UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology

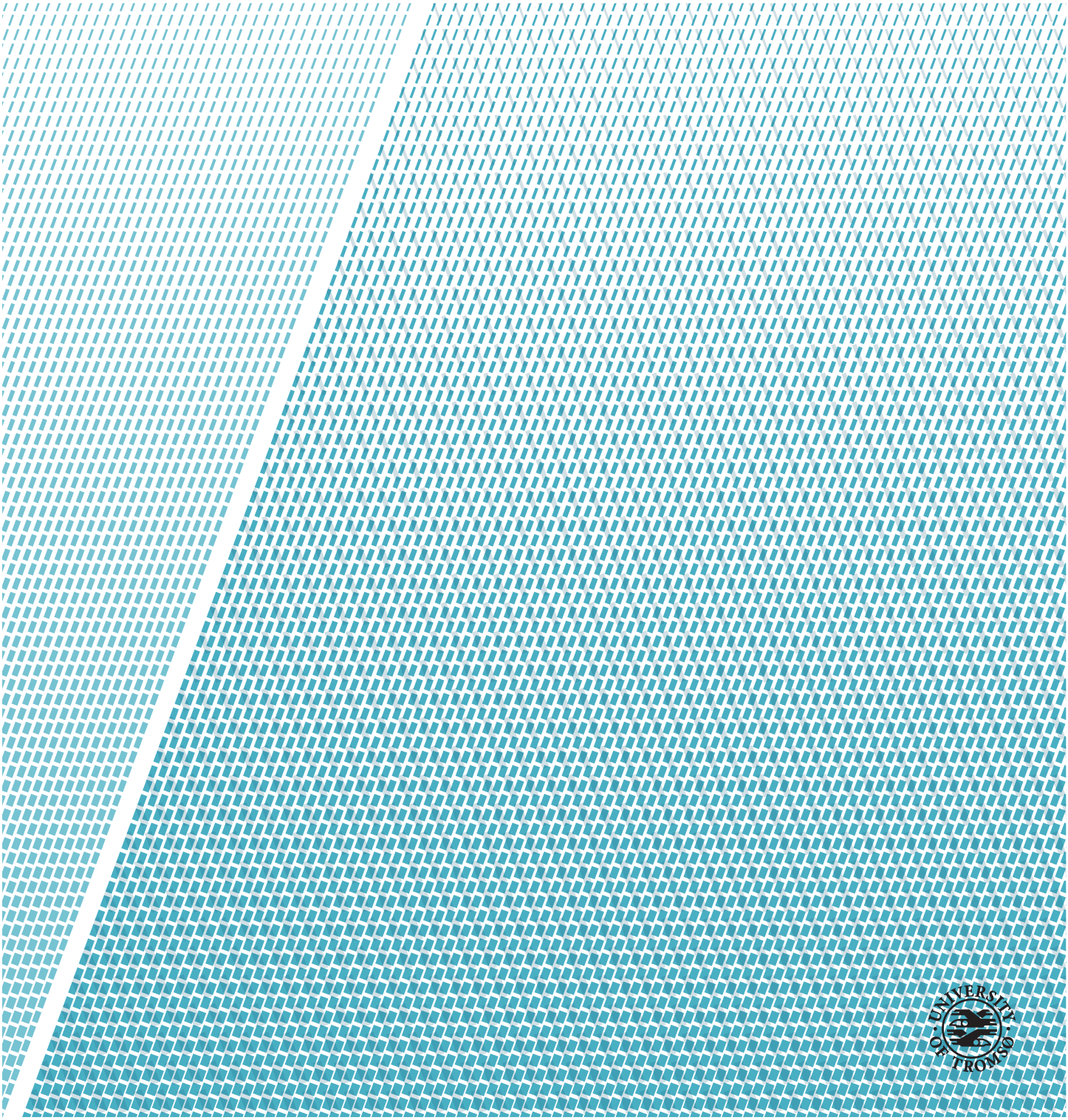
RS - Seismic Processing and Web-Visualisation

Simplifying visualization of Seismic Data

—

Tom Pedersen

Master thesis in Computer Science
May 2014



Abstract

The University of Tromsø - The Arctic University of Norway (UIT) is conducting regular marine seismic acquisition cruises for scientific and educational purposes in the polar regions of the Norwegian Sea and beyond. Leading experts in the field currently employed by the Department of Geology at UIT, have found the current seismic visualization tools lacking in several fields. The available seismic software provides a multitude of settings and filters to improve visualization, but has limitations when it comes to user friendliness, and lacks or has shortcomings for data interaction. Because of these limitations the scientists are still reverting back to using thermal paper plots for seismic data interaction. The thermal printers in the possession of UIT are old, bulky and prone to mechanical failure, and are expensive to replace.

The work done in this thesis attempts to address the needs of the Department of Geology, and as a response to these needs presents the system RS. A system to process, visualize and interact with both "live" and previously recorded seismic 2D data. RS provides processing and filtering of seismic data, and presents this data in a web based user interface, using an Open Source JavaScript Tile viewer to visualize the seismic data. RS also consists of among other: a Go[1] two tiered backend system, a custom built tile maker, web sockets for bidirectional communication and NoSQL database storage.

The goal of the system is to provide a new visualization tool to replace or supplement current visualization platforms. RS does this by presenting a visualization client which can run on any device with a modern browser, giving every authorized person on a seismic vessel the ability to view and interact with seismic data.

To present seismic plots to the end user, RS uses binary seismic data acquired by existing seismic software and hardware. This data is a combination of the seismic data and a variety of metadata from sensors aboard the vessel. Data is filtered for noise, and cached before images are created. These images are served by a web server, and made available to the end user through his

1. http://golang.org/

preferred platform.

Acknowledgements

I would like to thank my Advisor John Markus Bjørndalen for his help and guidance throughout the year. In addition I would like to thank the Department of Geology, and especially Bjørn Runar Olsen and Anoop Mohanan Nair for sharing their knowledge of the field Reflection Seismology, and providing seismic data for testing and evaluation purposes. For helping with setting up the system on the FF Helmer Hansen, and also sharing his knowledge, I would like to extend my gratitude to Ronald Berntsen.

I would also like to thank all staff and students at the Department of Science and Technology for providing a great educational environment through five amazing years.

A special thank you goes to Wendy van Dreunen for first supporting me through my bachelor degree, and then for supporting me to start on a master while I still was motivated. I would also like to thank her for her extreme patience throughout these years and especially during her pregnancy with our first child, despise my many late nights throughout the last 9 months.

Contents

List of Figures

List of Tables

List of Abbreviations

BSON Binary JavaScript Object Notation

CSS Cascading Style Sheets

GC Garbage Collector

HTML HyperText Markup Language

IO Input/Output

JSON JavaScript Object Notation

MB megabytes

MS milliseconds

REST Representional state transfer

TMS Tile Map Service specification

TVG Time Variant Gain

UIT The University of Tromsø - The Arctic University of Norway

1

Introduction

Seismic acquisition is conducted daily by research organizations and various public and private companies. Seismic is a big money industry, responsible for increasing the odds of finding natural gas and oil, by providing a platform for increasing the knowledge of the location deposits of such natural resources. In addition studies are being conducted into among other: earthquakes, tsunami predictions and the composition of the earth and other bodily elements.

While seismology is the study of earthquakes and the natural seismic waves that originate from such events, reflection seismology is part of the branch of controlled source seismology. This thesis deals with data from marine reflection seismology acquisitions done by The University of Tromsø - The Arctic University of Norway (UIT). The seismic data is used for, among other things, scientific studies that include the study of the sediments layers below the seabed, and studies into the presence of gasses.

At present time UIT employs three scientific vessels for scientific exploration in the Arctic region, but for seismic surveys the FF Helmer Hansen is mostly used. This vessel is the property of UIT and conducts research 300 days a year[1]. The vessel is used by UIT, the University of Bergen and the University Center of Svalbard. The vessel is also available for charter by international research groups.

1. http://uit.no/nyheter/

Figure 1.1: FF Helmer Hansen, UIT research vessel

Acquisition of seismic data can be done with various seismic equipment, which in turn might come with its own software and proprietary formats. This thesis presents RS, a seismic processing and viewing platform, which aims to provide scientist's a single platform to visualize and interact with seismic data. The system allows for multiple data formats to be converted into one, and displayed via a web-interface. RS runs in the background, and does not interfere with normal operations. A system crash in RS will not affect ongoing seismic surveys.

During the creation of RS, real seismic data from seismic expeditions was provided for evaluation purposes. A further evaluation was conducted aboard the FF Helmer Hansen in April 2014, where RS was able to visualize seismic data collected during a seismic cruise.

1.1 Marine Seismic Acquisition Process

The process of acquiring the seismic imagery starts with the source of the seismic wave. This source could consist of a number of available technologies, including: air guns, sparker, and boomer sources. These have a different way

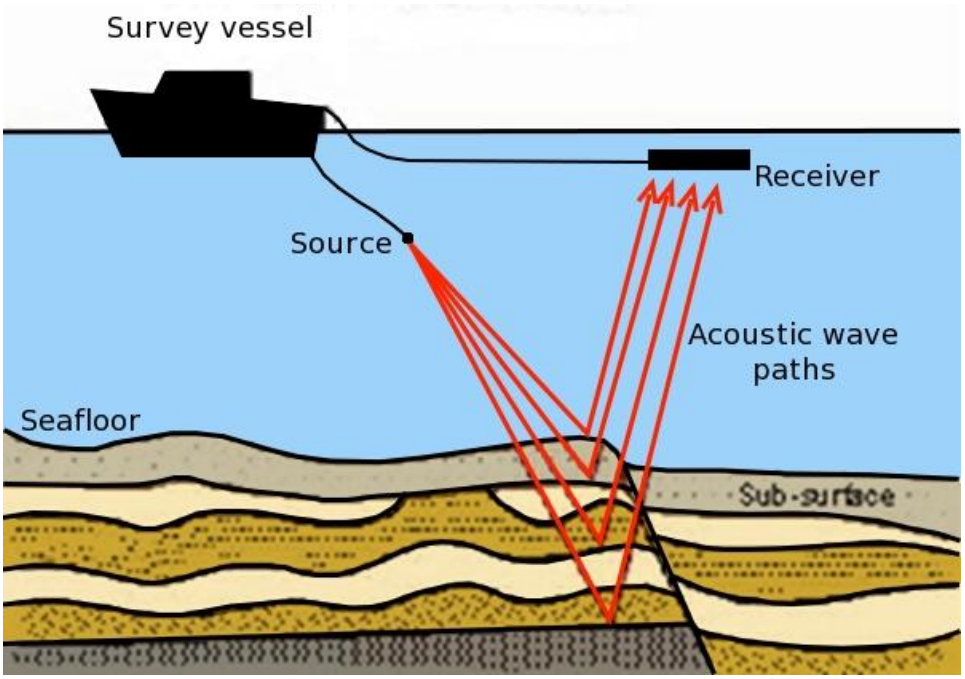


Figure 1.2: Reflection Seismology (http://www.soes.soton.ac.uk/)

of generating the seismic waves. These sources are towed behind the seismic vessel, and shoots acoustic seismic waves downwards into the seabed. On a separate line towed behind the vessel, further behind the seismic source, is a string of hydroponic sensors. Each shot or ping from the source and subsequent collection of signal from the sensor group, is called a trace. A series of traces form a shot gather. An example shot gather can be seen in figure 1.3. This data was created and visualized using RS.

The energy for marine seismology consists of P-waves or Primary waves. These waves are able to travel through any matter, even liquid. These waves will travel through the water and hit the seabed, compressing the material it passes through. At the interface between the seabed and the sea, some of the energy will be reflected back up to the sensors. The rest of the energy will be refracted downwards in an angle given by the density change of the two layers. This refraction can be determined using Snell's law [8]. The refracted energy will continue in the same angle until it again hits the interface between two layers with different density, here a part of the energy will again be reflected towards the sensors, and the rest refracted.

The change in density between layers is what can be seen on the final seismic image, darker spots indicates a bigger change in density then lighter spots. The seabed is such a dark spot which indicates that there is a substantial

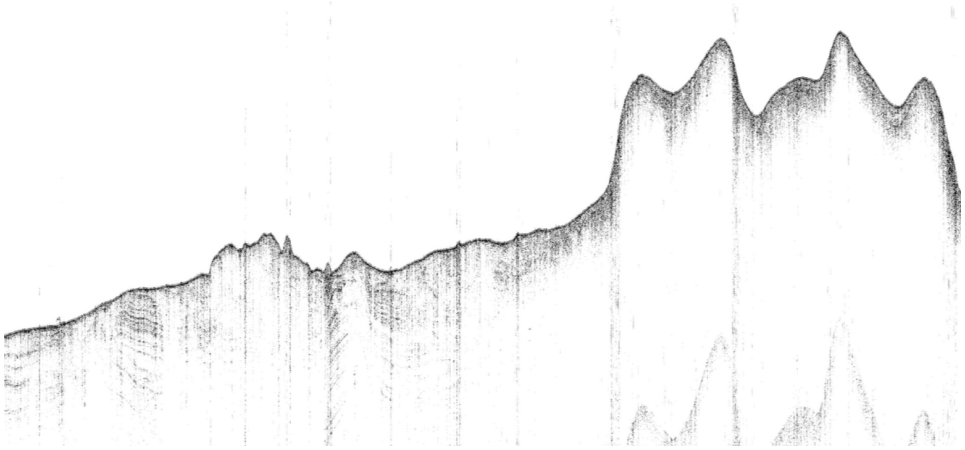


Figure 1.3: Shows a seismic trace from evaluation expedition April 2014, visualized with RS

difference in density between the seabed and the liquid of the ocean.

1.2 Current State of Visualization

To understand the need for a system like RS, a look into the current state of seismic visualization is required. At present time there are two ways seismic data can be visualized; thermal paper plot and the software provided by the seismic system's manufacturer. A seismic vessel has a thermal printer located in the operational room which can print seismic traces via software available on the vessel. These can be printed live, a line at a time, representing a trace. The printers can also print data from stored seismic data. The plotters currently in use at UIT are old technology from the seventies and eighties, and are big and bulky. Even so they produce nice plots of the collected data in the form of thermal paper rolls half a meter wide and many meters long.

