of assignments, comparisons, arithmetic operations etc. is a constant.
- The running time for a loop is equal to the running time of the statements inside the loop times the number of iterations.
- The running time for a group of nested loops is the running time of the statements multiplied by the product of the sizes of all the loops.

[15]

O-notation gives us a language for talking about the comparative efficiency of algorithms. The definition of O-notation is constructed so as to achieve three important properties:

1. Focusing on the dominant term for large problem sizes.
2. Ignoring the lesser terms and ignoring what happens on small problems.
3. Ignoring the constant of proportionality.

Standard Complexity Classes:

Constant	$O(1)$
Logarithmic	$O(\log n)$
Linear	$O(n)$
N log n	$O(n \log n)$
Quadratic	$O(n^2)$
Cubic	$O(n^3)$
Exponential	$O(2^n)$
Exponential	$O(10^n)$

[16]

3.1.1 3D Visualization

There are a number of possible techniques for 3D visualization of the sonar prediction data. Two methods I have explored involve isosurface visualization:

1. Using the Marching Cubes algorithm to generate the outer surface of the sonar performance data structure by using the values in a volume grid. This technique is described in chapters 2.1 and 4.4.2.
2. Using the outer sonar performance values from each bearing to draw squares or triangles with OpenGL. This is described in chapter 4.3.

The Marching Cubes algorithm proved to be too inefficient on data grids of this size. A 50 x 50 resolution of the data generated a grid with size 100 x 100 x 42 (various depths), and a 100 x 100 resolution generated a grid with size 200 x 200 x 76.

The Marching Cubes algorithm runs through the entire grid for each computation to find the surface boundary. The algorithm uses interpolation to find new grid points between existing ones.

Using algorithm analysis to compare the efficiency of the algorithms, the O-notation for the Marching Cubes is:

$$T(N) = O(N^2 * M)$$

N being the loops running through the j- and k-indices of the 3D matrix, and M being the loop running through the i-index. Because of various depths the number of iterations of the outer loop (i-index) will vary.

In addition there are two smaller loops nested inside these three, one that goes through the 256 different possible combinations of intersection points in a voxel, and one drawing triangles between three vertices.

The computation of grid points being inside or outside the isovalue was done when the grid was initialised.

The second algorithm runs through two loops; the outer loop runs through the collection of bearings from which data are available, and the inner loop runs through the outer column of each 2D data matrix containing the data. When data for 360 bearings are provided, no interpolation is necessary.

The O-notation for this algorithm is:

$$T(N) = O(N * M)$$

This last technique proved to be the most efficient, and is therefore used in the final solution.

For both techniques, different colours were used to visualize different levels of sonar performance values (colour mapping – see chapter 2.1). Signal excess and

transmission loss data use the same colour scale, consisting of 13 colours. Probability of detection data uses another colour scale, consisting of 8 colours. The colour scales used are the same as those used in the acoustic model LYBIN (see chapter 1.1).

3.1.2 Volume visualization

I have also explored two methods of volume visualization of the sonar prediction data:

- Visualizing the entire volume using points, cubes or spheres at each grid point in a volume grid, alternatively a reduced number of grid points.

The O-notation for this algorithm is:
$T(N) = O(N^2 * M)$

Using cubes to visualize the volume provides a solid structure, which gives a good impression of the data when making vertical slices. When no slices are made, the cubes within the structure are invisible and therefore superfluous, and require a lot of unnecessary computer processing to be generated.

Using points and different levels of detail (i.e. number of grid points visualized) provides a more efficient visualization technique. I have experimented with three levels of detail, where a higher number means more details:

Lod 1 – every 4th grid point is visualized.
Lod 2 – every 3rd grid point is visualized.
Lod 3 – every 2nd grid point is visualized.

The O-notation from the last algorithm can be divided with 4, 3 and 2 respectively. With a level of detail of 3, the user gets a fairly good impression of the data in the volume (see figure 4.13).

Level of detail (LOD) is explained in chapter 4.1.

- Visualize different levels of sonar performance values using isosurfaces and transparency.

The O-notation for the marching cubes algorithm must be multiplied with the number of isosurfaces. In addition three nesting loops (i-, j- and k-indices of the 3D matrix) calculating whether each grid point is inside or outside the isovalue must be added for each isosurface.

Using isosurfaces and transparency might be an efficient visualization technique when the surfaces are smooth. A demonstrator was made using cylinders, which gave a good result (see figure 4.22).

However, visualization of sonar performance data does not provide smooth surfaces. Experiments were conducted using the Marching Cubes algorithm with different isovalues on different data types (see chapter 4.4.3 – figures 4.23 through 4.28). The data values are strongly varying, which results in many tiny isosurfaces. This is especially true for signal excess data. Isosurfaces of transmission loss data seem to provide a better result.

This visualization technique provides little or no additional information on the sonar performance data compared to the volume visualization technique described above. In fact, it must be applied with caution to avoid a confusing impression of the data.

3.2 Software architecture

Active sonars use an array of projectors to transmit acoustic pulses into the water. An active sonar signal transmission is known as a ping. A ping may be simply the transmitted pulse or sequence of pulses, or it may be the total time between transmissions. The term "ping" is therefore ambiguous, but the meaning is usually made clear by the context in which it is used. The ping interval is an alternative term for the time between transmissions [1].

In this context, the term ping is used for one transmission of acoustic signals. The ping interval may vary.

In order to predict the performance of a sonar sensor in a specific environment, an acoustic model is run to estimate the performance. For this thesis in particular, the Royal Norwegian Navy acoustic model LYBIN (see chapter 1.1) was used in generating the sonar performance data.

The modelled sonar performance data are saved in binary files, one for each ping, with data sorted by bearing. In addition an information file is provided, containing the following information:

- The depth of the sonar
- The position of the frigate at each ping.
- Number of pings
- Range and depth of the area from which data are provided
- Resolution of the data
- Number of bearings
- The bearings from which data are provided

The position of the frigate is used to calculate the course and speed of the frigate between each ping (see figure 3.8).

3.2.1 Interpolation application

The sonar performance values and depth values are interpolated into a volume grid (see chapter 4.1). This is a time consuming task, and is therefore separated from the visualization. The interpolation application reads data from each ping, interpolates the data to a volume grid, and writes the interpolated values to new files, one for each data type and ping. The size of the grid is calculated according to the resolution of the data provided, and the maximum depth of the area.

The interpolation application needs to be run only once each time a new dataset is provided.

The data types read are:

- Transmission Loss
- Signal Excess

- Probability of Detection
- Seabed Profiles

Signal excess and transmission loss data are provided in sound pressure intensity units, and are converted to decibel (dB) by the application. Probability of detection data are provided in percentage, and seabed profile data in metres, and neither need any conversion.

Figure 3.1 provides an overview over the classes in the Interpolation application.

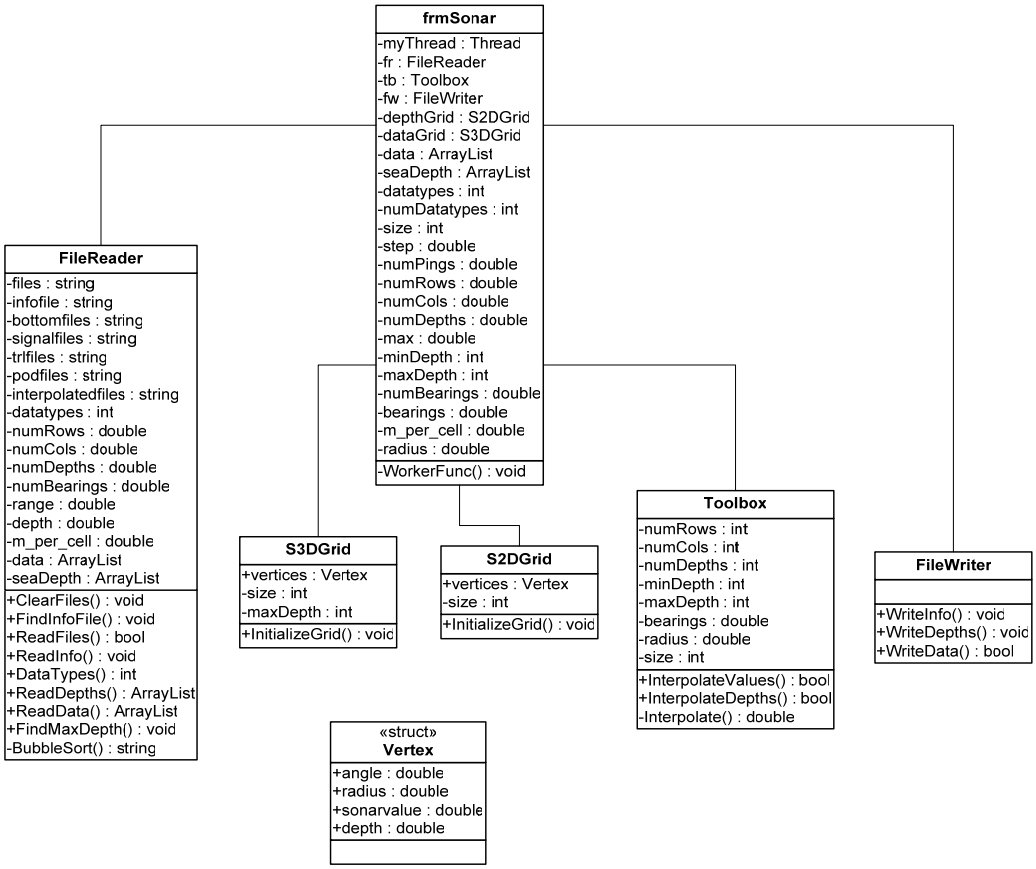


Figure 3.1 UML Class Diagram for the Interpolation application

Figure 3.2 provides an overview over the method-calls and interaction between classes in the Interpolation application. One S3DGrid for each data type is generated. The methods Read Depths, Find Max Depth and Read Data are repeated for each ping. The seabed profiles are necessary for the visualization of any type of data. If these files are missing, the application will exit with a message.

If any of the data types are missing, the interpolation application leaves out this data type from the interpolation process. An information file for the visualization application is written; containing the data types available, maximum sea depth in the sonar range area for each ping and size of the volume grid.

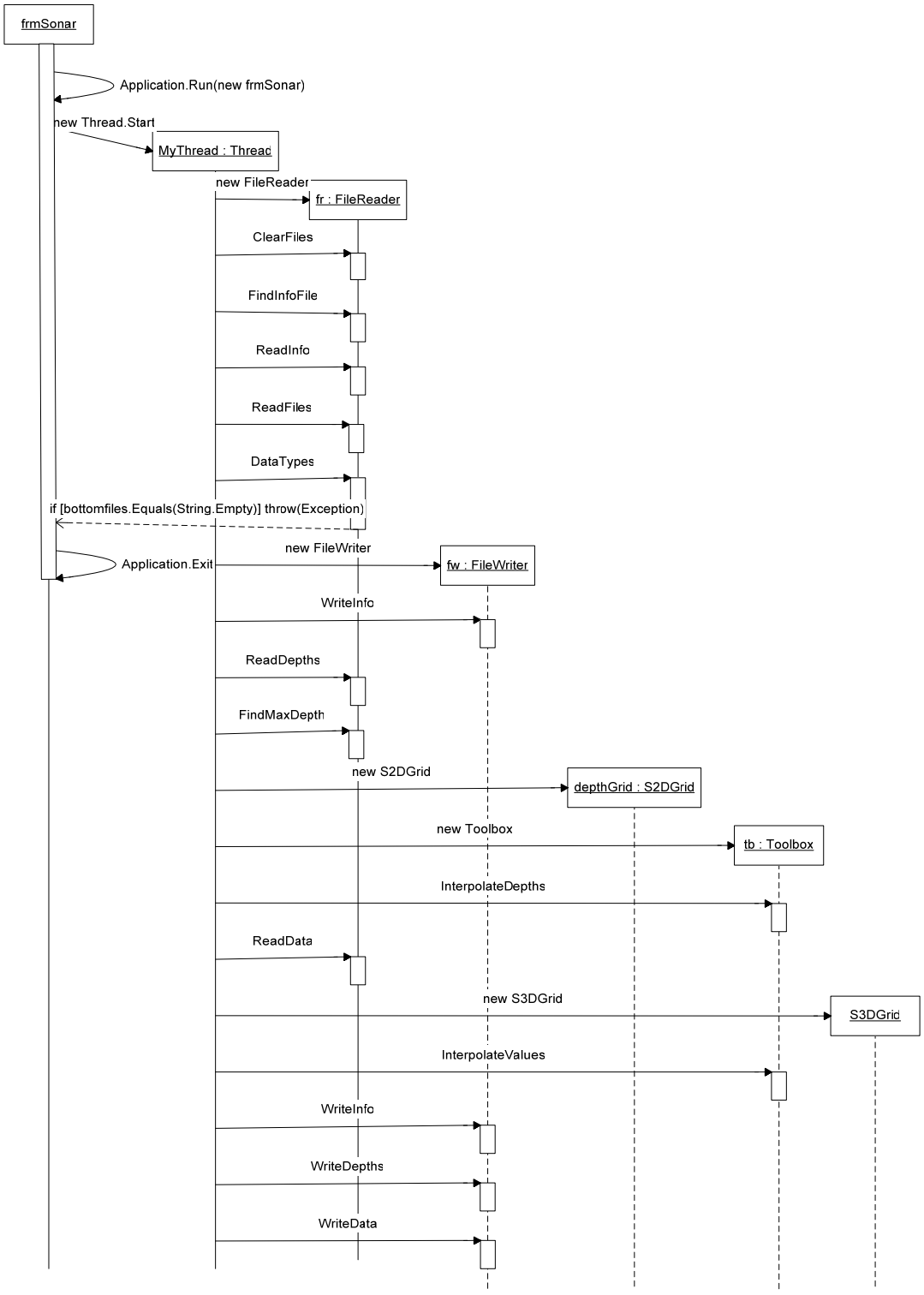


Figure 3.2 UML Sequence Diagram for the Interpolation application

The user can choose to cancel the interpolation process. The Thread is aborted, and a message to clear all files is sent to the File Reader object (see figure 3.3).

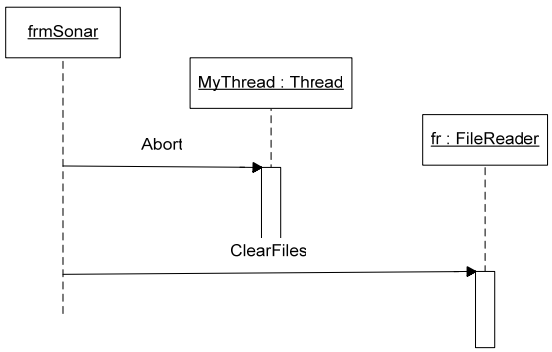


Figure 3.3 UML Sequence Diagram for cancelling the interpolation

3.2.2 Visualization application

The original matrix representation of the sonar performance data and the interpolated values are both available for the visualization application.

If the files with interpolated data exist, the values are read into volume grids, one for each available data type and ping. The interpolated depth values are read into the same grids.

The volume grids are used for visualization of horizontal slices, and for volume visualization with points and Marching Cubes. The original matrix representation of the sonar performance data is used for visualization of vertical slices, and for surface visualization.

Figures 3.4 and 3.5 provide an overview over the classes in the Visualization application.

