Report by Gerard Nothnagel

The Visualisation of Radio Frequency Interference

Supervised by Michelle Kuttel
Co-supervised by Sarah Blyth and Anja Schroder

<table>
<thead>
<tr>
<th>Category</th>
<th>Min</th>
<th>Max</th>
<th>Chosen</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement Analysis and Design</td>
<td>0</td>
<td>20</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Theoretical Analysis</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Experiment Design and Execution</td>
<td>0</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>System Development and Implementation</td>
<td>0</td>
<td>15</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Results, Findings and Conclusion</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Aim Formulation and Background Work</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>Quality of Report Writing and Presentation</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>Adherence to Project Proposal and Quality of Deliverables</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall General Project Evaluation</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Total marks / Overall Comments 80 / 80

Department of Computer Science
University of Cape Town
2014
Abstract

The increased utilisation of radio frequency bands by devices, coupled with the growth of technological resources, is exacerbating radio frequency interference (RFI) in radio astronomy observations. RFI is an ongoing issue at the MeerKAT site (located in the Karoo, South Africa) and the radio frequency (RF) environment has to be constantly monitored to prevent the loss of data due to RFI. For this purpose, large amounts of data are currently collected in hourly segments. An initial visualisation prototype has been developed to help make sense of the data, but there are opportunities for improvement. In this project, an alternative RFI visualisation is proposed which aims to assist the team responsible for managing RFI at the MeerKAT site. We first describe potential visualisation and interaction techniques, and thereafter present a visualisation framework suitable for a browser-based analysis tool. We followed an iterative design process to identify effective and useful methods for visualising RFI. A responsive and interactive visualisation was developed, which visualises an hour of data using web technologies. User testing revealed that the visualisation is effective at showing affected frequency channels, and that the interaction techniques were useful for analysing data sets.

Acknowledgements

Michelle Kuttel, for frequently assisting with our writing and time management.
Sarah Blyth, for recruiting participants for the user study.
Christopher Schollar, for providing test data and visualisation ideas.
Contents

1 Introduction ..................................................................................................................5
2 Background ..................................................................................................................6
  2.1 The MeerKAT Monitoring System ........................................................................6
  2.2 Scientific Visualisation .........................................................................................9
  2.3 Conclusions ...........................................................................................................20
3 Design .......................................................................................................................22
  3.1 Design Goals .......................................................................................................22
  3.2 System Architecture .........................................................................................23
  3.3 Filter Design ......................................................................................................24
  3.4 Visualisation Design ............................................................................................27
4 System Implementation .............................................................................................30
  4.1 Implementation Languages and Libraries .........................................................30
  4.2 Filter Implementation ..........................................................................................30
  4.3 Data Retrieval and Parsing ................................................................................32
5 Visualisation Design and Evaluation: Iteration 1 .......................................................34
  5.1 Overview .............................................................................................................34
  5.2 Visualisation Design .........................................................................................34
  5.3 Implementation Iteration 1 ..............................................................................38
  5.4 Evaluation ..........................................................................................................43
6 Visualisation Design and Evaluation: Iteration 2 .......................................................44
  6.1 Overview .............................................................................................................44
  6.2 Visualisation Design .........................................................................................44
  6.3 Implementation Iteration 2 ..............................................................................47
  6.4 Evaluation ..........................................................................................................56
7 Results: Final Design ................................................................................................59
  7.1 Overview .............................................................................................................59
  7.2 Visualisation Walkthrough ...............................................................................60
  7.3 Extended Design ...............................................................................................61
  7.4 Discussion ..........................................................................................................62
8 Conclusions and Future Work .....................................................................................63
9 Appendix ....................................................................................................................64
  9.1 Code Snippets .....................................................................................................64
  9.2 Benchmarks ........................................................................................................70
  9.3 Visualisation Walkthrough ................................................................................72
1 Introduction

The continued growth of technological resources worldwide is increasing the utilisation of radio frequency bands, which could interfere with radio astronomy observations [1]. Interference management aims to reduce the impact of radio frequency interference (RFI) and prevent the loss of data contaminated with RFI [2]. This is especially the case for the South African MeerKAT radio telescope, a precursor to the Square Kilometre Array (SKA) telescope, which will be among the most sensitive radio telescopes in the world [3].

At the MeerKAT site, large amounts of data are received at an interval of once per second [4]. The data consists of a series of power values (the strength of a signal) observed over a fixed frequency range and the power values are used to detect RFI. In addition to the raw power data, they keep logs of over-range events, which are times at which the spectrum analyser received a signal that could not be quantised. Christopher Schollar has prototyped a system running both detection algorithms and a simple visualisation of the data at MeerKAT. Signals received by the antenna are processed in different stages, which includes flagging data classified as RFI and visualising activity in the radio frequency (RF) environment.

Visualisation is an effective tool for reasoning about data: it adds an additional layer of abstraction to data, reducing it to a level suitable for cognitive processing [5, 6, 7]. There is opportunity for improving both the detection and the visualisation of RFI in this prototype system. A new RFI visualisation must address three challenges. Firstly, because of the large volume of data, the data size should be reduced intelligently. A visualisation should also display a good overview of the data. Lastly, unreliable data should be distinguishable from trusted data. Our research was motivated by these challenges, and entailed finding effective and useful methods for visualising RFI.

This report has the following structure. In the Background chapter, we present a survey of visualisation techniques that are applicable to time series and large data sets, as well as interaction techniques that allow visualisations to go beyond static displays. In the Design chapter, a visualisation architecture is proposed, together with a design and evaluation strategy, which serves as a framework for the subsequent Development chapter. The final design is presented in the Results chapter, and future developments are considered in the Conclusions chapter.
2 Background

In this chapter, we first describe the MeerKAT monitoring system, and then present a survey of visualisation and interaction techniques that could be used for monitoring RFI. With large volumes of data being generated at a growing rate across a variety of disciplines, extracting useful information from data sets is becoming increasingly more difficult [8, 9]. Visualisation, which leverages the highly tuned ability of the human visual system to detect patterns, spot trends and identify outliers [9, p. 1], can help in this regard. Kosara [10] outlines the goals of scientific visualisation as enabling the exploration, analysis, and presentation of information such that the user gains a greater understanding of the data. We look at techniques for visualising time-series and large data sets, as well as interaction patterns that could improve the ability of visualisations to answer visual queries. Lastly, we describe evaluation strategies that could assist in determining the effectiveness of visualisations.

2.1 The MeerKAT Monitoring System

RF data at MeerKAT is collected by an on-site antenna over a frequency range of 50MHz-850MHz. Spectra are captured in one-second intervals, each containing 14200 64-bit power values. If an over-range occurred during an observation, the event is logged and stored in a separate file. Raw data has a lifetime of two weeks and is stored in hourly portions. That is, an hour’s worth of data consists of a 2D array, with dimensions 3600x14200. Only aggregates of the raw data are permanently stored, whereas RFI detected in the data is stored permanently. Currently, data at MeerKAT is being viewed from two perspectives:

- Activity in the RF environment during a single moment in time (the spectrum graph, Figure 2.2).
- Changes in the RF environment over time, with a condensed view of RF activity for each time instance (the bar and waterfall chart, Figure 2.2).

These perspectives broadly define the visualisation needs at the site – which are addressed by a line graph, bar chart, and waterfall chart. A waterfall chart uses a colour-shaded matrix display to represent a data set. Each entry in the matrix is mapped to a region or rectangular cell, and the value of that point determines the colour of the area. The benefit of this technique is its ability to display a large amount of information compactly. This does come with a penalty of reduced precision, as quantitative values are not accurately deduced from colour intensities or shade variations.

Other authors apply similar visualisations to RF data. Ellingson et al. [11] used time-frequency plots or waterfall charts (Figure 2.1) combined with line graphs to visualise RF data at Arecibo. Multiple series are shown within the same chart area. Joardar [12] implemented the same graphs for RF visualisation at the Giant Metrewave Radio Telescope (GMRT), but included a transpose to three dimensions (Figure 2.1). Three-dimensional views reduce the amount of visible data, as peaks might occlude regions behind them. By providing multiple views of the data set, users can switch between two and three dimensions depending on the type of visual query they have. In the following sections alternative visualisations are explored, which could improve the
visualisation of RFI for MeerKAT. The MeerKAT system visualises large data sets of time series data. We therefore review visualisation strategies for time series, as well as techniques for dealing with large data sets in the next two chapters.

Figure 2.1: Waterfall charts applied to data from Arecibo [11] (top) and GMRT [12] (middle and bottom). The top and middle chart shows frequency on the x-axis and time on the y-axis; shades or hue values are used to encode power values. A colour scale is added to indicate how points are mapped to hues or shades. The bottom chart is a transpose of the middle waterfall chart to three dimensions, and uses height as an additional encoding for the power values.
Figure 2.2: The current visualisation prototype used to monitor RFI at the MeerKAT site. Usage permission received by Christopher Schollar. The top chart shows a spectrum graph; the middle chart shows a bar chart, which displays times at which over-range events occurred; the bottom chart shows a waterfall chart. Each horizontal slice through the waterfall chart can be mapped to a spectrum chart, with power on the y-axis and a frequency range on the x-axis.
2.2 Scientific Visualisation

2.2.1 Visualisation Design

It is important to investigate the context in which a visualisation will be used, especially to understand the needs of the target audience. Developing a visualisation is a process entailing experimentation, user feedback, and multiple design iterations before suitable visual encodings can be identified. Thus, it is necessary to search through a large set of possible visualisations before discovering a satisfactory visualisation [13].

Several models for visualisation design have been proposed [14, 15]. The design process is divided into phases, and for each phase, a separate evaluation strategy is applicable. Munzner’s nested model has four levels: characterise the visual queries using domain vocabulary, abstract the domain tasks to general queries in information visualisation, design visual encodings and interaction techniques, and design algorithms to implement these techniques [15, p. 921]. Separating the design process eases the process of validation [15]. Many visual tasks are agnostic to the underlying data, but there are exceptions [15]. Thus, caution is taken when generalising visual queries.

Fry’s [14] model splits visualisation design into finer grained stages (Figure 2.3), but shares similarities with Munzner’s model. Both emphasize that the different stages are not strictly sequential: a design requires refinement, and consequently multiple iterations. Refinement is the process of further clarifying the representation of data, by changing the visual encodings or emphasising certain data features [14]. While designing, implementing and evaluating insights are generated. The feedback can be used to improve how the data is represented, and to improve the visualisation’s interaction techniques.

![Figure 2.3: Ben Fry’s visualisation design framework [14]. Feedback from one phase provides new insights, which in turn assist in improving other phases.](image-url)
2.2.2 Visualising Time Series

Time-oriented data is intrinsic to many domains, and many different visualisations have been proposed that are either domain specific, or applicable to a wide range of data (see Aigner et al. [16] for an extensive survey on time-oriented visualisation techniques). Line graphs were first introduced by William Playfair in the 18th century [17]. Since then, they have become a common technique for visualising temporal data [18].

2.2.2.1 Horizon Graphs

Line graphs are effective when displaying a single time series, but become cluttered when multiple series appear on the same graph [19]. The horizon graph is a possible solution to this problem [20]. In this method, a chart is split into what Tufte [21] calls “small multiples”, in order to display a collection of graphs on a small screen. An illustration of the process of transforming a line graph to a horizon graph is given in Figure 2.4 and Figure 2.5.

![Figure 2.4: A traditional line graph (left) is converted to an area chart (right). Both the area above and below the x-axis is divided into three bands, each with a different hue.](image1)

![Figure 2.5: Bands below the x-axis are flipped (left). Finally, bands are collapsed to the x-axis (right).](image2)

Few [22] suggests that the increase in data density in a horizon graph offers more benefits than drawbacks. The reduction of chart height increases the amount of data displayed on a screen, freeing up vertical space, while using small multiples reduces clutter. Discerning patterns, exceptions, and variations within the data using only line charts (left chart in Figure 2.6) is not well facilitated. Conversely, using compactness and varying hue and saturation (right chart in Figure 2.6) reduces the perceptual effort of making such observations.

Heer et al. [19] argue that there are transition points where the compactness of a chart creates a significant drop in estimation accuracy. They found that using four or more bands with different saturation made estimations tedious. Test subjects tasked with making estimations complained about the difficulty thereof and found it tiresome. Using two bands was optimal, placing the least strain on users. The use of three bands yielded comparable results.
2.2.2.2 Stream Graphs

Stacked graphs are an aggregation technique for visualising multiple time series (Figure 2.7, left). They are similar to stacked bar charts, but approximate continuity rather than depict values at discrete steps. Stream graphs, a type of stacked graph, were originally developed to visualise trends in music listening over time [23] (Figure 2.7, right). The intent behind the development was to increase the legibility of individual layers of stacked graphs, particularly when dealing with numerous layers. The three critical components of a stacked graph are the shape of the overall silhouette, the layer ordering, and layer colouring.