The software provided by the manufacturers can also display live seismic data. The analog signal arriving in from the seismic survey is digitalized and displayed on a screen on the seismic vessel. This data will be inserted into a proprietary format which may or may not be well documented. One of the problems with these programs is their inability to go back in time, not only during the time of seismic acquisition, but also during playback of previously collected seismic data. During a live feed or during playback, data will flow from right to left. When new data appears it will push the data on the left side out of the screen's boundary, without any way to go back. The only way to see data already gone from the screen would be to restart the playback from the

Figure 1.4: Thermal plotter (http://delta.geo.uib.no/)

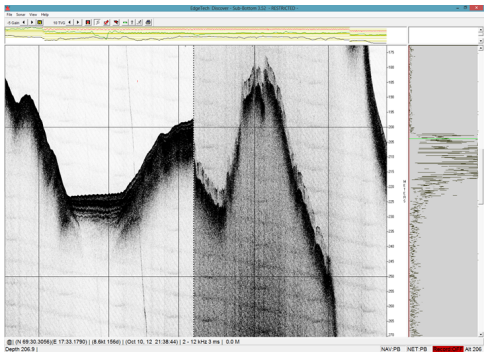


Figure 1.5: Screenshot of Edgetech software

beginning of the file, or in the case of live visualization, to make the program end the file and let it insert newly collected data into a new file. Even so you still have the issue of playback being quirky and not very interactive.

The limitations of these softwares are one of the main reasons for continuous use of thermal plotters. Because of the size of the thermal paper rolls, the scientists will take a roll of seismic paper and roll it out on a large table or on the floor and work on it; making annotations and discussing over this roll of paper. Although this might be adequate, this must be something which should be done equally good or better on a computer. The image in figure 1.6 illustrates this perfectly.

The limitations of the current softwares and the aging plotters cause the department of Geology to be interesting in acquiring new software for visualizing their seismic data. They wanted interactive software to visualize and interact with both live and old seismic data. RS provides this and more. It provides an interactive tool for visualizing seismic data; it does so from a web-based application available to any personal device, anywhere on the

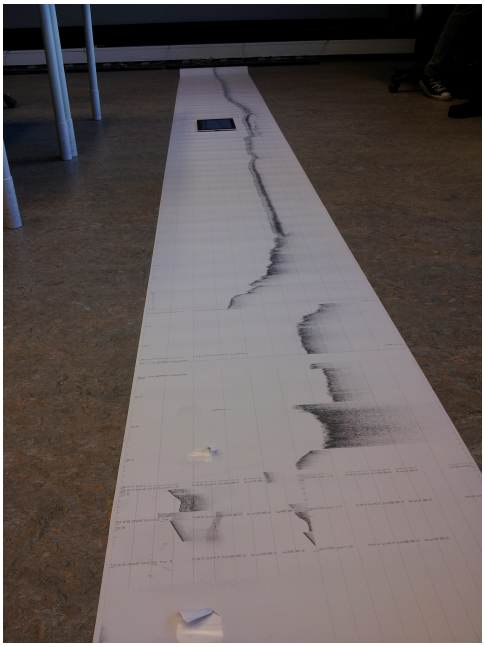


Figure 1.6: A roll of Thermal paper, rolled out on the floor with a IPad on it for scale(The roll was to long to roll out completely)

seismic vessel.

1.3 Contributions

This master thesis contributes by:

1. Providing scientists and students a new tool for visualizing and interacting with seismic data.

2. Using web technology via a tile viewer to present the seismic data is believed to be a novel approach. It provides the end user the ability to analyse and interact with the seismic data with any modern device with a modern browser; this without storing large binary files on his personal device.

3. Demonstration that RS can perform as a processing and visualization tool by evaluating the system by measuring: processing latency and round trip latency. An on-site evaluation is also performed, and an evaluation by a leading expert in the field of seismic acquisition.

1.4 Conclusion

RS provides the scientists with a fast seismic processing and visualization tool. It adds another visualization platform to view seismic data, which is both responsive and easy to interact with.

During evaluation of RS, it was found that it can process 5-10 minutes worth of seismic data into viewable content in just above 4 seconds. This content can be visualized in less than a 1.1 seconds from request time, on both personal computers and personal smart phones. The content presented has been judged as being of good quality from a leading expert in the field, comparable in image quality compared the existing visualization tools. The quality could still be improved by implementing more advanced seismic filters.

RS provides a platform which can be built upon to create an even better tool for processing, visualizing, interacting with and analysing seismic data.

2

RS

2.1 RS overview

Before delving into the architecture, design and implementation of RS, this chapter will give a short overview of the RS user interface, and the functionalities this interface provides. The main functionality revolves around the visualization of shot gathers. This seismic data all originates from proprietary binary formats, but has been parsed, restructured and processed in RS into single format that can be visualized. In the current build the user is presented with a list of shot gathers that in the applications are named maps. This is a list of all available shot gathers currently stored on RS. This list can be seen in image 2.1.

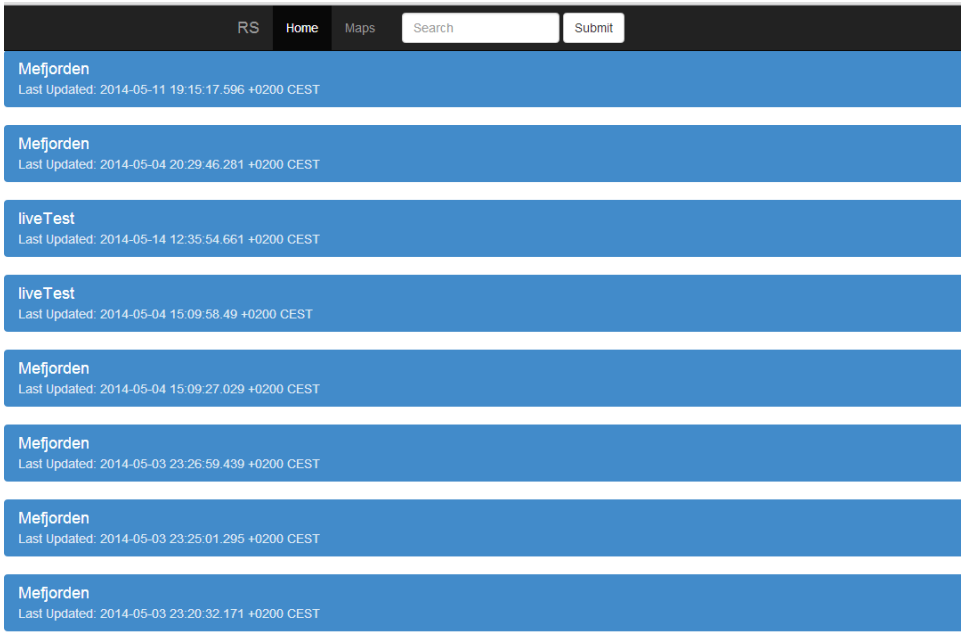


Figure 2.1: List of Available Seismic Shot Gathers

From this list the user can chose any available shot gathers, and will be presented with the actual seismic data. This is data represented as images, which is stored on a server. No data is stored or processed on the client, although the image viewer has built in caching of images. An image of the main user window can be seen in figures 2.2, 2.3 and 2.4. They show screenshots both from a desktop computer and two smartphones.

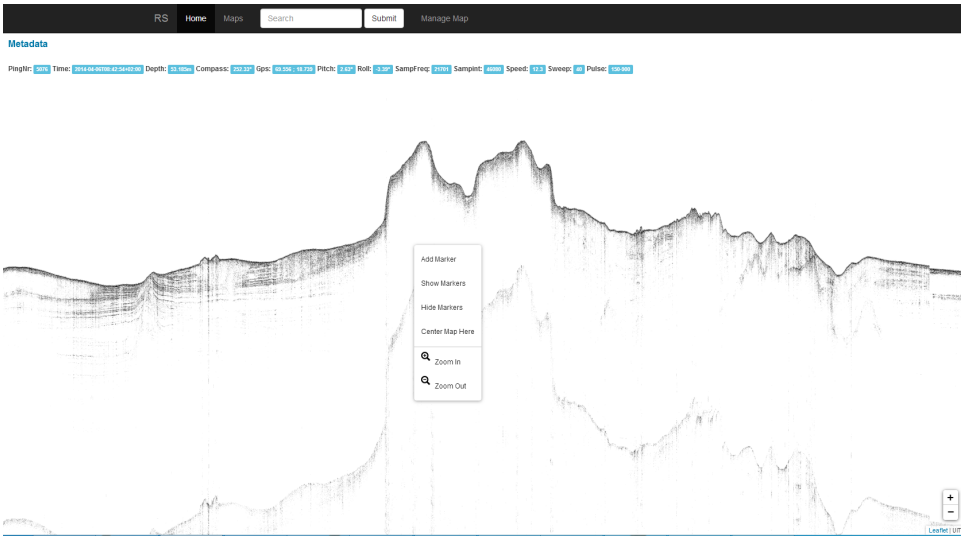


Figure 2.2: RS on Desktop

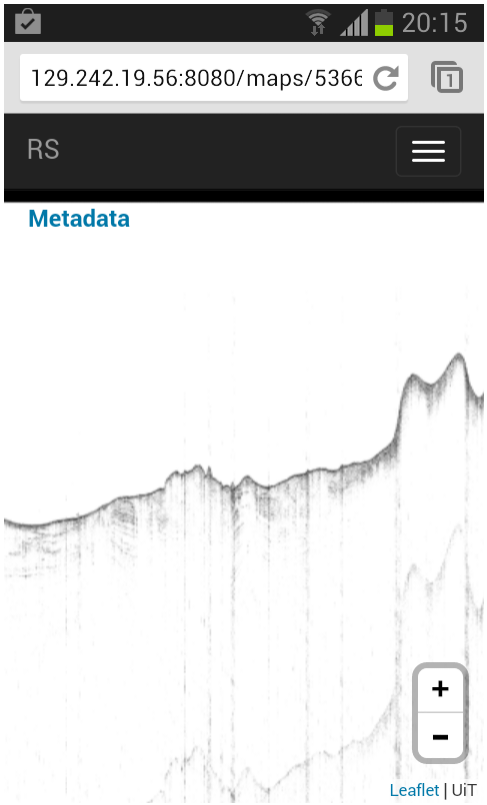


Figure 2.3: RS on Android

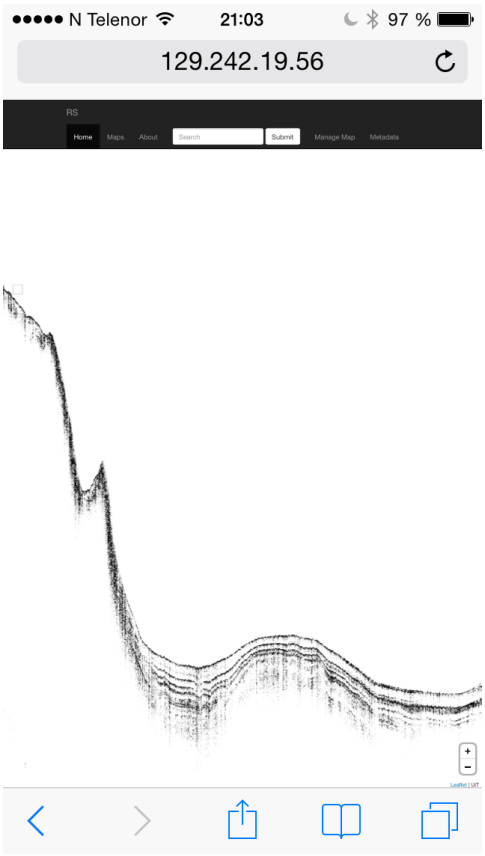


Figure 2.4: RS on Iphone

Here the user sees tiles of images that is created and stored to allow for zooming in pre-defined levels. Interaction with RS is very smooth on modern devices, but older smartphones and pads might lack the processing power to properly interact with RS. This is due to the JavaScript library used, Leaflet[1]. Leaflet does some processing on the client side that might cause the user experience on older devices to be less than optimal.

From the main interaction point, the user can set markers by double click or double tap, or with the use of a right click or long hold click via a context menu. The user can choose to annotate the marker via the same context menu. The context menu is visible in figure 2.2. Creation of markers are an interactive process, any marker put on the seismic shot gather will instantaneously be visible to all users with this shot gather open. Such a marker along with its annotation is visible in figure 2.5.

1. http://leafletjs.com/

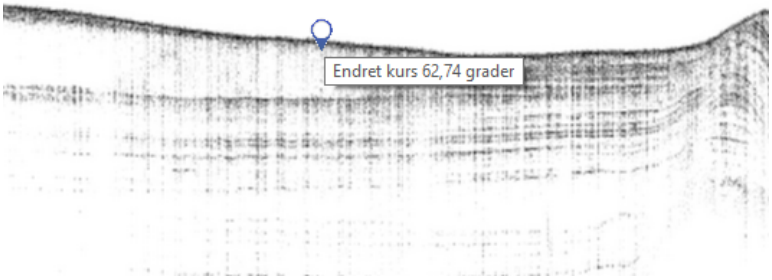


Figure 2.5: Screenshot of Annotation in RS

If a user wants to have a full overview of annotations made or make modifications on the shot gather metadata, he can used the Manage Map feature. This feature presents the user with some managing options and all markers and annotations on the seismic data in question. The manage feature can be seen in figure 2.6

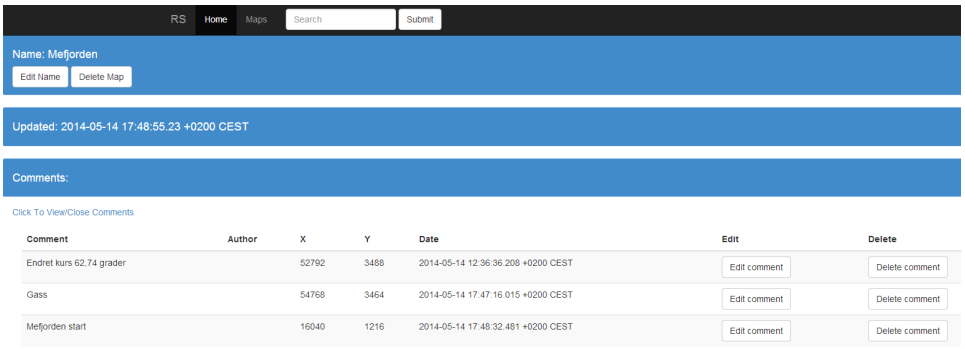


Figure 2.6: Screenshot Manage Feature in RS

3 Architecture

RS is a client-server system for retrieving and processing seismic data from existing seismic formats. Processed data is made available for visualization and interaction through commodity devices. Figure 3.1 illustrates the architecture on a high abstraction level. This chapter explains the idea behind the different system layers and their place in the overall system architecture.