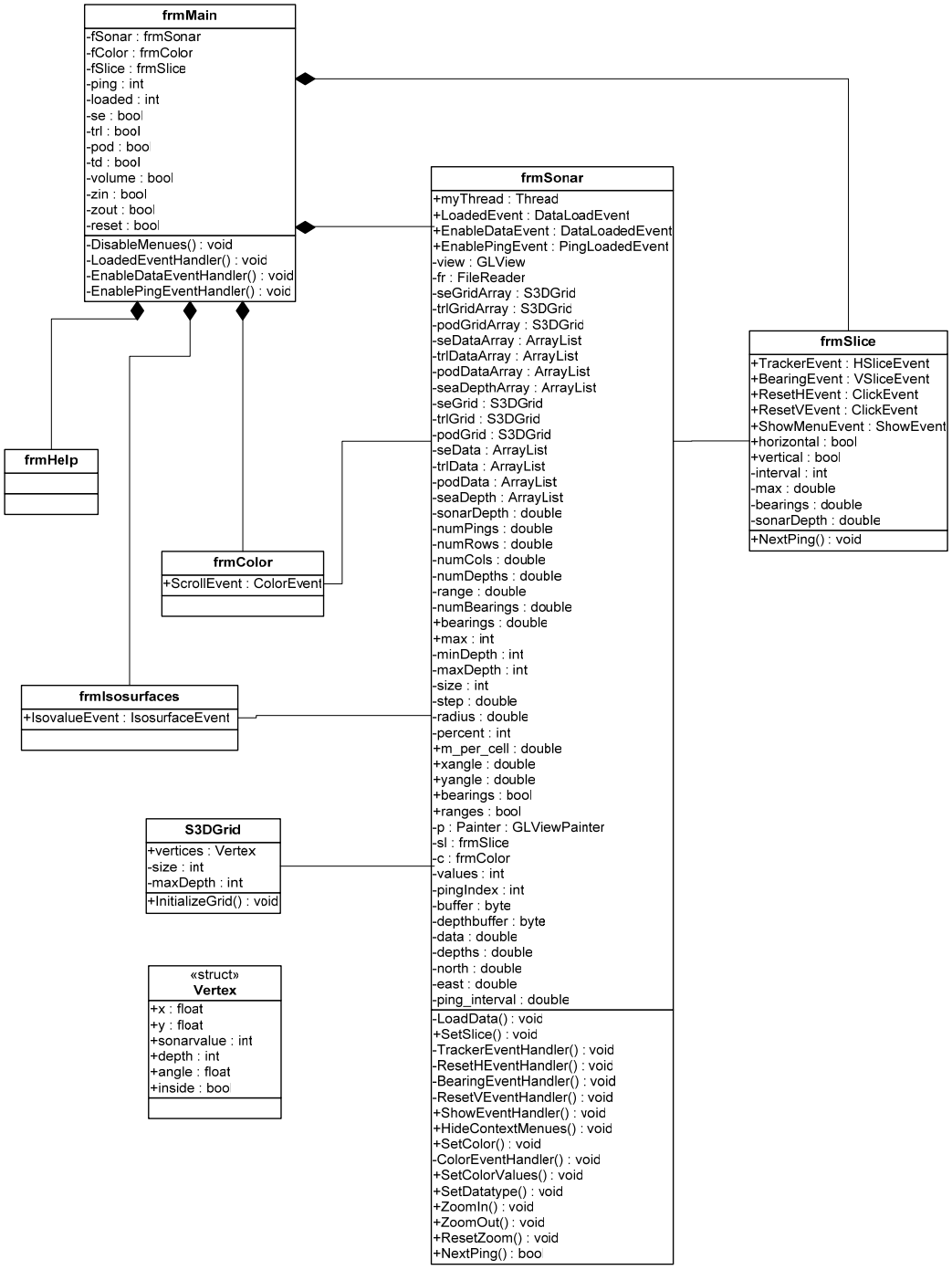


Figure 3.4 UML Class Diagram for the Visualization application

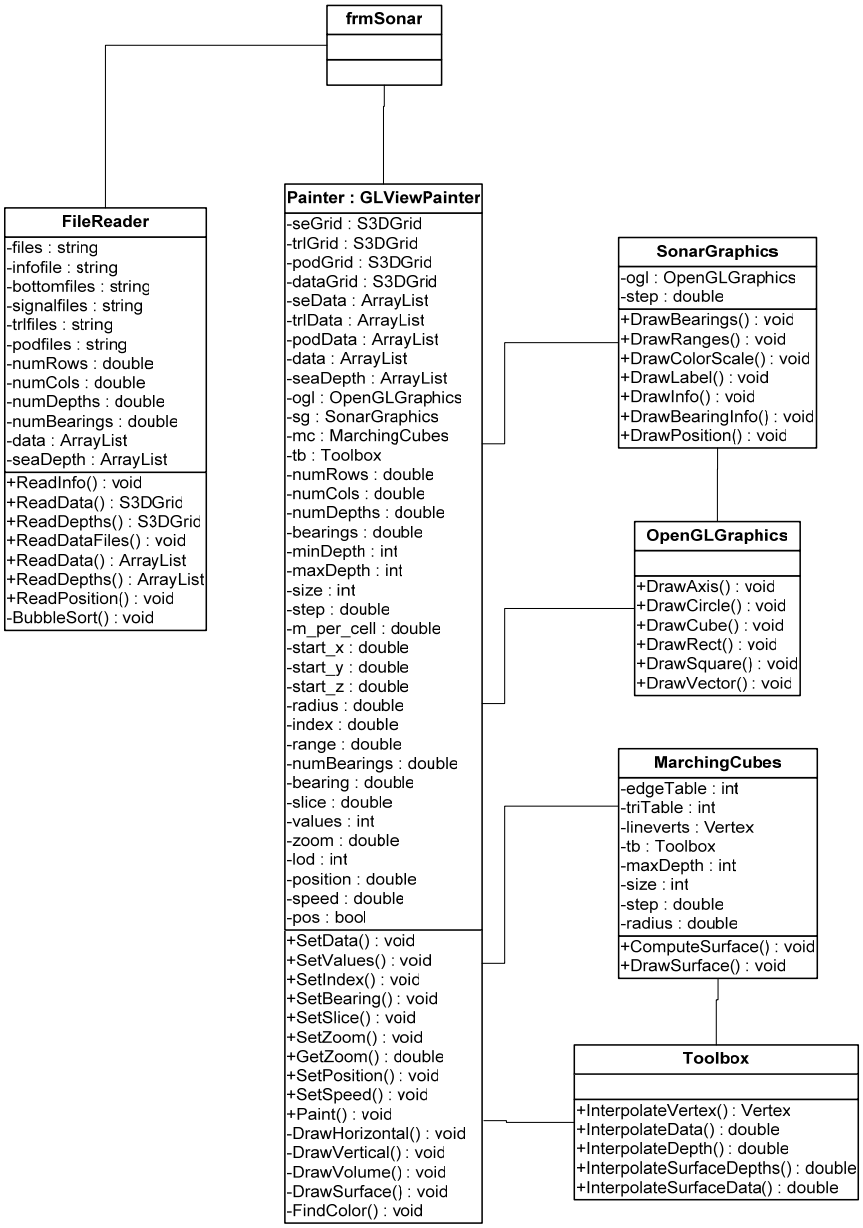


Figure 3.5 UML Class Diagram for the Visualization application continued

Figure 3.6 provides an overview over the initialisation of windows in the Visualization application. The main window starts the application, and initialises the child windows. When a menu item for choosing data type is clicked, the default values for the chosen data type are set.

When the user changes the colour settings in the Colour window (frmColor), makes horizontal or vertical slices in the Slice window (frmSlice), or chooses isovalues for isosurfaces in the Isosurfaces window (frmIsosurfaces), events are raised by these windows and handled by the visualization window (frmSonar) (see the class diagram in figure 3.4 – events and event handlers).

If the user clicks the menu item for Help, a help window is initialised.

If the user clicks the menu item for View -> Isosurfaces, a window for choosing isovalues and number of isosurfaces is initialised.

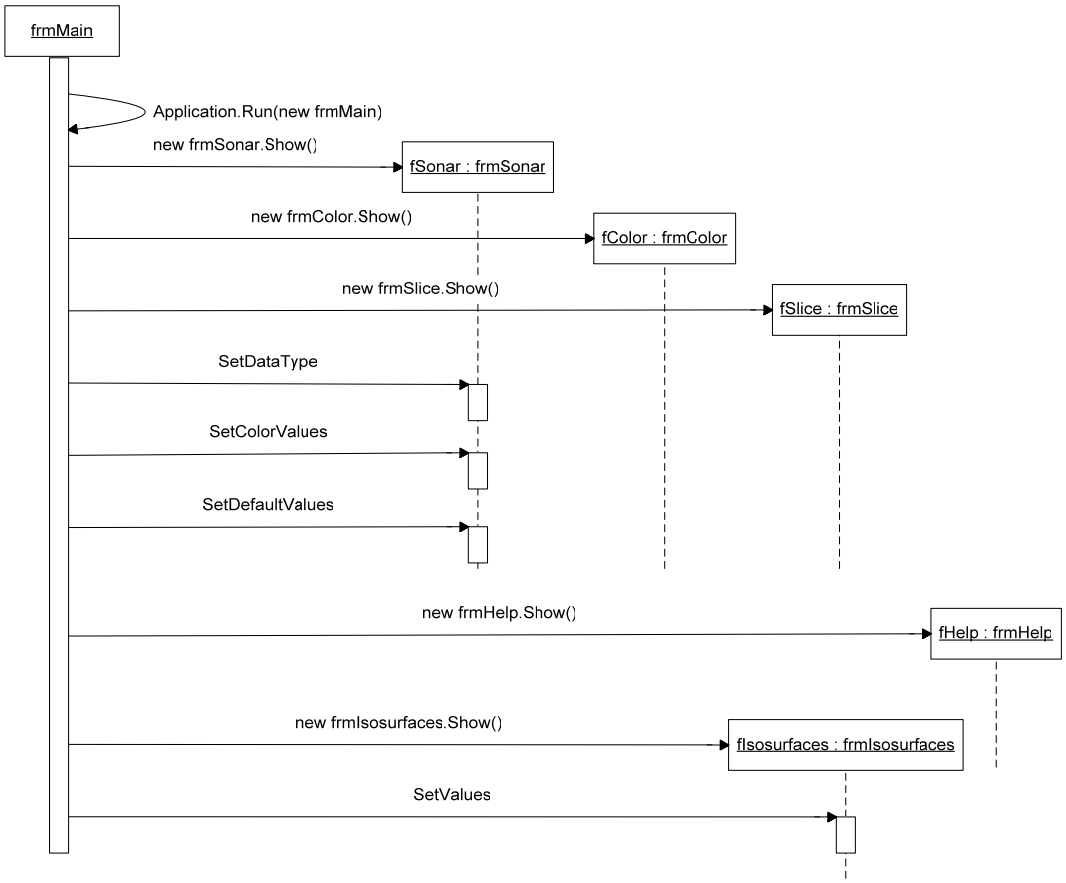


Figure 3.6 UML Sequence Diagram 1 for the Visualization application

Figure 3.7 provides an overview over the method-calls and interaction between classes as the Visualization application is started.

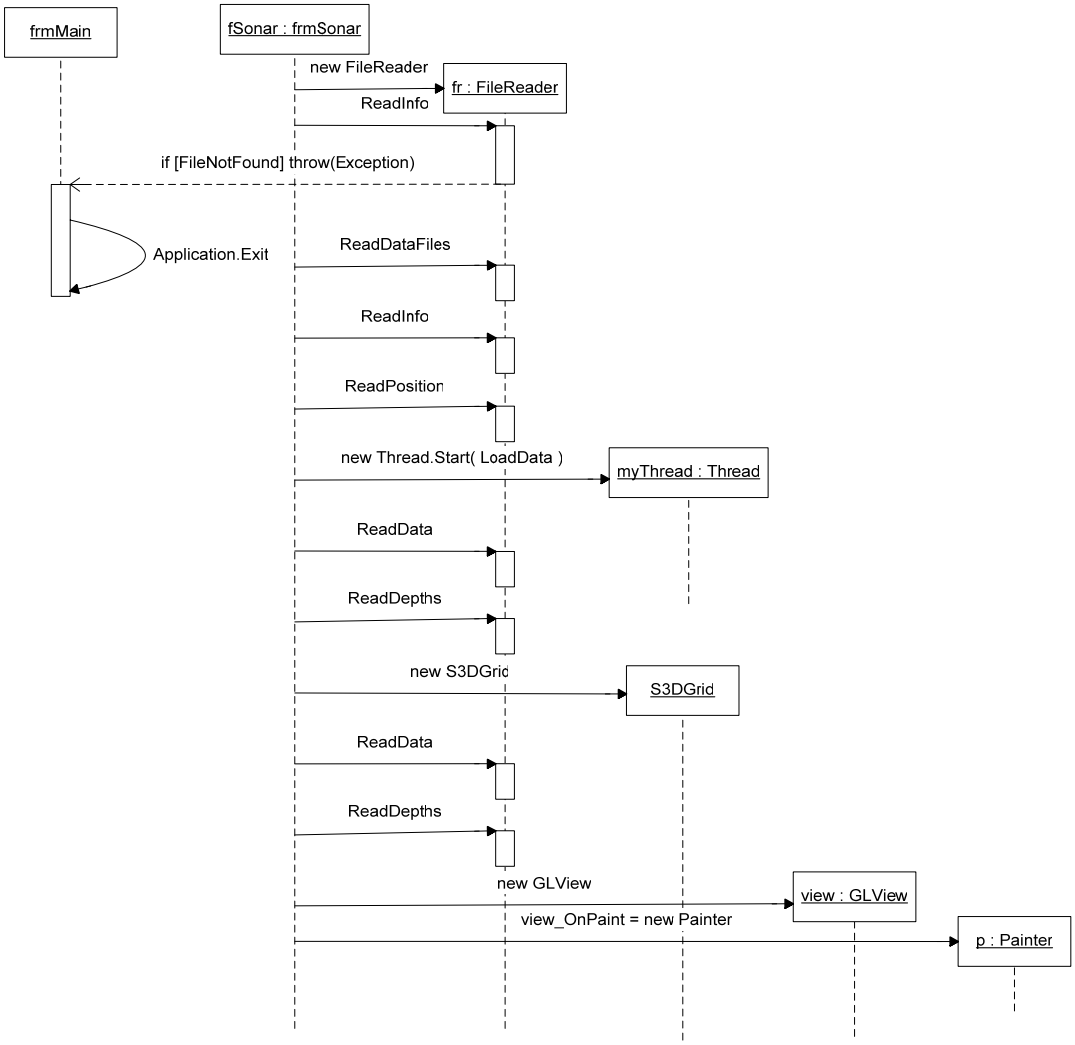


Figure 3.7 UML Sequence Diagram 2 for the Visualization application

If the files with interpolated data do not exist (the interpolation application has not been run, or the interpolation process has been cancelled), the visualization application will exit with a message that no files can be found.

The first Read Info method reads the information file from the interpolated data. If this file does not exist, an Exception is thrown by the FileReader object, and caught by the main window by exiting the application.

The methods Read Data Files, Read Info, Read Data and Read Depths read the original data into 2D matrices, one for each bearing. The matrices are collected in an Array List, which holds data from all bearings for each ping.

One S3DGrid for each data type is generated, and the last Read Data and Read Depths methods read interpolated data into the grids.

When the application starts, a thread starts in the background and reads data from all pings into arrays of Array Lists / grids. When data from the first ping is read, the menu items for choosing the data types available are enabled, and these data can be visualized and explored. Menu items and toolbar buttons for choosing next / last ping or automatic update of pings (see chapter 3.3 on Graphical User Interface) are disabled until all data are read. The status of the data reading is shown in a status bar.