Through a series of algorithmic transformations, each operating on a critical component, layer legibility can be improved. The justification of the use of layer stacking is inspired by Tufte’s [21] macro-micro principle: information can be shown on multiple layers, both on a macro and micro scale. The first goal is to show an overview of the data by stacking series (giving their sum). The second goal is to ensure legibility of individual layers [23]. Unfortunately, the difficulty of comparing layers with different slopes can cause misinterpretations, and summing layers do not always lead to sensible interpretations (e.g. summing temperature).
2.2.2.3 Discussion

Javed et al. [24] conducted user studies to determine user performance on visualisation techniques for multiple time series. They found that shared-space techniques had better results for finding local maxima (a technique where a single graph is used to display multiple data sets). Split-space techniques were best for visual tasks requiring a dispersed visual span, but less appropriate for tasks requiring focus on a local area (a technique where small multiples are incorporated in the visualisation). It is thus difficult to construct a single visualisation that is effective for all visualisation tasks. There are trade-offs between using traditional time series and techniques such as horizon and stream graphs. Choosing one technique over the other depends on the type of information users want to extract from the data sets, as well as the size of the data set.

2.2.3 Visualising Large Data Sets

Large data sets introduce challenges to the task of visualisation [25, 26]; problems that become more apparent as data size increases [26]. Techniques applicable to smaller data sets are not always scalable to larger data sets [27, 26], and thus large data sets require consideration in their own right.

There are two inherent complexities in visualising Big Data: the human visual system and limited screen space [25]. Regarding the former, there is a limited amount of objects that can be displayed on a screen without exceeding the cognitive capacity of a human. While the size of data sets are growing, the human bottleneck remains a constant [26]. Consequently, the visual elements chosen to represent a data set must be intelligently selected to result in a meaningful interpretation of the data. This is especially the case for large data, where there are numerous potential representatives of the data set.

The second issue of screen space has been addressed in several ways [28, 29]. One solution is to increase the size of the display device (Figure 2.8) as well as the display resolution: projects such as hybrid-reality display environments (HRDE) are able to juxtapose interrelated data sets and display them on a single window [30]. Although immense display devices hold potential for data analysis, their effectiveness is hindered by the limitations of human visual acuity [26].

![Figure 2.8: A real-time visualisation of a molecular-dynamics simulation using a hybrid-reality display environment. Taken from [30].](image)
To improve the scalability of visualisation techniques, Choo et al. [25] propose the use of computational methods, such as clustering and dimension reduction, to provide compact and meaningful information about the raw data. This includes methods from machine learning and data mining. Unfortunately, these methods are computationally intensive and not practical for real-time visualisations. The next subsections briefly discuss techniques more suitable for real-time interaction.

2.2.3.1 Low Precision Computation

To lessen the computational burden, Choo et al. [25] suggest the use of low precision computation, with the motivation that humans can only perceive a certain amount of precision visually. Calculating beyond our perceived precision, for visualisation purposes, is a waste of computation. Additionally, display devices can only represent a certain level of precision. Any precision beyond which can be displayed does not contribute to the effectiveness of the visualisation. An example of this can be seen in Figure 2.9.

In addition, iterative algorithms, such as clustering algorithms, might reach a sufficient degree of accuracy during early iterations. Subsequent iterations that do not make significant changes to the visualisation (relative to what humans can perceive or display devices can represent) are unnecessary. Termination conditions should consider these constraints; computation should be parameterised with display resolution as a factor.

Figure 2.9: Scatterplots for facial-image data, where 1420 data items were visualised. Each data item consists of an 11264 dimensional vector (plots taken from [25]). There are only two pixel displacements between the two figures. The left chart used Single-precision computation; the right chart used double-precision computation.

2.2.3.2 Spatial Displacement

Large data sets with a high dimensionality are often ambiguously represented in lower dimensions, resulting in point occlusion [31]. This occurs when there are more data points than pixels, or there is a many-to-one mapping from data space to image space. One technique for minimising occlusion is jittering, which adds small random variables to data points with identical locations in screen coordinates [32]. However, due to random adjustments to point locations, jittered visualisations could be difficult to interpret and correlations could be lost [31].
Rather than using random displacement, a form of topological distortion is used by Keim et al. [33] to minimise overlap by intelligently offsetting data points. They make use of Voronoi tessellation and k-means clustering to organise points in image space. This retains neighbourhood relationships and point positions are kept close to their original locations (Figure 2.10).

Figure 2.10: Scatterplots of telephone services, taken from [33]. The x-axis depicts call duration; the y-axis call cost. Both figures are from the same data set. The left figure has no displaced points; the right figure has topological distortion applied to it.

2.2.3.3 Sampling

Sampling selects a subset of the data to be displayed, reducing data volume rather than altering data attributes [28]. Two such methods are described by Bertini and Santucci [34]. Uniform sampling maximises the amount of different data densities while preserving the magnitude of the difference. It can effectively preserve the intensity of data differences. Non-uniform sampling maximises the amount of different data densities, while altering the magnitude of the differences. Features such as zooming allow for a detailed view of a portion of the image space, but do so at the cost of losing an overview of the data set. Uniform sampling and data density maps enhance image readability, but areas with a low density are poorly visualised and structural information such as patterns are lost. Non-uniform sampling addresses both these problems and provides a detailed overview of a data set without changing the image size, while preserving structural information.

2.2.3.4 Discussion

Spatial displacement techniques are efficient in reducing occlusion in some cases, but do not preserve spatial relationships [28]. Although these techniques exploit unused screen real estate, if there are more points than pixels, they only create new overlaps [31]. Sampling can miss important structures or outliers in the data set, but there are techniques for minimizing these effects. A summary of the discussed techniques are tabulated (Table 2.1). It is clear that no one technique solves the problem of visualising large data sets. Multiple authors [31, 33] use hybrid methodologies to compensate for shortcomings of a particular technique, but might do so at the expense of increasing computational cost.
Table 2.1: Capability summary of some techniques for visualising large data sets. Entries 1-5 are adapted from [35]. Special cases are denoted by + and are discussed in the source. Clustering is scalable for large data sets, but not necessarily for real-time interaction [25]. The table includes alpha blending or “change opacity” which was not discussed. Alpha blending assigns to each data point a degree of opacity, allowing multiple points to be displayed at identical screen coordinates.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Scalable</th>
<th>Avoids Overlap</th>
<th>View Overlap Density</th>
<th>Keeps Spatial Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Non-Uniform Sampling</td>
<td>Yes</td>
<td>Possibly</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>2. Clustering</td>
<td>Yes</td>
<td>Possibly</td>
<td>Possibly</td>
<td>Partly</td>
</tr>
<tr>
<td>3. Point Displacement</td>
<td>No</td>
<td>Yes*</td>
<td>No</td>
<td>No*</td>
</tr>
<tr>
<td>4. Topological Distortion</td>
<td>No</td>
<td>Possibly</td>
<td>No*</td>
<td>Possibly</td>
</tr>
<tr>
<td>5. Alpha Blending (change opacity)</td>
<td>No*</td>
<td>Partly</td>
<td>Yes*</td>
<td>Yes</td>
</tr>
<tr>
<td>6. Low Precision Computation</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

2.2.4 Interaction Techniques

The two core components of a visualisation are the representation and interaction techniques used to facilitate visual analysis [36]. Yi et al [36] argue that interaction has received far less attention than representation, and yet interaction allows a visualisation to go beyond static or autonomous displays. The need for interaction increases when faced with large data sets. We hypothesise that interaction techniques are an essential part of each visualisation and will assist the experts in answering visual queries. The following interaction patterns are applied to the visualisations: details-on-demand, overview and detail, filtering, semantic zooming, and linking and brushing. A brief description and motivation for each technique is given in the following subsections, coupled with illustrative examples applicable to the visualisations in this project.

2.2.4.1 Details-on-Demand

It is impractical to provide in-depth detail for every data point visualised, due to limited screen real estate and the possibility of overwhelming users [37, 25]. Rather, an overview of the data can be shown and interaction can be used to allow users to request additional data without changing views (Figure 2.11). This technique is referred to as details-on-demand [38] and can be useful when a visual query requires very specific knowledge of a subset of the data, or information not explicitly shown in the visualisation.
2.2.4.2 Overview and Detail

When different levels of details exist in a visualisation, restrictions imposed by the display device might prevent a complete and sufficiently detailed view of the data from being shown [40]. Even if such a view could be displayed, it would increase the difficulty of distinguishing between information that is relevant or irrelevant to the visual query. Overview and detail is a technique for partitioning the display area into regions: one for contextual information and another for detailed information. The context area (or context brush, Figure 2.15) allows an overview of the data at the cost of reduced detail, while a more detailed region foregoes contextual information to supply an increased level of detail.
2.2.4.3 Filtering

Not all visual queries require the user to be conscious of every data point. Filtering is a technique used for removing points that do not assist in answering a visual query \[38\]. Often filtering is implemented as Boolean expressions applied to data attributes, for example, using disjunction to filter in selected criteria. In the context of an RFI visualisation, it could be used to distinguish between different RFI sources or to toggle the visibility of RFI or untrustworthy data in the representation. When such techniques are executed as rapid, interactive, and reversible actions, they can be referred to as dynamic queries. Shneiderman \[38\] suggests that when a user performs a dynamic query, the representation should ideally be updated within 100 milliseconds. Thus, it is necessary to consider the response time of interactions during evaluation (described in Section 3.4.3). An example of filtering is given in Figure 2.13.

![Figure 2.13: "NBA Draft", by Adam Pearce \[41\]. Drafted players are encoded as bars and a slider is used to filter players according to their rating. The top chart shows the top 40 players per year; the bottom chart is filtered to show the top two players per year only. Filtering makes it easier to locate players of a certain rating.](image-url)
2.2.4.4 Semantic Zooming

At different levels of detail, a user typically finds it useful to see different information [42]. Semantic zooming changes the representation of the objects in view depending on the degree of magnification (or zoom level). This differs from geometric zooming, which only enlarges or decreases the size of objects as the user changes the zoom scale. In cases where data points are aggregated and mapped to a single object, semantic zooming can allow the object to be deconstructed into its constituent parts as the user magnifies the view (Figure 2.14).

![Figure 2.14](image1)

**Figure 2.14**: Filtering of information as implemented by Taxi Stockholm [43]. Fine detail is aggregated into a representative unit. As the user zooms in, the constituent parts making up the whole are revealed.

2.2.4.5 Linking and Brushing

A single visualisation provides only one view of the data. To overcome the shortcomings of a single visualisation, multiple visualisations can be linked together in a single display. The collection of visualisations represent the same data, but data might be projected to different dimensions (Figure 2.15) or a different representation scheme could be used (Figure 2.16). By interactively selecting the data in one plot (brushing), the linked representations are updated to reflect the selection across multiple plots.

![Figure 2.15](image2)

**Figure 2.15**: The multivariate Iris data set visualised as scatterplots [44]. The left plot is the original; in the middle and right plot, a user brushes a subplot, and the selection is reflected in the other 15 subplots.
2.2.4.6 Discussion

All of the techniques we described could be useful for RFI visualisations. Because of the large volumes of data collected at MeerKAT, there is a risk of placing a cognitive burden on users. Techniques such as details-on-demand, overview and detail, and filtering could reduce this burden. They allow detailed information to be hidden until the user requests more data, instead of discarding the data permanently. Similarly, zooming allows the user to control the level of detail, while the discussed techniques could support the zoom interaction by providing contextual information. If there are multiple charts present, they could be linked together to respond to user interactions simultaneously. Charts therefore cooperate while users answer visual queries.

2.2.5 Visualisation Evaluation

How can it be determined whether a visualisation has achieved these objectives? In Johnson’s [27] list of Top Scientific Visualisation Research Problems, quantifying effectiveness is listed as second. Researchers, such as Kosara et al. [46], advocate the appliance of user studies. Johnson recommend the scientific method of observation, hypothesis formulation, and evaluation to determine efficacy.

Hullman et al. [47] note that evaluation models for visualisation often solely rely on measurements such as user response time and response accuracy. The intent is that focusing on these attributes optimises cognitive efficiency, and hence the ability to rapidly and accurately communicate information. This approach has been criticised, as it does not sufficiently capture the complex nature of visualisations [48]. User performance measures are frequently inconsistent, which makes it difficult to choose between visualisation designs [49].

A lack of confidence in the efficacy of visualisations hinder their integration into the systems they were intended for [50]. Carpendale [50] believes this is because visualisations are seldom tested beyond simple data sets, simple tasks, and university students. Using real users, real tasks, and large complex data sets would increase the validity of a visualisation [50]. Lam et al. [51] describe evaluation questions and methods for seven visualisation scenarios that could be applicable to this project. Of particular interest is the Visual Data Analysis and Reasoning scenario, which evaluates user tasks such as data analysis, decision-making, and knowledge discovery. We adapt this model in our user experiments.
In order to reduce user distractions and interruptions, we also need to consider the response time of interactions. A survey by Kalwasky [52] tabulates three ranges for interaction response time and the effect it has on users. The threshold ranges are incorporated as target values for the latency associated with unit tasks (Table 2.2).