3.1 Arcitecture Overview

The system can be split into four components: a data collector[1], backend, front-end and clients. The data collector is responsible for collecting input files from various sources, and makes it available to RS. The backend formats the heterogeneous data into an appropriate format, here data is filtered and made available as images and metadata to the front end. This is done via the file system. The client requests content made available from the Front End server.

The various components are split up into separate processes to make it easy to swap out components without rewriting a lot of code. Every separate component except from the client, works independently. The client needs data from the Front End to work properly. The data collector should be able to

1. Note that the data collector isn't currently implemented

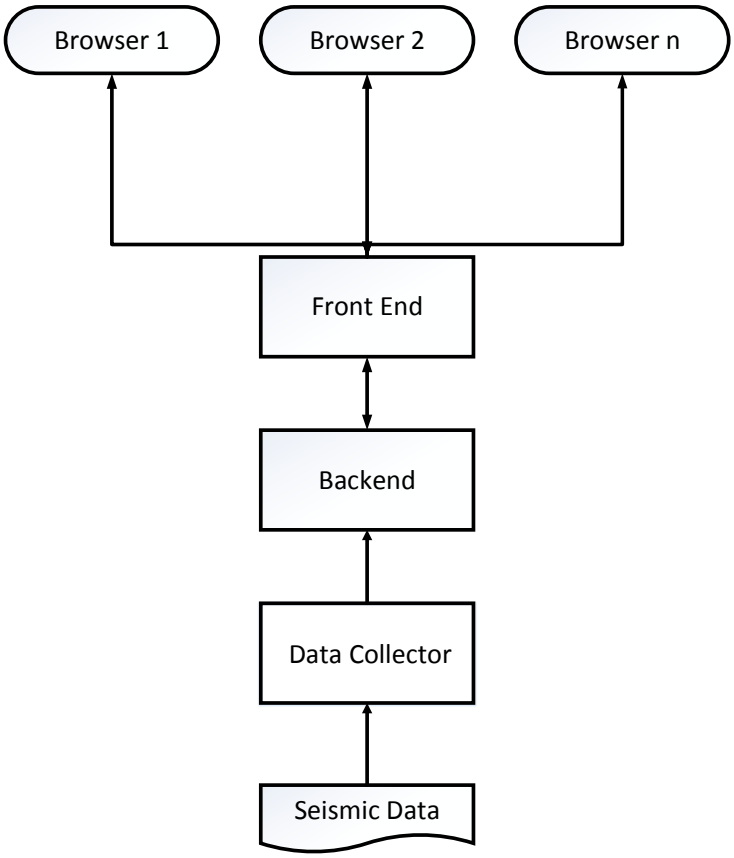


Figure 3.1: RS general architecture

run separately, even if the backend crashes, collecting data that later can be used by the Backend when it again is operational. The backend can process and create data for visualization without the Front End, as the Front End is independent from the Back End. Separating these components also makes the system more robust as a crash in a single process system would crash everything.

The architecture would not scale up to the millions, but is more than good enough for the intended usage. It was thought that a more complex architecture would only complicate the system without adding anything valuable. The system is intended for a seismic vessel with a relative tiny crew, and a relatively small contingent of scientists and students who both are the intended target audience.

4

Design

This chapter goes into more details than the architecture presented in the previous chapter. It delves deeper into the Architectural components and explains the interaction between and within these components. Presenting the design built upon the architecture.

4.1 Data collector

RS uses binary data created from existing software as data input. This data comes in various formats. Currently RS support seismic data in 2 formats; Elics' Delph format[1] and Edgetech's JSF format[2].

The seismic data originates from analog signals, and are converted to digital by Analog-to-digital converters. This creation of binary data is done on separate machines, from now on called the source machines.

Together with the seismic data, these files also contain relevant metadata such as: time of day, geographic position, speed of the vessel, pitch and roll of the vessel. Which metadata types are available is dependent on the format. Edgetech provides rich metadata for each trace, while Delph has lim-

1. No online documentation
2. http://www.edgetech.com/docs/

ited metadata available, and zero documentation on how this metadata is structured within the binary data. RS metadata is thus not available for this format.

As RS currently depends on these pre-created binary files to produce visualizations of the seismic data, it cannot provide a live feed such as the source programs can. How this can be achieved is discussed in the future work chapter, chapter 9.

4.1.1 Data Acquisition

The softwares responsible for creating the binary input files are located on the source machines, running on the same closed network as RS. These files are accessible to RS via remote access to folders on the source machines. The current version of RS does not support automatic transfer of files, but it needs manual copy from the source machine. It is envisioned that RS will support either a user configuration of a source folder, or force the user of source software to store new seismic data in a pre-defined folder on the source machine. Both scenarios are possible, but also have drawbacks. A third option is to go around the source softwares and formats and used data directly from the analog source. This will be discussed in chapter 9.

These source softwares allow for configuration of the output size of the seismic data, effectively ending a seismic file when it reaches a given size limit. From this point the new seismic data will be written to a new file. It is presumed that future versions of a source software will allow for this configuration. By limiting the size of the binary data files, these will be available sooner for processing on RS. RS needs a steady input of these binary files to simulate "live visualization, but is designed to handle data seismic data of any size.

4.2 Backend

Figure 4.1 shows the inner design of the backend, together with its connection to the Data collector and Front End.

4.2.1 Data Conversion

There is no limit on how many formats RS can support, and new formats that can be supported in the future. To be supported, a future binary file format needs to be parsed with a custom parser that alters this data into a uniform

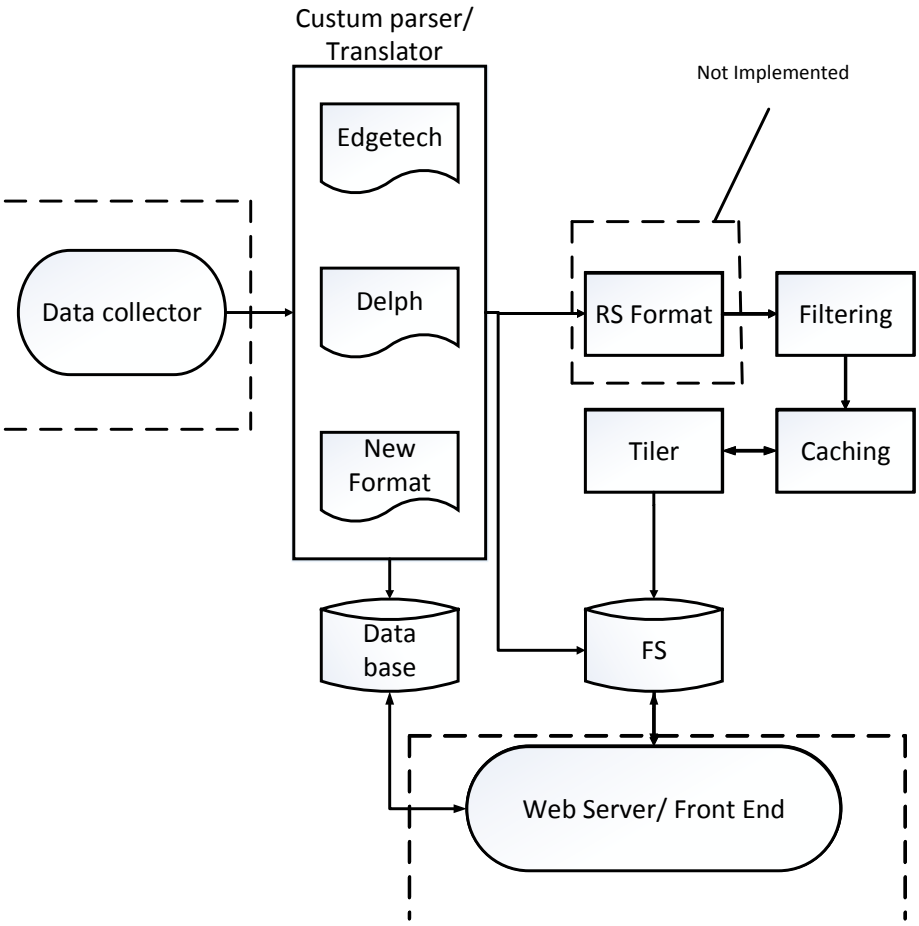


Figure 4.1: RS Backend Design

format usable to RS. This is to avoid working with many formats during the seismic processing and creation of visualizations.

Metadata from the source formats are stored separately in a database. The

reasoning for this design choice is to accommodate fast and reliable storage on creation, and fast retrieval when metadata is requested from the web clients. A future planned RS format will also contain this metadata to work in RS without any dependencies to the source formats.

In a future version a new binary format will is presented, RS format. This format will be comprised of both metadata and the actual uniformed seismic data. The RS format will be stored to disk, for the possibility to redo parts of or the entire seismic shot gather with different filters. The ability to apply new or a different set of filters to the dataset can uncover new and interesting details that the default filter might not. The default filter might hide interesting features that might otherwise be present with different filters. This will be discussed more in chapter 9.

4.2.2 Data Filtering

Data from a seismic acquisition contains, not only the representation of the reflections, but also noise. Noise in the signal can be caused by a number of either human contributed or natural sources. Surface waves are a common source of noise; these waves can also tug seismic equipment due to the seismic vessel having to change speed. Other noise factors might be: the swell created by the seismic vessel itself, current, faulty equipment or nearby seismic activity. No matter the cause, removing noise to give a better noise to signal ratio is crucial for improving the readability of seismic data.

The data is exposed to a series of filters. Each filter implements an algorithm for removing noise originating from a specific source. This is a standard filter series, that might be modified by user in a separate configuration file. The filters that is used will be discussed in section 5.2.2.

4.2.3 Data Caching

As data flows in via user configured sizes, ranging from small files containing a few traces, to GB's of data from hours of data collection, uniform and filtered seismic data is cached before being ready for visualization. The system needs a certain amount of traces to start creating the visualization, and will cache up the incoming data in memory until it has enough. The caching will be explained thoroughly in the next chapter, but it has to be mentioned that the cache is the temporary memory storage from which the image tiles are created.

4.2.4 Metadata Storage

Metadata is stored in a database, ready to be fetched when the user requests it. The metadata is inserted into the database while parsing the binary input files. The metadata for a separate trace is stored as documents in MongoDB[3]. These documents are stored in a single collection of documents. RS currently supports the following metadata to be visualized.

- Longitude and Latitude
- Compass Heading
- Pulse Frequency (start, end)
- Sweep Length
- Depth
- Pitch and Roll
- Time (year, day, hours, min, sec)
- Course
- Speed
- Sampling (interval, frequency)

Again this is only true for Edgetech binary files, but some of these or others could be valid for Delph binary files at a later stage in the development process. MongoDB collections are schema less, so a different format could support alternative or additional metadata types.

4.2.5 Tiles

Data is visualized as tiles to the end user. The tiles are small images, and combining these images forms a larger image. This thesis will from now on present these images as tiles as it correctly describes the images. RS conforms to the method used by Tile Map Service specification (TMS) [4] to split a larger image up into a pyramid of images, enabling different zoom levels to be

3. http://www.mongodb.org/
4. http://wiki.openstreetmap.org/wiki/TMS

visualized to the end user. The tiles can be seen in figure 4.2. A tile is 256 x 256 pixels. A tile from a zoom level below is comprised of 4 tiles merged into one, while still being the same total size, 256 x 256 pixels. This pattern is followed all the way until zoom level 1, which is the last zoom level.

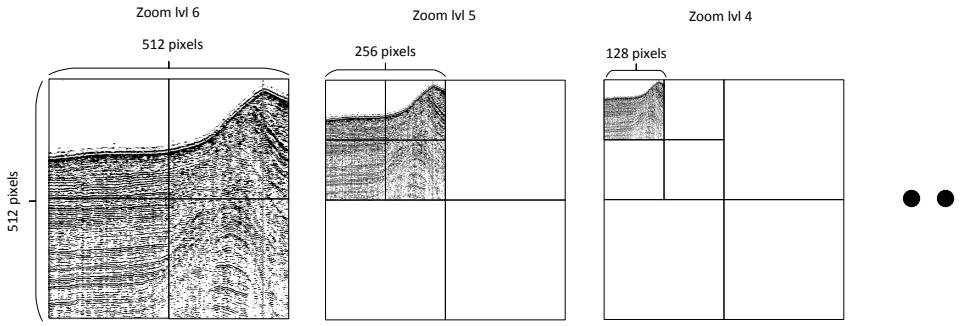


Figure 4.2: RS tile zoom structure, images taken from RS. Image on the left shows 4 tiles, this is needed to make 1 tile on zoom level 5. This data will be part 1/4 of the tile on zoom level 4.

The tile creator collects data from the cache and creates data to be visualized by inserting them into the file system. The tile creator creates tiles for each zoom level needed to interactively visualize the data. The Data is resampled and re-cached for use on different zoom levels. How and why this is done will be discussed extensively in the next chapter.

4.2.6 Storage

File System Storage

RS stores the following data on the servers file system: tiles and appended version of original files. The tiles are stored in the pyramid form mention mentioned earlier and is visible in figure 4.3.