Data from all pings are read into memory to enable a fast visualization of data from different pings. Data are compressed from type double to integer to reduce the size of computer memory needed to store the data.

The status of the reading and enabling of menu items and toolbar buttons are handled by events raised by the visualization window and handled by the main window (see the class diagram in figure 3.4 – events and event handlers).

A GLView object is generated, and its OnPaint event is set to a Painter-object. Whenever a change is made, the view object is told to refresh itself, and the Painter-class Paint method is invoked. The Painter-class handles all OpenGL drawing. Some of the draw-methods are moved to separate classes – SonarGraphics, OpenGLGraphics and MarchingCubes (see the class diagram in figure 3.5).

3.3 Graphical User Interface

The FFI Simson evaluation and simulation tool is integrated into Teleplan's Military Mapping Application Maria [17], and I have chosen a similar design for the graphic user interface of my visualization tool, shown in figure 3.8. The image shown in the visualization window is a horizontal slice of signal excess data from the 3rd ping available. The speed and course of the frigate is calculated using the difference in the frigate's position in this and the last ping, and visualized with a velocity vector pointing in the direction of the frigate's course. The velocity vector is also available in 3D for all 3D and volume visualizations.

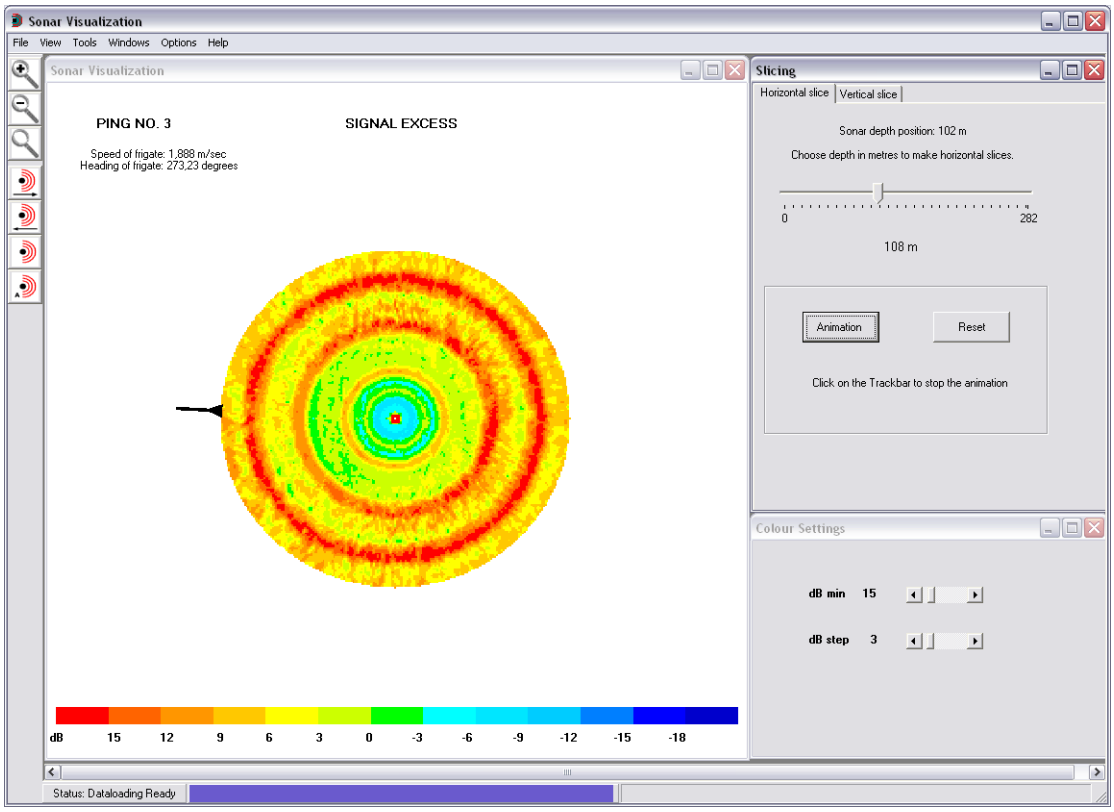


Figure 3.8 Graphical User Interface showing horizontal slicing

The GUI uses MDI (Multiple Document Interface) with a Parent Form/main window and various Child Forms/windows. The main window contains menus where the user can:

- File-menu:
  - Choose data type to visualize
  - Exit application
- View-menu:
  - Choose 3D or volume view
    - Volume with points with different levels of detail
    - Volume with isosurfaces and transparencies

    - Show a toolbar with tools for zoom and ping
- Tools-menu:
    - Zoom in / out, reset zoom
- Windows-menu:
    - Open the different windows if closed
    - Show all open windows
    - Select which window should be active
- Options-menu:
    - Choose next / last ping
    - Choose automatic update of pings
- Help-menu:
    - Open a Help-window

The automatic update of pings is not visualized in real time. The case data provided with the Master's degree project have a ping interval of 30 seconds. A visualization with 30 seconds between each update would be far too slow, and the pings are updated every second.

Child Forms:

- Visualization window
  The main child form is the visualization window. Default values are applied to the colour scale used.
- Colour Settings
  The user can choose which minimum value in dB and step between values in dB (intervals for the colour scale) will be used for the visualization of signal excess or transmission loss. When visualizing probability of detection, these options are disabled.
- Slicing
  The user can choose horizontal or vertical slicing.
- Isosurfaces
  The user can choose number of isosurfaces (maximum 3) and which sonar performance values to be used as isovalues.
- Help
  The user can choose help topics from a drop down list.

Horizontal slicing:

Moving the thumb in the track bar and in this way choose the depth in metres where the slice should be made makes horizontal slicing. The maximum value for the track bar is the maximum sea depth in the area for which sonar data are provided. The interval between depths where slices can be made is determined by the resolution of the data visualized.

Clicking the Animation-button can also make horizontal slicing. The thumb of the track bar will move, changing the horizontal slice visualized. Stop the animation by clicking the track bar, and resume animation by clicking the Animation-button again. The Reset-button will reset the track bar and the visualization window to 0 (sea surface).

In horizontal slicing mode, a context menu is available in the visualization window. The user can choose to show or hide bearing lines and range circles (5 circles; i.e. if the range of the sonar is 10000 m, one circle per 2 km). Bearings 0°, 90°, 180° and 270° are labelled using transparent gifs as texture in OpenGL. In figure 3.9 both bearings and ranges are shown, they may be shown or hidden individually as demonstrated in figure 3.10.

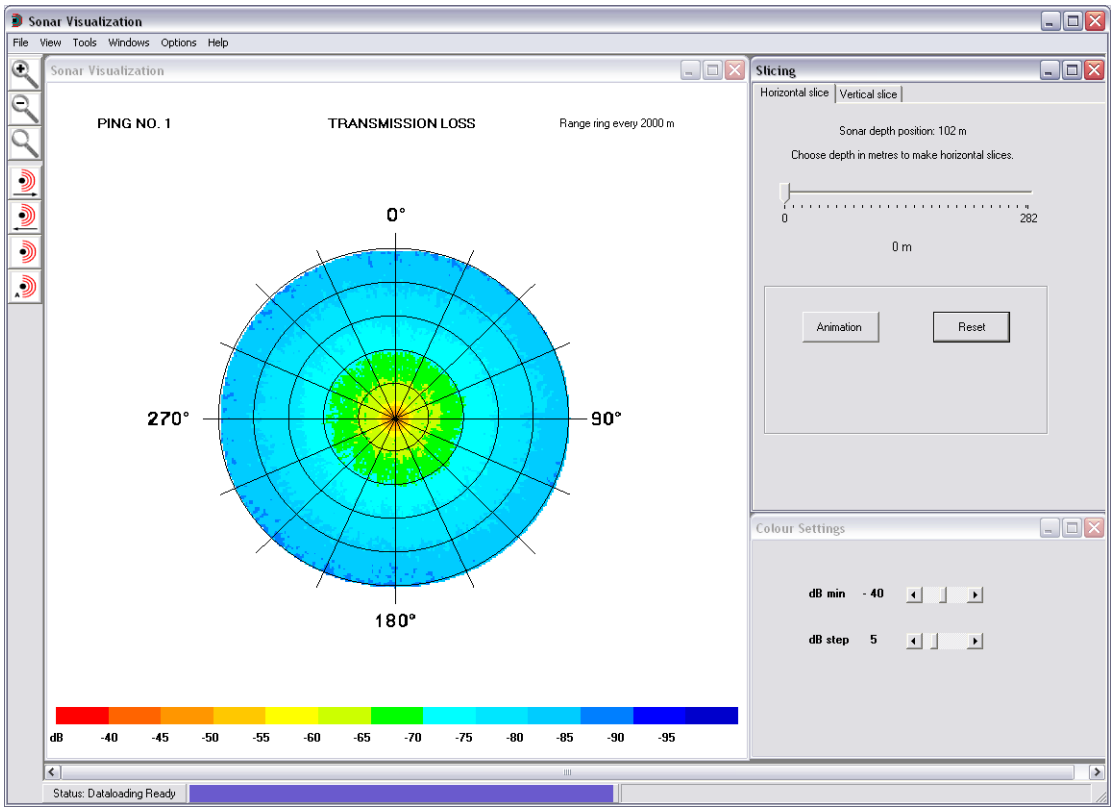


Figure 3.9 Horizontal slice showing both bearings and ranges

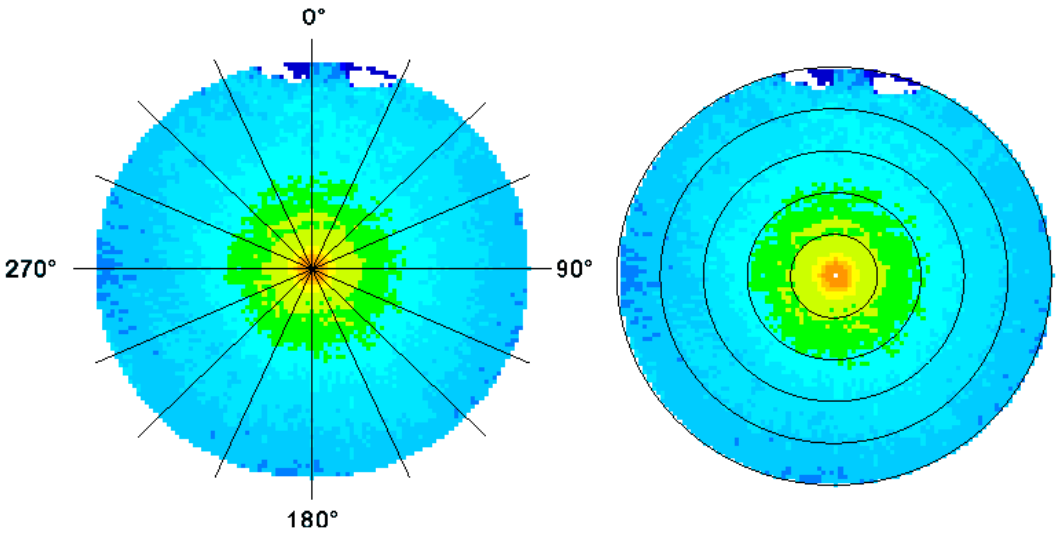


Figure 3.10 Horizontal slice showing either bearings or ranges

Vertical slicing:

The user chooses the size of the slice to be made by typing in the size in degrees. If the user does not write anything in this field, the default value 90° is chosen. If typing a number equal to or greater than 360, the actual degree is calculated with modulus 360, and shown in the size field. Correct input is limited to integer numbers using a regular expression.

The bearing to make the slice from is chosen by moving the thumb in the track bar. A slice can be made from / to bearings 0 through 359. If there are no data provided for the selected bearings, data are interpolated from the nearest bearings (see chapter 4.1). The slices can be rotated about the y-axis, and the orientation in space is indicated with a figure showing sonar bearings 0°, 90°, 180° and 270°, with the bearings of the slice indicated in red, moving as the slice is moved.

In figure 3.11 a vertical slice of 135° has been made from 115° (right hand side of the figure) to 250° of transmission loss data, and zoomed in for a better view. The data are from the towed array sonar, and the red point in the middle of the structure identifies the sonar's depth position.

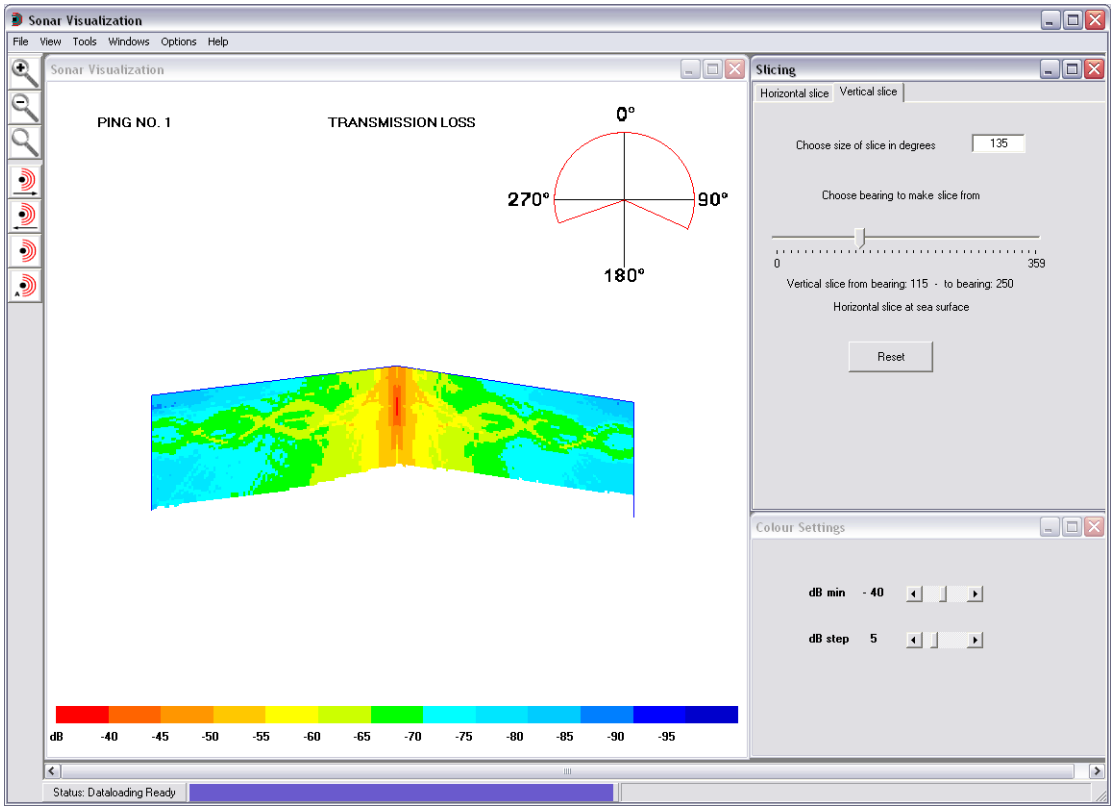


Figure 3.11 Graphical User Interface showing vertical slicing

In vertical slicing mode another context menu is available in the visualization window. The user can choose to show or hide a 3D view of the sonar data structure as the vertical slices are made (see figure 4.11). When 3D view is hidden, a line is drawn around each bearing to separate them visually.

Any horizontal slice made is visualized together with the vertical slices, and the user can toggle between horizontal and vertical slices using the tabs. In figure 3.12 a horizontal slice at sea depth 56 m has been made previous to the vertical slice from 115 ° to 250 °, this time of signal excess data.