Table 2.2: The effect of latency (measured in seconds) on users. Adapted from [52].

<table>
<thead>
<tr>
<th>Latency</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0 - 0.1 s</td>
<td>The ideal time range. Users perceive the results of interactions as immediate. Most unit tasks are required to fall into this range.</td>
</tr>
<tr>
<td>0.1 - 1.0 s</td>
<td>Within this range, users can still maintain a flow of thought without feeling interrupted. After 0.1 seconds, users might feel they have lost contact with the data.</td>
</tr>
<tr>
<td>1.0 - 10 s</td>
<td>The time limit for users to remain attentive is 10 seconds, and the risk of distraction increases greatly thereafter. For interaction techniques that require longer processing, a progress indicator could be useful.</td>
</tr>
</tbody>
</table>

It is clear that there are many aspects affecting the efficacy of visualisations. We thus employ both quantitative and qualitative methods in our evaluation strategies (response time and expert feedback, respectively). However, due to the complexity of visual analysis, we rely more on qualitative results.

2.3 Conclusions

Discovering novel ways of exploiting modern computing is an important challenge in increasing the effectiveness of visualisations: the human bottleneck will remain constant while data size increases. It is thus necessary to find ways to compensate for both human and technical limitations. Regardless of the underlying data, there are common attributes of a good visualisation:

- Interesting structures, outliers, and patterns in the data set should be preserved. Displayed data should give a fair representation of the data set.
- Visualisations should minimise the cognitive effort required to make such observations.
- Multiple perspectives of the data should be shown with different levels of detail.

A selection of techniques specific to time series and large data sets have been covered in this review. For time series, using small multiples and the macro-micro principle is advisable. Compared to traditional graphs, stream and horizon graphs achieve some improvements. To compensate for their weaknesses, use them in conjunction with other visualisations.
Big Data techniques address visualisation difficulties by parameterising computation, by altering data attributes in image space and through data reduction. Combinations of these methods can be used as they operate at different stages of the visualisation pipeline, but add to the computational cost. The ability of visualisations to answer visual queries can be ameliorated by adding interaction, since interaction offers methods to manipulate data attributes and levels of detail. None of the techniques offers a complete solution to the information need at MeerKAT. However, they are able to provide additional features that are not being taken advantage of by the current system. This includes showing a detailed picture without losing an overview, comparing multiple data sets, and maximising usage of screen real estate.
3 Design

In this chapter, we discuss the design of the visualisation system. This includes the system architecture, operations performed on the data, and methods for evaluating the effectiveness of visualisations. The visualisation software is accessed through the web using a client-server architecture. Data is stored on a server and passed to the client upon request, where it is processed and displayed within a web browser. The visualisation pipeline is has two aspects:

- The design of the server-side data filter, which reduces the raw data size and converts the data to an intermediate format suitable for the visualisation.
- The design of the client-side visualisation, which runs within a browser.

The filter is an essential component in the pipeline, since the visualisation cannot run without the data being pre-processed. Thus, the filter is designed during a preliminary phase before the visualisation design and development iterations initiate. We use Heer and Agrawala’s Reference Model [53], which provides a separation between the data and visualisation model, as a template for the visualisation framework (Figure 3.1).

3.1 Design Goals

The set of desired features of the visualisation have been documented by Christopher Schollar (the client). Since there is a large volume of data, the primary objective is a visualisation that will assist in answering visual queries without adversely affecting the response time of user interactions. This section is concerned with the visual analysis tasks performed on these large data sets (or the requirements of the visualisation), and the desired properties of the data filter. The visualisation task taxonomy of Amar et al. [54] is adapted to describe the visual analysis tasks and requirements, which are:

- A distinction between power values that were flagged as RFI, and power values in which no RFI was detected.
- The retrieval of power values for a given time and bandwidth channel.
- The ability to place constraints and conditions on the bandwidth channels, power values, and period of the data being viewed. Examples include viewing the frequency range or time period with the most or least RFI activity, or specifying a range for the bandwidth, power values, or time instances.
- Display aggregates or extrema of the data: averages, maxima, minima, number of occurrences of RFI, or percentage of RFI in the data.
- View anomalies: rare or infrequent RFI occurrences or the presence of RFI sources previously unobserved, and display over-range events (untrusted data).
- Find clusters, similar regions, or recurring patterns in the data. For example, are there clusters of a certain power value range, or a recurring RFI pattern for a specific bandwidth channel?

The data filter does not affect the type of visual queries, but can affect the answers to these queries. Thus, the data that is sent from the server to the client must provide a fair representation of the raw data to prevent inaccurate observations. For the
purpose of RFI monitoring, data should be sampled such that points affected by RFI receive preference over uncontaminated data. Furthermore, the output of the filter should be in a format compatible with the visualisation components.

3.2 System Architecture

In Figure 3.1, the data source generates raw data: HDF5 data sets containing power values and a CSV file containing a list of over-range events (times at which the power values cannot be trusted). The RFI detection component uses this raw data to generate an RFI mask. Each raw data point will have a corresponding value in the mask file, which indicates whether the data point was flagged as RFI, and possibly indicate the type of RFI that was detected. The filter takes as input the raw data extracts a subset of the data. Ideally, the mask file should be processed by the filter and thereafter be sent to the server, but the inclusion of the mask file into the visualisation pipeline is left for future work.

Figure 3.1: An illustration of how the different components of the project are connected together. Back-end processes are indicated by red arrows; processes initiated by the front-end are indicated with a blue arrow. Processes that were not developed are indicated with grey dashed lines.
In the initial step, the raw data is converted to a format compatible with the visualisation framework and stored on the server. Once a client sends a request, the aggregated data and over-range file is sent from the server to the client. The data is further processed within the web browser by the visualisation. Data points are mapped to visual encodings, which are used to construct one or more views. The output is a graphical display, with interface elements to facilitate interaction. Interface elements allow the user to change views or the visual encodings of a view. The visualisation with its view and control components (Figure 3.2) are the core of the project. Data filtering (Figure 3.1) forms an integral part of the pipeline, but a sophisticated implementation thereof is left for future work.

3.3 Filter Design

The aim of the filter is to reduce the amount of client side processing required before graphical output is displayed. Only a subset of the data is transmitted from server to client. The filter does not include server side RFI detection. However, the previous thresholding algorithm (similar to the one implemented by Christopher Schollar) has been included as a component within the visualisation. Consequently, the filter only operates on raw data to produce aggregates, whereas RFI is detected in the front-end. The filter is not intended to replace the implementation used by the RFI management team. It is only used as a temporary measure, as we do not have live access to the streamed data from the MeerKAT site.

The thresholding algorithm calculates whether the value of a data point falls within an acceptance range. If a point falls outside of the range, it is considered RFI. The acceptance range boundary is constructed such that points greater than a threshold value from the mean of a frequency channel are flagged. Sampling data by extracting the maximum in a neighbourhood reduces the risk of missing potential RFI (compared to random sampling).

Data reduction is performed without knowledge of the visualisation. Information, such as the time and frequency at which a point was sampled, has to be explicitly encoded in the output file for the visualisation to interpret how data was sampled. The data is not sampled at consistent time and frequency intervals: the maximum in one neighbourhood could have been sampled at a different time or frequency than the maximum in an adjacent neighbourhood. The visualisations require a consistent scale.
to be useful for comparisons. That is, the distance between points in screen coordinates should correspond to a scalar multiple of the distance between them in data space. An approximate time and frequency can be chosen for each point, which would result in consistent time and frequency intervals in the output file. For each sampled point, the time and frequency of the top left corner (the reference point) in the neighbourhood is assigned to a point sampled in the region.

```javascript
1. // sampleRows specifies the bin height; sampleColumns specifies the bin width
2. function sampleData(data, sampleRows, sampleColumns){
3.   var rows = data.numberOfRows;
4.   var columns = data.numberOfColumns;
5.   var yStep = rows/sampleRows; // sampling neighbourhood height
6.   var xStep = columns/sampleColumns; // sampling neighbourhood width
7.   var aggregate[1][sampleColumns]; // the aggregated power data
8.   for(var y = 0; y < rows; y += yStep){
9.     for(var x = 0; x < columns; x += xStep){
10.    var maxPower = verySmallNumber;
11.    for(var i = y; i < y + yStep; i++){
12.      for(var j = x; j < x + xStep; j++){
13.        var power = data[i][j];
14.        maxPower = max(power, maxPower);
15.      }
16.    }
17.    aggregate[y][x] = maxPower;
18.  }
19.}
```

Figure 3.3: Pseudo-code for the sampling algorithm.

Figure 3.4: The format of the CSV output file. The first column is reserved for timestamps, and the first row from the second cell onwards for the frequency value at which the point was sampled.

<table>
<thead>
<tr>
<th>Time (HH:mm)</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00</td>
<td>250 450 650 850</td>
</tr>
<tr>
<td>00:15</td>
<td>1 2 9 7</td>
</tr>
<tr>
<td>00:30</td>
<td>3 4 7 6</td>
</tr>
<tr>
<td>00:45</td>
<td>0 5 0 8</td>
</tr>
</tbody>
</table>

Output with bin resolution 2x2

<table>
<thead>
<tr>
<th>Time (HH:mm)</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00:00</td>
<td>4 9</td>
</tr>
<tr>
<td>00:30</td>
<td>5 8</td>
</tr>
</tbody>
</table>

Figure 3.5: Example output of the sampling algorithm for a bin resolution of 2x2. The data (left) is divided into 4 regions, each with a width and height of 2 units, resulting in a reduced version (right). The top left corner is chosen as the reference point in each sample neighbourhood.
The thresholding algorithm requires the mean and standard deviation of the frequency channel (a column in the power data, Figure 3.5) for a point to determine the presence of RFI. Because the aggregation method is applied to a region (multiple columns) rather than on single channels, the statistics must also be calculated over a region. After the data is parsed in the visualisation, it is downsampled to form a list of objects – each with a different bin resolution. These objects are used by the zoom interaction to switch between levels of detail. Thus, for the thresholding algorithm to be correctly applied across multiple resolutions, a statistics file should be calculated for each object.

The filter produces a collection of CSV files of a specified bin resolution (i.e. the number of rows and columns that should be in the output file). These files are sent to the client upon request and parsed within the client to ensure compatibility with the visualisations. The file format specification ensures the visualisation can read the files correctly. An additional requirement is that, for lower resolutions to be calculable from the power value CSV file, the retrieved file’s resolution must have dimensions satisfying the constraints:

\[
\text{rows} = rm^{n-1} \\
\text{columns} = cm^{n-1}
\]

Where \( n \) is a positive integer representing zoom depth, \( m \) is a positive integer representing the zoom factor, \( r \) is the row count of the lowest bin resolution in the list of zoom levels (or zoom objects), and \( c \) is its column count. The greater \( m \) is, the greater the difference in bin resolutions between zoom events. The zoom interaction has three zoom levels and thus \( n \) is set to 3. We set \( m \) equal to 2, which allows each successive downsample step to halve the data if it is a row or column, or to divide the data by 4 if it is a 2D array:

**Initialisation:**
\[
\text{rows}_3 = r2^{3-1} = 4r \\
\text{columns}_3 = c2^{3-1} = 4c \\
\text{columns}_3 \cdot \text{rows}_3 = 16cr
\]

**Iteration 1:**
\[
\text{rows}_2 = \text{rows}_3 / 2 = 2r \\
\text{columns}_2 = \text{columns}_3 / 2 = 2c \\
\text{columns}_2 \cdot \text{rows}_2 = 4cr
\]

**Iteration 2:**
\[
\text{rows}_1 = \text{rows}_2 / 2 = r \\
\text{columns}_1 = \text{columns}_2 / 2 = c \\
\text{columns}_1 \cdot \text{rows}_1 = cr
\]
The statistics and over-range data are halved, whereas the power values are reduced by a factor of \(2 \cdot 2 = 4\) after each downsample iteration. It is possible to calculate aggregates for the power values client-side, since the maximum is preserved over consecutive reductions. The standard deviation cannot be calculated without the raw data, and it is therefore precomputed on the server by the filter for each bin resolution (568x400, 284x200, and 142x100).