Database

Besides storing the actual tiles in the file system, two types of metadata are stored in separate database tables. This is the metadata representing the complete shot gather, and the metadata that correlates to each trace. The database is a single MongoDB instance running in a non-replicated and non-shared environment. MongoDB is a NoSQL document store that stores

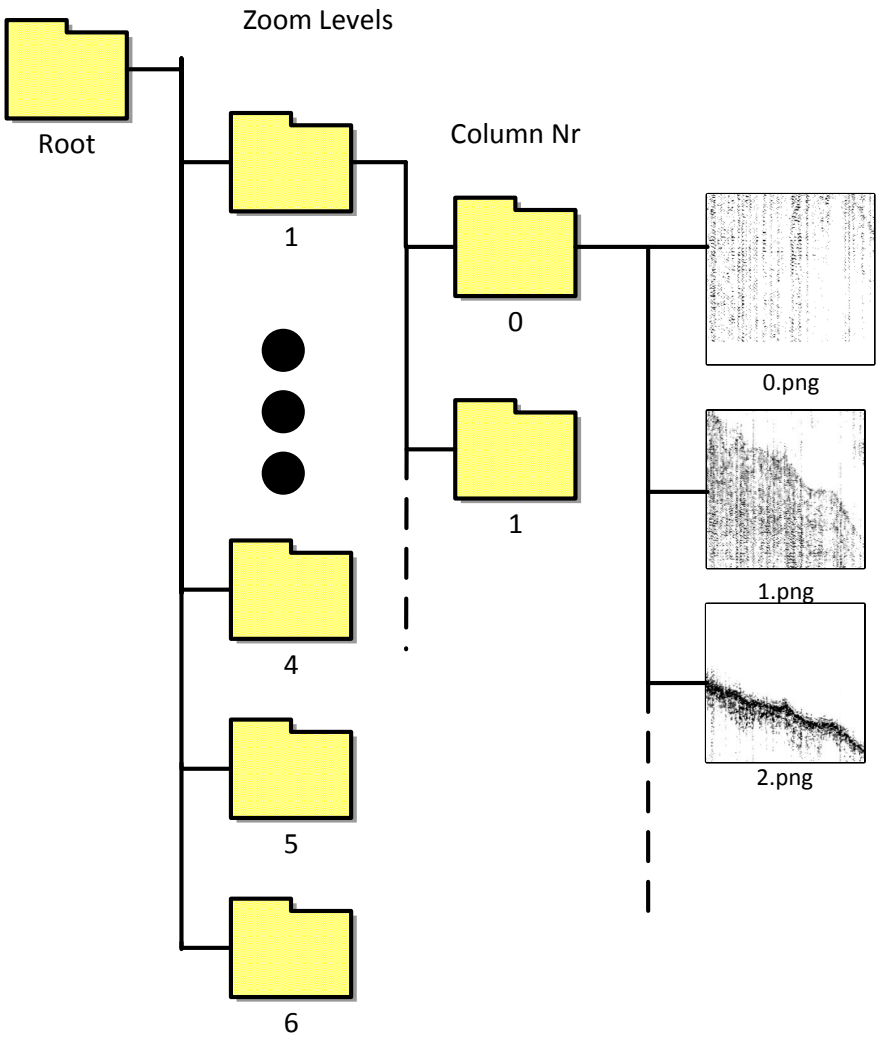


Figure 4.3: RS file structure for tiles

data or documents as Binary JavaScript Object Notation (BSON). For better availability and durability the database should be replicated and be located

on separate machines, but at this stage of development the mongod[5] instance runs on the same machine as the server. To support faster reads, MongoDB supports indexes. Every document is automatically indexed by its id, but further indexes are created to support faster reads for the most common read patterns. Creating indexes will harm the write speed but not enough to ignore their power. This increase in insertion time is due to the fact that every entry has to be added to a collection, and also added to an index. A drawback in this design is complicating the export of RS data from the system, as tables from MongoDB would have to be exported separately for data to work as intended on a separate server.

4.3 Front End

The Front End server is located on the same machine as the backend. Alternatively the front end could have resided on a separate machine. It is not connected to the internet, but is connected to the vessels local network. This is a closed network. It can serve multiple browser clients that also are connected to this network. This included any mobile device or personal computer with a modern browser, which supports JavaScript and websockets[6].

4.3.1 Websockets

Websocket is a light weight, Bi-direction and full duplex communication channel over TCP. It is a relative new communication technology, created under the HTML5 initiative. It is considered to be the next evolutionary step after Ajax[7]. It is a technology which is not created to replace Representional state transfer (REST), but can work alongside REST. Websocket is initialized by the client, but after initialization both client and server can communicate over a single bi-directional and full duplex channel. It has an open connection after initial http handshake (until timeout or closed by either part), and have a minimum of 2 bytes header size after initial handshake, this means that websockets is light weight and fast.

There are no pre-defined structures on which messages should be built. Any communication structure needs to be defined by the developer. RS has a websocket message structure based on CRUD[8]. This structure is shown in

5. http://docs.mongodb.org/manual/reference/program/mongod/
6. http://www.websocket.org/
7. https://developer.mozilla.org/en-US/docs/AJAX/
8. http://docs.mongodb.org/manual/crud/

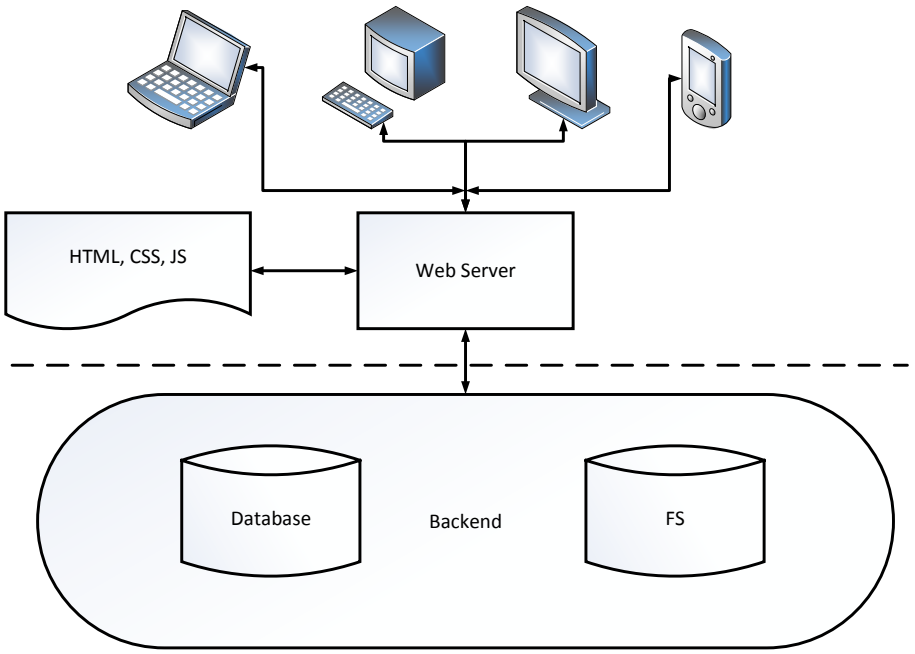


Figure 4.4: RS Front End

table 4.1.

4.3.2 Web Server

The server serving the clients is a multi-threaded web server. It handles requests for HyperText Markup Language (HTML), tiles, JavaScript and Cascading Style Sheets (CSS). The system uses the websocket solution for handling requests which otherwise could be handled by a REST interface. These are POST, PUT, and DELETE requests. The reason behind this design choice is to

Operation	Type	Data	Description
CREATE	coord	x, y, comment	For given shot gather, and updates all clients with shot gather open
READ	coords	-	Returns all markers to client
READ	metadata	Trace number	Returns metadata for trace
UPDATE	imageName	New name	Updates and returns new value
UPDATE	coord	x, y, comment	Updates a comment on coordinate
DELETE	coord	x, y	Deletes a marker

Table 4.1: RS Websocket Interface

Resource	Description
GET /	Returns a list of all seismic maps available, in future this will be a front-page for users
GET /maps	Returns a list of all seismic maps available
GET /maps/:id	Returns seismic map with unique id(bson id string)
GET /maps/:id/manage	Returns page to manage a seismic map, information includes tags and comments etc.
POST /search/query[a]	Return list of maps based on the query

[a] An exception to keeping Post requests on the websocket is the search request

Table 4.2: RS Rest Interface

create dynamic web pages which can be altered by interaction by the current user, another user, or the server. This can now be done without having to create a new connection for every request, thereby saving time on initialization.

The server has no login mechanism. This is by design, and is based on the specification given by the Department of Geology. Any user should be able to do anything, and everyone on the vessel with access to the secure network, is considered trusted. It is feasible that a future version of RS would run on land. This would allow authorized personnel on land to view data collected from various vessels over a long period of time. This would possible require a login mechanism, but the login could be to access the network, not RS itself. Even so, when dealing with sensitive data it would be prudent to have additional security mechanisms in place.

The webpages are requested through a restful interface, where the web server handles requests on a pre-defined port. This interface can be seen in table 4.2.

4.3.3 Client

The end user interacts with RS via the web application. The main content of the client is made up of HTML, JavaScript and CSS, which is served from the server. For static content the client connects via the restful interface, but for potential dynamic content the clients connects via the websocket. The client connects to the server via the websocket and is connected to it as long as he stays on content related to a specific seismic map. Note the use of the term "map", this will be the term used for the shot gather when dealing with RS visualization.

Creating the RS client as a web application, automatic creates cross platform compatibility. It is now available on any operating system that supports a modern browser. Any modern device which has some processing power can use RS as intended.

Leaflet

For visualization of the seismic data the choice fell on the JavaScript library Leaflet[9]. Leaflet is an open source tile viewer, which can be compared to among others: Google's Google Map[10], Openlayers[11] and Mapbox[12]. It is designed for interactive maps with an emphasis on quickly allowing creation of mobile friendly custom maps with simplicity, performance and usability in mind. In can for instance be used with the maps provided by openstreetmap[13]. With maps in mind, Leaflet supports; amount other, georeferencing, tile layers, markers and overlays.

Even with the design emphasis being on actual maps, Leaflet can be used to visualize non map images. To use Leaflet as a zoomable image viewer it is required to present the image as tiles, small images representing a whole image, and present these tiles in the pyramid structure shown previously in figure 4.3.

9. http://leafletjs.com/
10. https://developers.google.com/maps
11. http://openlayers.org/
12. https://www.mapbox.com/
13. http://www.openstreetmap.org/

5

Implementation

This chapter delves further into the details of the components presented in chapter 3 and was further explained in chapter 4. Implementation details for each component are presented, important algorithms explained, and some implementation choices discussed.

5.1 Backend

5.1.1 Data Conversion

Edgetech

The Edgetech binary file is a set of data messages. Each message begins with a header of 16 bytes. This header contains information like message type and message size. RS uses the messages with message type 0x80, which contains the actual seismic data. These messages start with a metadata header of 240 bytes, which forms the basis for RS's metadata. The rest of the message, message size - 240 bytes, are the seismic data. This seismic data consists of 2 parts, a real and an imaginary part.

RS stores part of the metadata in the MongoDB database, it currently uses 34 bytes of the original 240 bytes. The real and imaginary part is converted to a complex trace[12]. Each pixel value keeps the original 16 byte value.

The actual seismic data from the Edgetech is converted into the uniform

format and made available for further processing.

Elics Delph

The data collected from the Delph system consists of 2 files. One file that shows similar characteristics as Edgetech's binary format, as it provides metadata and then the trace data in a series of messages. The second file contains pure metadata. It provides information about the first file, and user settings.

Currently there are little metadata from the Delph format that can be used by RS. The binary Delph format presented has no known metadata specification; beyond than that it contains some positional and time data.

The actual seismic data is in a compressed format. Each colour value that can represent a pixel is stored as a nibble. This compressed storage prevents exact representation of the Delph format on RS. More on this subject is discussed in chapter 9.

The parser extracts the necessary data to from the binary file and inserts it into the same uniform format as with Edgetech, 16 bits per pixel value.

5.2 Data Caching and Tiling

The tile creation process was implemented in two versions. Version one was a prototype, which was implemented using Go[1] and Python[2]. It created a whole image of all the seismic input data from a single file. This version scaled poorly, and was quickly discarded. One of the main motivations for scratching this version was the false belief that the Department of Geology was without software that could visualize live seismic data. This meant that the tiling needed to be done quickly, to accommodate this "live" visualization. This belief was due to miscommunication, but the result was a much better and faster implementation.

The second implementation was a fast, in-memory scheme that dealt with caching the data for each zoom level, and used these in-memory data caches to create images for each zoom level. This implementation has some issues when it comes to excessive memory usage, both in total memory use and peek memory use. A little more about this will follow in the next chapter, where

1. http://golang.org/
2. https://www.python.org/

memory usage is evaluated.

5.2.1 Prototype

The system parsed a file from the source machines and created an image of the entire seismic data within this file. This image was split into tiles with the help of a python program named gdal2tiles; which originates from the GDAL translator library[3]. This was done to quickly show the validity of choosing leaflet as a feasible way to visualize the data. Gdal2tiles proved to be an easy way to create tiled images in the pre-required format. When a new file was introduced into the system, it was again parsed and a separate image created. This image was then stitched together with the large image from the first file and totally re-tiled with gdal2tiles. The more files the system received the slower it could process the final data. A version that attempted to only create the new tiles was attempted, but quickly stopped when the complexity of the operation was uncovered. The final visualization was aesthetically pleasing as it was blessed with the image algorithms provided by GDAL. The performance on the other hand was far from satisfactory.

This version scaled poorly due to a number of circumstances. Firstly, gdal2tiles is a single threaded python program. It has a multi process implementation available, but it was unstable and crashed constantly. Secondly, due to the nature of how gdal2tiles creates tiles. It created tiles based on a large image, albeit being slow, worked fine for large single files. But for partitioned files it proved difficult to create tiles with the correct name, and there would be a visible line between tiles between files. This was due to the fact that the last column of a file would potentially be only a few pixels wide. In retrospect, the full image from a file could have been stretched so total width modulo tile width (256) was 0.

5.2.2 In-memory Version

This section will begin with an overview of the implementation and later follow up with more detailed implementation details where it is deemed necessary.

This version uses an in-memory cache system, where data for all levels of zoom are created and cached in memory as uint16 bytes slices. A slice[4] is built on top of the array literal, and provides a flexible data structure. Instead

3. http://www.gdal.org/gdal2tiles.html
4. http://blog.golang.org/slices

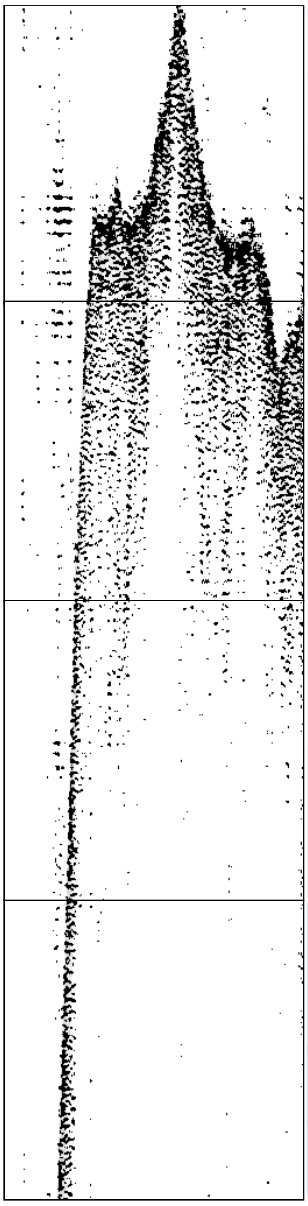
of passing the underlying array as a value, the slice literal allows you to pass the slice or part of the slice as a reference. Each zoom level has its own cache, and the tiles for a level are created from its corresponding cache. RS takes advantage of having 16 GB of available ram on the server machine, and this allows for relatively large data structures to be stored in memory. Going back in time this might not have been feasible as memory often was limited by the underlying architecture.

