Visualisation of Animal Tracking Data

Dissertation Document

James Walker
521866@swan.ac.uk

May 2011

Abstract

The study of animal behaviour is a key area of biological research. Studying the deep underwater behaviour of marine wildlife pose special challenges. Advances in technology make it possible to attach tri-axial accelerometer recording devices to animals which remotely record movement. Biologists at Swansea University use these devices for collecting data on animal behaviour. Currently this data is plotted on large multiple two dimensional time series plots. By looking for signature patterns and combining plots together this can be mapped to an animal activity. However, manual inspection of graphs is very time consuming with no automatic way to select areas of interest and is commonly subject to errors. Advanced visualisations have been created which assist in and enhance data analysis including the automatic extraction and visualisation of animal behavioural patterns. Data is visualised in a more meaningful and intuitive way to biologists. Visualisations created include: spherical scatter plots, multidimensional histograms and animal posture diagrams among other advanced visualisation techniques which enhance the understanding of marine wildlife behaviour.

Project Dissertation document submitted to the University of Wales, Swansea
in Fulfilment for the Degree of Bachelor of Science

Swansea University
Prifysgol Abertawe

Department of Computer Science
University of Wales Swansea

Contents

1 Introduction

Animal behaviour in the wild is a key area of biological research. Technological advances have made possible the attachment of miniature devices to animals which record their movement and location environment without direct human surveillance. Animals that previously could not be observed in the wild due to the presence of humans or the environment, especially underwater are now indirectly observable [15]. Biologists at Swansea University are at the forefront of this research technique, using tri-axial accelerometer recording devices along with other measuring instruments. These devices are called Daily Diaries (Can be seen in Figure 1) and as the technology matures they will shrink in size, enable more measurements, as well as become cheaper and more prevalent.

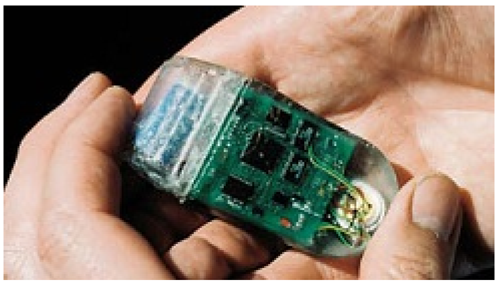


Figure 1: A Daily Diary device as used by the Biology Department at Swansea University. [2]

Daily Diaries record accelerometer data in 3 axes along with environmental data such as temperature and pressure. The use of accelerometers to investigate animal behaviour is currently very low [5]. Most biologists are using Radio Frequency Identification Devices (RFID) or Global Positioning System (GPS) to track animals. However, these only track the geographical positions of animals and not behaviour, also these devices do not work underwater. The Daily Diaries record actual animal movement over a long period of time, potentially days. Movement is recorded as an acceleration value which can be split into orientation and direction components. Orientation is specified in the form of the pitch and roll of an animal. All this data can be recorded in a variety of different local environments and is interpreted by biologists to identify an animals activity.

Daily Diary devices weigh 42g in air, 12g in water and have a maximum dimension of 55x30x15mm [5], approximately the same size as a match box. This makes the devices ideal to be used on a variety of different animals. Biologists at Swansea University have collected large amounts of data from: Badgers, Turtles, Sharks, Penguins, Cheetah's and Llamas [5]. This list is by no means exhaustive. The Daily Diaries can be attached to on land animals as well as animals that fly or swim [5]. Attaching a device to an animal is usually achieved by placing a collar around the animal to hold a device firmly in place. Figure 2 shows a device attached to a Penguin.

This is a new technology and is in its infancy, once the devices become more founded they will open up many more areas for research. For instance, collected data could be applied to conservation issues. If this data were combined with GPS data then it should be possible to determine not just an animal's territory but its use i.e. for hunting or breeding. Ultimately, this could enable a habitat to be conserved according to its importance to the animal [5]. The possibilities of this technology really are endless and it is a very exciting research area to be involved in.

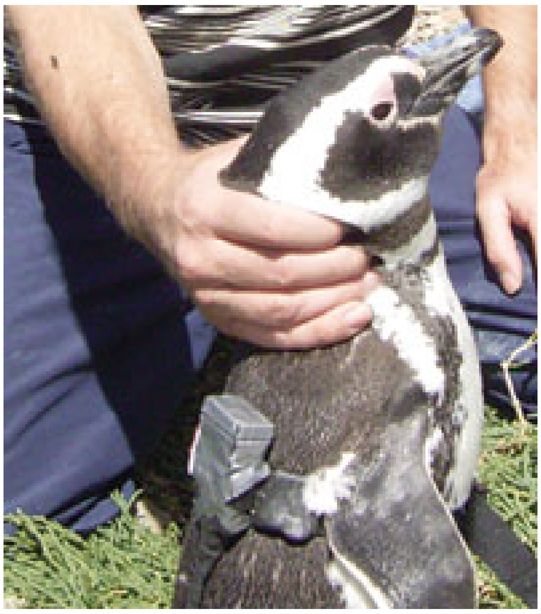


Figure 2: A Daily Diary device attached by a collar around the body of a Penguin [15].