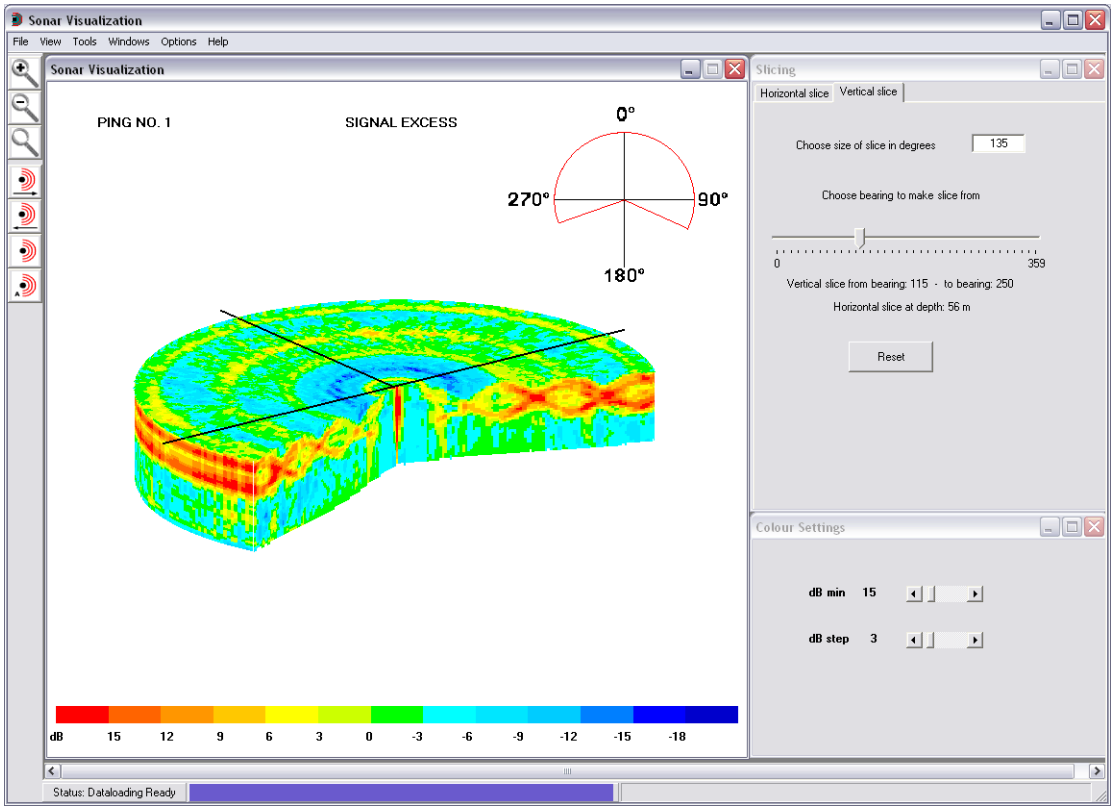


Figure 3.12 Graphical User Interface showing horizontal and vertical slicing in combination

The Reset button will reset the size of the vertical slice to default 90°, from 0° to 90° bearing, and the horizontal slice to sea surface level.

Isosurfaces:

When the user chooses Isosurfaces from the View menu, an additional window is opened. The number of isosurfaces (maximum 3) to visualize is chosen from a drop down list. The sonar performance values available for the chosen data type is loaded into a list box when the window opens, and the user can choose a number of values corresponding with the number of isosurfaces.

In figure 3.13 two isosurfaces with isovalues –60 dB (red) and –70 dB (orange) have been chosen for transmission loss data. The colours used for the visualization of isosurfaces, are red with an alpha-value of 1.0, and orange and yellow with an alpha-value of 0.75 (see chapter 4.4.3 for further results and explanation of the choice of colours). The colour scale used for visualization of horizontal and vertical slices and 3D and volume view with points is not used for isosurfaces and therefore no longer visible in the visualization window.

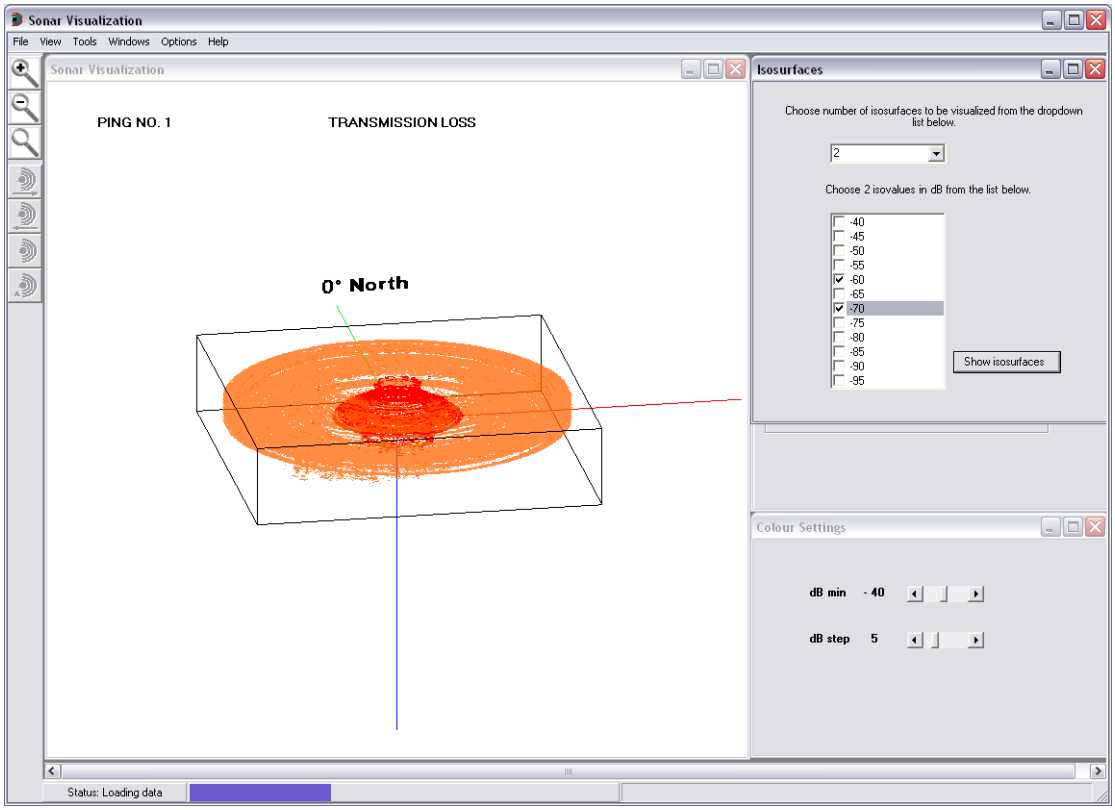


Figure 3.13 Graphical User Interface showing isosurfaces

The figure shows the display as the data are still loaded into memory, and the toolbar options for choosing next ping, last ping, reset ping and automatic update of pings are disabled.

4 IMPLEMENTATION

4.1 Interpolating sonar performance values to a volume grid

A volume grid provides a tool for efficient volume visualization. Volume datasets are generally much larger than 3D surface datasets (such as polygon mesh surfaces). Level of detail (LOD) is an important mechanism for achieving a high level of performance in a 3D virtual world. It balances the quantity (extent) of an object with its quality (detail). Different levels of detail may be implemented in different ways, for instance using images with different resolutions.

A volume grid provides a tool for implementation of an efficient lod-algorithm. In order to reduce the computation time for a volume visualization application, the volume dataset can be reduced by dividing the number of points in the volume grid, e.g. by 2 at the time. The number of divisions must be balanced against the request for quality of the image visualized.

A volume grid also provides a tool for 3D surface visualization algorithms, such as the Marching Cubes, described in chapter 2.1.

The sonar performance values simulated in the acoustic model LYBIN (see chapter 1.1) are saved in 2D matrices, one for each bearing, with one dimension representing different ranges from the sonar (x-coordinate) and the other representing sea depths (y-coordinate). Each cell of the matrix contains a value representing the sonar performance in this particular point (range/depth). A set of matrices contains values from a set of bearings/depths, and constitutes a 3D structure.

Each horizontal slice in the 3D structure represents a given depth, and the slice forms a fan with one dimension representing the range of the sonar, and one dimension representing the bearing as in a polar coordinate system. The range dimension contains discrete sonar performance values, simulated for evenly separated points in range. This is illustrated in figure 4.1.

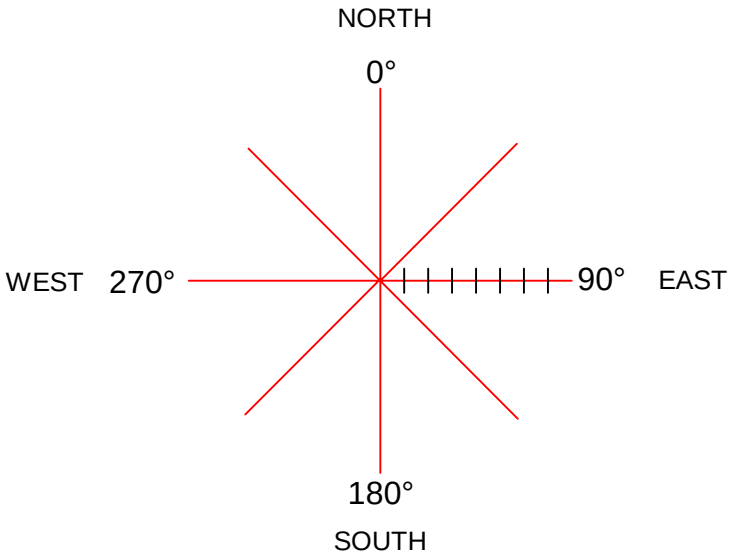


Figure 4.1 Sonar data structure for one sea depth represented using modified polar coordinates

For an explanation of the angles in figure 4.1, see figure 4.5.

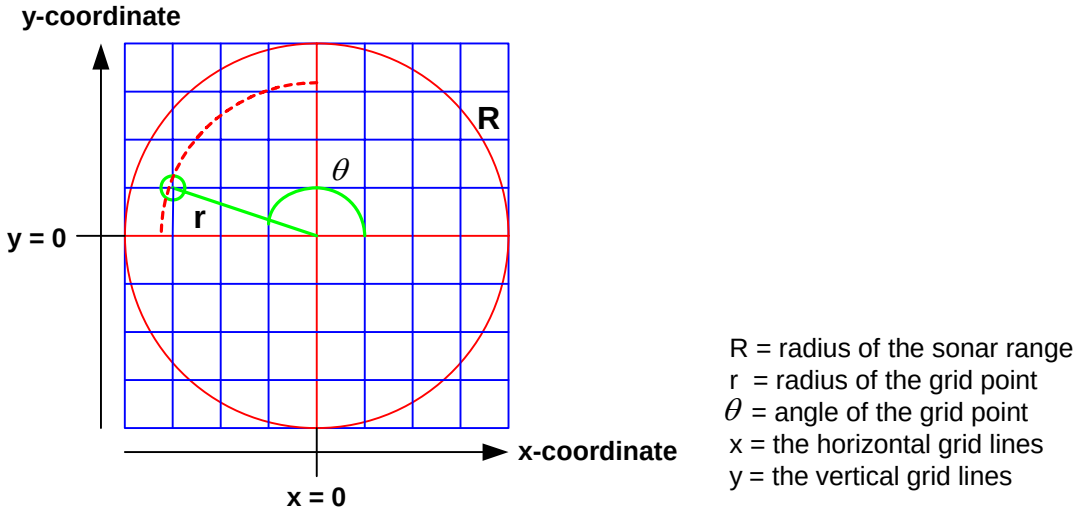


Figure 4.2 Calculating radius and angle (polar coordinates) for grid-points (Cartesian coordinates)

The sonar data structure for one horizontal slice is illustrated with red in figure 4.2 (simplified to show the main sonar bearings). The sonar performance values are sampled into points, arranged in a volume grid. The grid lines are illustrated with blue lines. Each point in the grid is assigned a sonar performance value by applying linear interpolation between the two nearest sonar data bearings.

The algorithm for interpolating a horizontal slice of the cube is:

1. Calculate the angle and radius of a given point in the grid (illustrated with green in figure 4.2) using the following equations:
   a. R = radius of the sonar range
      x = the point on the horizontal grid line
      y = the point on the vertical grid line
      x, y = grid point; cross point between horizontal and vertical grid lines
      Radius of the grid point: r = $\sqrt{x^2 + y^2}$ * R
   b. Angle of the grid point: θ = (arccosines$(x/r)/2\prod$) * 360
      Arccosines computes the angle in radians.
      The angle is then computed into degrees, to compare with the sonar bearings in degrees.
      If y < 0 (lower half of the circle) θ = 360 - θ
2. Find the corresponding point in the nearest angle on either side of the grid point using the radius of the grid-point (dotted red line in figure 4.2).
3. If these points lie between value-points, interpolate intermediate values, one for each angle.
4. Interpolate an intermediate value for the grid-point from the values on the nearest angles, using the grid-points angle.

Interpolation is done using a linear interpolation algorithm:

When interpolating between value points on one angle (step 3 in the algorithm), r is the radius of the grid-point, r_α and r_β are the radius of the two nearest values in the sonar angle's data-table, α and β are the values, Δr is the step between values in the table, and r_γ is the new value. This is illustrated in figure 4.3.

When calculating the new value, the values α and β are weighted as to their corresponding points' distance to the grid point. In figure 4.3 r_α is closer to the grid point than r_β, and more of the value α will be calculated into the new value.

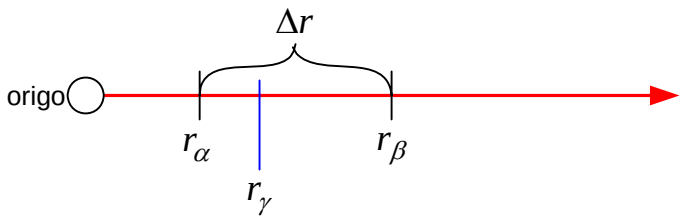


Figure 4.3 Interpolation in the sonar range

When interpolating between the sonar angles (step 4 in the algorithm), the radius r is replaced with θ, which is the angle of the grid-point. θ_α and θ_β are the angles of the two nearest sonar angles, α and β are the values, $\Delta\theta$ is the step between angles in

the sonar data structure, and θ_γ is the interpolated value in the grid-point. This is illustrated in figure 4.4.

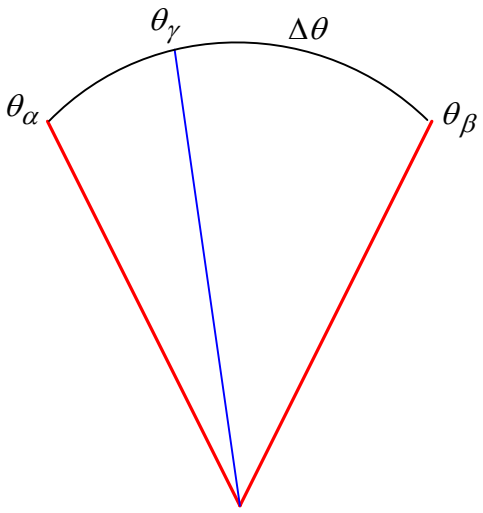


Figure 4.4 Interpolation between the sonar bearings

The interpolation algorithm uses the following equation:

$$(1)\ r_\gamma = \frac{r - r_\alpha}{\Delta r} * \beta + \frac{r_\beta - r}{\Delta r} * \alpha$$

Grid-points outside of the sonar data structure area (the radius of the grid-point is larger than the radius of the sonar data structure), can either be set to an unknown value, or extrapolated values. Extrapolated values are not used, since these grid-points are outside of the sonar's range (red circle in figure 4.2). The value NaN (not a number) is used.