Before the tiles are created, the entirety of the filtered seismic data is appended to the level 6 cache. From the level 6 cache the 1:1 tiles are created. These are tiles that hold the same amount of data points as the original seismic data. This is a statement that is close to the truth, but dependent of the speed of the seismic vessel and the time frequency of traces, the tiles might need to be stretched in width with a factor based on these parameters, the stretching factor. Currently this factor is hard coded into a configuration file, but should be made dynamic if development on RS continues.

For RS to start making a column of tiles, it needs a data set of seismic data corresponding to 256 traces. This number is divided by the stretching factor, so for a stretching factor of 4 there is a need for 64 traces. Note that only the data from the level 6 cache need stretching. The data in lower caches originates from the level 6 cache and is thus already stretched.

If there is a zoom level below the current level, the dataset that is used to create the tiles, is resized to half height and half width. This means that the dataset is reduced to 1/4 of the total size. This downsizing will produce a dataset of the scale 1:4, from the dataset it was reduced from. The downsizing is done by a average algorithm, which is explained in section 5.2.2. The data used for tiling, which is either the stretched data, or data from a higher level zoom is presented to the tiler. This data is called the column, as they represent a column of images in a bigger image set. The column is split into tile sized datasets, 256 x 256 pixels, and each tile is created into an image within a separate goroutine[5] to utilize the multi cores on the server machine. How the slice structure lies in conjuncture to the placement of those values as pixels are shown in figure 5.1. In this figure the slice is visualized as an image. It can be seen that data that becomes pixels in an image, are located in a structure that starts at top left and continues vertically in terms of the final image. This goes against that standard that has values going from top left and horizontally, again in terms of the image. The reason for this can be traced back to the way the input data is collected and placed in binary files, a trace at a time. And with a potential height difference of traces it would be very difficult and costly to re-arrange the values.

5. http://golang.org/doc/effective_go.html



0	16	32	48
1	17	33	49
2	18	34	50
3	19	35	51
4	20	36	52
5	21	37	53
6	22	38	54
7	23	39	55
8	24	40	56
9	25	41	57
10	26	42	58
11	27	43	59
12	28	44	60
13	29	45	61
14	30	46	62
15	31	47	63

Figure 5.1: Pixel data structure in slice, This can be seen as a column of tiles, scaled down to 4 x 4 pixels to show the concept.

A tile created within a goroutine is then stored and named dependent on: its position in the slice, its zoom level, and its column number. At any zoom level, the first column slice will be numbered 0, and stored in a folder so named. In

this folder the tiles or images are stored. This folder will lay underneath the zoom level folder, which will be named in accordance to its zoom level. For level six this folder will be named "6". The tiles themselves are named from 0 to n. Tile 0 being the tile on the bottom of the column.

After all goroutines has started creating tiles, the tiler will check if there is enough data to create tiles for the level below. If there is, it will create a column for this zoom level as well, and continue down the zoom levels until there isn't sufficient data in a cache belonging to the zoom level in question. When the system cannot make more tiles it will attempt to fetch another file for data conversion and subsequent tiling. If another file is found, it will be parsed to see if it follows the same name convention as the previous one. If it does, it belongs to the current data set. If not, it is a new data set coming up in the pipeline. This last case will trigger the first map to end, forcing creation of all tiles from all remaining data.

Cache

The caches used to hold the data are in memory slices stored in a Go struct[6], and an instance of the cache is passed as argument to the tile creator. After each unique identified map has completed its course and a new map with another identity is being processed, a new cache will be initialized. This is done to allow freeing the reference to the previous cache, and thus allow this allocated memory to be garbage collected. After a part of a cache has been used to create a column, it will be removed from the cache. Any new data belonging to this zoom level will be appended to this cache slice.

Filtering

The data acquired from external sources are filtered using various techniques. The goal is to improve the signal to noise ratio, and present visualizations that can be useful to the end user. Data is mainly filtered using band pass filters. Both high and low frequencies consist of noise. The most noteworthy are the low frequencies as they are represented as black on an image. Although high frequency noise is close to white, they can be removed using a low pass filter. Although not as visible as the black pixels, it is still noise. The band pass filtering is very basic and makes all pixels over and under a given value white. If you filter too much you lose some of the signal, and if you filter too little there will be important details hidden behind noise. This cut of value can be tweaked in a configuration file. But it is envisioned that this should be done

6. http://www.golang-book.com/9

via the client.

Another filter present is a median filter. This filter is used to smooth out pixel data. Although it is not a perfect filter, it smooth's out an otherwise jagged pixel image. It also fills in some gaps that could be due to signal loss or a low trace shot frequency. The value of a pixel is calculated by taking the median value of a 2d window of 9 values, where the value itself is the middle value and adjacent values form the remaining. An example of the median algorithm is seen in figure 5.2. The current implementation does not resample the edge values.

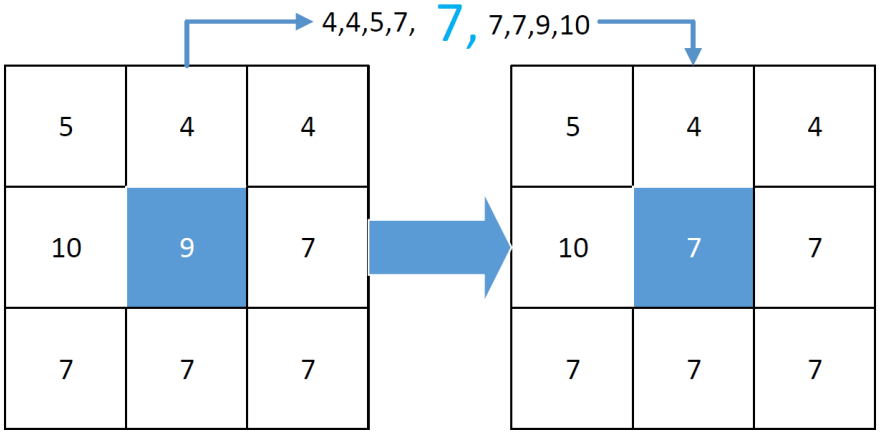


Figure 5.2: Median filter

Tile Creation

The tiles are created from the pre-filtered and cached data. After the external data is loaded in to the system and into the cache, the tiler will go through each cache, one by one, starting at the level 6 cache, creating a column, and check the next cache. By design the tiler does not complete every column and subsequently every tile for a level before moving to the next cache level. This is done to not delay the creation of tiles of levels below, but to gradually create more content on each level. The basis for this implementation is the idea that there was a need for "live" visualization. And this implementation would better simulate such "live" creation of content.

Data for the level below is created by resampling the data from the current zoom level. The data is resampled into half the original size, half width and half height. For the tiler to create tiles for a level below the current level, it

requires 2 column of the current level. This means that a column of tiles will not be created until there is enough data to create one. A consequence of this is the delayed tile creation for level 1; as it requires 32 level 6 columns until it will have enough data to create a complete column of tiles. This implementation choice was done with execution speed in mind. It was the belief at the time that re-creating tiles or appending to tiles would slow the live visualization process. But none the less, there is no loss in data; the creation of lower level tiles is just delayed.

A future version of RS might support creation of partial columns, where a portion of the column is created when there are data available. This would present the end user with data for all levels quicker, but would slow down the overall process as columns and tiles would need to be recreated or be appended to.

For an append to work, the tiles that are images would need to be decoded, before new colour values could be added, finally these images would need to be re-encoded.

Resampling

As stated before, data for a cache below the current level needs resampled data to be able to create tiles. The resampling algorithm in us is an implementation of an average interpolation algorithm. It resamples a pixel based on the average values of itself and 3 neighbours. This can be seen in 5.3. This solution is not optimal in terms of quality, but it provides good quality and fast down scaling of images. A Nearest Neighbour solution was also created but failed to meet the expected image quality. Other solutions that could be implemented in the future are among other: bilinear interpolation, Bicubic interpolation or a Lancos algorithm.

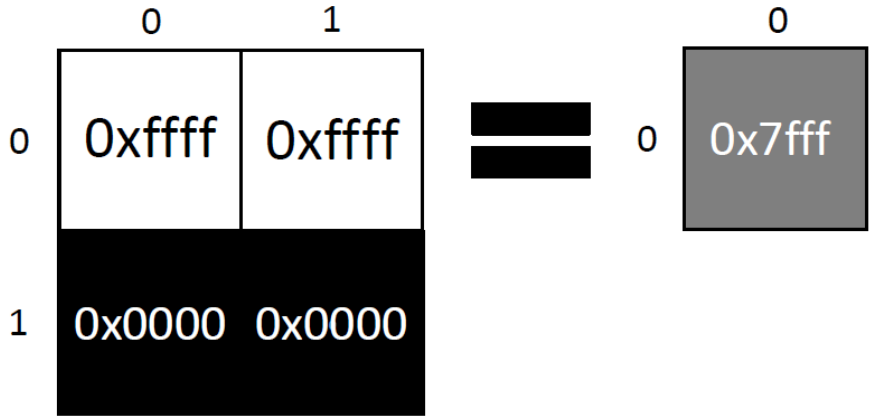


Figure 5.3: Average interpolation

If data needs to be stretched, they are stretched using a linear interpolate algorithm. It stretches the column x times in 1 dimension by approximating the values in between pre-existing pixel values by using these existing values. The algorithm takes 2 known colour values and estimates the colours between these values. This is done by drawing an imaginary linear line between the known points and calculating the slope by using the following mathematical formula:

$$m = \frac{y_1 - y_0}{x_1 - x_0}$$

The x_0 value can be random but x_1 is x_0+ distance in positional value on final image. The Y values are colour values. After calculating the slop, any x value in between x_1 and x_0 can be calculate with the use of the same formula and one of the original points.

User configuration change

To accommodate alternating depth during the acquisition of seismic data and subsequent binary data creation at the external data source, the external source supports changing the stored height for each trace. RS supports such a configuration change, both in mid file and between files.

For an increase in data size in mid file, the entirety of the file is re-run from the beginning. The part that has a lower height will be padded for a uniform height for all traces. This is a quick fix solution to support this feature.

Handling a decrease in trace height is easier as it only involves padding. Redoing data in this case would make no sense as it would cause data loss.

This solution is sub-optimal. A new and better solution is planned and will be presented in chapter 9.

5.3 Front End

As mentioned in in section 4.3.2, the Front end consists of a web server handling requests on a REST interface, and on an open websocket. This server is constructed by using the server provided by the http package[7] in Go. This server is multi-threaded, and can handle enough requests to handle the low user mass expected to use RS. It serves HTML by utilizing the template package[8], this allows for static HTML to be combined with data from Go structures. This static content is requested through the REST interface.

5.3.1 CSS

The CSS of RS's client is strongly influenced by the Bootstrap[9] framework, which provides easy to use and aesthetically pleasing CSS and JavaScript. It also supports a responsive web design by adapting to the user changing his window size or using a device with small resolution. It was not deemed necessary to provide a self-made and complex CSS implementation, while bootstrap and similar implementations where available. Besides bootstrap, some CSS was written to accommodate Leaflet within the layout.

5.3.2 Websocket

The websocket implementation on the server side is influenced by the websocket implementation found at [10]. It differs slightly from this by adding a map identifier. This identifier is used by the front end to only push relevant data to the viewer of a certain map.

7. http://golang.org/pkg/net/http/
8. http://golang.org/pkg/html/template/
9. http://getbootstrap.com/
10. http://gary.burd.info/go-websocket-chat

5.4 Web Client

The web client is created from the HTML, CSS, and JavaScript served from the Front End.

When the user has connected to a map, he sees the shot gather. Here the user interacts with an instance of Leaflet. This instance is responsible for fetching tiles from the server based on the: map id, the zoom level, and client viewpoint. Here the user can annotate the map via a right click or long hold menu. Markers are visible on the screen via a Leaflet overlay. Markers can be inserted by any client, and will be quickly visible on all clients viewing the same map. This is done via the websocket. Any user can annotate the markers, leaving notes as he sees fit. Unfortunately there is no collaborative writing via the websocket when writing notes. This means that there might be a conflict if two clients annotate on the same marker at the same time. This is temporarily solved by letting the last save overwrite the first. Users can also view and hide all markers via the same menu.

The client supports zooming in or out. This results in fetching tiles belonging to a higher or lower zoom level respectively. Leaflet caches tiles to avoid re-fetching tiles from the server.

The user can view and interact with any map, with no regard for other users. He can search for old maps via a search bar, get a list of all available maps, or manage a specific map, altering map name or notes made by any user.

When a client requests a seismic map via the restful interface, a websocket connection is also created. This is also true for connecting to the GET /maps:id/manage. The server saves the map id corresponding to the map the client is connected to. When a client requests something, the server will know which map the client is referring to, without the need for this to be specified at every message. This also allows the server to potentially send new information to all clients that have the map in question open. The websocket message system used by RS is designed based on CRUD (Create, Read, Update, and Delete) which are the major function types available in a database. All communication after initialization is done with JavaScript Object Notation (JSON)[11].

For adding right click or long hold menu system, RS uses the leaflet extension Leaflet-Contextmenu[12].

11. http://www.json.org/
12. https://github.com/aratcliffe/Leaflet.contextmenu

6

Evaluation

The goals of the evaluations in this chapter is to both evaluate the system with respect to scalability, processing speed and visualization quality; and by doing this, show the suitability of the system as a tool for visualizing seismic data.

RS is all about visualization and interaction, and with that in mind, both run time on the data creator, and the round-trip latency for loading the seismic map to the web application are important areas to evaluate.