Collected data is currently displayed on two dimensional intensity verses time graphs using commercial graphing software, allowing analysis by biologists. Animal behaviours are determined by looking for signature patterns in the graph plots. Usually biologists have to align up to 5 graphs and analyse the data attributes in parallel [4]. This is challenging to the biologists as it is a lot of information to interpret with a typical data set consisting of over 1 million entries, with no automatic way to select areas of interest from the graphs this can consume a lot of time and produce a significant number of errors [4]. Clearly a lot of information has to be processed by the biologist during data analysis. Currently, large amounts of research time as well as the skill of the biologist is absorbed in interpreting collected data.

It is vital for the biologists to have a better method of visualising this data, trawling through graphs for days on end is clearly not acceptable or by any means feasible. Biologists should be able to retrieve knowledge from the data produced by Daily Diary devices with ease. The aim of this project has been to address these problems and produce a solution for the biologists. However, this is by no means simple and posed several challenges to the project. One of the biggest challenges being the size of the data sets. Visualising millions of data items and still being able to extract knowledge is challenging.

This project has created a tool which produces three dimensional visualisations which enable biologists to identify animal behaviour. Specifically, we have looked at creating visualisations for Accelerometer and Compass data. Visualisation techniques produced make data analysis easier, less error prone and less reliant on the skill of the biologists. Our methods are beneficial to biologists as it is now easier to understand and extract knowledge from collected data. Visualisations created also enable knowledge to be extracted which was previously not possible, such as, finding patterns that exist in the data, for example, common animal orientations. Our visualisation product also includes useful interaction techniques, such as, being able to play back through a data set in real time. This adds clarity to visualisations, enabling the biologists to watch visualisations grow and see the progression between time steps. These visualisation techniques will save biologists a huge amount of time and allow them to gain a greater insight into collected data than previously possible.

The dissertation is split up into the following sections:

- Section 2. Presents previous work on this topic and related research material.
- Section 3. Outlines the project specification including the requirements of the project and the technology used throughout the project implementation.
- Section 4: Demonstrates designs for class structures and the Graphical User Interface.
- Section 5. Introduces the project plan and timetable, consisting of: a risk analysis, the software development model used and the time plan outlined at the start of the project.
- Section 6: Presents our software product, including what has been implemented and how.
- Section 7: Outlines the testing process applied to visualisations and software. This is to safeguard visualisations created are correct and the application produced is reliable.
- Section 8: Summarises the presented work and wraps the project up.
- Section 9: Discusses future work for the project and possibilities for more visualisation research in this field.

2 Background

2.1 Related Work

2.1.1 Visualisation of Sensor Data from Animal Movement [4] - Edward Grundy, Mark W. Jones, Robert S. Laramee, Rory P. Wilson and Emily L.C. Shepard.

The paper 'Visualisation of Sensor Data from Animal Movement' covers very similar ground to the research aims and goals of our project. This paper provides an excellent starting point, not only to discover information about the monitoring devices attached to the animals but also about already implemented visualisations of collected data. Research work presented was produced in collaboration with the biology department at Swansea University with an aim to create visualisations that would assist in finding animal behavioural patterns.

Monitoring devices are the preferred method of monitoring animal behaviour in the wild as opposed to human observation. 'Animals may behave differently in the presence of humans, they generally cannot be observed all the time (and sometimes hardly observed at all) and because, even if the animal is visible, it is often difficult to quantify behaviour, particularly with respect to intensity' [4]. These reasons mean animal monitoring devices are very important to biologists in this research area.

The devices used by the biologists at Swansea University are called Daily Diaries or otherwise known as tri-axial accelerometer devices. They are very small, light weight (Figure 3 shows a device attached to a Leatherback turtle) and can be used over long periods of time, making them ideal for monitoring animal behaviour. Daily Diaries record acceleration (g) and magnetic field strength (mV) in 3 axes along with hydrostatic pressure (mBar), light intensity (mV) and temperature (°C). Collecting this data makes it possible to determine the movements and location environment of the animal it is attached to. Accelerometers are positioned to the dorso-ventral axis (heave), anterior posterior axis (surge) and the lateral axis (sway). 'These axes are analogous to the X, Y and Z axis in Cartesian co-ordinates' [4].

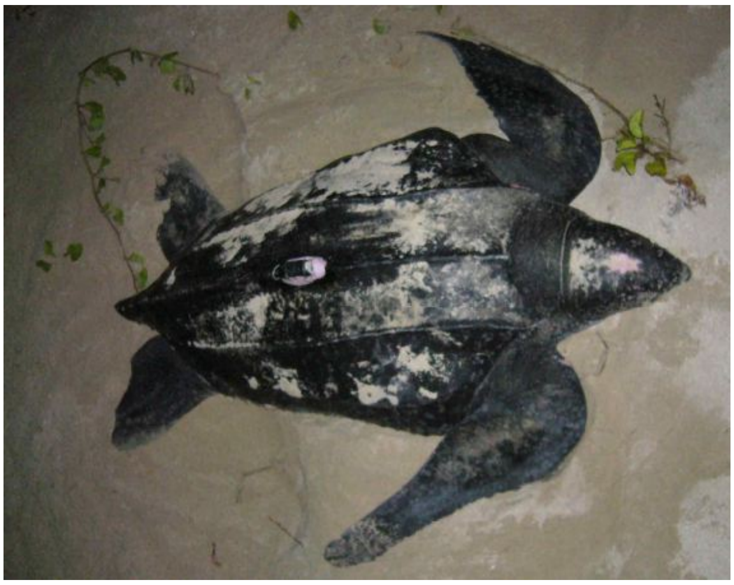


Figure 3: A Leatherback Turtle with a Daily Diary device attached [4].