3.4 Visualisation Design

The first set of visualisation designs is chosen from a list of techniques selected according to their ability to facilitate the visual analysis tasks identified (see Section 3.1). Since it is unlikely that one visualisation can answer all visual queries, each visualisation focuses on a subset of the list of tasks. Visualisation mock-ups for the tasks were created using tools available in Microsoft Excel or Google Charts, and visualisations by other authors could be used as a reference to determine how suitable a chart type is for answering visual queries. These mock-ups were presented to the client for approval during a design iteration, and the goals for the subsequent development iteration were decided. This strategy is repeated for the two iterations that follow the preliminary development phase. Upon completing a development iteration, the ability of the visualisation to answer visual queries is evaluated by the client (first iteration) or by both the client and domain experts (second iteration).

Instead of only examining aspects of a visualisation in isolation (visual encodings or interactions) and measuring user performance, we consider a visualisation’s ability to support analytical tasks as a whole. We adapt the Visual Data Analysis and Reasoning scenario of Lam et al. [51] for evaluating the proposed visualisations (see Section 2.2.5). Domain experts familiar with RFI visualisations are the target group for the evaluation questions. The expectation is that, during their experience with domain tools, experts would have likely developed a set of patterns around the elements of their task [55]. They are more likely to point out which features of the visualisation are beneficial or detrimental in the context for which the visualisations are used.

3.4.1 Design Approach

A data representation technique will be proposed to the client and project coordinator that describes the basic form the data will take (for example, a scatterplot for showing correlations), and interaction patterns will be matched to the data representation. If the client and coordinators approve of the concept, the refine and interact loop initiates (Figure 3.6).

![Figure 3.6: A representation is chosen at the start of an iteration and there are two iterations in total. After a representation is chosen, incremental improvements to the representation and interaction techniques are made for a duration of between 2 and 3 weeks. The figure was adapted from [14].](image-url)
During this loop, which lasts between two and three weeks, the concept is implemented and incremental improvements are made to the visual encodings and interaction techniques. Changes to the representation could present new opportunities for interaction, and changes to the interaction could require the representation to be reconsidered. Once both the visualisation and its respective interaction patterns are implemented, the visualisation will be evaluated by the client and domain experts. The output of the evaluation methods are both quantifiable metrics and subjective feedback from the experts. The output will be used to influence the next iteration, and thus the representation and types of interactivity of the next visualisation. Each of the two iterations consist of an implementation and evaluation phase, and stakeholders (the client, project coordinators and domain experts) are to be kept involved before and at the end of an iteration.

3.4.2 Test Data

The visualisations are tested with real data sets, all of which were collected at the MeerKAT site by the RFI management team. The data at the MeerKAT site is collected throughout the day, where the data for each hour is stored in a single file. The raw data file contains a table consisting of 3600 rows (one row for each second) and 14200 columns (one column for each frequency channel). Each cell contains a power value associated with a time and frequency (implicitly encoded by the row and column numbers, respectively). Raw data is stored for up to two weeks, but it is possible to acquire data collected over a longer period if necessary. After two weeks, only aggregates of the raw files are stored. Visualising the raw hourly data files is the objective of the project, and a single file is visualised per session.

3.4.3 Preliminary Evaluation

It is required of a visualisation software to be stable and well developed before experts can perform visual analysis tasks [51]. Preliminary tests will be performed before the visualisations are evaluated by the experts to ensure it is usable in its intended environment. The tests are constructed to determine whether the visualisation can maintain interactivity within an acceptable latency threshold when used with a raw data file. Each visualisation has its own set of visual tasks. To complete a visual task, a series of unit tasks need to performed, such as selecting a portion of a context brush to restrict the range on an axis. This might be done to discover extrema in a neighbourhood, or to compare trends over multiple graphs for a certain time range. Unit tasks are not unique to visual queries, as is seen from this example.

For a given visualisation, the latency of each unit task is measured in milliseconds, and it is determined whether the latency falls within a threshold (see Section 2.2.5, Table 2.2 for the threshold ranges). There are many sources of latency in a web visualisation [56], but only the factors within the scope of the project are mentioned:

- Query processing
- Data transfer from the server to the client
- Data processing within the client
- Rendering the data
Unit tasks only become available once the data has been sent from the server to the client. The visualisation is therefore only concerned with latency due to data processing and rendering of the data points.

3.4.4 Expert Evaluation and Interviews

Five experts from the University of Cape Town’s Astronomy Department are used to evaluate the visualisation. A form listing visual queries, visualisation components, and interaction techniques is given to the experts. The experts enact the process of using the visualisation on a real data set, answering the visual queries associated with the visualisation. The visual queries drawn from both those listed in Section 3.1, and any insights that are suggested by the experts. The following questions are asked:

- Which insights did the visualisation best support? Which insights were not adequately supported? The experts will be asked to answer visual queries while working on the tool and to give feedback on the visualisation’s ability to answer them.
- Which interaction patterns were the most useful? Were they useful or a hindrance? The ability of the interactions patterns to assist in answering the visual queries are evaluated.
- How does the visualisation support generating information about the domain? Are the visualisations too general and should they be more targeted towards the domain?
- What was the quality of the data analysis experience? Was the interplay between visual output and interaction intuitive and easy to use? Was there anything that felt lacking in the visualisation?

After the experiment, the experts are asked to give verbal and written feedback about the visualisation and interaction techniques. The critique of the visualisation is either associated with the interaction patterns that accompanied the visualisation or how the underlying data was represented. If the experts make suggestions regarding either, the visual encodings and interactions will be adapted during the next iteration, or be incorporated in a new visualisation depending on the scale of the suggestion. If it is found that a certain interaction or representation component was useful, the component and its dependencies will be further developed. A disliked or infrequently used component can be discarded or set as optional.
4 System Implementation

The visualisation is expected to be accessible through a web browser in order to free users from the task of managing a local copy of the raw data. Figure 4.1 illustrates the components of the visualisation pipeline. Data parsing and initialisation occur before charts are created and are independent of the representation. These components were developed separately from the visualisation iterations during a preliminary development phase, while the representation and interaction components were developed during two design iterations. An extended design is proposed in Section 7.3, but the design was not implemented.

The filter (back-end) and the visualisation (front-end) were developed separately and do not interact directly. Instead, a common data format is used: the output of the filter is stored as CSV files on the server, while the client requests CSV files when launching the visualisation. An ideal filter would listen for new data and stream data directly to the visualisation as requests are made, rather than writing data to an intermediate format.

4.1 Implementation Languages and Libraries

The filter was only developed as a temporary measure and Java was chosen as the programming language out of convenience – a complete back-end is left for future work. Since the visualisation executes client-side, the previous prototype was developed entirely in JavaScript. Web visualisation libraries offer a wide range of features: from reusable charts to basic functions such as shape drawing and axis construction. Due to the specific requirements of the visualisation, it was necessary to develop the visualisation components without using existing templates. Hence, the libraries were chosen for their ability to provide basic drawing functions. The only exceptions were the usage of dat.GUI [57] and Pace [58], which required minimal configuration before being usable. The JavaScript libraries that the visualisation incorporates and the primary functionalities that were used are listed in Table 4.1.

4.2 Filter Implementation

The filter is implemented as a Java program that converts a single HDF5 file (raw power data) to a CSV file (aggregated power data). Additionally, the mean and standard deviation are calculated for each column in the aggregated power data. JHDF5 was used to read the HDF5 files and create a 2D array holding all the power values, which is then aggregated within the Java program according to a specified bin resolution. Due to the size of the raw files, reading entire file sets into memory to serve multiple client requests would not scale even for a small group of users. Thus, the aggregates are precomputed for a fixed bin resolution and stored on the server, rather than being dynamically calculated as requests are made by the client. However, precomputed data limits the number of views that can be constructed from the data. A visualisation allowing the retrieval of arbitrary bin resolutions would require a more sophisticated filter.
Table 4.1: The libraries incorporated in the visualisation and matched to usage cases. Entries marked with an asterisk were used during early iterations, but were replaced by libraries that are more suitable for the visualisations in this project.

<table>
<thead>
<tr>
<th>Library</th>
<th>Iteration</th>
<th>Visualisation Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>D3 [59]</td>
<td>1 – 2</td>
<td>Axes construction, line and area drawing, text labels, context brushing</td>
</tr>
<tr>
<td>D3-Tip [60]</td>
<td>1 – 2</td>
<td>Colour scale mouse-over annotations</td>
</tr>
<tr>
<td>dat.GUI [57]</td>
<td>1 – 2</td>
<td>Visualisation control interface</td>
</tr>
<tr>
<td>Heatmap.js* [61]</td>
<td>1</td>
<td>Waterfall chart area (rectangle drawing)</td>
</tr>
<tr>
<td>P5* [62]</td>
<td>1</td>
<td>Waterfall chart area (rectangle drawing)</td>
</tr>
<tr>
<td>Pace [58]</td>
<td>1 – 2</td>
<td>Page load progress</td>
</tr>
<tr>
<td>PIXI [63]</td>
<td>2</td>
<td>Waterfall chart area (rectangle drawing)</td>
</tr>
<tr>
<td>Two* [64]</td>
<td>2</td>
<td>Waterfall chart area (rectangle drawing)</td>
</tr>
</tbody>
</table>

Figure 4.1: An overview of the visualisation pipeline implementation. The visualisation components consist of representation and interaction components. Bin resolutions indicated in the format: columns x rows (e.g. 142x100).
Unlike the raw file, which only contains power values, the CSV files contain additional information to indicate how the raw data was sampled. The sampling algorithm (Section 3.3) converts the raw power data to an aggregate file, and three separate files containing the mean and standard deviation for different bin resolutions. Raw data for the over-range events are stored in CSV format and thus need no conversion. The example files in Figure 4.2 are derived from a raw file with a resolution of 4x4; the aggregated file was binned to a resolution of 2x2; the statistics file was calculated for a bin resolution of 2x2, hence it has two rows.

```plaintext
1. // a CSV file version of the raw power data (sample regions separated by shades)
   1, 2, 9, 7
   3, 4, 7, 6
   8, 5, 0, 8
   0, 0, 0, 0

2. // CSV files on the server (with white space added for readability)

3. // CSV file contains power data for the max bin resolution
   10. Time, 250, 650
   11. 00:00:00, 4, 9
   12. 00:30:00, 5, 8

4. // stat CSV file
   15. frequency, mean, std
   16. 250, 4.5, 0.5
   17. 650, 8.5, 0.5

5. // over-range events CSV
   20. 00:00:00

Figure 4.2: The CSV files containing precomputed values. The top file contains the sampled data (aggregates); the middle file contains information about the median and standard deviation of a sampled region; the bottom file indicates any over-ranges that occurred during the time range.
```

4.3 Data Retrieval and Parsing

Data retrieval from the server can be time-consuming. The size of the transferred data can be large, and JavaScript by default executes code sequentially inside a single thread. A time-consuming code block would freeze the interface until its execution has completed. Mike Bostock’s D3 visualisation library includes methods for asynchronously retrieving data residing on the server. The server should support PHP for such requests to work. Asynchronous retrieval allows the rest of the page to load while the data is fetched in the background. It is possible that the main thread reaches a code block dependent on the data before the data has loaded.

To prevent this, D3 allows the specification of a call back function that is executed only once the data has completely transferred to the client. The call back function is used to convert the data to multiple JSON objects, each of which represent a different bin resolution and visualisation view (Figure 9.1 gives an example JSON object). Access to data objects is limited until all the fields are initialised. Encapsulating the data as a JSON object simplifies the task of working with multiple files or views, since a single variable can be used to keep track of the active data object (the object being
visualised) while the inactive objects are stored in a list until a user changes the current view.

The server only sends the highest bin resolution available in the repository, whereas lower resolutions are calculated within the browser. Each data resolution is added to a list and, if a user triggers a zoom event, and hence a view of a different bin resolution, the appropriate data object can be retrieved from memory rather than being computed on the server. The extracted data object is then set as the new active data object. Fields within the object provide the information necessary to update the representation to reflect the new data object.

The data is downsampled iteratively using the algorithm in Figure 9.3. Once a lower resolution is created, it is set as the parent data for the next iteration. During the next iteration, the parent data is used to create another downsampled object. This process repeats until the minimum zoom level is reached. Power values are downsampled with the same algorithm used for binning raw data in the server-side filter. The over-range events are a special case of the data-binning algorithm where the 2D input array has only one row, and points in the data are either true or false. For clarity, a simplified version is given as a function in Figure 9.3. Instead of finding the maximum in the neighbourhood, over-ranges require the logic OR operator to be applied to each value. If there is a true value in the parent bin neighbourhood, a true value will be propagated to all lower resolutions.