An alternative method would be to place the entire grid inside the sonar data structure, but then some values would be lost.

When using the angles of the sonar performance values to compare with the angle of the grid-point, one needs to consider that the angles of the sonar are computed clockwise, with $0°$ pointing north, $90°$ pointing east etc. (see figure 4.1), whereas in standard polar coordinate system the angles of a circle are computed counter clockwise. This is corrected with the equation:

$$(2)\ \text{Angle: } \theta = \left(360° - \theta + 90°\right) \text{modulus } 360°$$

The polar coordinate system and the sonar bearing description are illustrated in figure 4.5.

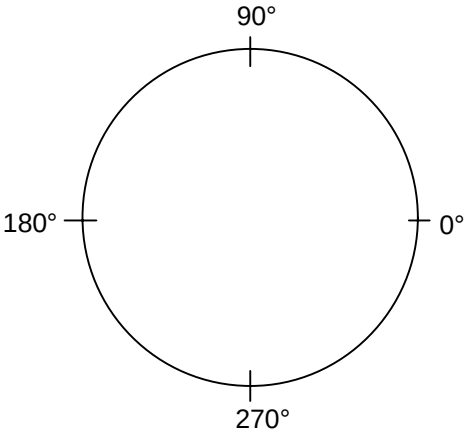
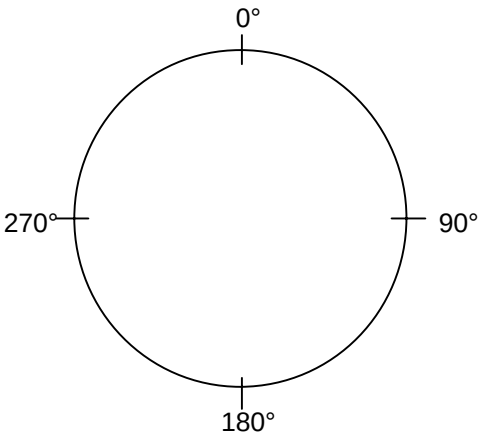


Figure 4.5 Angles of a circle in polar coordinates



Bearing description of sonar

4.2 Horizontal and vertical slicing

The data representation of the volume grid is a 3D matrix; m[i, j, k], with i representing the depth dimension ranging from 0 to the number of matrix cells that corresponds to the maximum sea depth in the area covered by the sonar range, j representing the vertical dimension ranging from 0 to n, and k representing the horizontal dimension ranging from 0 to n. n is the size of each horizontal slice of the cube (illustrated as a red square in figure 4.6). The number of grid points in each horizontal slice is n x n. The size of the volume grid is n x n x max depth. Each combination of m[i, j, k] represents a grid point.

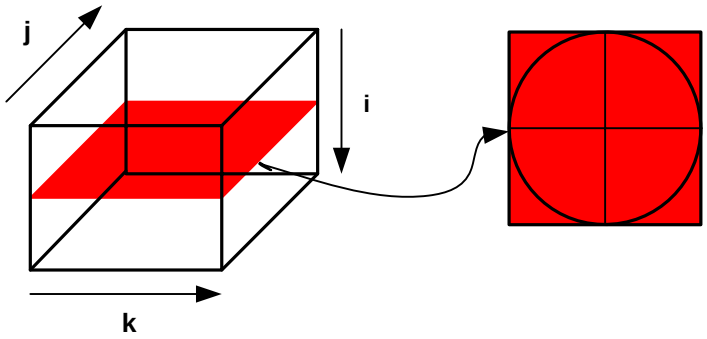


Figure 4.6 Data representation of volume grid

The data representation for each grid point is a struct Vertex, containing the x- and y-coordinates, angle, and interpolated sonar performance value and sea depth for the grid point, in addition to a Boolean value for information as to whether the grid point is inside an isovalue used for isosurfaces.

A horizontal slice is made in the data structure by fixing the i-index. Fixing the j-index or the k-index respectively makes vertical slices.

A vertical slice can be made parallel with the bearings, independent from the grid. The data from all bearings available are read into 2D data matrices, which are later interpolated to a volume grid. The data matrices can be used when a vertical slice is made (see figures 3.11, 4.11 and 4.12). If there are no data available for the chosen bearing, sonar performance data and sea depths are interpolated from the two nearest bearings using the linear interpolation algorithm described earlier (1).

Horizontal slicing could also have been performed on the original 2D matrix dataset. But since a volume grid with interpolated values is already available, this provides a more efficient tool for making horizontal slices on the i-axis.

For the lower horizontal slices (between min. and max. sea depth), the sea depth at each grid point is calculated from the seabed profiles using the linear interpolation algorithm described earlier (1). As with sonar performance values, the sea depth for a given grid point is interpolated using its angle and radius, and finding the

corresponding point in the nearest angle on either side of the grid point. Seabed profiles are provides for the same sonar bearings as the sonar performance values. Grid points below the seabed are not visualized (see figure 4.9).

Figure 4.7 shows the top horizontal slice of the volume grid (sea surface level) with transmission loss values interpolated from 360 bearings. The data are sampled in a range of 100 to 10000 m from the sonar, and the white area in the middle is the area close to the sonar where no performance values are simulated. The colour scale used has 13 colours representing sonar performance values from more than –40 dB (bright red) to less than –95 dB (dark blue), with intervals of 5 dB. The user can choose min value in dB and step between values in dB (intervals for the colour scale). See chapter 3.3 on user interface.

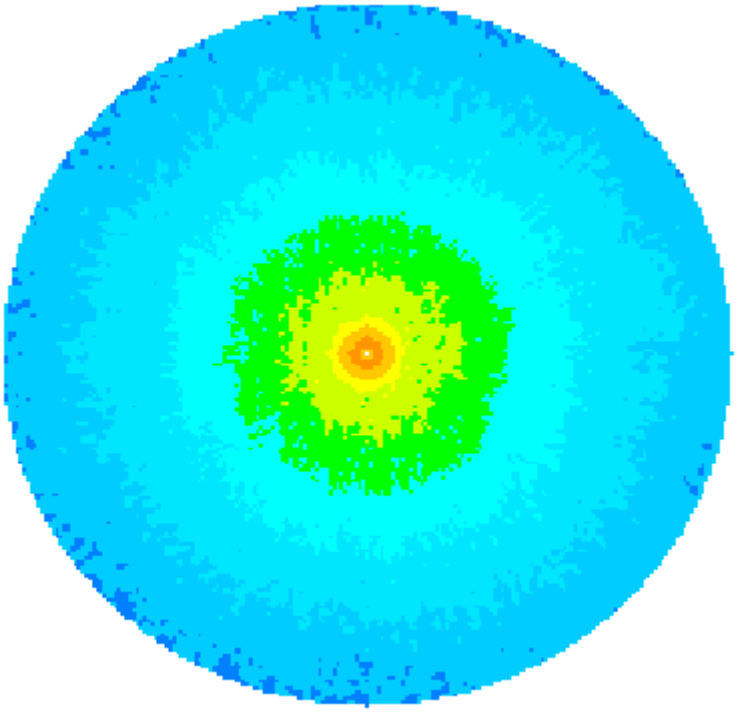


Figure 4.7 Horizontal slice at sea surface level

Figure 4.8 and 4.9 shows horizontal slices of the volume grid with sonar performance values interpolated from the same data as figure 4.7, and with sea depths interpolated from the seabed profiles. There are 50 depth-cells in the matrix, each representing a depth interval of 6 m. The slice in figure 4.8 represents a sea depth of 102 m (i-index 17 in the 3D matrix), which is the depth of the sonar. This is indicated with bright red.

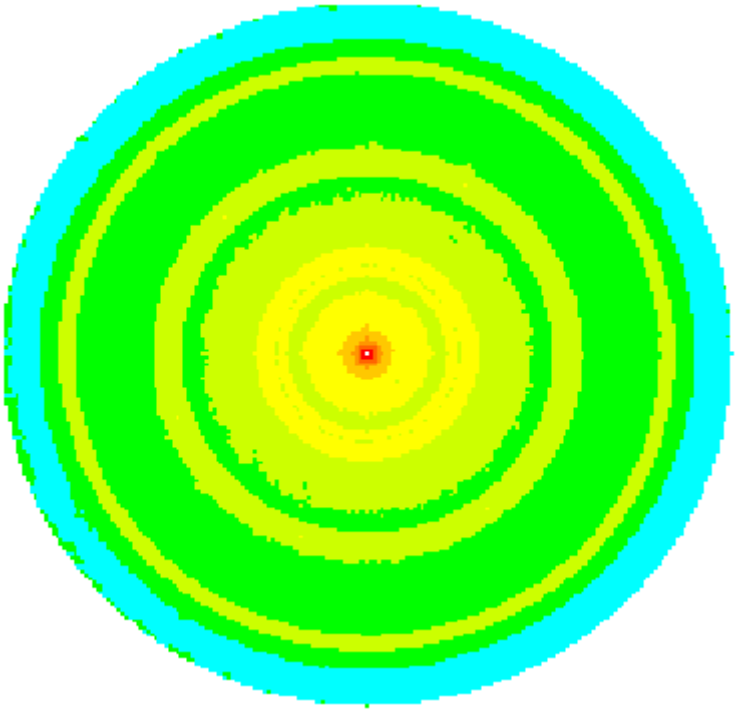


Figure 4.8 Horizontal slice at sonar depth

The slice in figure 4.9 represents a sea depth of 240 m (i-index 40 in the 3D matrix). Grid points below the seabed are not visualized (white areas).

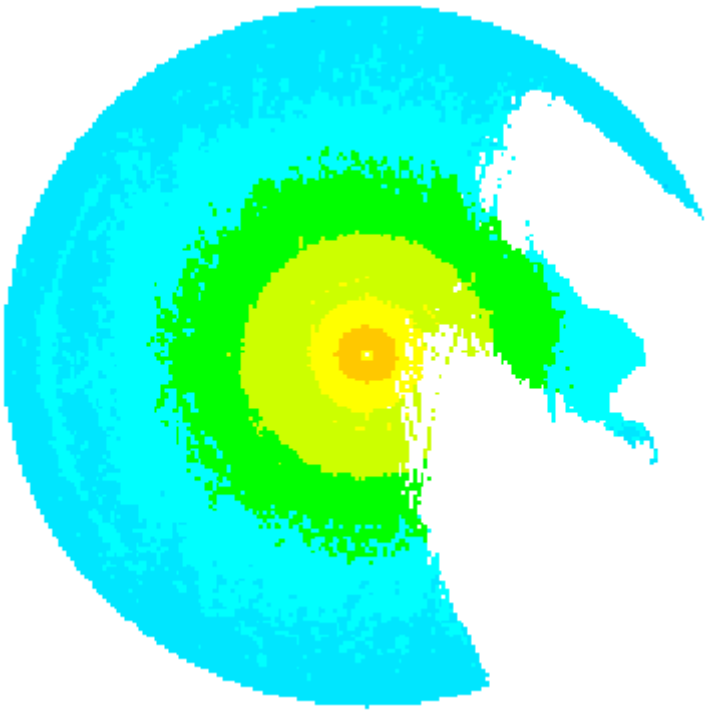


Figure 4.9 Horizontal slice at 240 m depth

4.3 3D and volume visualization

A 3D visualization of the sonar performance data has been made with the outer values from each bearing. The values are collected from the last column of each 2D matrix into a new 2D matrix. Then they are visualized column by column from the sea surface to the seabed, rotating the structure one degree between each column.

The sea surface is visualized using the top horizontal slice of the volume grid. Any points below the seabed or between vertical slices (see figures 4.11 and 4.12) are omitted from the visualization algorithm, and no transparency is applied. The resulting structure is shown in figure 4.10.

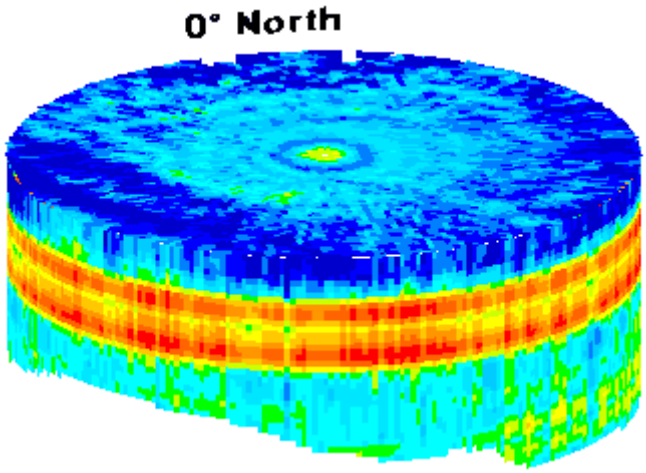


Figure 4.10 Surface of signal excess data

The sonar performance structure can be rotated about its y-axis, and the orientation in space is indicated with a label showing 0° North (in sonar bearing description – see figure 4.5). The label is made using a transparent gif as texture in OpenGL. It is part of the structure, and will therefore rotate with it.

In figure 4.11 the same technique has been used to generate the outer surface of the sonar data when a 90° vertical slice from bearing 95° to bearing 185° has been made. The sea surface is again visualized using the top horizontal slice of the volume grid, this time with all grid points within the slice transparent. The edges of the slice are visualized using the 2D matrices of original data, as shown in figure 3.11.

The black lines on top of the sonar data structure represent bearings 0°, 90° and 270° respectively, and are meant as a help to keep track of the orientation as the vertical slice is moved and the structure is rotated.

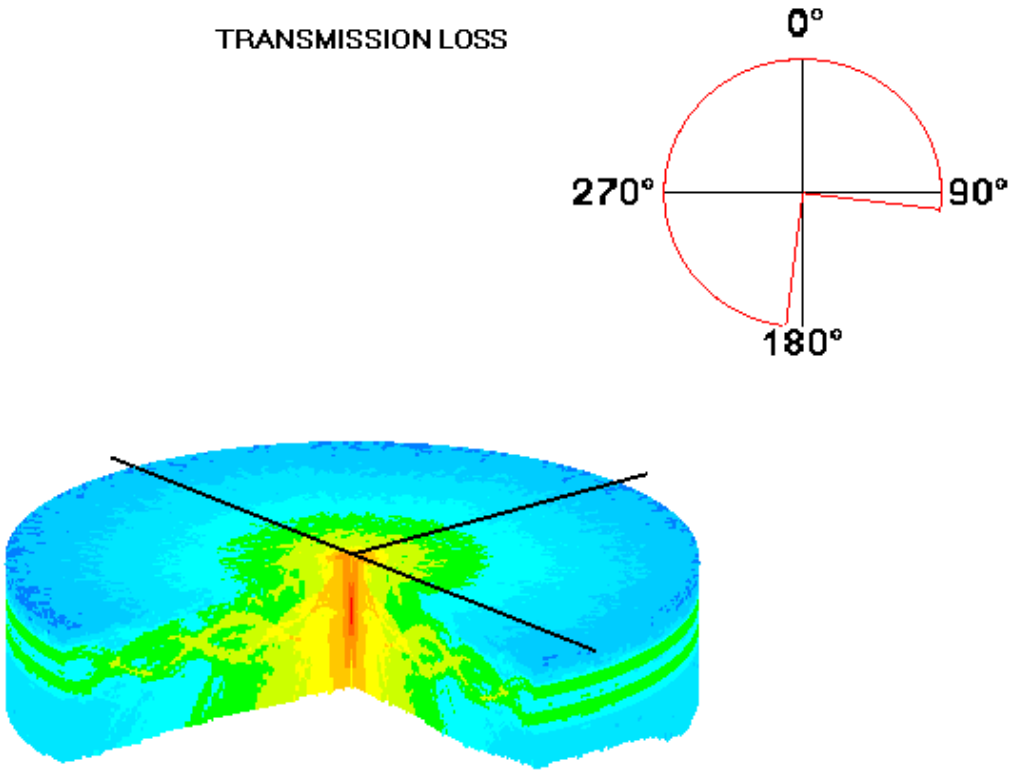


Figure 4.11 Surface visualization after vertical slice

Figure 4.11 has been generated using data with resolution 100 x 50, and with data from all bearings (0 through 359). Figure 4.12 has been generated using another data set, with resolution 100 x 100, and with data from every 10^{th} bearing only. For all bearings where data are not provided, sonar performance data and sea depths are interpolated from the nearest bearings, using the linear interpolation algorithm described earlier (1).