Data is collected at a relatively low frequency over a long period of time. Frequency depends on the animal as 'the recording frequency has to be high enough to provide at least 5 data points per repetitive behaviour cycle (e.g. per wing-beat or stride)' [4]. Acceleration data consists of two components; static acceleration and dynamic acceleration. Static acceleration gives the orientation of the animal and is 'derived from animal posture with respect to the gravitational field' [4]. The dynamic component gives the acceleration vector and indicates the animals movement.

The paper provides an example of data recorded from a device attached to an Imperial Cormorant. 'Measurements were taken at 8Hz for 8 hours and 40 minutes resulting in 249,988 measurements. Data is recorded at 22-bit resolution onto 128mb flash memory'. 'The accelerometer has an accuracy of +- 0.06g with a sensitivity of range 2.21g' [4].

Currently acceleration data collected is represented in three 2D time-series plots (intensity against time) with each recording having its own graph (Figure 4 shows an example scenario). In addition to this, environmental data is often shown alongside the acceleration plots to assist in data analyses. Plots are usually aligned via the x-axes. Commonly it requires at least 5 plots to help identify an animal's activity. Animal behaviours are found by manually combining the data to identify and label activities, however, this is challenging and takes 'interpretive effort and expert knowledge from the user [4]'.

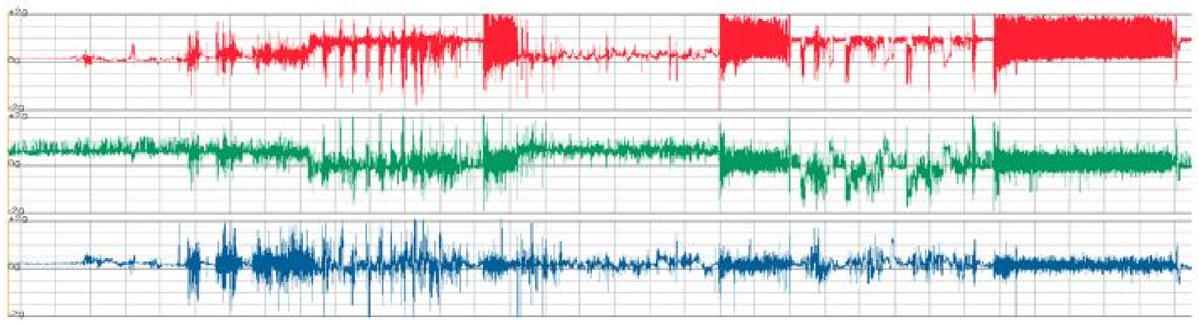


Figure 4: Acceleration data from a device attatched to an Imperial Cormorant plotted on 2D data plots [4].

Identifying animal behaviours via the 2D time-series plots is very 'error-prone due to (a) three components of two acceleration sources present in a single signal, (b) the signal being presented in the temporal domain (which can be long), (c) no simple visualisation of body orientation to assist the deductive process, (d) signal noise, and (e) slight variances in animal behaviour' [4]. This paper aims to produce visualisations which make it easier to interpret data and reduce errors in labelling behaviour.

No previous work is presented in the form of visualisations for data produced from daily diary devices. However, the paper takes a brief look at research in the use of accelerometers. In the field of automatic pattern recognition methods, accelerometers are often used. Applying pattern matching techniques to accelerometer data makes accelerometers suitable for use in pervasive computing, that is, using accelerometers as an input device to detect user input such as gestures.

Grundy et al. have implemented visualisation techniques with a main focus on visualising accelerometer data collected from daily diaries. The visualisations created include: spherical scatter plots, spherical histograms, clustering methods and feature based state diagrams of data. Visualisations were shown to biologists at Swansea University for review and were found to be very useful. They have since been adopted by the Swansea University Smart Tag research group and it is expected that these visualisations will provide new insights into data collected.

2.1.2 Identification of Animal Movement Patterns Using Tri-axial Accelerometry [5] - Emily L.C. Shepard et al.

The paper "Identification of Animal Movement Patterns Using Tri-axial Accelerometry" focuses on identifying the behaviour of animals, using 2D time series plots of acceleration data recorded from Daily Diary devices. It is necessary to understand how to identify animal behaviour using the existing visual aids in order to create visualisations which enhance data analysis. Shepard et al. present data recorded from 12 different species of animals. Collected data is examined throughout the paper to assist in teaching the skill of indentifying animal movement.