The server calculates the mean and standard deviation and stores it in three files. Within the client, a sorted array holds the data of the three files: the highest bin resolution (maximum zoom level) is placed in the last position; the lowest bin resolution (minimum zoom level) in the first position. While the loop iterates through zoom levels backwards, starting from the maximum zoom level, the current zoom value in the loop can be used to retrieve the correct statistics data from the array.
5 Visualisation Design and Evaluation: Iteration 1

5.1 Overview

The visualisation design and evaluation iterations focused on designing representations for the data, and designing interaction methods to manipulate and augment the representations. At the end of this iteration, the results were presented to the client for critical feedback. Thereafter, the goals for the next cycle were discussed with the client. Before the first development cycle (described in Section 0), we presented design concepts to the client. From the discussion, the following goals for the iteration are decided:

- The development of a waterfall chart (or heatmap).
- The inclusion of a configurable RFI thresholding algorithm. Detected RFI should be distinguishable from uncontaminated data.
- The addition of supportive charts that allow unclear or unavailable information in the waterfall chart to be revealed.

The waterfall chart was chosen because it displays data compactly – particularly data points with three attributes (in this case frequency, time, and power). Frequency channels affected by RFI are clearly shown, while patterns in the RF environment are exposed. Two charts were chosen to support the waterfall chart: a bar chart and spectrum graph. The spectrum graph can display the rows in the heatmap with higher accuracy, while the bar chart shows aggregated information.

5.2 Visualisation Design

The main component of the visualisation is a waterfall chart, and the interaction techniques of the previous section are applied to improve the waterfall chart’s ability to answer visual queries. For this reason, supportive charts were added which depict the power values with increased detail. The heatmap is designed to occupy the largest percentage of the display area, while the remaining area is reserved for additional charts (Figure 5.1).

Power values are encoded as shaded rectangles, and the width and height of the rectangles are equal for all cells. The area dedicated to the heatmap area is determined by the resolution of the display device, but the ratio of width to height of the area is fixed. Ideally, the whole visualisation should fit within the display device without the user needing to scroll. For devices with a high display resolution, the width and height are scaled up. The area for devices with a lower resolution are downscaled, but with a lower bound imposed to ensure readability of the charts and to eliminate clutter (e.g. overlapping text labels).
Figure 5.1: A design of the chart layout. The heatmap occupies the most significant portion of the display area (left); the remaining area is divided into two and reserved for additional charts (right).

The heatmap alone is unable to allow a user to answer all the visual queries from Section 3.1. The shade variations that encode power values in the heatmap do not allow precise values to be perceived. To increase the precision, a spectrum graph is added that encodes the power values as a line. The x-axis depicts frequency values; the y-axis depicts power values. Thus, the data visualised corresponds to a row in the power data. The height of the line above the x-axis for a given point is a scalar multiple of the difference between the minimum power value and the value of the point.

Because the heatmap visualises a large number of rows in the data, displaying all the rows in the spectrum graph would clutter the graph. Linking and brushing can be used instead: when a user hovers over the heatmap, the row touching the cursor can be set as the visualised row (Figure 5.4). In addition, the spectrum graph can show the user why data was flagged as RFI. A threshold region can be drawn in the background of the chart. If a point in the spectrum falls above the region, the point is considered RFI.

The bar chart has a bar for each shade in the palette. Since there are eight different palettes (Figure 5.2), there are eight different bar chart configurations. The height of the bar chart indicates the percentage of power values that were binned to that shade. This allows the most active frequency range to be seen with ease. To indicate which palette is used for the heatmap area, a chart legend can be added in the form of a colour scale. The colour scale in Figure 5.5 is sorted from light to dark, to match how values are mapped to shades in the heatmap area. An example of all the charts discussed is given from Figure 5.3 to Figure 5.5. The same data shown in Figure 5.3 is used across all plots, and a clear relationship between the different visualisations can be seen.
Figure 5.2: The eight colour palettes for the heatmap area. Colour Brewer was used to generate the RGB values used for the palettes.

Figure 5.3: Two waterfall chart mock-ups. The top chart shows power values as shaded cells. The bottom chart does the same, but overlays the cells containing an RFI point with red.
Figure 5.4: A spectrum graph mock-up (top), with the row (bottom) in the waterfall chart it visualises (bottom row in Figure 5.3).

Figure 5.5: A bar chart mock-up (top) for a colour configuration with five shades (bottom). The bar height indicates the percentage of points binned to the matching shade in the palette. The bottom shades function as a chart legend. See Figure 5.3 for the corresponding waterfall chart.
5.3 Implementation Iteration 1

The preliminary phase was dedicated to developing a filter, data retrieval and parsing mechanism. This code base was reused for all consecutive iterations, with minor changes made throughout iterations to increase the ease of integrating it with the visualisation components. All charts made use of D3, while only the waterfall chart used heatmap.js and P5.

5.3.1 Waterfall Chart

The first version of the chart consisted of several components that were constructed using multiple libraries:

- D3 was used for SVG manipulation, and creating and labelling axes.
- Heatmap.js and P5 were used to construct two versions of the heatmap area. Due to a change in visual encodings, heatmap.js was replaced by P5.

P5 requires the specification of an HTML DOM element that will serve as a parent node for the canvas element that the libraries generate. The properties of the parent node are used to initialise the canvas. The heatmap requires two additional parameters that are not listed within the data object: normalised data points and a colour gradient. Frequencies are mapped to a point between \([0, \text{width}]\), while time values are mapped to a point between \([0, \text{height}]\), by normalising the values using the algorithm in Figure 9.2.

Colour configurations are independent of the data set and are thus separately stored. To determine the shade of a point or region, power values are normalised and mapped to an integer range. The particular range a value is mapped to depends on the active colour configuration. For each colour configuration, there is a matching palette. In total, there are eight palettes. The palettes are identified by the number of grey shades they consist of (Figure 5.6). A single palette is stored as an array, where each item in the array is a colour shade drawn from the respective palette, and items are sorted in decreasing order according to hex values of the shades.

Consequently, the first index in the array corresponds to the lightest shade in the palette while the last index corresponds to the darkest shade. The shade of a point can thus be retrieved by mapping the value of the point to an array index, which is used to access a shade in the array. As there are eight palettes in total, each point has eight shades associated with its value (one from each palette). At start-up, the shade values are calculated for the default palette (seven shades). If a user changes the number of bins, and thus the palette, the shade values are calculated for that palette and the results are stored in a JSON object. This allows the hex value to be retrieved from the shade matrix during a render loop or display update, rather than recalculating the matching shade for each display update.

After the shade matrix is calculated, the heatmap area can be constructed. The area is tiled starting from the top left corner of the heatmap area (Figure 9.4). Example output of the algorithm for the two different approaches taken can be seen in Figure 5.8. D3 is able to generate axes and appropriate tick marks in SVG format, if given the
correct parameters. These parameters are the length of the desired axis in pixels, the minimum, and maximum of the values for which the axis is generated, and a scaling function that maps a point from data space to pixel coordinates. The scaling function normalises the data points to lie within the range $[0,\text{length}]$ and assigns to each point in data space a corresponding point on the axis.

1. // JSON object holding the 8 palettes. Only the first 3 are fully shown
2. var palettes = {
3.   2: [12434877, 6513507],
4.   3: [15790320, 12434877, 6513507],
5.   4: [16250871, 13421772, 9868950, 5395026],
6.   5: [ ... ],
7.   6: [ ... ],
8.   7: [ ... ],
9.   8: [ ... ],
10.  9: [ ... ]
11.};
12. // an example usage
13. var activePaletteID = 3,
14. range = [0, 1, 2], // normalised values lie herein
15. normalisedPowerValue = 1;
16. var palette = palettes[activePaletteID];
17. var hex = palette[normalisedPowerValue]; // hex = 12434877

Figure 5.6: The JSON data structure holding the colour palettes. The hex values are shown as decimals.

Figure 5.7: The end result of the waterfall chart after the first iteration. The heatmap area was rendered using P5 and the axes built using D3. For this example, a palette with 7 shades was used.
Figure 5.8: A comparison of the output of heatmap.js (left) and P5.js (right) for the colour palettes with a shade count of 3, 5, 7, and 9 (from top to bottom). The x-axis represents the frequency range; the y-axis represents the time range. Dark shades encode a higher power value; lighter shades encode a lower value.
5.3.2 Bar Chart

Each shade to which power values are mapped has an associated range, and if the value of a point falls inside the particular range, it is assigned that shade value. The bar chart displays the percentage of points that were mapped to a shade, and thus the bar chart has the same number of bars as the shade count in the colour palette chosen by the user.

While data is normalised, or binned to a shade, a data structure keeps tracks of the number of points binned to each shade. Additional information, such as the maximum and minimum value binned to the shade, and the infimum and supremum of the shade range is kept. The percentages are used to determine the bar heights, while the additional information is displayed when the user hovers over a bar in the chart. Each palette requires a separate bar chart configuration, since the information represented by the chart depends on the active palette. The configuration for a specific palette is constructed as in Figure 9.6, and has to execute for each of the eight palettes.

It should be noted that a small value of 0.001 is added to the global maximum before values are normalised. That is, the range is \([\text{globalMin}, \text{globalMax} + 0.001]\). This is because the floor of a normalised value is used to calculate an array index within the shade palette. Without the small increment, all points except the global maximum will be less than the number of shades, while the global maximum is mapped to an integer index equal to the number of shades. Because shades are stored as an array, the upper limit is one less than the number of shades. Therefore, the global maximum should be mapped to a value one less than the shade count. By adding a small increment to the range, the global maximum will be mapped to a value just below the shade count, of which the floor value is the correct array index. The whole of the bar chart was implemented in D3, and all the components were created as SVG elements. D3-Tip was used to create the popups. A sample output for a palette with seven shades is shown in Figure 5.9.

![Figure 5.9: The resulting bar chart for a palette of seven shades. A user has hovered over the second bar, revealing additional information about the values binned into the region.](image-url)
5.3.3 Spectrum Graph

Similar to the bar chart, all of the components were encoded as SVG elements using D3. The axes are constructed using the same process used in the waterfall chart. The graph line can be drawn using the algorithm in Figure 9.5. The reference point for the graphics object is the top left corner of the screen. Hence, the normalised value is subtracted from the height, as in line 13 and 20 in Figure 9.5, to prevent the graph from being displayed upside down. The algorithm for drawing the threshold region is similar to the line drawing algorithm, but instead of drawing one line, points lying on the upper and lower limit of the threshold region are calculated, and a polygon is drawn using those points (Figure 9.7).

Since the polygon is closed, the first and last point of the polygon should be the same point. The polygon points are added by iterating through the line points clockwise. The first half of the array is filled from left to right, and corresponds to the upper line bordering the area; the second half is filled from right to left and corresponds to the lower line. A complete example, illustrating the spectrum line and threshold region, is shown in Figure 5.10 and Figure 5.11. The data used to draw the line in the spectrum graph corresponds to a row in the power data. Brushing and linking is used to determine which row to visualise. When a user hovers over the heatmap area, the mouse position is mapped to a row in the data and the function in Figure 9.5 is called with the row as a parameter.

![Figure 5.10: The spectrum graph (blue) with the threshold region drawn in the background (grey). Here, the threshold is symmetric.](image)
5.4 Evaluation

Before the second development iteration commenced, the result of the cycle was submitted to the client. We discussed the potential of the visualisations to answer visual queries, by comparing a visualisation to the queries from Section 3.1. Of the three visualisations, the waterfall chart and spectrum graph displayed the most useful information. The waterfall chart displayed a clear distinction between channels affected and unaffected by RFI, whereas the spectrum graph made it clear why data was flagged as RFI. The bar chart did not contribute significantly to the answering of visual queries, and the queries that it answered were not considered important. Instead of a bar chart, the client requested a different chart and the addition of interactions to the other charts that would assist in answering queries that are currently difficult to derive from the current visualisations. The functionalities that the client requested were:

1. The percentage of RFI occupying a channel over a time range should be displayed.
2. The waterfall chart should show when an over-range event occurred. An over-range affects an entire row, and thus the affected rows need to be distinguishable from safe rows.
3. An exact value at a point in the waterfall chart should be shown when a user requests it.
4. The ability to focus on a subset of the data or to change between different levels of detail.
5. The ability to view historical data or change time and frequency ranges.
6. Linking charts together: if a user interacts with one chart, the other charts should be updated to reflect the effects of the interaction.

The goals for the next development and design cycles were derived from the six requests made by the client.
6  Visualisation Design and Evaluation: Iteration 2

6.1 Overview

The second development cycle had two primary objectives: replace the bar chart with an occupancy plot and continue developing the previous visualisation by adding interaction techniques. Both objectives were derived from a discussion with the client. The bar chart did not generate particularly useful insights about the data. It was not discarded, but an option to switch between the bar chart and occupancy plot was added, with the occupancy plot being the default chart shown. The popups that were displayed upon hovering over the bars in the bar chart were instead set to appear whenever a user hovers over a shade in the colour scale. The client preferred to have a more robust and complete visualisation, rather than the development of a separate visualisation. Hence, the code base from the previous iteration was reused, and interaction techniques were added to improve the answering of visual queries and ease of access to information.