Without creating images from the seismic data there would be nothing to visualize for the end user. It is therefore important this data is available as fast as possible without compromising the quality of the images.

A user of a system like RS would also require low latency when interacting with the server, the user should not be have to wait for content for a time frame which is perceived as long. The user also requires low latency when interacting with the data. This involves both perceived lag when sliding back and forth on a seismic shot gather, and latency when dealing with content manipulation and subsequent feedback.

Harry Shum [6] said that "250 milliseconds (ms), either slower or faster, is close to the magic number now for competitive advantage on the web". This means that if a user needs to wait for more than 250 ms he is likely to leave and use a competitor’s site.

Time Required For	Millisecond
One beat of a hummingbird's wings	20 ms
A single frame of a projected film	42 ms
A website loading delay to discourage visitor	250 ms
The blink of an eye	400 ms
A baseball pitched at 99 mph to reach the plate	417 ms

[a] Numbers gracefully borrowed from [6]

Table 6.1: Consumer expectation of delivery speed

If the response time is greater than 10 seconds it will break the thinking continuity of the user, but a response delay of around 2 seconds will still allow for continued focus [7].

There are no other web based solutions available for UIT that can provide similar services to RS. But if the system is perceived as slow the user might choose to discontinue the use of RS, and revert back to the other visualization tools available, thermal paper plots and source software's visualization. Although the goal is not to be as fast as stated in [6], it is something to strive for. And the numbers from [7] give added indication of the human tolerance for delay.

Without visually pleasing images, a system like RS would not be used to a greater extent. Perception of image quality is often highly subjective, but it is important that the users who use RS perceive the images as being of a good quality. Subjective image assessment is accepted to be the most effective and reliable [9]. Good quality in this case is determined to be quality which is comparable to the alternatives, thermal paper plots and existing software. A quality that is perceived as inferior to what is already available is likely to discourage the continued use of a system.

In the book Color imaging: vision and technology [3] Joyce E. Farrell stated that: "Since our customers are the final arbiter of image quality, we consider their subjective image quality judgments to be key to the success of our imaging product". Besides subjective evaluation there are a number of other methods of evaluating images which are discussed in [9], [13] and [4], but in the context of this thesis it does not make much sense to attempt to evaluate based on such techniques.

6.1 Evaluation Environment

The system is evaluated using a server machine bought in by the Department of Geology at UIT, for the sole purpose of running RS on a seismic vessel. It is a blade suitable to be inserted into one of the racks available in the operational room of a seismic vessel. The machine consists of the following hardware and software:

- Intel Core i7 3.1 GHz
- 16 GB ram
- 7200 RPM Hard disk
- Windows 7 64bit

Unless otherwise pointed out, all numbers provided in this evaluation is created by using 30 measurements.

6.2 Data Processing Runtime

In this section, the data creator will be evaluated based on the latency when data is available to RS, to when visualization content is available to the end user in form of images.

As discussed earlier, data available to the system can be available in a multitude of sizes. These sizes are user-determined. The seismic data can either be split into smaller chucks by the source software, to allow for a more continuous data creation by RS, or be made available for RS when a seismic run has completed. It is likely that such a run will take many hours, and will then present RS with a file of potentially many Gigabytes. In the last scenario there will be no new data for RS to process until the run is over, with a consequence of no available "live" visualization of the current seismic cruise.

For a system like RS to work as intended during "live" data acquisition, it would need a steady input of data. A file of 14 megabytes (MB) is the data created from minutes or 10's of minutes of seismic acquisition. Megabyte per minute is based on a two parameters. The first being depth, and the second being window stored. In shallow waters, there will not be much data stored per trace, and it will take longer to collect 14 MB, but in deeper water each trace would contain more data, and thus the time taken for 14 MB to be collected would be shorter. To prevent unimportant data to be stored, only a

Megabytes[b]	Mean[c]	Std[d]
4	2.13	0.162
14	4.04	0.389
46	14.24	0.710
86	24.37	0.856
123	33.98	0.872
190	51.03	1.261
297	80.85	2.507
429	121.82	2.758
701	284.69	72.365
953	328.52	8.987
1829	717.59	32.836

[a] Time from seismic data to be available until all images is available to user
[b] Input files size in MB
[c] Average in seconds
[d] Average standard deviation in seconds

Table 6.2: Edgetech runtime for creating all tiles from input file

window of the complete trace is stored in the final binary format.

The system will be evaluated based on input size, the lowest being 4 MB, the biggest being 1829 MB. This file was the biggest available for evaluation, but file sizes such as this would be unlikely during a seismic cruise. A more likely size would be 14 MB, which was used during on-site evaluation of RS discussed later in section 6.6.

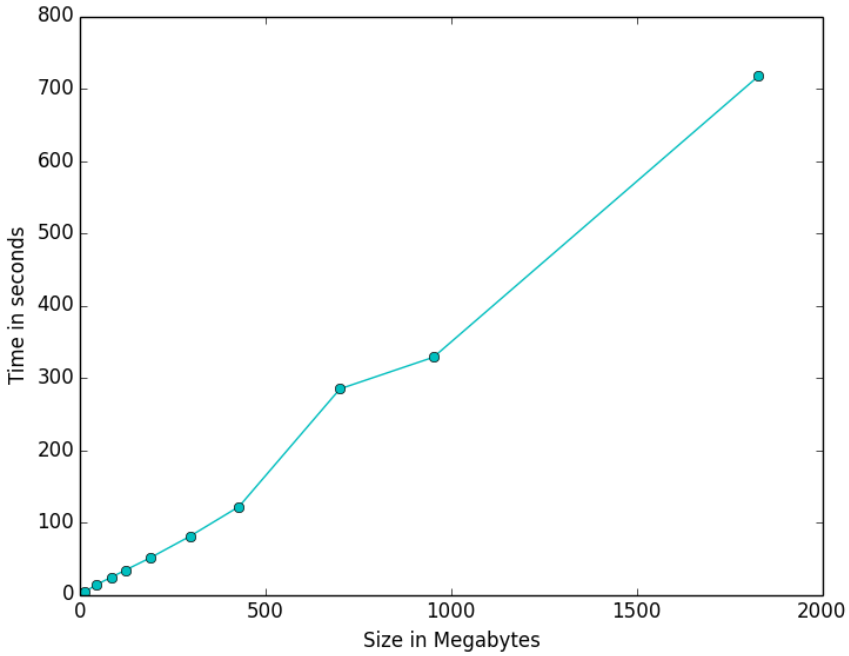


Figure 6.1: Average runtime backend, create all tiles from a File

Megabytes[b]	mb/s[c]
4	1.878
14	3.467
46	3.230
86	3.529
123	3.620
190	3.723
297	3.673
429	3.522
701	2.462
953	2.901
1829	2.549

[a] Megabytes divided by time
[b] Input size in MB
[c] Megabytes per second

Table 6.3: Edgetech runtime speed

6.2.1 All Images Ready

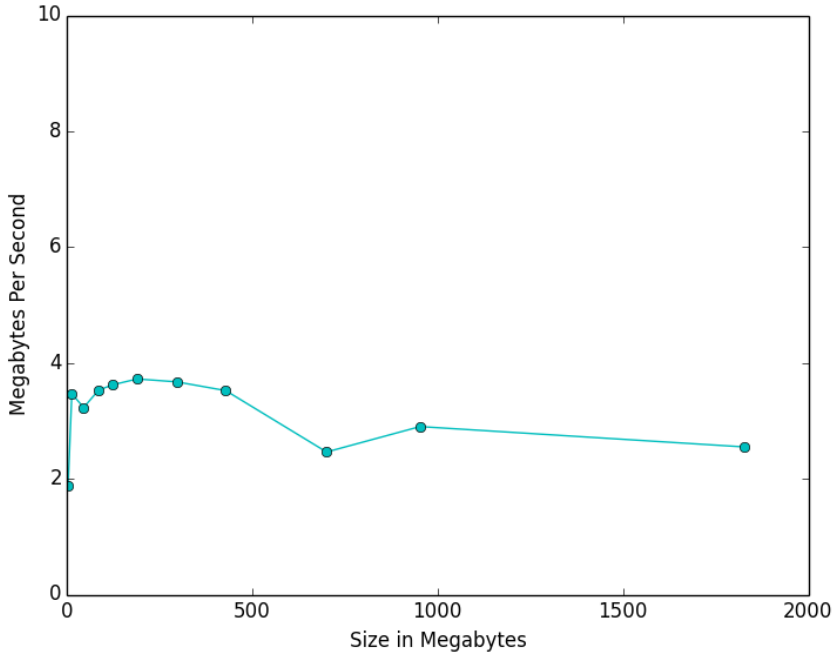


Figure 6.2: Megabytes divided by time, mb/s

Figure 6.2 and table 6.1 shows the runtime results, when taking the time for RS to create all image tiles from a file. Figure 6.2 and table 6.3 provides a different view of the results by dividing file size by time measured. From the numbers it can be see that RS scales up to relatively well up to file sizes of 1829 MB, but the scalability is not linear as might be perceived in the graph in figure 6.1. The numbers provided in figure 6.3 show that files in the ranges above 429 MB provide a gradually decreasing execution speed. A potential reason for such decreasing execution speed can seen by looking at the measurements done for total memory usage. Measurements of CPU load is not taken into account, as it was observed that the CPU load at the measurement time was very constant, around 10-16 percent total system utilization, and the size of the input files had no bearing on this load.

Megabytes	Total Memory Consumption GB	Standard deviation GB
4	0.48	0.586
14	2.03	0.009
46	4.08	0.004
86	7.57	0.008
123	10.92	0.009
190	17.25	0.279
297	28.01	0.543
429	38.56	0.015
701	87.23	20.102
953	131.74	27.4556
1829	340.40	92.2569

[a] Memory consumption from acquisition to all tiles are available

Table 6.4: Total memory utilization image creator

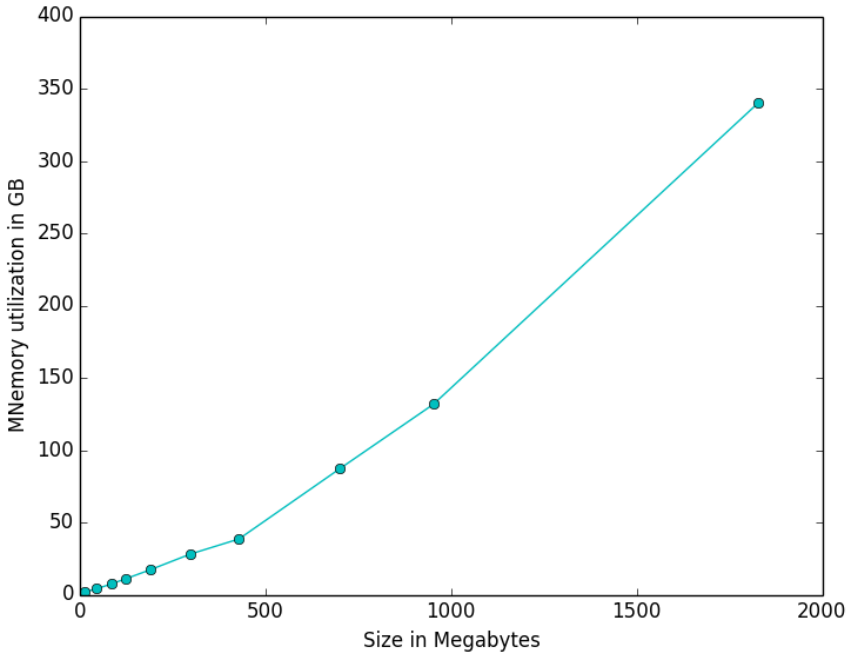


Figure 6.3: Backend memory usage in GB for all images created

By looking at the usage in memory in figure 6.4 it can be seen that the Go implementation uses a lot of memory to create the finished visualization. A

Megabytes[b]	Mean[c]	Std[d]
4	NA	NA
14	1.943	0.121
46	11.404	0.187
86	18.316	0.856
123	24.910	0.595
190	33.982	0.561
297	37.674	0.875
429	69.431	1.935
953	142.664	4.225
1829	271.648	10.719

[a] Runtime from acquisition to first tile is ready for request
[b] Input size in MB
[c] Average in seconds
[d] Average standard deviation in seconds

Table 6.5: Edgetech runtime image creator one tile

big contributor to this is Go built-in append[1] function. This function is used to append two slices to each other. Instead of allocating the exact memory needed, it will increase the memory held by the slice by x2 if size of slice is less than 1024 elements, and by x1.25 if over 1024 elements. This will account for some of the high memory usage.

By looking at the memory numbers in table 6.4 and the graph in figure 6.3 it follows the same pattern as when measuring processing time in figure 6.2 and table 6.1. The increase up until files with size up to 500 MB is linear, but for input files over this size, the increase is growing at a gradually higher pace. This gives reason to believe that there is at least some relationship between high memory usage and the total processing time.

From the standard deviation column of table 6.2 it can be seen that calculation speed for the image creator is very stable, except for the files with size 701 and 1829 MB. These files had a handful of consecutive runs which largely deviated from the standard. What causes this deviation remains unclear. This is of course in an optimal environment with no requests on the Front End, with also runs on the same machine. Given a steady stream of requests, it is likely that these times would increase slightly as the tile creator would have to compete for CPU time and disk Input/Output (IO) with the web server.