Total acceleration is recorded in all 3 axes. Acceleration consists of static and dynamic components. However, one of the challenges is extracting these values. Static acceleration is used to determine the posture of an animal. Dynamic acceleration is used to derive the movement of an animal in its respected axes. Components of acceleration can be used individually or combined to recreate the movement of an animal. Biologists are mainly interested in the dynamic component of acceleration as movement is more useful for determining an animals activity. Animal patterns however, can only be resolved if the sampling frequency is high enough and the device is attached properly to prevent noise [5].

Static acceleration 'is a measure of the incline of the accelerometer with respect to the earth's gravitational field' [5] this can therefore be used to determine the angle of the animal it is attached to. Angles computed are analogous to the pitch and roll angles commonly used in flight dynamics. Figure 5 shows how the pitch changes as the angle of a penguin changes its orientation. Static acceleration values range from 1 to -1g and are extracted by subjecting the raw total acceleration values to an appropriate pass filter [5]. A moving average filter is suggested for extracting the static component. Once extracted, the static component can be used to derive the pitch and roll by converting to degrees.

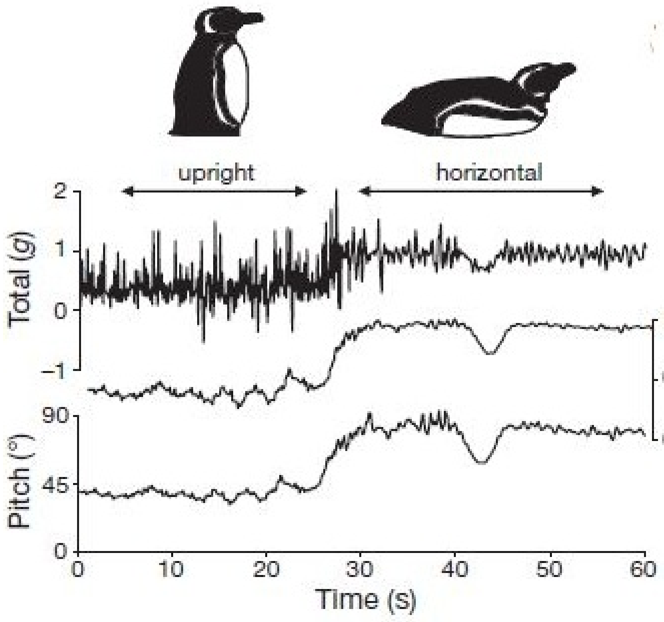


Figure 5: Graph displaying the body pitch of a Magellanic Penguin and its relation to total and static acceleration in the heave axis [5].

Pitch and roll can be derived from the static components of the heave and sway. The pitch is equal to the arcsine of the static heave acceleration component and the roll is equal to the arcsine of the static sway component. An animal is upright when the heave is equal to 1g and the surge is 0g. Pitch and roll are very important for classification of patterns [5]. For example, a Penguin up right for hours must be on land, where as if a Penguin is horizontal it could be swimming (in water) or sleeping (on land).

The dynamic component of acceleration 'represents the change in velocity as a result of body motion' [5], useful for determining the movement of an animal. It is computed by subtracting the earlier found static acceleration values away from the raw total acceleration logged by the daily diary devices. Newtons law states that 'for every action there is an equal and opposite reaction'. 'The movement of a limb in one direction will produce an opposite movement in the trunk' [5]. For example, presented in figure 6, a wing stroking downwards will result in a dynamic acceleration of the body upwards. The value oscillate's between each stroke cycle as the wing moves up and down. Dynamic acceleration can not only determine the direction of an animal but increases in frequency and/or amplitude can also show increases in effort which usually means an increase in speed.

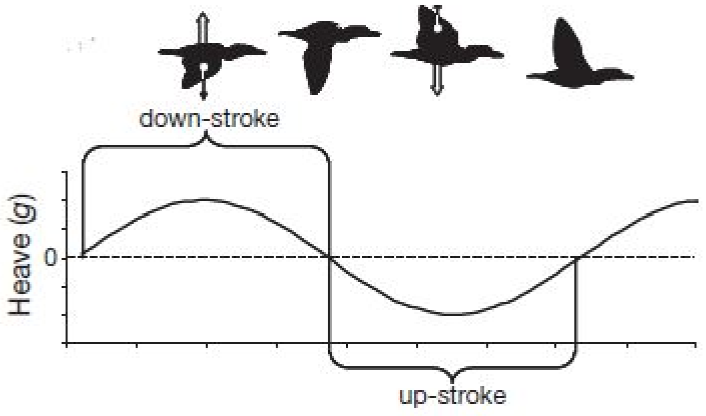
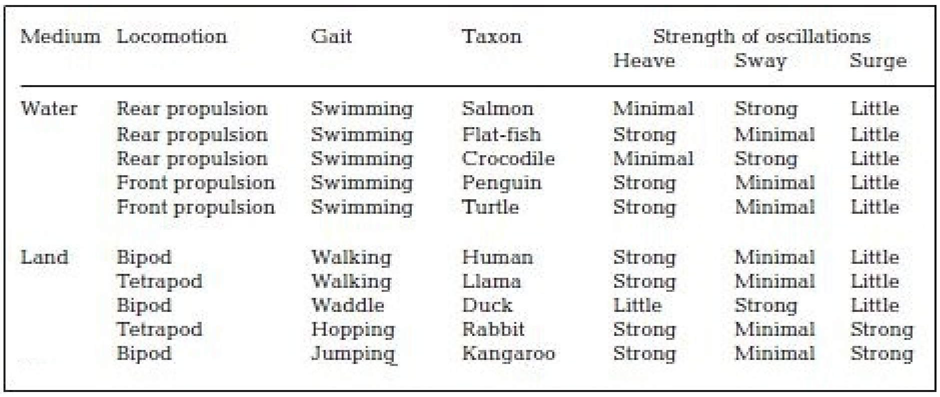