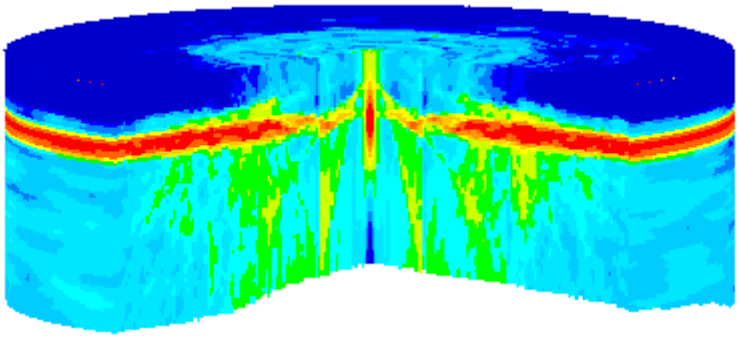


Figure 4.12 Surface visualization of signal excess data with interpolated values

Figure 4.13 shows a volume representation of transmission loss using points at every two grid-points (lod 3, described in chapter 3.1.2).

The sonar performance structure can be rotated about its y-axis, and the orientation in space is indicated with a label showing 0° North (in sonar bearing description – see figure 4.5). The label is made using a transparent gif as texture in OpenGL. It is part of the structure, and will therefore rotate with it.

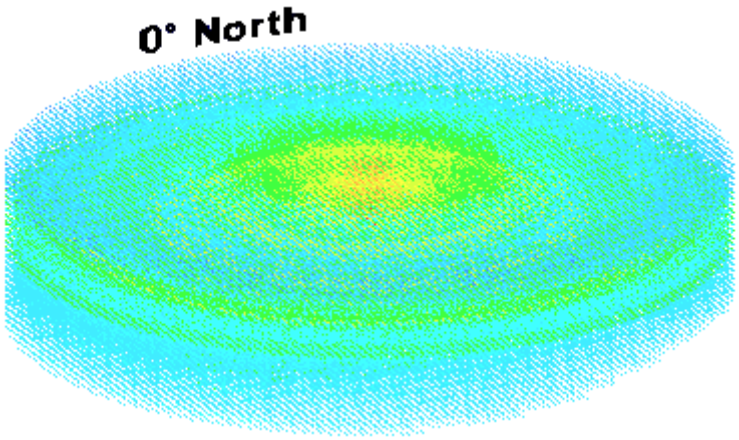


Figure 4.13 Volume visualization using points

4.4 Experiments

4.4.1 Case data

The default resolution of data generated in LYBIN is 50 x 50 (50 range-elements and 50 depth-elements for each bearing). Figures 4.10 and 4.21 have been generated from data with this resolution. Most figures from the visualization application in this report have been generated from data with resolution 100 x 50 (100 range-elements and 50 depth-elements), which results in smoother images of horizontal slices. All data have been generated with a range of 10000 m, i.e. each cell represents a range of 100 m. The following three figures are from visualization of signal excess data with resolution 100 x 100.

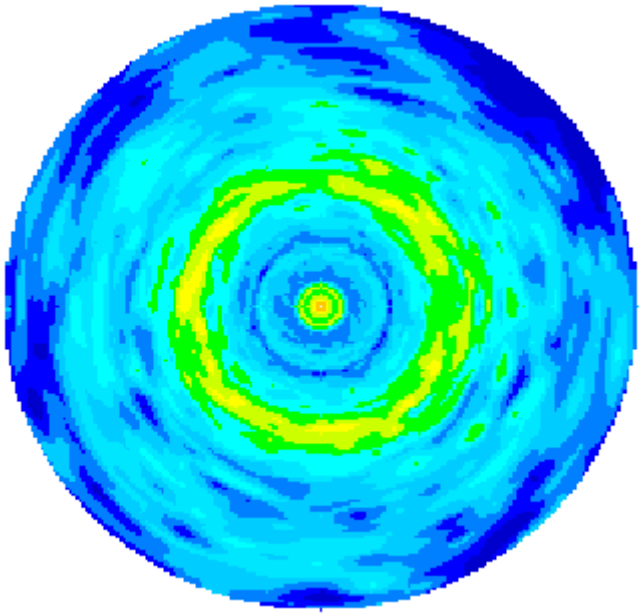


Figure 4.14 Horizontal slice of signal excess data at 47 m depth

Sound speed profiles (SSP) are graphs of the speed of sound in the sea against depth. The SSP depends on the location, the season, the time of day and the weather. A typical deep sea SSP is divided into layers:

- Surface layer
- Seasonal thermocline
- Main thermocline
- Deep isothermal layer

Between the main thermocline and the deep layer there is a sound speed minimum, where sound tends to be focused by refraction. The depth at which this focusing occurs is known as the deep sound channel (DSC). To exploit this channel, the source is placed close to the minimum and, because the spreading is cylindrical, very long-range propagation is possible [1].

The sonar has been positioned at a depth of 78 m, in a deep sound channel. This results in strong signals, visualized with bright red in figures 4.15 and 4.16.

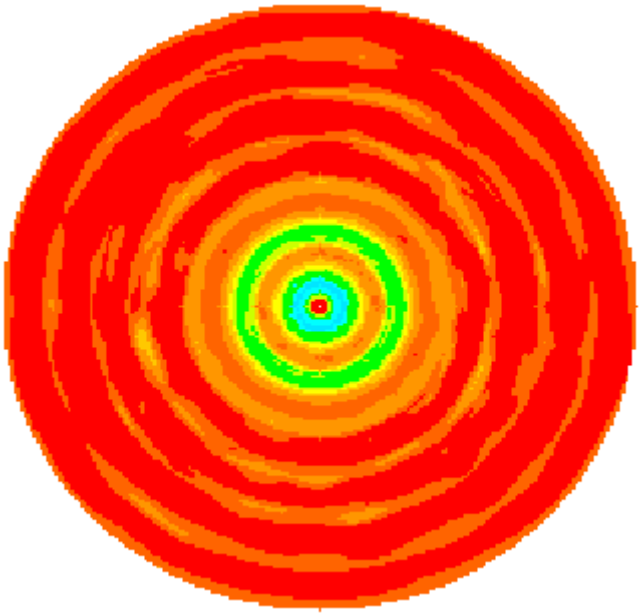


Figure 4.15 Horizontal slice of signal excess data at sonar depth

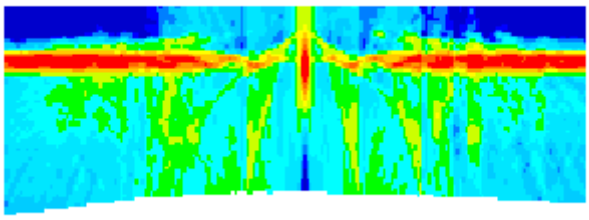


Figure 4.16 Vertical slices of signal excess data

The dataset with dimension 100 x 100 use the same number of cells to represent a range of 10000 m and a depth of 400 m respectively. The resulting images of vertical slices and 3D or volume data structures have proportions that deviate from reality. The dataset that has been used throughout most of this report has a dimension of 100 x 50, representing a range of 10000 m and a depth of 300 m. The resulting images deviate a little less from reality.

Figure 4.17 shows a horizontal slice of signal excess data at 108 m depth. This is close to the depth position of the sonar at 102 m.

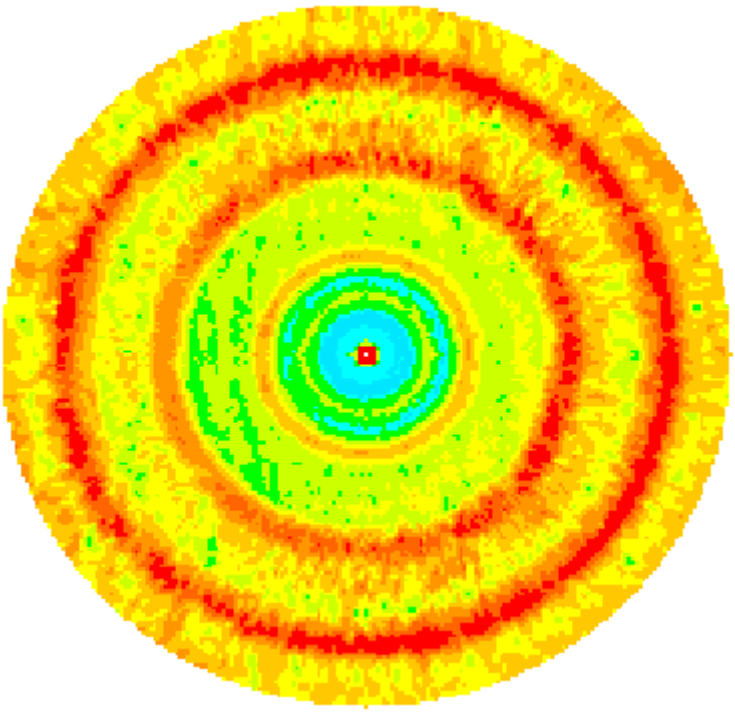


Figure 4.17 Horizontal slice of signal excess data at sonar depth

The information provided by probability of detection data closely follows signal excess data. High signal excess values mean high probability of detection. The PoD data have percentage units, where the highest signal excess data give a probability of detection of 98 percent or more. Figure 4.18 shows a horizontal slice of probability of detection data at 103 m depth. The dataset is the same as in figure 4.17.

Figure 4.18 also shows the colour scale used for probability of detection data.

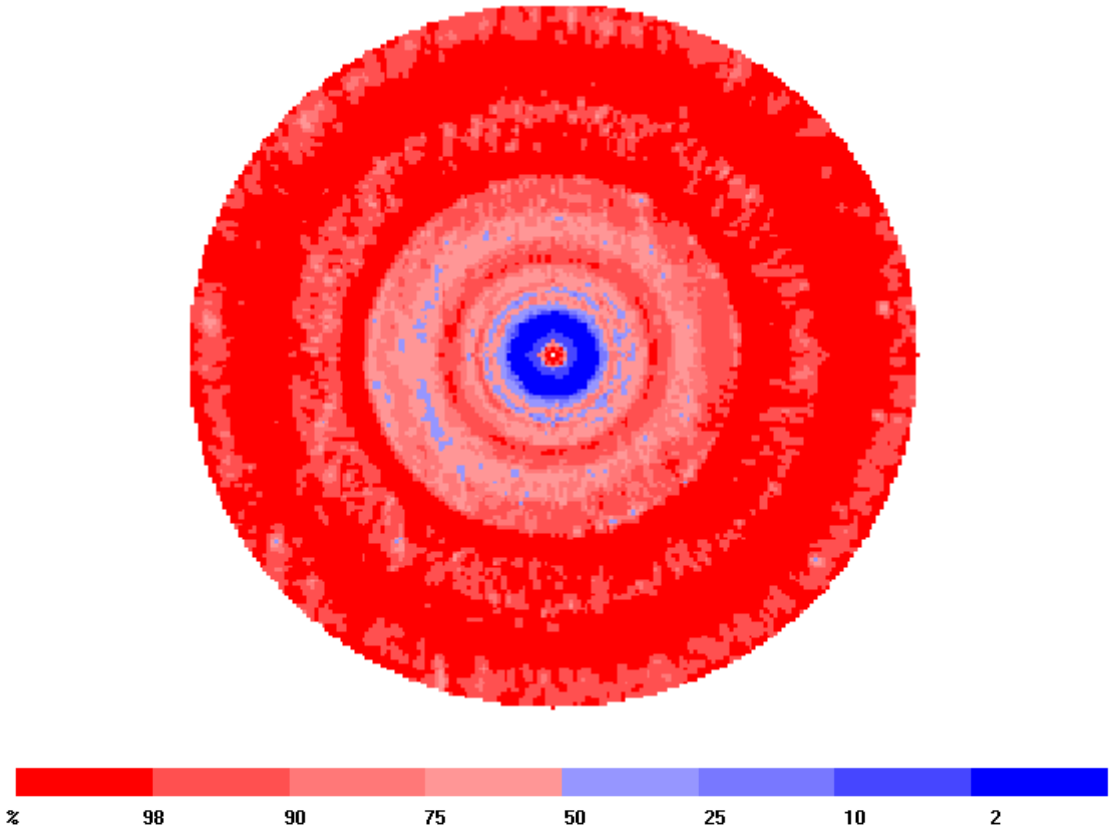


Figure 4.18 Horizontal slice of probability of detection data at sonar depth

Figures 4.19 and 4.20 show vertical slices of signal excess and probability of detection data from the same dataset.

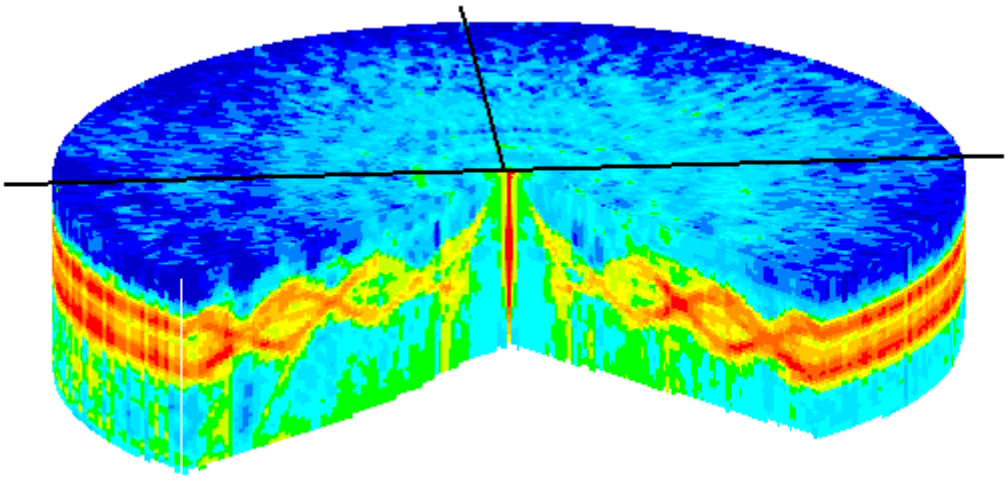


Figure 4.19 Vertical slices of signal excess data

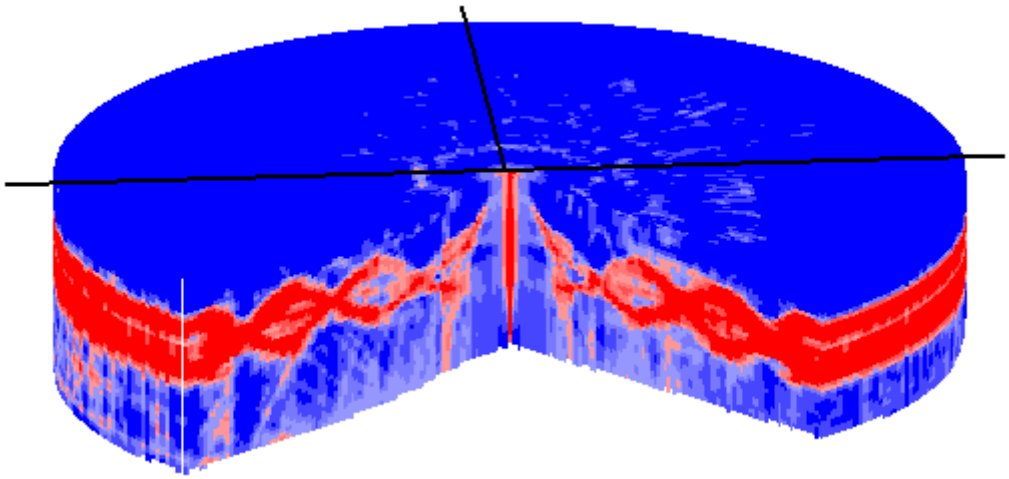


Figure 4.20 Vertical slices of probability of detection data

4.4.2 3D and volume visualization

A 3D visualization of the sonar performance data can be made with the marching cubes algorithm (described in chapter 2.1) generating the outer surface of the sonar data structure. Figure 4.21 has been generated using the radius of the sonar data structure as isovalue, and checking whether the radius of each grid point is inside the isovalue. For every edge between any two grid point where one is outside and one is inside the radius, the x, y, angle, sonar- and depth values are interpolated into a new vertex on the edge.