1. http://golang.org/pkg/builtin/

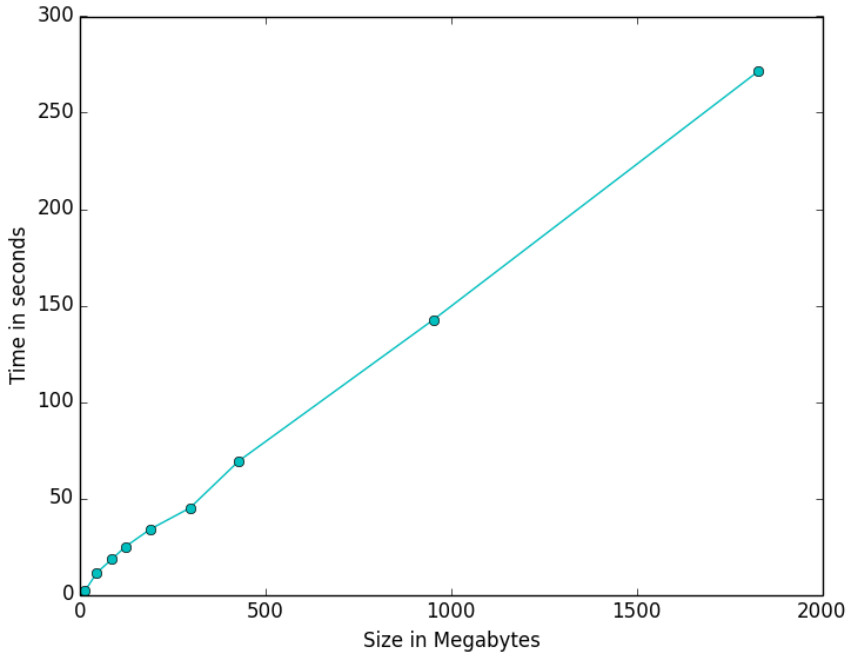


Figure 6.4: Runtime on Backend from data acquisition to first tile/image is ready for request

6.2.2 One Image Ready

The end user does not have to wait for all tiles to be created to start interacting with the seismic data. He can request images as soon as they are ready in the backend. With that in mind it is also prudent to measure the time it takes to process the seismic data, and create only one image, the first tile for zoom level 6. By looking at the table 6.5 and figure 6.4 it can be seen that this part follows a more linear pattern then creating all images. This is also true if we study the memory usage in table 6.6 and the graph in figure 6.5. Here it again follows the pattern of time taken, in this case linear.

Megabytes	Total Memory Consumption GB	Standard deviation MB
4	0.479	0.586
14	0.95	1.874
46	3.05	0.003
86	5.71	11.482
123	8.08	0.005
190	12.56	0.007
297	19.81	35.937
429	27.88	0.005
953	64.44	80.841
1829	124.00	84.937

[a] Memory consumption from acquisition to first tile/image ready for request

Table 6.6: Total memory utilization for image creator, one Tile/Image

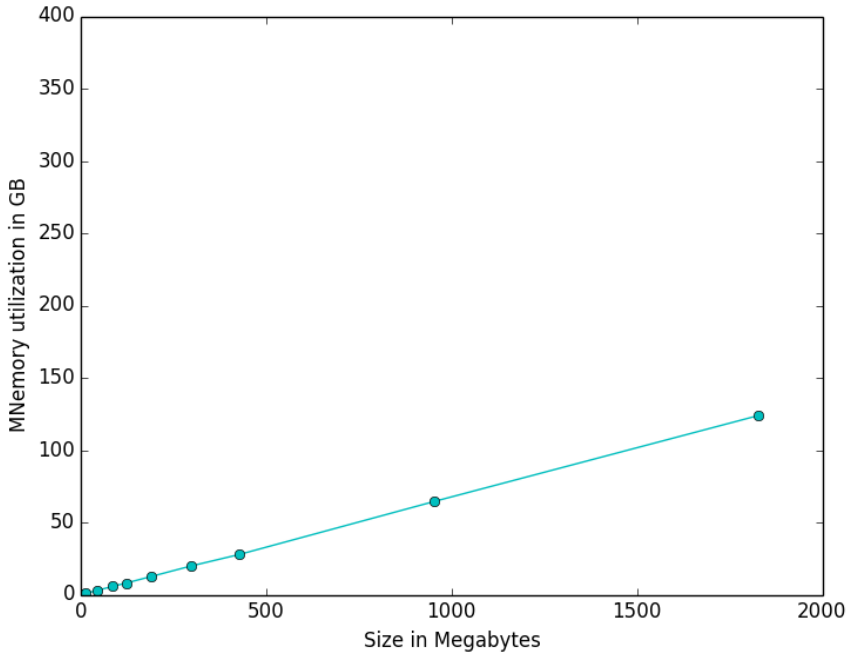


Figure 6.5: Total memory usage in GB for 1 image created

By looking at the numbers and graphs in this section and section 6.2.1 it can therefore be concluded that something during the creation of images are causing a high memory usage. And this high memory usage might contribute to the extra time it takes to complete building these images.

The windows version of Go, v1.22, has issues with the Garbage Collector (GC). Previously used memory is not release back to the operating system without forcing the GC to run. Without this "hack" the memory will reach the max allotted memory fast, and start paging to disk. But even with forcing the GC to run at certain sections of the code, all unused memory is still not released. Go version 1.3 promises to have a 100 percent precise GC, and hopefully this will fix some of the issues related to memory. Although this does not affect the total memory usage, it might cause less memory to be occupied by the RS backend at any given time.

6.3 Round-trip Latency

This section evaluates the round-trip time for loading RS's most important feature, the seismic data. Measuring the time it takes for all tiles to be loaded is important for showing RS as an application which meets the standards which one might expect for a web application. To take into account the expected use pattern on the vessel RS will be employed, this evaluation was done on PC's with cable network and mobile devices connected to a wireless network.

The round-trip time is measured from the user clicks a link for a random map, until all JavaScript on the client has loaded. In between these two points in time are among other: the request to server, server preparing data to be sent, a database call, sending of data, and the important processing of the data on the client side. The client side processing also includes a setup for the websocket and a call on this websocket to fetch relevant markers.

It must be noted that the measurements were done at optimal test conditions, with only one user and no other user initiated programs running on the server or on the clients. The server was not processing seismic data at the time. It would be expected that such events would have increased the times measured slightly. The measurements will also be higher the first time a device loads in static content like CSS and JavaScript. Content such as this is cached on the client to avoid unnecessary file transfers. These files are neither bundle together or minimized, so the client requests much more data then he would had the files been be optimally bundled and minimized.

This evaluation was conducted with 2 personal computers, the first being the machine that runs RS, and the second a 4 year old laptop. The server machine was included in the evaluation as is likely that it will also be used as a client to visualize seismic data from RS. The 2 mobile devices consisted of a 3 year old Samsung Galaxy S2, and a new LG Nexus 5. All measurements where done

Device	Round-trip[a]	Client loading time[b]
i7 3.1[cd]	139	49
i5 2.26[d]	293	124
Samsung S2[e]	1039	423
Nexus 5[e]	678	219

[a] round-trip in milliseconds
[b] Client processing and loading time in milliseconds
[c] Running at the same machine as server
[d] Cable network
[e] Wireless network, Eduroam

Table 6.7: Round-trip latency for a seismic map request

running the newest version of Google Chrome[2], version 34.0.1847.131. The evaluation was done with 15 measurements for each device. It is a given that testing wireless against Cable could result in different numbers for the two PC's, but this was found irrelevant for this evaluation. This is due to the use patter expected on the seismic vessel, where PC's run on cable and mobile devices run on the wireless network.

As was expected the blade with the best hardware, and cable network was the fastest to display the seismic data, the old i5 computer still did good, and the total time discrepancy from the i7 is mostly due to the added client loading times. Both devices had an acceptable round-trip time, under and or far away from the magic 250ms mentioned in table 6.1. The mobile devices where expected to be slower, but even at a little over a second total time the aging S2 was not perceived as slow. The Nexus did a little better, especially on the client side due to the improved hardware available.

This evaluation proves that the RS client can perform well, even on smart phones. In addition to this evaluation, a leading expert in the field of seismology did not perceive the latency as high when evaluating the system on the S2.

Interaction with the server through the websocket is very responsive, it is perceived as instantaneous, and a small evaluation case with heavy load provided content within milliseconds.

2. http://www.google.no/intl/no/chrome/browser/

6.4 Image Quality

In this section, images originating from RS are compared with images from both the Edgetech software and a digitalized Delph image from a plot on thermal paper. The images from RS consist of actual seismic data collected by UIT. The images are subjected to different filters with increasing cut-off frequencies.

Providing images with high quality is important to the user. Without images with a quality equal or better than expected, the user might choose to revert back to thermal paper or the software's which comes with the seismic equipment. With that in mind, an emphasis was put in to create images of similar quality to what was available. As these software's provide a number of methods to: filter away noise, resample data, change gain levels and sharpening images; it was important that RS also supported similar features.

Images in RS are filtered for noise using techniques like, low and high pass filters. These filters discard high frequency and low frequency signals respectively. In the case of images, high values are white and low values are black.

In figure 6.6, the effect of adding a high pass filter to unfiltered seismic image can be seen to affects the quality of the visualization. By studying the visualization of the unfiltered data in this figure it is evident that data contains a high degree of noise. By adding a high pass filter and subjecting the unfiltered data to various degree cut-off values, it can be seen that the images are losing more and more noise, by tuning up the cut-off frequency up. It is also evident that applying a too high cut-off value removes potential signal data.

Figure 6.7 shows more filters being applied to the same image. In the second image from the left, the median filter is applied, and the lines of the image become a little clearer. On the third image some of the noise has been filter on the high value scale by applying a low pass filter. In addition a high pass filter with a high value has been added. This should become a very white image, but when making all present colours which currently isn't filtered away, black, the image show a lot of details. On the right hand side of figure 6.7 an image from the Edgetech software is displayed. Notice that this is an image taken from the exact seismic shot gather as the other 7 in these two figures. There are some differences in scale, but it shows what RS is up against when it comes to image quality. The Edgetech image looks very nice, and has been subjected to filters and image manipulation which differs from RS images.

There are unfortunately not any example for thermal plotter plots using the same dataset, but to illustrate the quality of these thermal plotter visualization,

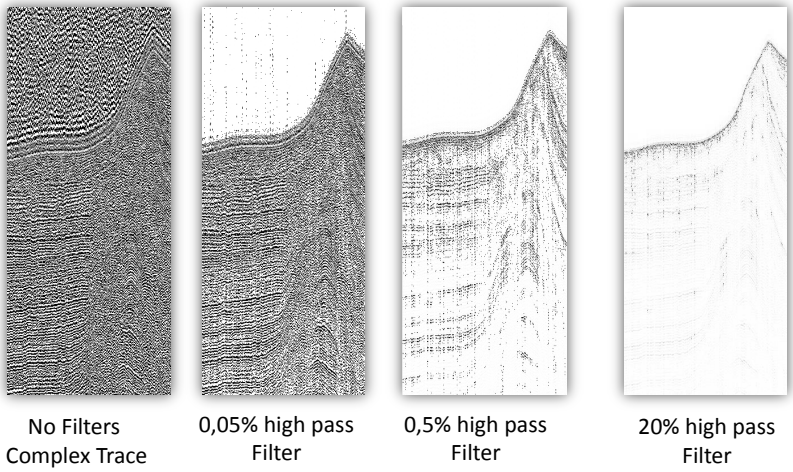


Figure 6.6: Compares different cutoff frequencies for high pass filter

an image of an unrelated plot from an unknown location can be seen in figure 6.8. From this figure is can be seen that the thermal plotter images from Delph can show seismic images with good quality, with a high signal to noise ratio. As this is the visualization that the scientists are accustomed to, it should be taken into consideration when evaluating RS visualization quality.

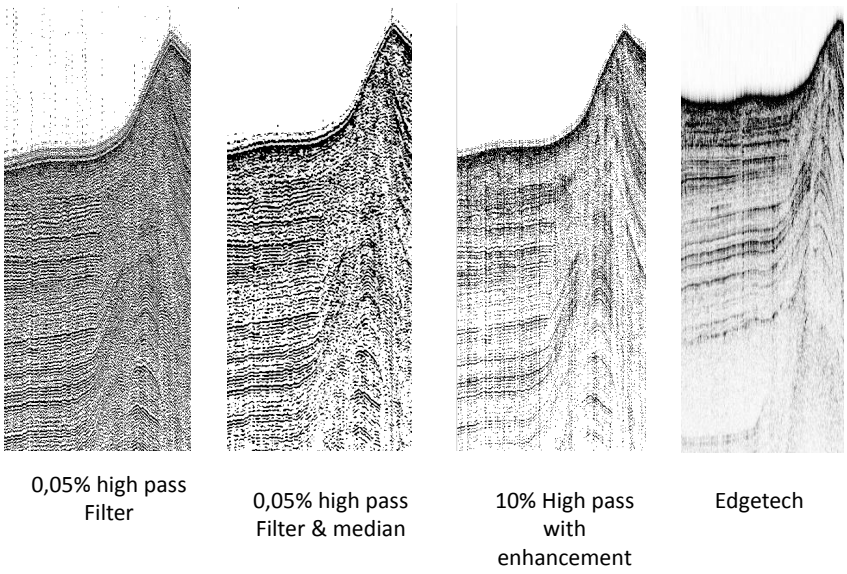


Figure 6.7: Compares for aditional filters and Edgetech image

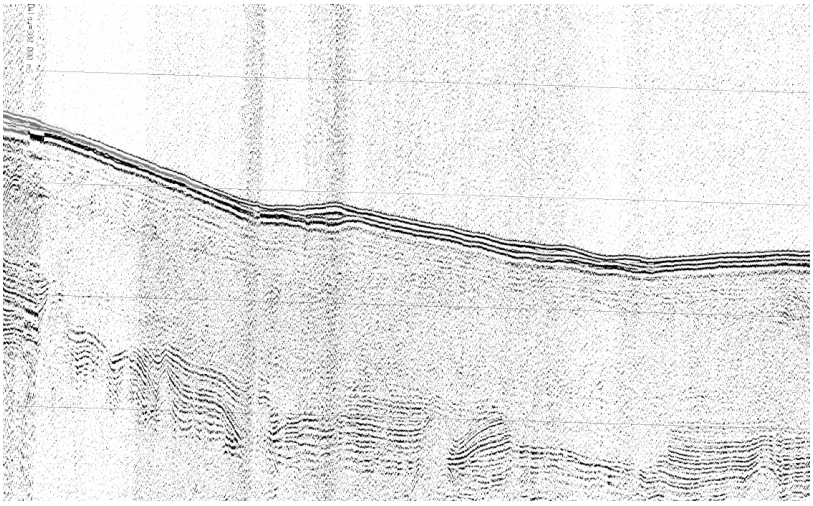


Figure 6.8: A scanned copy of a Delph thermal plot

In his PHD thesis [2] from 2010, Thomas Elboth said: "A decade ago, only applying band pass filter in certain areas to attenuate noise might have satisfied us". There is a lot of work left in this section, new filters should be developed to create both sharper and smoother images, filter out noise and give options for alternative views. More about this future work will be discussed in Chapter 9. It is evident when looking at the images provided in figures 6.6 and 6.7 that there is a lot of work still to be done in terms of creating an optimal visualization environment to use for scientific and educational purposes, but RS has taken the first step of Creating such a tool.