Figure 6: A graph showing the change in heave acceleration data values during the flight of a bird. As the bird flaps down the acceleration value is positive and as the bird flaps up the acceleration value is negative [5].

Overall there are many techniques for resolving an animals behaviour. Fast Fourier transformations, band-pass filters and shape analysis can be applied to describe patterns. However, simple rules define expected patterns across and within species, Figure 7 shows this.



Medium	Locomotion	Gait	Taxon	Strength of oscillations		
				Heave	Sway	Surge
Water	Rear propulsion	Swimming	Salmon	Minimal	Strong	Little
	Rear propulsion	Swimming	Flat-fish	Strong	Minimal	Little
	Rear propulsion	Swimming	Crocodile	Minimal	Strong	Little
	Front propulsion	Swimming	Penguin	Strong	Minimal	Little
	Front propulsion	Swimming	Turtle	Strong	Minimal	Little
Land	Bipod	Walking	Human	Strong	Minimal	Little
	Tetrapod	Walking	Llama	Strong	Minimal	Little
	Bipod	Waddle	Duck	Little	Strong	Little
	Tetrapod	Hopping	Rabbit	Strong	Minimal	Strong
	Bipod	Jumping	Kangaroo	Strong	Minimal	Strong

Figure 7: A table displaying some simple rules that define behavioural patterns in the listed animals. [5]

Moving on from identifying basic activities, finally the paper presents detecting more advanced behaviours. Identification of movement for more precise behaviours requires patterns of acceleration in all 3 axes, although the procedure for identifying behaviour is the same. Figure 8 shows a male Ovis Musimon (a type of mountain goat), involved in a clash/head-butt which is preceded by a run. When the goat is motionless the acceleration axes are stable. There is a strong heave value and medium surge and sway values towards the end of the graph, this indicates the goat has started charging. Finally, peaks are present in all the acceleration axes, the goat has clashed with another goat.

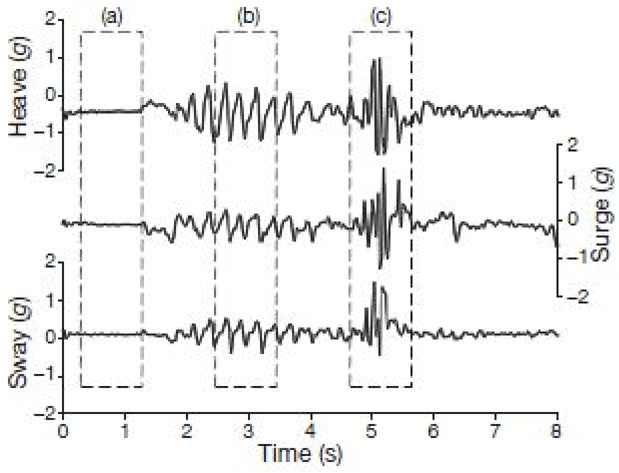


Figure 8: A mountain goat involved in a clash/head butt. Stage (a) shows the animal stable. Stage (b) shows the animal charging and stage (c) shows the animal involved in a clash with another goat [5].

2.1.3 Prying Into the Intimate Details of Animal Lives: Use of a Daily Diary On Animals [19] - Rory P. Wilson, E. L. C. Shepard and N. Liebsch

The paper 'Prying into the Intimate Details of Animal Lives: Use of a Daily Diary on Animals' covers similar ground to previous research papers reviewed, describing an overview of the "major features and operating modes of the Daily Diary devices". However, Wilson et al. also discuss features of the Daily Diaries not present in previously reviewed literature, such topics include: compasses for reading activity and using Daily Diaries as a dead reckoning system. Wilson et al. also discuss features we are now already familiar with: Tri-axial accelerometry for resolution of behaviour and limitations of Daily Diary devices. This paper review will focus on the topics previously not discussed to gain a greater insight into how the tri-axel compasses work and there uses.

Daily Diary devices include a tri-axial compass which can be utilised to calculate the heading of an animal with respect to magnetic North to within 1 degree of accuracy. As previously seen with tri-axial accelerometers, compasses can also be used to determine animal behaviour. "Compasses utilised in the Daily Diaries are so accurate that they need not be converted into an animal heading or orientation with respect to the earths magnetic field, instead the direct value, logged by the sensing devices is often enough to show an animals activity" [19]. Although, it is more reliable to convert to degrees as the "sensitivity of the compass varies with its orientation with respect to the earths magnetic field' [19]'. Figure 9 shows an example of how compass data may show behavioural patterns. As the compasses can show changes in body orientation, little to no fluctuation in wave patterns show low activities, indicating little animal movement. Large wave patterns show big changes where the animal is constantly moving, for example, running.