The sea surface is visualized using the top horizontal slice of the volume grid. All grid points below the seabed have their alpha values set to 0, and are therefore transparent.

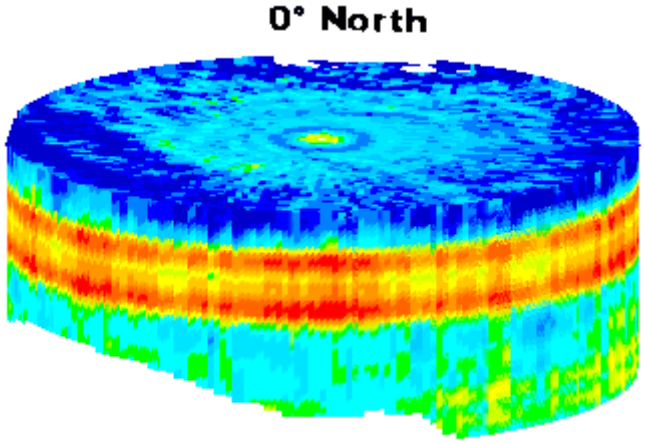


Figure 4.21 Surface visualization of signal excess using Marching Cubes

The visual result is practically identical with the one in figure 4.10, but this visualization technique is much more inefficient.

For an efficiency analysis of this algorithm, see chapter 3.1.1.

A possible volume representation of the sonar performance data is to generate 2-3 partial transparent isosurfaces, representing different levels of sonar performance.

This idea was tested using three cylinders with different colours and alpha-values; the inner cylinder is red with an alpha-value of 1, the middle cylinder is orange with an alpha-value of 0.75, and the outer cylinder is yellow with an alpha-value of 0.5.

The surface of each cylinder is clearly visible, as figure 4.22 shows.

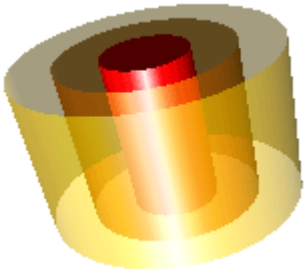


Figure 4.22 Isosurface test

Using the same technique on sonar performance data does not give the same clear result. Figure 4.23 has been generated on transmission loss data with resolution 100 x 50, and with –60 dB (red) and –70 dB (orange) as isovalues, these values represent values 5 and 7 on the colour scale (see figure 3.9) respectively.

The x-, y- and z-axes are visualized with red, green and blue respectively. The boundaries of the grid are visualized as black lines.

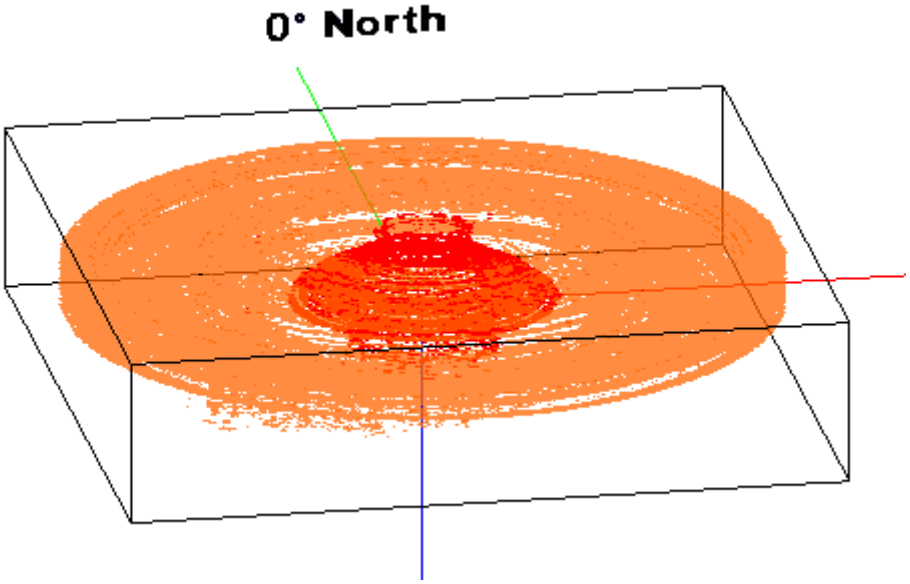


Figure 4.23 Isosurfaces of transmission loss data

In figure 4.24 only one isosurface with isovalue –60 dB has been generated from the same dataset. The result is one almost continuous isosurface, with a few additional points scattered around.

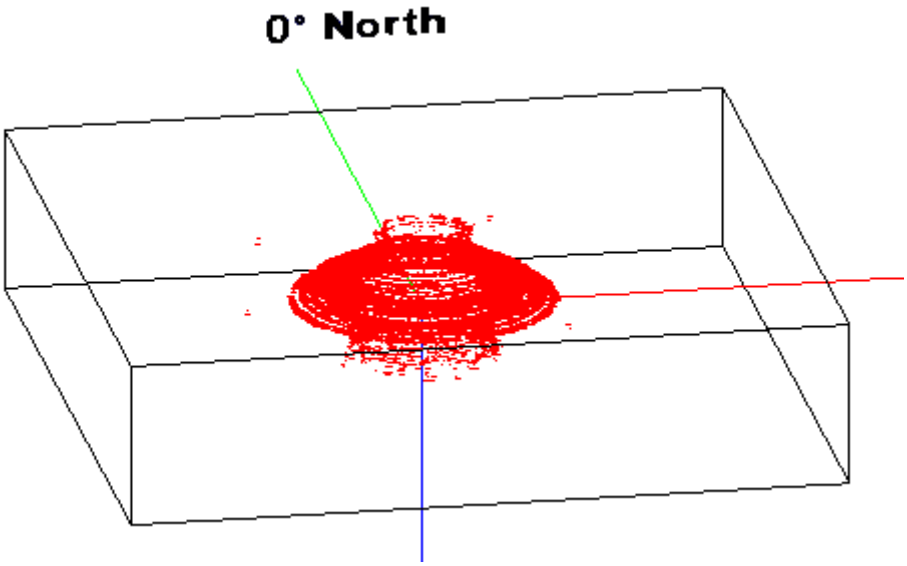


Figure 4.24 Isosurface of transmission loss data

For signal excess data a larger percentage of the data has high values, but these values are scattered around the area, resulting in many tiny isosurfaces instead of one continuous. In figure 4.25, 12 dB (red) and 0 dB (orange) have been used as isovalues. These values represent values 2 and 6 on the colour scale (see figure 3.8) respectively.

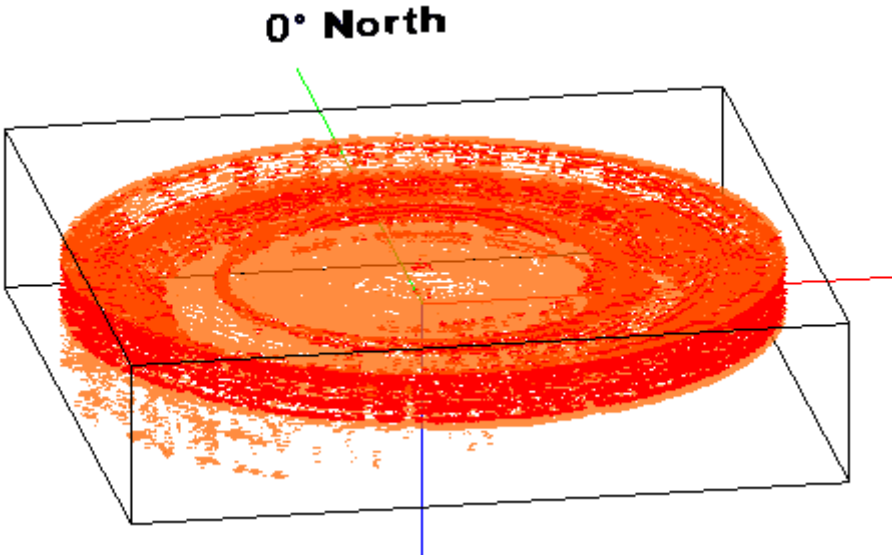


Figure 4.25 Isosurfaces of signal excess data

In figure 4.26 an additional isovalue of –18 dB (yellow) has been used. The result with weaker signals near the surface coincides with horizontal and vertical slices of the same data.

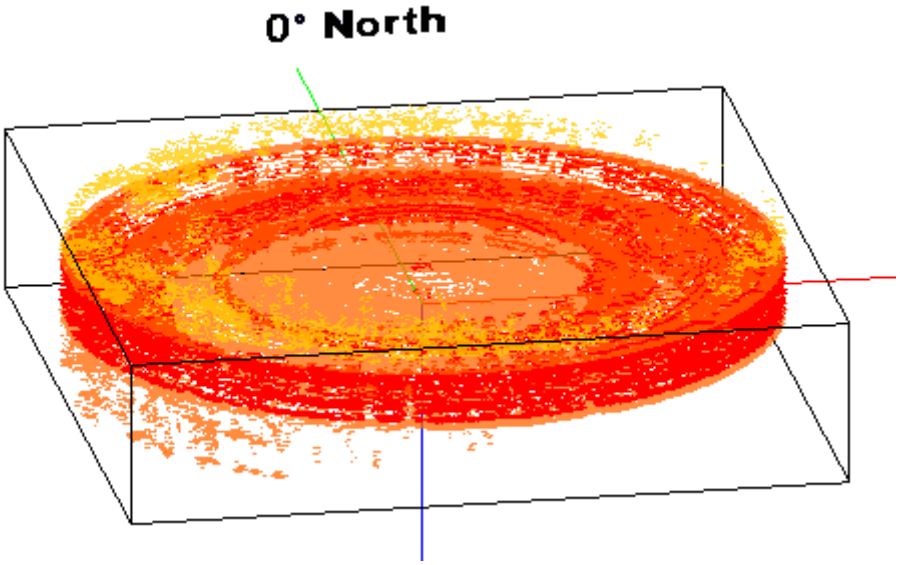


Figure 4.26 Isosurfaces of signal excess data

Figures 4.27 and 4.28 show isosurfaces of probability of detection data. In figure 4.27 98 percent (red) has been used as isovalue.

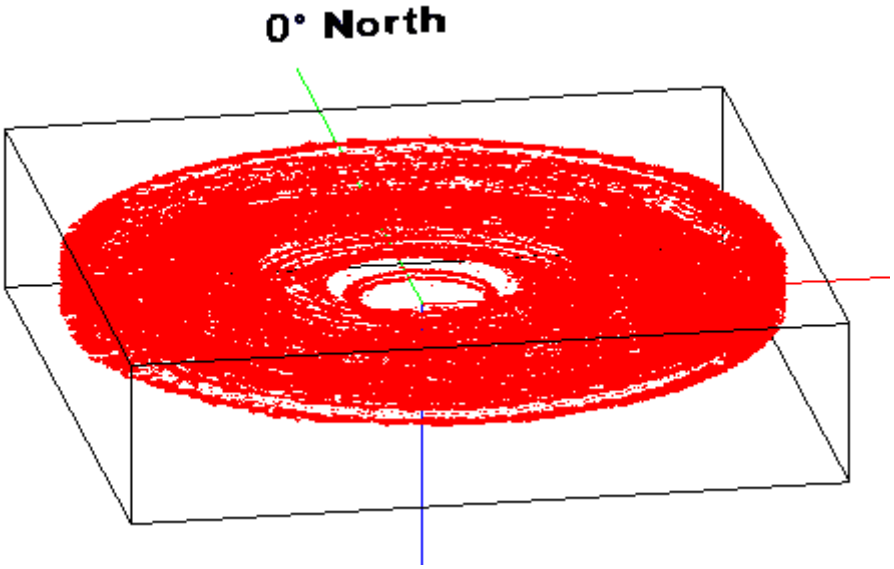


Figure 4.27 Isosurface of probability of detection data

In figure 4.28, 98 percent (red), 50 percent (orange) and 2 percent (yellow) have been used as isovalues. These values represent values 1, 4 and 7 on the colour scale (see figure 4.18) respectively.

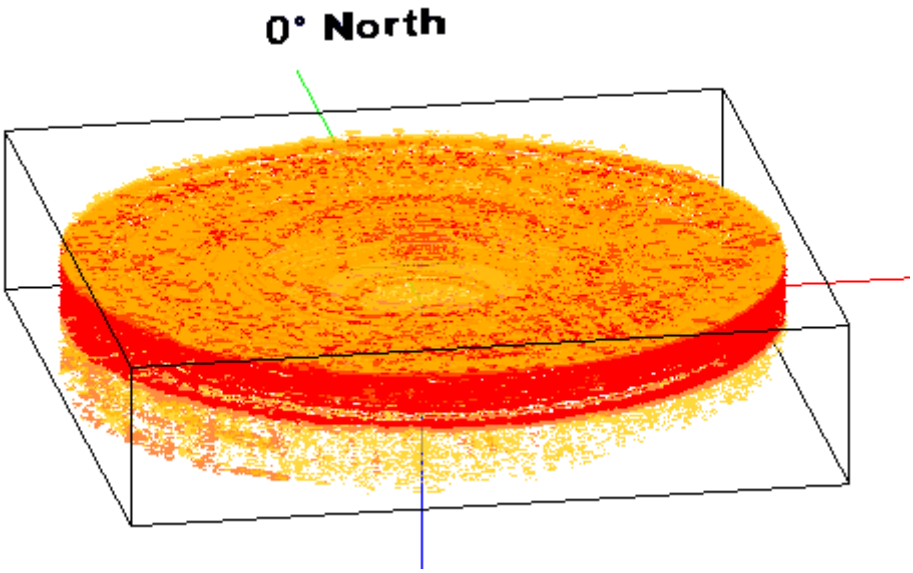


Figure 4.28 Isosurfaces of probability of detection data

I have also experimented with other colours, but red, orange and yellow seem to provide the best result. The red is clearly visible through the orange, and using yellow for the weakest signals is more appealing to the eye than for instance blue.