6.2 Visualisation Design

The charts in Iteration 1 visualised power data (waterfall chart, spectrum graph) or displayed a summary of the power data (bar chart). In contrast, an occupancy plot visualises detected RFI exclusively. It uses the same x-axis as the spectrum graph and waterfall chart (a frequency range), but has percentages on the y-axis. This is because the RFI occupancy is calculated for each channel. That is, for each channel (a column in the data) the percentage of points in that channel that were classified as RFI by the detection algorithm is shown. It thus displays which channels are the most frequently occupied by RFI (Figure 6.2). The benefit of this is reading exact occupancy values, and viewing a condensed view of RFI activity over a time range.

Only RFI was shown in the previous heatmap. To indicate over-range events, cells falling in an affected row are overlaid with over-range. A single hue is used, but the shade or tint by which a cell is overlaid depends on the power value. Darker shades are assigned to higher power values, while lighter shades are assigned to lower power values (Figure 6.1).

![Figure 6.1: The eight colour palettes for the heatmap area. Colour Brewer was used to generate the RGB values used for the palettes.](image-url)
Interaction techniques were adapted to improve the ability of the heatmap to answer visual queries:

1. Details-on-demand. The heatmap area only displays shades, while the axes provide a method of deducing time and frequency values of a cell. A popup is shown whenever a user hovers over the heatmap. The popup displays the exact power value, time, and frequency of the cell.

2. Filtering. Differences in hue are used to distinguish between RFI (red), over-range events (orange) and clean data (shades of grey). An interface can be added with options to filter RFI and over-range events in or out, depending on the visual query.

3. Zooming. An overview of the power data is given, while the user is allowed to increase the zoom level to display finer detail, or decrease the zoom level to return to a lower bin resolution with coarser aggregation applied to power values.

4. Linking and brushing. Brushing is used to pan within a zoom level, and a popup is shown whenever the cursor hovers over a cell in the heatmap. By linking the charts together, these interactions are reflected in the spectrum graph and occupancy plot.

5. Overview and Detail. While the user moves between different levels of detail or pans within a level, a context map can be displayed which indicates how the current level relates to the data as a whole, and thus providing contextual information. The brushes can also be used to give contextual information (see Section 2.2.4.5).

Together, these techniques compensate for limitations present in the first iteration of the visualisation. Examples of these techniques are illustrated in Section 2.2.4, and how they were implemented is detailed in Section 6.3.
Figure 6.2: Two waterfall chart mock-ups (top and middle) and the corresponding occupancy plot mock-up (bottom). The waterfall charts show detected RFI in red and over-ranges in orange. The occupancy chart displays the percentage of RFI present in each channel (a column in the waterfall charts).
6.3 Implementation Iteration 2

During the iteration, Two was used instead of P5 to render the heatmap area. Due to only a marginal performance gain, Two was shortly thereafter replaced by Pixi, which increased the responsiveness of the visualisation. The algorithm used to draw the heatmap area remained the same, but it was still necessary to redo a significant portion of rendering loop and visualisation initialisation to facilitate the new libraries.

6.3.1 Occupancy Plot

The plot does not display the occupancy for all of the data, but only for the points currently being rendered on the heatmap canvas. If a user interacts with the waterfall brushes or changes zoom levels, the occupancy plot is updated to reflect the occupancy for the new view. This means that each time a user interacts with the waterfall chart or changes the RFI threshold, the occupancy for each channel has to be recalculated. The algorithm used to calculate the occupancy is given in Figure 9.8. It iterates through the power data, which is supplied as a parameter, by going through the columns in the outer loop and through the rows in the inner loop. Each time a point in the column is flagged as RFI, the matching value in an array is incremented. After the inner loop completes, the occupancy is calculated by dividing the number of points flagged with the total number of points in the column. The result of the two loops are used to update the occupancy plot. The axes for the chart could be constructed with the same procedure used in the previous charts. A complete example of the plot is given in Figure 6.3 and Figure 6.4.
6.3.2 Zoom

The bin resolution of the aggregated power data CSV file determines the number of zoom levels accessible. The rows and columns of the CSV file are set equal to the highest bin resolution in the zoom sequence (see below). Three zoom levels were implemented for the waterfall chart, each differing from the adjacent level by a factor of two along the rows and two along the columns:

- zoom level 1 (142x100)
- zoom level 2 (284x200)
- zoom level 3 (568x400)

The first level is set as the default and can be displayed within the entire heatmap area. The heatmap area displays exactly $142 \cdot 100 = 14200$ points, irrespective of the zoom level. Thus for higher zoom levels with a greater number of points, only a subset of the points is displayed. The number of levels were limited to three because of the memory requirements of storing more levels.

After creation, the objects are added to a list (see the downsampling algorithm, Figure 9.3). The objects in the list are sorted by increasing order according to the object’s zoom level. A variable can thus be used to keep track of the currently viewed object’s zoom level. The variable is then incremented or decremented depending on whether the user zooms in or zooms out, respectively. An object with a matching zoom level is drawn from the list, and is set as the new visualised data. This update affects all charts in the visualisation, except the bar chart:

- The appropriate subsets of the data have to be passed to the charts.
- The axes of the waterfall chart, spectrum graph and occupancy plot have to be updated. For this, the extrema in the data subsets are needed.
When a user zooms, the mouse position in pixels is mapped to a coordinate pair relative to the waterfall chart area (Figure 6.5). All space above and to the left of the area are ignored. This coordinate pair can be mapped to a column and row index in the data (a relative position in data space). The mouse position forms a bounding box with the reference point of the canvas. The dimensions of the bounding box are used to calculate which subset of the data is displayed after the zoom event. The heatmap area (or viewing window into the data) contains only a subset of the columns and rows of the data (except when viewing the lowest zoom level). Hence, the top left coordinate \((x_1, y_1)\) is not necessarily equal to \((0,0)\), as there might be rows above and columns to the left. Several steps have to be performed before the viewing window is correctly positioned in the new data after a zoom interaction.

The first step is to take the width of the bounding box (the horizontal red arrows in Figure 6.7) and calculate the number of columns between the mouse position and the left edge of the box. The height of the bounding box (the vertical arrows in Figure 6.7) is used to calculate the number of columns between the top edge and mouse position. These offset values will be used to find the relative position in data space, and to align the viewing window.

![Figure 6.5: The blue rectangle indicates the bounding box that can be constructed from the bottom right corner formed by the mouse pointer and the top left corner (reference point) of the canvas. The relative position is used to determine which subset of the data is displayed after the user zooms.](image)

In the next step, the offset values are added to the number of rows and columns between the top left corner of the bounding box \((x_1, y_1)\) and the top left corner in data space \((0,0)\). The result is the relative position in data space of the current zoom level \((x_r, y_r)\). The relative position is then normalised to acquire the relative position in the new zoom level’s data space \((x'_1, y'_1)\). The viewing window is translated such that the top left corner is located at this position (Figure 6.8).
Lastly, the viewing window must be aligned such that the position to which the user zoomed is located at the same pixel coordinates in the new zoom level. To do this, the bounding box dimensions are subtracted from the four corners of the viewing window. This aligns the viewing window correctly and the effect of zooming is thus predictable to the user (Figure 6.9).

Since each lower zoom level is an aggregate of the higher levels, zooming in does not simply enlarge cells in the heatmap area. Rather than geometric zooming, each cell is effectively divided into four cells that represent the four greatest power values in the region they were sampled from (Figure 6.6). The final coordinates form two ranges: a column range \([x''_1, x''_2]\) and a row range \([y''_1, y''_2]\). The end-points of both ranges are used to extract a subset of the data stored in the JSON objects. In particular, the rows are used to determine the time range; the columns are used to determine the frequency range; both the columns and rows are used to determine the power data subset (since it is a 2D array). The maximum and minimum of these subsets are passed to the chart axes, which allows them to be rescaled to match the new data.

![Figure 6.6: From left to right, the effect of zooming in on the heatmap area (blue rectangle indicates the zoom focus region) Finer detail is revealed with consecutive zooms (or hidden when zooming out).](image)
Figure 6.7: Convert from pixel coordinates (right) to a column and row coordinate (left).

<table>
<thead>
<tr>
<th>Old zoom level and data</th>
<th>Viewing window</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td>2</td>
<td>(x_1, y_1)</td>
</tr>
<tr>
<td>3</td>
<td>(x_2, y_1)</td>
</tr>
<tr>
<td>4</td>
<td>(x_1, y_2)</td>
</tr>
<tr>
<td>5</td>
<td>(x_2, y_2)</td>
</tr>
<tr>
<td>6</td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Figure 6.8: Convert column and row pair in old data to a position in the new data. Translate the viewing window’s top left corner to that position.

<table>
<thead>
<tr>
<th>New zoom level and data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

Figure 6.9: The converted pixel units from Figure 6.7 are used to translate the viewing window. The viewing window is now aligned correctly over the new data.

<table>
<thead>
<tr>
<th>New zoom level and data, viewing window translated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>
6.3.3 Brush, Overview and Detail

A brush and context map was added to the waterfall chart to support the zoom interaction. Whenever a user changes the zoom level to a value greater than 1, only a small percentage of the total number of points in the data will be shown. A brush would allow the user to pan within the data, and thus change focus without losing context. The brush thus has a dual purpose:

- By linking the brush with chart data, a change in the brush position would be reflected in the charts, and hence enable panning.
- Since the brush size relative to the scroll area is the same as the viewing window size relative to the data it lies within, it provides contextual information.

The brush requires charts to be linked; panning within the data would change the focus of the viewing window, and thus the data of the occupancy plot and spectrum graph. There are two brushes added: one for panning with respect to time (right brush), and one for panning with respect to frequency values (bottom brush). Both brushes use the same procedure, but update different end ranges within the data. The technique is explained for the bottom brush, but holds equally for the right brush. A strategy similar to the one applied for zooming can be used, but instead of mapping the mouse position to data space, the brush offset ($\delta$ in Figure 6.10) is mapped to data space. The offset is used to translate the viewing window and update the remaining charts.

The bottom brush (Figure 6.10) is positioned within the range $[0, x_{\text{max}}]$. This range corresponds to column indices in the 2D power data array. The range $[x_1, x_2]$, which makes up the brush length, is fixed across panning interactions and is identical to the column range of the viewing window. All charts, except the bar chart, are set to respond to brush interactions. The two brushes only provide contextual information for one axis and zoom level at a time.

To show context for both simultaneously, a context map is added to the bottom right corner of the chart. The map only becomes visible when a user hovers over the brushes or within close proximity to them, to avoid occluding a significant portion of the heatmap area permanently. A shadow area is placed behind the map to provide a clear separation between the context map and the heatmap. The map is set to display the lowest zoom level at all times, in order to provide an overview of the data.

The relative position of the viewing window in the current zoom level is indicated by drawing a transparent rectangle over the map. The four corners of the rectangle are linked to the two brushes. If a user moves the bottom brush, the rectangle will move horizontally; if a user moves the right brush, the rectangle will move vertically. Together with the map, the moving rectangle enables the user to informatively pan through a region of interest.
Figure 6.10: The bounding box is extended to the full viewing window. The corner points are linked to the brush corner points. Moving the brush changes the brush end points, and thus the frequency range end points, which updates the charts.

Figure 6.11: The context map and transparent rectangle displaying the relative position of the viewing window (bottom). The heatmap area represented by the rectangle is shown above each map.
6.3.4 Details-on-Demand

For the heatmap area and colour scale, an event listener is attached that monitors the cursor position and determines which webpage elements are being hovered over. When the user hovers over a point in the heatmap area, the same procedure used in the zoom interaction is used to map the mouse position to a coordinate in data space. This mapping produces a column and row coordinate pair, which provides a method of accessing the 2D power data array, time array and frequency array. These arrays are accessed, and the three values are used to update the three entries in the popup whenever the cursor moves over the heatmap area (Figure 6.12). The popup is hidden as soon the cursor leaves the heatmap area, to avoid unnecessarily occluding the chart area.

![Figure 6.12: The popup displayed when hovering over the heatmap area.](image)

Similarly, the colour blocks in the colour scale can be mapped to an array containing the same number of objects as shades. Each object holds information about the shade:

- The percentage of points that were binned to the shade.
- The minimum and maximum power value binned to the shade.

A popup is displayed whenever a user hovers over a shade in the scale (Figure 6.13), and the matching object in the array is retrieved and used to update the popup entries. Together, these interactions allow precise information to be displayed upon request.

![Figure 6.13: The popup displayed when hovering over a shade in the colour scale.](image)
6.3.5 Profiling

When a user performs an interaction, a sequence of function calls are executed that update the representation and data object. The nontrivial functions and code blocks were logged to ensure the response time of interactions is below a threshold of 100 milliseconds. JavaScript’s High Resolution Time API \cite{65} was used to calculate time differences between events. The execution time of the zoom and brushing mechanism were logged for three runs, and each run executed an interaction 60 times. A summary of the results is given below:

- All of the logged interactions were below the threshold.
- The worst possible run for the zoom interaction was 10.23398 ms, while the worst run was 9.35498 ms.
- The worst possible run for the brush interaction was 7.05500 ms, while the worst run was 8.81799 ms.