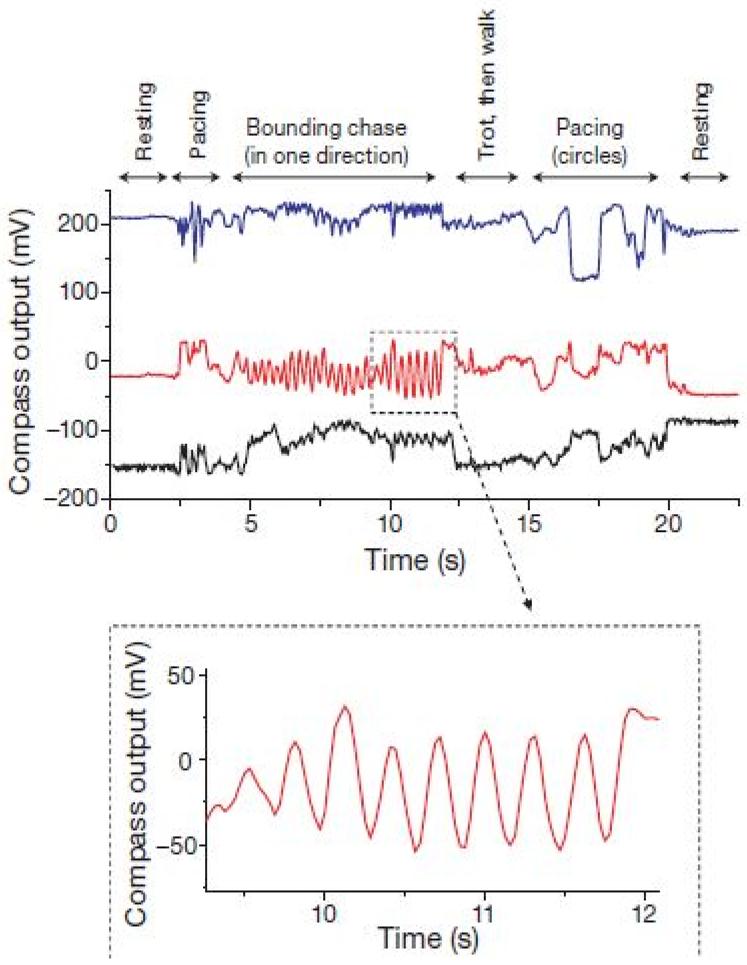


Figure 9: Time intensity plots of Tri-Axil compass data showing the relationship between compass data and animal behaviour [19].

Using compass and acceleration data combined it is possible to create a dead-reckoning system. Given a starting position (where device was released) with knowledge of speed (Dynamic acceleration), heading (Compass data) and change in height (Pressure) it is possible to "derive new positions with respect to those previously known" [19]. Figure 10 shows an example of a dead-reckoning trace computed using this technique. The use of such a dead-reckoning technique allows the Daily Diaries to derive animal locations regardless of their locale environment. This is especially impressive in environments such as underwater where other technologies like GPS fail. However, one of the problems with this technique is errors often accumulate over time, "the degree of these errors depends critically on whether the equipped animal is terrestrial, Volant or aquatic" [19].