6.5 Evaluation by Expert

To decide whether the images from RS can hold up to its competition, a leading expert in the field of seismic acquisition was asked to evaluate the image quality. The leading expert found the image quality very pleasing. The images produced by RS were similar to what the Delph software could provide. An example of an image from Delph in form of a paper plot can be viewed in figure 6.8. Even though the images have similar qualities to Delph images, the RS images was still lacking in quality when compared with both Delph and Edgetech images. Besides the filtering done in RS, these two programs provide additional or better filters to improve the image quality. The expert also suggested some filters that could be implemented in a future version of RS.

The expert was very pleased with the functionalities of RS and the ability to look at the seismic data on his personal smart phone. The application performed to expectation when it came to user-friendliness and interaction smoothness. Even though he was happy with the presented product, he missed the functionality to re-filter sections of the seismic data. In addition he has numerous suggestions when it came to added or improved visualizations and functionalities. Some of the suggestions for improvement will be discussed further in chapter 9.

6.6 On-site Evaluation

RS was evaluated on the FF Helmer Hansen in April 2014, on a short seismic cruise. Seismic data was collected from departure from Tromsø, to Mefjorden (east of the island Senja), and back again. This data was collected using the Edgetech software. The Edgetech software was configured to produce 14 megabyte output files. This provided RS with new seismic data at a pace

varying between minutes and 10's of minutes, depending on the sea depth. The files where processed and images where ready in around 4 seconds after a file was presented to RS. During this evaluation RS successfully processed all input files and created visualizations to the scientists and students aboard. This evaluation shows that RS can perform as a visualization tool on a seismic vessel, producing interactive and available data to any interested user on board the seismic vessel.

At the time of evaluation RS did not have the image quality it has in its current build, and due to suggestions given by scientists aboard, the image quality on RS was later improved.

7

Related Work

Providing content as tiles via a web application or another application is not a novel way to represent images, there are numerous application that does exactly this. For web applications images can be presented via a number of both licenced and open sourced JavaScript libraries. Of the most commonly known are Google's Google Map[1], Openlayers[2], Leaflet[3] and Mapbox[4]. The common use case for such libraries is to display actual map data, be it a World Map, or just a map of the local area.

Using these libraries for images other than maps is not unheard of, albeit a little less common. An example is using google maps as an art viewer[5]. Another example is Zygote[6] which displays a 3d models of the human body. A third example is the Genome Projector[7], which displays genes as tiles. Leaflet itself was even used to create a gigantic xkcd[8] cartoon[9]. This proves that libraries such as Leaflet can be used for numerous things, also to visualize seismic data.

1. https://developers.google.com/maps
2. http://openlayers.org/
3. http://leafletjs.com/
4. https://www.mapbox.com/
5. http://www.google.com/culturalinstitute/project/art-project
6. http://www.zygotebody.com/
7. http://www.g-language.org/g3/
8. http://xkcd.com/
9. http://xkcd-map.rent-a-geek.de/

Finding work that is truly related to RS is challenging, as there are no known web applications for seismic data which is in the same mould as RS. With that in mind the most related work from the field of seismology, is the software that come with the seismic hardware, Edgetech, Elics Delph and similar systems.

The first to study is the Edgetech software which is one of the software's available for UIT. It is software used for acquisition, processing and visualizing seismic data. It is powerful software providing the user with numerous settings to filter seismic data for improved visualization. Filter included are among other, the Gain filter, Time Variant Gain (TVG) and swell filter. As this is commercially licenced software there is no description of the architecture, design or implementation. It created to run on Microsoft Windows PC as a desktop application.

The second program UIT has available is Elics Delph. It is also used for acquiring, processing and visualizing seismic data. Information available about this software is very limited. It is also a proprietary system, but filters it uses can be seen from the output header file. It uses Swell filters, high pass filter, low pass filter, gain filter and a TVG filter. As with the Edgetech software It is also a Windows based desktop application.

The project OPENTECH from dGB[10] is another seismic viewer which provides visualization in 2D and 3D. It provides a desktop interface which allows for viewing and interacting with the seismic data in a way that can be compared to RS. This is a program written in C++, and is available as an open source product.

10. http://www.dgbes.com/

8

Concluding Remarks

This thesis presented RS, a system for processing, visualization and interacting with seismic data from potentially a multitude of sources. RS was created in response to a demand for better seismic visualization and interaction by the Department of Geology at UIT. The scientists have for a long time been working with inadequate visualization tools in form of old thermal plotters, and software which lacks the required features for proper interaction with the seismic data.

This thesis has contributed by:

1. Providing scientists and students on a seismic vessel a new tool for processing, visalizing and interacting with seismic data.

2. Providing an application available on heterogeneous platforms, by providing seismic visualization through a web application.

3. Demonstrating the capabilities of RS in an evaluation. Showing RS as a promising seismic processing and visualization tool.

The system presented provides components which processes and filters seismic data, before making it available to clients via a web interface. The web interface is implemented to support mobile devices. It provides seismic data as image tiles, for zoomable content, and provides markers and annotations as an overlay to the tiles created.

RS was evaluated during an actual seismic acquisition cruise, and was further evaluated experimentally to asses RS's contribution towards being a new and improved visualization tool for seismic data. During evaluation on the research vessel FF Helmer Hansen and further evaluation of RS, RS proved to process minute's worth of seismic data in just over four seconds. This allows the seismic data to be used for seismic studies quickly. The RS client was evaluated, and it was shown that under optimal conditions RS could provide seismic data to a client in less than 1.1 seconds.

The evaluation also showed that RS can perform well in terms of visualization quality when compared to existing visualization tools. Further evaluation was done by a leading expert in the field of seismic acquisition. He was pleased with the visualizations RS provided, and is eager to continue the development of RS. He was interested in adding features to the user interface, and improving the filtering and collection of data. He also concluded that the client was very responsive, even on a three year old smartphone.

9

Future Work

9.1 RS Format

RS does not currently store data into a RS format, and therefore cannot re-filter data, without using the binary files from source software. This should be the first priority for future work. While the software from source binary files is being parsed, both the seismic data and metadata should be stored in a new format, RS format. The code needs to be slightly rewritten to support parsing of RS format, or use RS format out of the box in the tiler. For the last option the tiler needs to be slightly rewritten to support the uniformed data now containing metadata.

9.2 Delph Parsing

Delivered version of RS does support the Delph format, and provides a parser that transforms the Delph format into a uniform format and images. The image quality on the other hand is severely lacking. The image data in the binary data are stored in a compressed format. Each pixels data is located in a nibble. There is a need for a major rethinking of the Delph format parsing. Unfortunately there are little or no documentation on this format. In addition, there are metadata located within these files. This metadata setup is not specified beyond where it is located, and thus it is difficult to use this metadata in RS.

9.3 Web Client

9.3.1 Image Overlays

Besides displaying the image tiles, the images should have an overlay indicating depth and distance; this should be achievable with the use of leaflet overlays. It is envisioned that images of horizontal and vertical lines can be placed on the left and top section of the screen respectively. As a separate overlay or connected to this overlay there should be a number indicating relevant measurements. For this to work, placement of the lines and number have to be dynamically calculated as the user shifts his view or zooms.

9.3.2 Improved Websocket Updates

The envisioned role of the websocket solution in RS was to provide the user both quick communication, and support the user with dynamic content. This has partly been achieved, but in certain areas the user needs to manually update the web client to receive updates by different users. This could be fixed by directly adding data into the HTML via JavaScript. This has been done successfully in parts of the client code.

9.4 Filtering

9.4.1 Improved Filtering

Currently RS used simple filtering techniques to filter noise from the seismic data. The filters used are classified as band pass filters. Along with these, the images are treated with a median filter to smooth out images. A lot of work remains in this section, as these techniques are lacking in sophistication as described in [2].

Looking at the images from Edgetech, they are treated with a smoothing algorithm far superior to RS's median algorithm. It can be speculated that these have been treated with a Gaussian filter that adds blur and a subsequent sharpening technique.

Filters to remove noise from swell and doubles etc. could be included to improve the visualization of RS imagery. It can be speculated that such improvements might increase the total processing time for creating images, but visualization quality is key to analysing the seismic data.

Techniques that exits that could be looked into are among other: automatic Gain filters [11], stack filters [10], swell filters [5] and rectification filters

[1].

The idea behind the gain filter is to work your way down the image data, a trace at a time. Doing a median on a window sized set of this trace, and for each pixel, recalculate the value.

A version of the Gain filter is a TVG filter, the idea is that the pixel in terms of its position in the trace matters. The lower part on the trace will have less signal strength due to less reflection as the seismic wave continues downwards. This filter would add more gain on the lower parts than it would on the top part of a map.

The swell filter would remove effects of waves by adjusting the positioning of a trace of pixel data. This adjustment is based on a heave value available on board the vessel. This means that if the boat is at the peak of a wave, the traces affected would be shifted downwards. By following this pattern, you would get smoother lines, and remove wave like features on an area on the seismic image that in real life is flat.

9.4.2 User Defined Filtering

When filtering noise from the seismic data, it is difficult if not impossible to find a filter or filter set that works best in all situations. It would therefore be desirable to be able to re-filter data after it has already been processed.

If the user finds an area that he or she finds interesting, or believes a different filter could create better images, the user could mark it in the web client. The user could then choose between numbers of different filters and filter settings to get images exposed to these filters. The user should then be able to decide that a certain filter suits best for this area and potentially overwrite original images.

Storing data in the RS format would solve this. This disk stored binary format should provide all the data needed to redo filtering of any section of a shot gather.

9.5 Data Acquisition

Current build of RS does not support automatic collecting of seismic data. The data needs to be manually copied from the binary data creator to RS, via a folder on the private network. There are three ideas on how such an

automatic collection could work.

In the first, RS monitors a pre-defined shared folder on the source machine, holding track of which files have been read, attempting to open a new file with write permission at a given interval. If successful, the file is ready to be transferred, if not, RS will delay for a set time, and try again later. The negative about this approach is that it breaks the current use pattern. The scientists are costumed to choose which folder to store files, and thus prevent a further file management.

The second option is the same as the first, but for the fact that the target folder can be customized via RS user interface. The disadvantage here is the need to manage two programs when choosing a folder to store the seismic data, RS and the source software. There would be no consequence in forgetting to configure RS, other than RS being unable to do its work. This folder could be set in RS during a seismic run, and would allow RS to catch up quickly.

The third option is to write a separate monitor and push program on the source machines. The idea is that this program would monitor a folder, and push data to RS when it becomes available. This solution would also require a second configuration by the end user.

9.6 Use Analog Signal

RS is now dependant on the data from different seismic binary formats. An option would be to use the analog signal that originates from the sensors. RS would need access to these analog signals, and have access to an Analog-to-digital converter to convert signals from analog to digital. This task could be well suited for the Physics department of UIT.

9.7 Append Module

Ideally RS would need small binary files from source software's to provide a delayed live image viewer. This causes a conflict with the later use of these binary files. The use pattern after a seismic cruise is completed involves the use of a bigger file in the further research. Many smaller files would complicate this work. Although RS solves this by appending all input files into a big file, the scientists would like to have the option to append only a selection of the files. This could be done as a separate script, where the scientist specifies the files he or she wants, and a subsequent append is done only on these

files.

9.8 Export to jpg/png

A common use case of seismic data is exporting sections of this data to an image format. To accommodate this RS needs an option to export a section of the full visualization. RS could provide such a feature by creating images straight from the planned RS format data. Here RS would need to encode a big image out of an area specified by the scientist. As this does not involve tiling, a custom image creator would have to be written to support this feature.

9.9 Improved handling of Trace Depth Change

As discussed in section 5.2.2, a user configured change in trace length can happen at any moment inside a binary file. This is handled, but not good enough. This could have been handled by stopping a parsing of a file parsing when a trace height change is detected. The data already parsed into the correct format could be pushed to the cache. The cache would have to be emptied by the creation of tiles, from all available data. From this point, the data parsing could continue until a new potential shift occurs. This would cause a separation of the images with vertical edges, but would not require padding of data and re-parsing data as is the current practice. This would also give a similar visualization experience as thermal paper plots give, where a separation line would be visible between height alterations.

References

[1] AH Balch. Color sonagrams: a new dimension in seismic data interpretation. *Geophysics*, 36(6):1074–1098, 1971. 65

[2] Thomas Elboth. Noise in marine seismic data. 2010. 56, 64

[3] Joyce E Farrell. Image quality evaluation. *Colour imaging: vision and technology*, pages 285–313, 1999. 42

[4] Alain Hore and Djemel Ziou. Image quality metrics: Psnr vs. ssim. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 2366–2369. IEEE, 2010. 42

[5] M Jenkerson, R Houck, M Walsh, L Combee, AD Curtis, JE Martin, N Moldoveanu, A Özbek, CM Sayers, RC Walker, et al. Signal preserving swell noise attenuation using point receiver seismic data. In *2000 SEG Annual Meeting*. Society of Exploration Geophysicists, 2000. 64

[6] Steve Lohr. For impatient web users, an eye blink is just too long to wait. *New York Times*, 2012. 41, 42

[7] Robert B Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 267–277. ACM, 1968. 42

[8] Harris Onishi. Anisoropy in head waves in crosswell data. 3

[9] Thrasyvoulos N Pappas, Robert J Safranek, and Junqing Chen. Perceptual criteria for image quality evaluation. *Handbook of image and video processing*, pages 669–684, 2000. 42

[10] David G Schieck and Robert R Stewart. Pre-stack fk median filtering. 64

[11] Thomas H Shipley, Mark H Houston, Richard T Buffler, F Jeanne Shaub,

Kenneth J McMillen, John W Ladd, and J Lamar Worzel. Seismic evidence for widespread possible gas hydrate horizons on continental slopes and rises. *AAPG bulletin*, 63(12):2204–2213, 1979. 64

[12] M Turhan Taner, Fulton Koehler, and RE Sheriff. Complex seismic trace analysis. *Geophysics*, 44(6):1041–1063, 1979. 29

[13] Zhou Wang and Alan C Bovik. A universal image quality index. *Signal Processing Letters, IEEE*, 9(3):81–84, 2002. 42