A more comprehensive event log can be found in the appendix, Section 9.2. A bar chart of the tabulated results in the appendix is given below (Figure 6.14). The charts display the sum of the tables in the appendix.

![Zoom Response time](image1)

![Brush Response time](image2)

*Figure 6.14: Two bar charts that display the zoom and brush interaction response times.*
6.4 Evaluation

The second development cycle resulted in a visualisation with four charts and five different interaction techniques. These components were evaluated by the client and five domain experts from the University of Cape Town’s Astronomy Department. A walkthrough of the visualisation was given to the client. We discussed the capacity of the visualisation to function as a RFI monitoring tool, and determined whether visual queries are answerable. In addition, an evaluation session with the experts was hosted using the facilities provided by the Computer Science Department of the University of Cape Town. Each expert was given a machine to test the visualisation, and an evaluation form with four questions. While they were using the tool, the features of the visualisation were explained. Furthermore, they were given the opportunity to complete the form and give verbal feedback about the visualisation and interaction techniques. The questions that were asked are:

1. Which insights about the data did the visualisation best support?
2. Which insights were not adequately supported or what features were missing?
3. Were the interaction techniques useful? What did you think worked best?
4. Were any of the interactions a hindrance? Was anything missing, and what interactions would you add?

For each question, the experts had to consider visual queries. A list of the visual queries are given below, together with the feedback from the client and the experts. In the list, several strengths and weaknesses of the visualisation are revealed. Further suggestions made by the client and experts are given thereafter.

6.4.1.1 Visual Queries

1. At what time and in which frequency channel was RFI present?
   The spectrum graph, occupancy plot, and waterfall chart clearly display the channels affected by RFI. Only the waterfall chart displays RFI instances over a time range, whereas the other spectrum graph displays RFI for a time instance (row in the data) and the occupancy plot gives a summary for a time range. The spectrum graph lacks the ability to view data over a time range (column in the data).

2. What is the power level of a detected RFI signal?
   Only the waterfall chart displays an exact power value with a popup. The spectrum graph allows a user to estimate a power value, but not a precise value. In addition to the techniques used in the current spectrum graph, the graph should make use of grid lines or popups to allow a user to read a value with higher accuracy.

3. What are the extrema of the data?
   The colour scale in the visualisation allows a user to hover over a shade, and to read the maximum and minimum value binned to the shade. However, these values are global. The extrema of the data shown in the viewing window of the waterfall chart could also be given.

4. How does the RFI detection algorithm work?
The visualisation has no information to help the user understand how the data was processed before being visualised. Such information should be added so that the users are able to trust the processes that were applied to the raw data.

5. How much of a band is affected by RFI?
The columns in the waterfall chart give a good overview of the number of RFI occurrences, and the occupancy plot provides percentage values summarising RFI occurrences for a channel. The spectrum graph only shows how much of the band is affected for a time instance.

6. What percentage of time on each channel is RFI present?
After the first iteration, the client requested an occupancy plot for this purpose. They found the plot easy to interpret and understand.

7. Which data points are reliable?
Over-range events are clearly visible and easily separable from the rest of the data. However, the aggregation algorithm in the filter ignores over-range events: a power value affected by an over-range might be passed through to the CSV file. This should not happen.

6.4.1.2 Expert and Client Feedback

Only three zoom levels are available. The client would prefer the ability to increase the zoom up to the same level as the raw data. While the brushes do allow panning, a more intuitive way could be used, such as dragging the chart area with the mouse cursor. This would also enable diagonal panning through the data. The experts suggested that the brushes be redesigned, as the brushes might be confused with the colour scales that are often placed adjacent to waterfall charts (for an example, see the waterfall charts in Section 2.1).

RFI is detected within the client and the interface allows the thresholding algorithm to be adjusted dynamically. However, because three statistics files are calculated for each zoom level, the output of the detection depends on the zoom level. The detection should instead be consistent for all zoom levels.

There is no help or information available for first time users. Both the experts and client suggested that an additional web page be made that guides first time users through the visualisations and interactions. During the evaluation experiment, none of the participants was aware of the brushing and zooming interactions until it was explicitly pointed out to them through a demonstration.

Four of the five participants expressed satisfaction with the visualisation components. In particular, they enjoyed the ability to see affected channels, to see how significant the detected RFI are, to configure the detection algorithm and enjoyed the responsiveness of interactions. All participants indicated that there are still features that they would like added. The collection of suggested features are listed below:

- Choice of axes for the charts
- The ability to manually flag data as RFI within the visualisation
- The ability to iterate through multiple data sets, and make comparisons between data sets
- An indication of the percentage of cells affected by over-ranges
- An arbitrary choice of colours, and not just grey shades
- The addition of histograms
- The ability to manually specify axis and bin ranges
- Binning could be according to time and frequency, not just power values
- An auto-hiding control interface, and buttons in the main layout
- An event log window
- Crosshairs on the waterfall chart
- Save the charts as images
- Save selections
- An interactive context map

With the currently implemented features, the visualisation could be used to monitor RFI. Although, minor adjustments would first have to be made to ensure the visualisation can be integrated with an existing system, and further testing should first be performed to determine whether an accurate view of the raw data is given. There are still limitations to the software, which were revealed during the evaluation. The suggestions made by the experts can be used to extend the usability of the visualisation. However, due to time constraints, these features would have to be left for future work. A concept design derived from a select few of the suggestions is presented in Section 7.3.
7 Results: Final Design

The culmination of the design and development iterations is a browser-based visualisation that is able maintain real-time interactivity for large RFI data sets. We present an overview of the design implemented in Section 7.1 and Section 7.2, while an extended design based on the expert feedback is presented in Section 7.3.

7.1 Overview

To view the visualisation, the website containing the visualisation should be hosted on a web server that supports PHP (to serve data requests), and it should be accessed from a web browser with JavaScript and WebGL enabled. The complete web visualisation, as is displayed immediately after loading the webpage, is shown in Figure 7.1. All of the components are embedded within a single webpage and can only be accessed by visiting the website. In total, nine components were developed:

- A GUI for configuring the visualisations (1)
- A linked waterfall chart (2)
- Context brushes (3)
- A context map (hidden)
- Mouse-over popups (hidden)
- A linked spectrum graph (4)
- A colour scale (5)
- A linked occupancy plot (6)
- A configurable bar chart (hidden)

![Figure 7.1: The default view which the user sees immediately after loading the webpage. The visible components are annotated 1-6, and match with the description in the bullet list.](image)

The waterfall chart is the main view of the data, while the other charts function as supportive views of the same data set. Only one HDF5 file is visualised during a visualisation session. Additionally, any over-range events during which the visualised data was collected is shown on the waterfall chart. A walkthrough of all the components is given in the following section.
7.2 Visualisation Walkthrough

Figure 7.2: The default visualisation settings. The left figure shows the hidden interface, which is shown by default; the right figure shows the expanded interface.

Certain components of the visualisation are configurable through a GUI:

- Waterfall overlays can be toggled by clicking the appropriate check boxes. Only the waterfall chart is affected by this change (Figure 9.9).
- The sensitivity of the thresholding algorithm can be set by dragging the threshold range. The waterfall chart, occupancy plot, and spectrum graph are affected by this change. (Figure 9.10).
- The number of shades used to bin the power values can be changed by dragging the appropriate slider. This affects the waterfall chart, bar chart, and colour scale (Figure 9.11).
- A bar chart can be shown instead of the occupancy plot by clicking the drop-down menu next to the Plot type option (Figure 9.11).

After the user has entered the desired settings, the interface can be hidden by closing the controls. The effects of changing parameters are illustrated for each component from Figure 9.9 to Figure 9.11 in the Appendix. The remaining features only become apparent after mouse or touchpad interactions are performed by the user:

- Hovering over the colour scale activates a popup displaying detailed information about power values binned to that shade (Figure 6.13).
- Hovering over the waterfall chart activates a popup that is displayed close to the cursor, which gives detailed information about that cell in the waterfall chart (Figure 6.12). Additionally, the spectrum graph is updated to show the row in the waterfall chart being hovered over (Figure 9.12).
- Dragging a brush pans through the data. A context map is shown in the bottom right corner to help guide the user (Figure 9.13).
- Scrolling while hovering over the waterfall chart changes the zoom level, which allows different levels of detail to be displayed (Figure 9.14).
7.3 Extended Design

The most notable limitation of the final implementation is that a visualisation session is restricted to one file. Additionally, it is not possible to know which file to inspect without first starting a new session and then viewing the data thereafter. This could be time-consuming if a user wished to view more than one data set. To compensate for the restriction, the previous design is extended to include support for multiple data sets, which allows the user to iterate through data sets seamlessly. It should be noted that this design requires a data set with RFI sources explicitly labelled, which is currently not available.

A bump chart is added below the main charts that uses bands to encode percentages of RFI in the data. Each band represents a source of RFI, while the width of a band encodes the percentage of points affected by that particular RFI source in the data. The bands can be ranked according to their width, and this ranking can be encoded by adjusting the position of the bands. Consequently, RFI sources with the highest occupancy would be positioned above sources with lower occupancies.

The horizontal axis is used as a timeline and files are arranged according to the time range during which power values were observed. By selecting a file, a new data set is loaded within the same session. Lastly, we add filtering techniques to the chart: a user can select the RFI source to be displayed in the main chart (Figure 7.3).

Figure 7.3: The bottom of the chart displays a bump chart. Currently, file 3 is selected. There are three bands, one for each RFI source. It is clear that the first source is the dominant RFI detected in the observed time range, while the second and third sources frequently change ranking. The user has filtered the chart to show the second source only.
7.4 Discussion

While the extended design does facilitate multiple data sets, it is difficult to create a visualisation that would accommodate all the suggestions of the expert in a single view. To prevent the layout from becoming cluttered with charts and features, the visualisation could be portioned into common use case scenarios. Since different users likely prefer seeing different views of the data, or would only focus on a subset of visual queries, having all the functionalities available within the same display would waste screen space. The ability to create custom layouts for use case scenarios could significantly condense the visualisation, and make the components of the visualisation more relevant to the task. This is not yet needed in the visualisation, but would become worth considering if the visualisation is extended. Furthermore, one of the experts suggested that the software should be generalised to work with use case scenarios for other sites and not only MeerKAT. This would require the implementation to be more data agnostic, as the current visualisation expects a strict data format.
8 Conclusions and Future Work

This project developed an alternative visualisation framework to assist the RFI management team in making sense of the large amounts of data at the MeerKAT site. A survey of visualisations applicable to time series and large data sets was presented, together with interaction techniques. By drawing design prototypes from the survey and by collaborating with the client, we decided upon a visualisation design to be developed. The back-end of the visualisation framework was developed during a preliminary development phase, while the visualisation designs were implemented during two development iterations.

Our contribution is a web visualisation that utilises technologies such as D3 and WebGL (accessed through Pixi). The main component is a waterfall chart that displays large data sets compactly, and one file is visualised during a browser session. The compact display of the waterfall chart foregoes high detail in order to show an overview of the data. To compensate for the loss of detail, we added two supportive charts that are linked to the waterfall chart. The supportive charts display the same data, but with increased detail. The waterfall chart can maintain interactive brushing and zooming to allow users to query the data set rapidly, while the supportive charts are updated dynamically as the user interacts with the chart and the adjacent brushes. A context map provides access to contextual information while the user pans through the data.

Experts found the visualisation useful for answering visual queries, such as seeing the channels affected by RFI and viewing why certain points were flagged as RFI. They suggested that the prototype be further developed before it could be deployed in a real-world context. The most useful features were the responsiveness of the visualisation, the linked charts and interactions, and the configurable detection algorithm. The visualisation was good at showing an overview of the data and providing more detail through supportive visualisations and interactions. However, there are still limitations in the software that were not addressed:

- Only one file is visualised, and comparisons between data sets is not possible.
- Users do not have access to live data and real-time events.
- An arbitrary bin resolution cannot be user specified – only by changing parameters in the filter can new resolutions be generated.