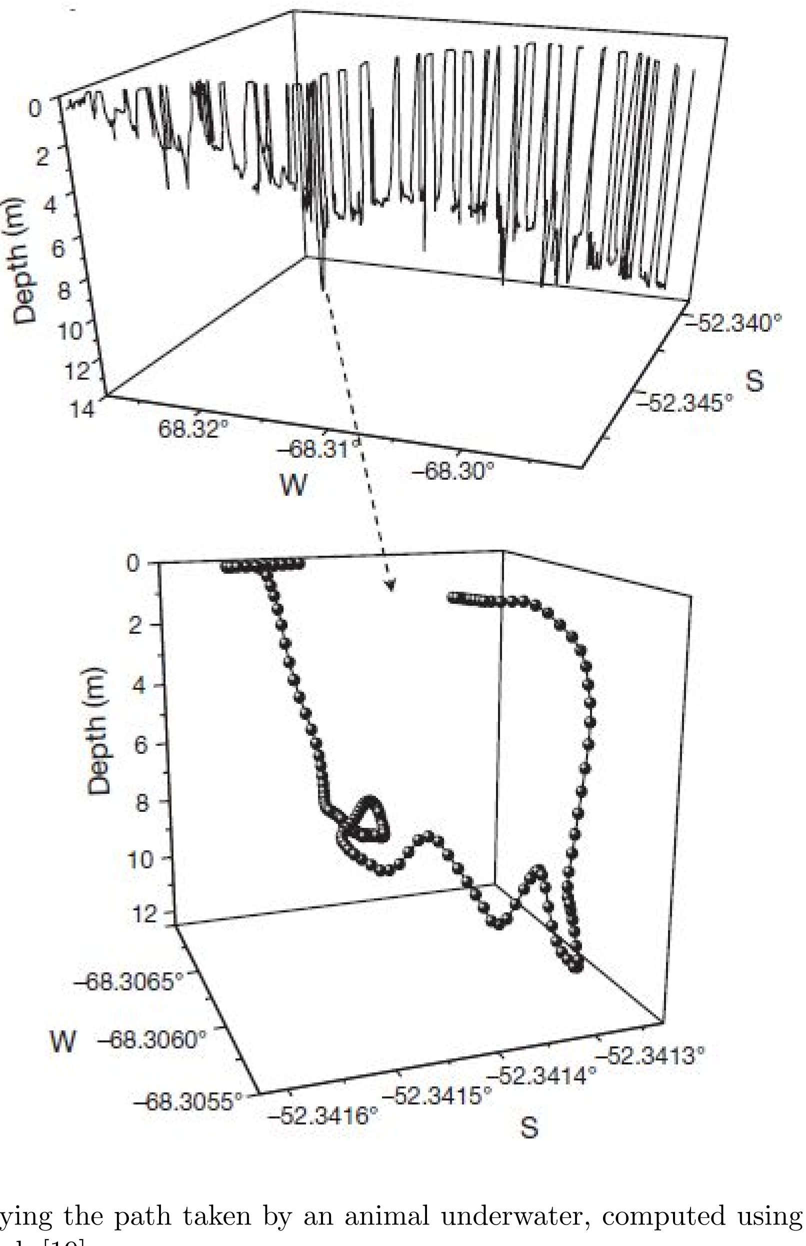


Figure 10: Graph displaying the path taken by an animal underwater, computed using a dead reckoning system approach [19].

Aquatic animals face the extra problem of drifting in water due to currents. This cannot be detected using the current sensors attached to the Daily Diary devices and as such additional measure have had to be put in place to measure speed. A small highly flexible paddle (Figure 11) has been added to the devices which bend backwards, allowing water flow to flow over it. An infra-red sensor then bounces light off the paddle to determine its angle. This logged measurement can then be used to measure an animals speed to a high accuracy, overcoming and reducing errors in aquatic species.

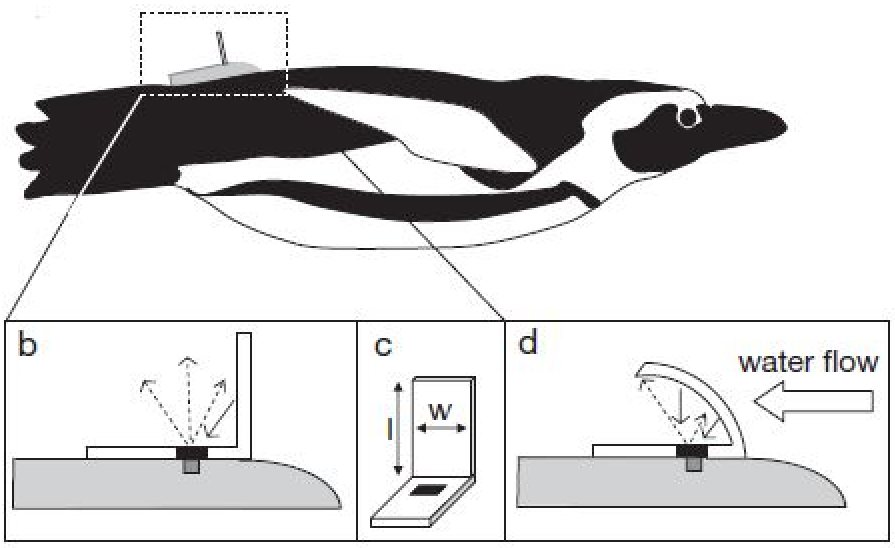


Figure 11: Paddle system introduced to the aquatic version of the daily diary device to determine the relative speed of an animal in water [19].

A worse problem exists in Volant species due to variable wind speeds being much greater than that from water currents [19]. Currently there is no sensor which overcomes these errors. Wilson et al. states the flight paths of birds can only be obtained with dead-reckoning if accurate, independent fixes are taken at regular intervals, such as with GPS [19]. This is not ideal when taking into considering some animals such as the Imperial Cormorant falls under both Aquatic and Volant categories.

Terrestrial animals do not suffer from drifting although it is a problem measuring speed. One approach is to use the overall dynamic body acceleration (Combined dynamic acceleration in three axis). Dynamic acceleration has a strong correlation to the indication of the speed of an animal, see Figure 12.