4.5 User feedback

Test users at FFI:

Jon Wegge	Scientist, sonar technology analysis Subproject manager for project Simson
Karl Thomas Hjelmervik	Scientist, underwater acoustic
Elin M. Dombestein	Scientist, environmental physics / underwater acoustic

My supervisor at FFI, Jon Wegge, has been involved in the entire development process of the visualization tool, and his remarks and requests along the way have had an impact on the final result. His main focus has been on user friendliness and unambiguous interpretation of the data visualized.

Some of the functionalities and additions to the visualization have been implemented on his request:

- Animation of horizontal slices
- Show and hide bearings and ranges on horizontal slices
- Figure showing bearings 0°, 90°, 180° and 270°, and bearings of the vertical slices made
- Show bearings on 3D representation of vertical slices
- Combination of horizontal and vertical slicing

During the last phase of development, all test users have evaluated the visualization tool with emphasis on:

- Have the requirements for the project been met (see chapter1.2)?
- User friendliness of the graphical user interface
- Evaluation of different visualization techniques for interpretation of the sonar performance data

The test users have each had a copy of the visualization tool to evaluate on their own without any help or explanation other than the Help-menu in the tool.
Some minor changes have been made to the GUI.

Conclusions from user feedback:

- The requirements for the project have been met
- The graphic user interface is intuitive and has a professional look. The colour-setting window gives the user opportunities to study different details of the data.
- Horizontal and vertical slicing are the visualization techniques that provide the best source of information.
    - Visualization of the inner structures of the volume provides little information compared to slices.
    - Visualization with points provides a fairly good impression of the data when the level of detail (see chapter 3.1.2) is high. With a lower level of detail this visualization technique is of little use.
    - Visualization of isosurfaces is poorly suited for sonar performance data alone, because of the richness of detail in the data. The use of

transparency might result in ambiguous interpretation of the data. This technique would be better suited for combinations of different sets of simulation data, e.g. for visualizing the range of the sonar as a function of the speed of the frigate. Instead of transparency, facilities for making slices would provide a better visualization of the inner structures.

The visualization application is a sonar performance prediction and analysis tool for data from the underwater volume covered by the sonar's range, whereas previous visualizations have been made as curves or 2D diagrams. The visualization is perspicuous and fast. There is no delay between visualization of slices and pings, which gives the user overview of the data quickly.

Sonar performance data visualized without any context, for instance seabed topography or track from the frigate, is mainly useful for underwater acoustics for interpretation of the data. The tool is especially useful for analysis of a sonars performance for different submarine ranges and depths in areas with varying seabed topography, e.g. in fjords.

Visualization of sonar performance data put into a context increases the information for the sonar operator and facilitates the interpretation of the data. This enables him or her to discard false alarms (signals from e.g. bottom reverberation and not enemy submarine), which would normally require time and resources to analyse.

5 SUMMARY

5.1 Conclusion

The ultimate goal of visualization is to convey the information effectively to help the viewer understand the data (ref. chapter 2.1). Volume visualization is very well suited for visualization of modelled sonar performance data, because it helps understanding large quantities of data by rendering them so that they can be viewed in their entirety. Sonar data from several pings are time varying data, and animations can show this variation in a natural way.

Different techniques of visualizing the inner structures of the sonar performance prediction data have been tried with various successes.

- Surface visualization with marching cubes proved to be too inefficient on volume grids of this size.
- Isosurfaces of different levels of sonar performance resulted in many tiny isosurfaces because of the richness of detail and the scattering of the data. The resulting images may cause ambiguous interpretation of the data.
- Visualizing the grid points with points and different levels of detail gives a good impression on the data if the level of detail is high enough. This visualization technique is also more efficient than using cubes or squares at each grid point.
- The visualization techniques that provide the best source of information, is horizontal and vertical slicing, with or without 3D surface view.

5.2 Further work

Modelled sonar performance data would be better suited for isosurfaces if the richness of detail was reduced. Isosurfaces used on combination of datasets, e.g. for visualization of the range of the sonar as a function of sonar depth or the speed of the frigate, would provide more useful and unambiguous information. The best result might be to generate solid surfaces without any transparency, and provide facilities for making horizontal and vertical slices.

Some improvements could be done on the graphic user interface by using one common panel for horizontal and vertical slicing in stead of two separate tab panels. The use of horizontal and vertical slicing in combination might then be more intuitive. An animation of vertical slicing might be added.

Colour settings for probability of detection could be added to give the user possibilities to explore details of the data. The options must then be limited to 0 – 100 percent.

6 BIBLIOGRAPHY

1. A.D. White
   Sonar for Practising Engineers
   John Wiley & sons Ltd 2002
   ISBN 0 471 49750 9
   Chapter nos.: 1.1 3.2 4.4.1

2. Nilsen / Tveit / Wegge: Visualization of Sonar Performance
   FFI/RAPPORT-98/02542
   Chapter nos.: 1.1 2.1

3. Microsoft .NET
   Chapter no.: 1.3

4. Microsoft Visual C#
   Chapter no.: 1.3

5. OpenGL
   Chapter no.: 1.3

6. Microsoft DirectX
   Chapter no.: 1.3

7. CSOpenGL
   Chapter no.: 1.3

8. Volume Visualization at NYU
   Chapter no: 2.1

9. Klaus Mueller
   Introduction to Volume Visualization
   Computer Science Department at Stony Brook University, State University of New York
   Chapter no.: 2.1

10. Schroeder / Martin / Lorensen
    The Visualization Toolkit – an Object Oriented Approach to 3D Graphics
    Prentice Hall 1996
    ISBN 0-13-199837-4
    Chapter no.: 2.1

11. Foley / van Dam / Feiner / Hughes
    Computer Graphics - Principles and Practice Second Edition
    Addison-Wesley 1996
    ISBN 0-201-84840-6
    Chapter no.: 2.1

12. Volume Visualization at NYU Gallery
Chapter no.: 2.1

13. Ernest Orlando Lawrence Berkeley National Laboratory Visualization Group. Available on the Internet on http://web.archive.org/web/20021201230457/www-vis.lbl.gov/site_info/gallery.html
Chapter no.: 2.1

14. Neider / Davis / Woo
OpenGL Programming Guide
Addison-Wesley
Chapter no.: 2.1

15. Noureddine Bouhmala
Notes from Analysis of Algorithm
Vestfold University College 2001 – 2002
Chapter no.: 3.1

16. Thomas A. Standish
Data structures, algorithms & software principles
Addison-Wesley 1995
ISBN 0 201 59118 9
Chapter no.: 3.1

17. Teleplan Maria
Chapter no.: 3.3

APPENDIX

CSOpenGL

This is how you set up your programming environment:

1. Download the csopengl.versionNo.exe file from http://sourceforge.net/projects/csopengl
2. The .exe file will extract all necessary files and install CSOpenGL.
3. In the VisualStudio.NET Start Page choose New Project, Visual C# Projects and Windows Application.
4. Locate the CSOpenGL.dll and CSOpenGLC.dll files in the CSOpenGL\source folder.
5. Copy the .dll files to the bin\Debug directory of your VisualStudio project.
6. Add a reference to the CSOpenGL.dll file to the project:
    - Menu Project
    - Add Reference
    - Browse to find the .dll file
    - Click OK to add a reference to the file

The CSOpenGL Namespace with all classes and members will now be visible in the Object Browser and IntelliSense, as if it were a part of the .NET Framework. To use the Object Browser, right-click on the code you want to explore and choose "Go To Definition", or you can use the menu View -> Others Windows -> Object Browser. Remember to include the statement "using CSOpenGL" in your code.

CSOpenGL will use the windows form as its canvas, and the GLView object is treated as a component in the form. As such, all members of the System.Windows.Forms.Control class are available. You can use mouse- and key-events to provide interaction with the user.

I experienced a problem when I tried to run my program on a computer that doesn't have VisualStudio.NET installed; the mfc70.dll could not be found. This can be solved in either of three ways:

1. Locate the mfc70.dll file in the C:\Windows\System32 folder, and copy it to the bin\Debug folder of your project.
2. Locate the source code for the CSOpenGLC.dll, which uses MFC, in the CSOpenGL\source\CSOpenGLC folder.
    - Open the project in VisualStudio.NET
    - Choose Properties from the Project menu.
    - Under General, Use of MFC, choose Use MFC in a Static Library
    - Rebuild the project, and use the new CSOpenGLC.dll in your project.
3. Rewrite the source code and make a new CSOpenGLC.dll without any link to MFC. The .dll in the bin/Debug directories in my applications has been changed in this way.

Code examples

```
using CSOpenGL;

namespace SonarVisualization
{
        public class frmSonar
        {
                private GLView view;

                public frmSonar()
                {
                        view = new GLView();
                        view.Parent = this;
                        view.CreateControl();
                        view.Dock = DockStyle.Fill;
                        p = new Painter();
                        view._OnPaint = p;
                        view.MouseMove += new MouseEventHandler(view_MouseMove);
                }

                public void view_MouseMove(Object sender, MouseEventArgs e)
                {
                        // do something, ex rotate object
                        view.Refresh();
                }
        }
}
```

The example code in the Paint method draws the sonar performance data volume using points at every grid point / lod = level of detail (see chapter 3.1.2).

```
public class Painter : GLViewPainter
{

public override void Paint(GLView view)
{
        glViewport(0, 0, view.Width, view.Height);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glShadeModel(GL_SMOOTH);

        glClearColor(1, 1, 1, 1);
        glClearDepth(1.0f);
        glEnable(GL_DEPTH_TEST);
        glDepthFunc(GL_LEQUAL);

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

        glEnable(GL_BLEND);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
        glPointSize(2.0f);

        for (int i = 0; i < maxDepth[ping-1]; i+=lod)
        {
                for (int j = 0; j < size; j+=lod)
                {
                        for (int k = 0; k < size; k+=lod)
                        {
                                if (i >= minDepth[ping-1])
                                {
                                        double depth = grid.vertices[i,j,k].depth / m_per_cell;
                                        if (i >= depth)
                                                continue;
                                }
                                if (grid.vertices[i,j,k].sonarvalue.Equals(Int32.MaxValue))
                                        continue;
                                FindColor(grid.vertices[i,j,k].sonarvalue);
                                glBegin(GL_POINTS);
                                        glVertex3d(x + step * k, y - step * i, z - step * j);
                                glEnd();
                        }
                }
        }
        glDisable(GL_BLEND);
}
}
```

Interpolating sonar performance data to a volume grid (see chapter 4.1):
Calculating angle and radius for each grid point:

```
// beregne verdier for det øverste snittet:
for (j = 0; j < size; j++)
{
        for (k = 0; k < size; k++)
        {
                Vertex v = new Vertex();
                x = start_x + step * k;
                y = start_y - step * j;
                v.radius = Math.Sqrt(Math.Pow(x, 2) + Math.Pow(y, 2));
                v.angle = Math.Acos(x/v.radius);                        // radianer
                v.angle = (v.angle / (2 * Math.PI)) * 360;              // regne om til grader
                if (y < 0) v.angle = 360 - v.angle;
                v.angle = (360 - v.angle + 90) % 360;                   // regne om til sonar-grader
                v.radius = v.radius * radius;
                v.sonarvalue = Double.NaN;
                v.depth = 0;
                vertices[0, j, k] = v;
        }
}
// verdiene kopieres til resten av gridet:
```

Interpolate values from the two nearest bearings (this code is simplified):

```
// interpolere mellom to verdier i nærmeste peiling til venstre:
value1 = leftBearing[i,(int)Math.Floor(grid.vertices[i,j,k].radius-1)];
value2 = leftBearing[i,(int)Math.Ceiling(grid.vertices[i,j,k].radius-1)];
leftValue = Interpolate(value1, value2, Math.Floor(grid.vertices[i,j,k].radius-1),
                Math.Ceiling(grid.vertices[i,j,k].radius-1), grid.vertices[i,j,k].radius-1, 1);

// interpolere mellom to verdier i nærmeste peiling til høyre:
value1 = rightBearing[i,(int)Math.Floor(grid.vertices[i,j,k].radius-1)];
value2 = rightBearing[i,(int)Math.Ceiling(grid.vertices[i,j,k].radius-1)];
rightValue = Interpolate(value1, value2, Math.Floor(grid.vertices[i,j,k].radius-1),
                Math.Ceiling(grid.vertices[i,j,k].radius-1), grid.vertices[i,j,k].radius-1, 1);

// interpolere mellom verdier fra venstre og høyre peiling:
newValue = Interpolate(leftValue, rightValue, leftAngle, rightAngle, grid.vertices[i,j,k].angle, 360/data.Count);
grid.vertices[i,j,k].sonarvalue = newValue;


private double Interpolate(double value1, double value2, double left, double right, double midpoint, double step)
{
        return (midpoint-left)/step * value2 + (right-midpoint)/step * value1;
}
```

Calculating the course and speed of the frigate:

```
// beregne fregattens kurs:
pos = 90 - (180 / Math.PI) * Math.Atan((north[ping-1] - north[ping-2]) / (east[ping-1] - east[ping-2]));
if (north[ping-1] - north[ping-2] < 0 && east[ping-1] - east[ping-2] < 0)
        pos += 180;
if (north[ping-1] - north[ping-2] > 0 && east[ping-1] - east[ping-2] < 0)
        pos += 180;

// beregne fregattens fart:
speed = Math.Sqrt(Math.Pow(east[ping-1] - east[ping-2], 2) + Math.Pow(north[ping-1] - north[ping-2], 2))
speed /= ping_interval;
```

north and east are arrays containing the frigates position at each ping.

Drawing a label using a bitmap as texture in OpenGL:

```
public void DrawLabel(string filename, double x, double y, double z, double width, double height)
{
        Bitmap bm = new Bitmap(filename);
        Rectangle rect = new Rectangle(0, 0, bm.Width, bm.Height);
        BitmapData bmData = bm.LockBits(rect, ImageLockMode.ReadOnly, PixelFormat.Format32bppArgb);
        // Locks a System.Drawing.Bitmap object into system memory.
        byte[] pixels = new byte[bm.Width * bm.Height * 4];
        Marshal.Copy(bmData.Scan0, pixels, 0, (bm.Width * bm.Height * 4));
        // Copies data from an unmanaged memory pointer to a managed byte array.

        // snu bildet opp-ned:
        byte[] new_pixels = new byte[bm.Width * bm.Height * 4];
        int pixelwidth = bm.Width * 4;                  // antall pixler * RGBA i hver rad
        int k = 0;
        for (int i = 1; i <= bm.Height; i++)            // antall rader i bildet
        {
                int index = pixels.Length - pixelwidth * i;
                for (int j = 0; j < pixelwidth; j++)
                {
                        new_pixels[index] = pixels[k];
                        index++;  k++;
                }
        }

        glAlphaFunc(GL_GREATER, 0.5f);
        glEnable(GL_ALPHA_TEST);
        glEnable(GL_TEXTURE_2D);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, (int)GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, (int)GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, (int)GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, (int)GL_NEAREST);
        glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, (int)GL_MODULATE);
        glTexImage2D(GL_TEXTURE_2D, 0, 4, bm.Width, bm.Height, 0, GL_RGBA,  GL_UNSIGNED_BYTE, new_pixels

        glBegin(GL_QUADS);
        glTexCoord2d(0, 0); glVertex3d(x - width/2, y - height/2, z);
        glTexCoord2d(0, 1); glVertex3d(x - width/2, y + height/2, z);
        glTexCoord2d(1, 1); glVertex3d(x + width/2, y + height/2, z);
        glTexCoord2d(1, 0); glVertex3d(x + width/2, y - height/2, z);
        glEnd();

        glDisable(GL_TEXTURE_2D);
        bm.UnlockBits(bmData);
        bm.Dispose();
}
```