The first limitation is addressed by the extended design, but has not yet been developed. The second and third limitation require a more robust back-end implementation and some changes to the front-end. These features could be considered in a future work, and a complete framework should include an improved detection algorithm and data filter as part of the back-end. We suggest that the limitations be addressed before the software is deployed for real-world usage.
9 Appendix

9.1 Code Snippets

```javascript
var dataObject = {
    isActive: true, // is this object the one being visualised?
    zoomLevel: 1, // the minimum zoom level ("zoomed out")
    numberOfRows: 2,
    numberOfColumns: 2,
    minFrequency: 250,
    maxFrequency: 650,
    minPower: 4,
    maxPower: 9,
    frequency: [250, 650],
    power: [
        [4, 9],
        [5, 8]
    ],
    time: [00:00:00, 00:30:00],
    overranges: [true, false], // an over-range only occurred at 00:00:00
    stats: [
        {
            mean: 4.5,
            std: 0.5 // population standard deviation
        },
        {
            mean: 8.5,
            std: 0.5 // population standard deviation
        }
    ]
};
```

Figure 9.1: An example JSON data object constructed from the three CSV files in Figure 4.2.

```javascript
function normalise(value, oldMin, oldMax, newMin, newMax){
    var rangeOld = oldMax - oldMin;
    var rangeNew = newMax - newMin;
    var ratio = (value - oldMin) / rangeOld;
    var normValue = (ratio * rangeNew) + newMin;
    return normValue;
}
```

Figure 9.2: The algorithm for normalising data.
```javascript
1. var zoomLevels = (maxZoom - minZoom) + 1, // = (3 - 1) + 1 = 3
2. dataset = new Array(zoomLevels),
3. statArray = new Array(zoomLevels);
4.
5. // zoom factor is hard-coded as 2, but can easily be generalised
6. function downsample (minZoom, maxZoom) {
7.
8.   var zoom = maxZoom,
9.   data = getMaxBinResolutionData(); // 568x400
10.}
11. while( zoom-- > minZoom){
12.   var rows = data.rows/2,
13.   columns = data.columns/2,
14.   frequencies = getHalf(data.frequency),
15.   time = getHalf(data.time),
16.   overrange = getORedHalf(data.overrange),
17.   power = sampleData(data.power, rows, columns), // see Section 3
18.   zoomLevel = zoom;
19.   stats = statArray[zoom - minZoom];
20.
21.   data = makeNewObjectWithDownsampledData();
22.   dataset.push(data);
23. }
24.}
25. // the input array is a float array with length divisible by 2
26. function getHalf(array){
27.   var half = new Array(array.length/2);
28.   for(var i = 0; i < array.length; i += 2){
29.     half[i/2] = array[i];
30.   }
31.   return half;
32.}
33. // the input array is a Boolean array with length divisible by 2
34. function getORedHalf(array){
35.   var half = new Array(array.length/2);
36.   for(var i = 0; i < array.length; i += 2){
37.     half[i/2] = array[i] || array[i+1];
38.   }
39.   return half;
40.}
```

Figure 9.3: The algorithm used to downsample the data from a bin resolution of 568x400. The loop
iterates three times in total.
1. // clear the heatmap area before the next render loop
2. graphics.clear();
3. 
4. for(var r = 0; r < rows; ++r){
5.     for(var c = 0; c < columns; ++c){
6.         // (x, y) coordinates of the top left corner of the rectangle
7.         var y = r * rectangleHeight,
8.             x = c * rectangleWidth;
9.         var point = data[r][c];
10.        var hex = palette[point.normValue];
11.        // use a graphics object to create and render the rectangle
12.        graphics.beginFill(hex);
13.        graphics.drawRect(x, y, rectangleWidth, rectangleHeight);
14.        graphics.endFill();
15.     }
16. }

Figure 9.4: The tiling algorithm, which draws rectangles from left to right and top to bottom starting from the top left corner. Shade values are retrieved as described in Figure 5.6.

1. // draws a line for the spectrum graph
2. function drawSpectrum (data, rowIndex, width, height, min, max) {
3.     var row = data[rowIndex],
4.         columns = data.numberOfColumns;
5.     var increment = width/columns;
6.     // clear the drawing area
7.     graphics.clear();
8.     var x1 = 0,
9.         y1 = height - normalise(row[0].value, min, max, 0, height);
10.    for(var c = 1; c < columns; c++){
11.        var point = row[c];
12.        var x2 = increment,
13.            y2 = height - normalise(point.value, min, max, 0, height);
14.        graphics.drawLine(x1, y1, x2, y2);
15.        x1 = x2;
16.        y1 = y2;
17.    }
18. }

Figure 9.5: A line drawing algorithm, which can be used to create the spectrum graph.
1. // returns a configuration for a bar chart
2. function binPoints (data, numberOfShades, globalMin, globalMax) {
3.  var rows = data.rows,
4.      columns = data.columns;
5. // a property holding the configuration for a bar chart
6. // each array has length numberOfShades
7.  var binProp = {
8.      min: [],
9.      max: [],
10.     supremum: [],
11.     infimum: [],
12.     sum: [],
13.      percent: []
14.  }
15. // bin the data points to a specified palette
16.  for (var r = 0; r < rows; r++) {
17.    for (var c = 0; c < columns; c++) {
18.      var point = data[r][c];
19.      var normValue = normalise(
20.        point.value,
21.        globalMin, globalMax + 0.001, // old range, in data space
22.        0, numberOfShades // new range, palette indices
23.      );
24.      var index = Math.floor(normValue);
25.      binProp.min[index] = Math.min(min[index], point.value);
26.      binProp.max[index] = Math.max(max[index], point.value);
27.      binProp.sum[index]++;
28.    }
29.  }
30. // calculate percentages, supremum and infimum
31.  var increment = (globalMax-globalMin)/numberOfShades;
32.  for(var i = 0; i < numberOfShades; ++i){
33.    binProp.percent[i] = sum[i]/(rows*columns);
34.    binProp.supremum[i] = globalMin + i*increment;
35.    binProp.infimum[i] = binProp.supremum[i] + increment;
36.  }
37.  return binProp;
38. }
39. 
40. Figure 9.6: The algorithm for binning points. The output is used to configure the bar chart. Each palette has such a bar chart configuration, and the algorithm is executed for each palette.
The function draws a threshold area for the spectrum graph.

Here, data corresponds to a row in the 2D power data array.

```javascript
function drawArea(data, threshold, rowIndex, width, height, min, max) {
  var row = data[rowIndex],
      columns = data.numberOfColumns;
  var increment = width/columns,
      secondLast = 2*columns - 1,
      x = 0;
  var polyPoints = new Array(2*columns + 1);
  for(var c = 0; c < columns; c++){
    var point = row[c];
    // std = population standard deviataion
    var yTop = point.mean + threshold*point.std,
    yBottom = point.mean - threshold*point.std;
    var yTopNorm = height - normalise(yTop, min, max, 0, height);
    yBottomNorm = height - normalise(yBottom, min, max, 0, height);
    // if the threshold is not symmetric:
    // yBottomNorm = height;
    // add top line points from left to right
    polyPoints[c] = new Point(x, yTopNorm);
    // add bottom line points from right to left
    polyPoints[secondLast - c] = new Point(x, yBottomNorm);
    x += increment;
  }
  // close the polygon
  polyPoints[secondLast + 1] = polyPoints[0];
  // clear the drawing area and draw area
  graphics.clear();
  graphics.beginFill("#d7d7d7");
  graphics.polygon(polyPoints);
  graphics.endFill();
}
```

Figure 9.7: An algorithm for drawing a closed polygon, which can be used to draw the threshold region.
1. // takes power data as input and calculates the percentage of RFI in each column
2. function getPercentRFI(data) {
3.   var rfi = new Array(data.columns);
4.   for(var j = 0; j < data.columns; ++j){
5.     rfi[j] = 0;
6.   }
7.   for(var i = 0; i < data.rows; ++i){
8.     var point = data[i][j];
9.     var rfiDetected = detectRfi(point); // detect using thresholding
10.    if(rfiDetected){
11.      rfi[j]++;
12.    }
13.   }
14.   rfi[j] = rfi[j]/columns;
15. }
16. return rfi;
17. }

Figure 9.8: The algorithm for calculating RFI occupancy for the frequency channels (or columns in the 2D power data array).
9.2 Benchmarks

9.2.1 Zoom

Each run consists of 60 updates, and a single update consists of a sequence of function calls. All values are in milliseconds.

<table>
<thead>
<tr>
<th>Update heatmap area and calculate RFI</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0.00599</td>
<td>0.00600</td>
<td>0.00599</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.04300</td>
<td>0.02199</td>
<td>0.02199</td>
</tr>
<tr>
<td>Mean</td>
<td>0.01038</td>
<td>0.00975</td>
<td>0.00923</td>
</tr>
<tr>
<td>Total</td>
<td>0.62299</td>
<td>0.58500</td>
<td>0.55399</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update brush size</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0.70300</td>
<td>0.70599</td>
<td>0.70299</td>
</tr>
<tr>
<td>Maximum</td>
<td>1.10500</td>
<td>2.46299</td>
<td>1.80100</td>
</tr>
<tr>
<td>Mean</td>
<td>0.79906</td>
<td>0.86584</td>
<td>0.83898</td>
</tr>
<tr>
<td>Total</td>
<td>47.94399</td>
<td>51.95099</td>
<td>50.33899</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update line chart</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>2.08800</td>
<td>2.06699</td>
<td>2.03299</td>
</tr>
<tr>
<td>Maximum</td>
<td>4.75799</td>
<td>4.14400</td>
<td>4.24100</td>
</tr>
<tr>
<td>Mean</td>
<td>2.75166</td>
<td>2.88033</td>
<td>2.7269</td>
</tr>
<tr>
<td>Total</td>
<td>165.09999</td>
<td>172.81999</td>
<td>163.61499</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update occupancy plot</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>1.27000</td>
<td>1.27999</td>
<td>1.24800</td>
</tr>
<tr>
<td>Maximum</td>
<td>2.18099</td>
<td>2.72600</td>
<td>2.97000</td>
</tr>
<tr>
<td>Mean</td>
<td>1.55108</td>
<td>1.60209</td>
<td>1.60176</td>
</tr>
<tr>
<td>Total</td>
<td>93.06500</td>
<td>96.12599</td>
<td>96.10600</td>
</tr>
</tbody>
</table>

The most time consuming run is calculated by taking the maximum of the three runs for each of the four functions, and adding them together. This gives:

$$0.04300 + 2.46299 + 4.75799 + 2.97000 = 10.23398 \text{ ms}$$
9.2.2 Brushing

Each run consists of 60 updates, and a single update consists of a sequence of function calls. The heatmap area update, RFI calculation, and occupancy plot function sequences are part of the brush update function sequence.

<table>
<thead>
<tr>
<th>Update heatmap area and calculate RFI</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0.00799</td>
<td>0.00699</td>
<td>0.00899</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.02399</td>
<td>0.02500</td>
<td>0.02500</td>
</tr>
<tr>
<td>Mean</td>
<td>0.01095</td>
<td>0.01147</td>
<td>0.01081</td>
</tr>
<tr>
<td>Total</td>
<td>0.48200</td>
<td>0.38999</td>
<td>0.29199</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update occupancy plot</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>0.62900</td>
<td>0.61699</td>
<td>0.63200</td>
</tr>
<tr>
<td>Maximum</td>
<td>1.51199</td>
<td>1.52099</td>
<td>1.73799</td>
</tr>
<tr>
<td>Mean</td>
<td>0.79202</td>
<td>0.74623</td>
<td>0.82796</td>
</tr>
<tr>
<td>Total</td>
<td>34.84899</td>
<td>25.3720</td>
<td>22.35499</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Update brush</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>3.12399</td>
<td>3.25199</td>
<td>3.30399</td>
</tr>
<tr>
<td>Maximum</td>
<td>6.06200</td>
<td>6.02800</td>
<td>7.05500</td>
</tr>
<tr>
<td>Mean</td>
<td>4.27165</td>
<td>4.06370</td>
<td>4.85833</td>
</tr>
<tr>
<td>Total</td>
<td>187.95299</td>
<td>38.16599</td>
<td>131.17499</td>
</tr>
</tbody>
</table>

The most time consuming run is calculated by taking the maximum of the three runs of the brush function. This gives:

7.05500 ms
9.3 Visualisation Walkthrough

Figure 9.9: The waterfall chart with two overlays set as active. The top chart shows RFI (red), while the middle chart shows over-ranges (orange). Both are shown at the same time in the bottom chart. Only the waterfall chart responds to this change.
Figure 9.10: Three different threshold settings and the resulting visualisations. The waterfall chart, occupancy plot and spectrum graph are affected by this change.
Figure 9.11: Three different bin settings: 3, 6, and 9 shades. The bin settings affect the waterfall chart, bar chart and colour scale. To view the bar chart instead of the occupancy plot, the bar chart must be set as active through the interface.
Figure 9.12: The spectrum graph (top) after the user hovers a row in the waterfall chart (bottom). The spectrum graph shows the areas where the spectrum falls outside or is close to the border of the threshold region, which causes the cells to be flagged as RFI.

Figure 9.13: Panning from affects all charts except the bar chart, as can be seen from the two figures. The context map guides the user through the zoom level.
Figure 9.14: Three different zoom levels. All charts except the bar chart respond to zoom events. From top to bottom, the zoom levels are one, two, and three. The user is zooming in to the top left corner.
10 References