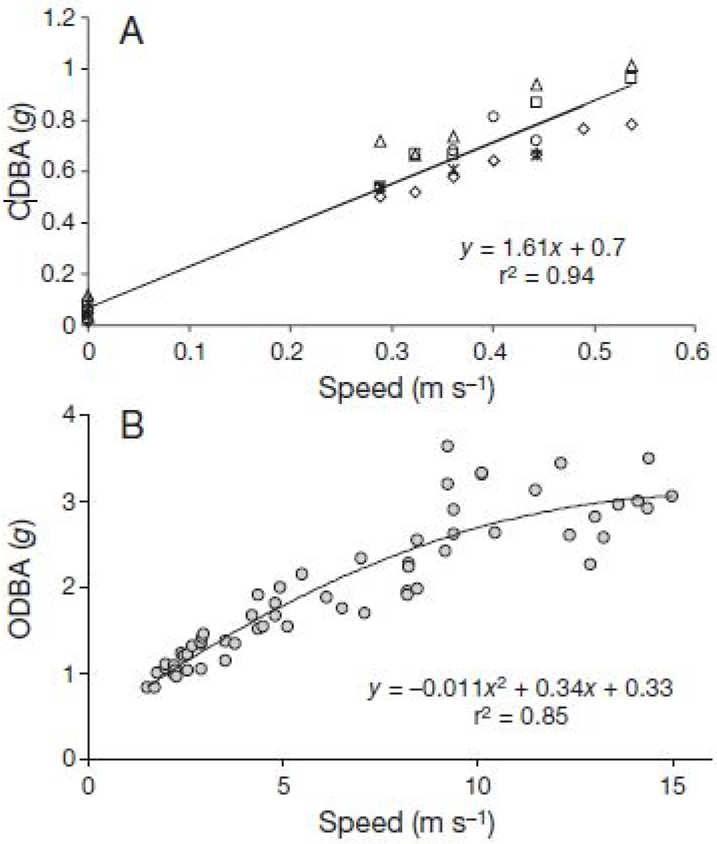


Figure 12: Graph of Overall Dynamic Body Acceleration (ODBA) verses the speed of an animal. There is a strong correlation between both of these values, indicating speed can be derived from ODBA [19].

Wilson et al. conclude the paper with a discussion of future prospects for the Daily Diary devices. A key idea mentioned is in the area of visualisation.

> ”We hope to be able to project an image of the equipped animal on the computer screen and then have that animal perform the activities undertaken by the free-living animal at the time it was equipped, but also to move through an appropriate environment. On land this can even be taken from NASA data that emulate a Google earth approach so that it should be possible to see the animal in its natural habitat. The conditions of the habitat (temperature, light, humidity etc.) can be superimposed”

Essentially creating a virtual 3D representation of the data, recreating animal movement in the locale environment the animal is situated.

2.1.4 Interactive Data Visualization: Foundations, Techniques, and Applications [11] - Matthew O. Ward, Georges Grinstein and Daniel Keim

The book 'Interactive Data Visualization: Foundations, Techniques, and Applications' provides an introduction to data and information visualisation techniques. Theory, practical details and tools necessary for building visualisations are also featured [11]. This book provides a useful resource for information on visualisation methods, and benefits the project by providing the necessary information to create visualisations which assist in data analysis. A brief summary of the introduction chapter follows.

Data visualisation is the method of displaying data in a graphical form which can allow a user to gain insight into data. "A single picture may contain a wealth of information and can be processed much more quickly than a comparable page of words" [11], meaning visualisations are an effective means of presenting data. "Pictures are also independent of local language, as a graph may be understood by a group of people with no common tounge" [11]. Visualisations are used on a day to day basis all the time throughout our daily lives, take for example viewing a weather chart or map of a region. These are both graphical representations of data, in this case, weather and geographical data.

Visualisations are important and if done correctly can provide a great insight into data being presented. They are useful in decision making, communicating results and discovering new knowledge. However, the choice of visualisation is vital. Choosing a visualisation method can impact on the decision making process. Different visualisation methods may distract the user from the truth of the data characteristics. An example of this is presented by Ward et al, data from a hypothetical clinical trial is displayed using various different visualisation techniques. Some visualisations were more effective than others during the decision making process of answering the question: Should the trial should be stopped. The presentation of a visualisation is also just as important. Figure 13 shows 4 data plots each displaying the same data but with different spacing between the axes. The graphs all show different conclusions to the data, therefore it is necessary to never make assumptions about a visualisation and perform user studies before putting visualisations in a working environment.

A visualisation process exists for designing new visualisations. The process starts with an analysis of the data available and the type of information the viewer hopes to extract from the resulting visualisation. To create a visualisation there needs to be a mapping from the data to a spatial domain. For example, numbers from a data set could be mapped to the y axes and the numbers index to the x axes, this would define a scatter graph. Many different visualisation techniques exist, each with different ways of mapping the data to a spatial domain. Dynamic visualisations allow the user to change attributes of the visualisations such as the colour, or the data selection be shown. The user should be able to customise a visualisation to their standards until they have achieved their goal in extracting the knowledge they seek [11].

The process of starting with data and producing an image is called a pipeline. Specifically we are interested in the visualisation pipeline. Figure 14 displays the pipeline, the process starts by acquiring data to be visualised. A format must be used that enables rapid access and easy modification of data. Next, a data selection is required, this involves identifying a subset of data to be visualised, either selected manually by the user or automatically by an algorithm [11]. Next, as previous discussed the data needs to be mapped to a spatial domain. Finally, scene parameters must be set, such as camera position, colours and map selection.

While creating visualisations it is important to bear in mind the limitations of the human visual system. It is essential that perception abilities be considered [11]. Visual illusions may occur from which the eye will not perceive the data as it is plotted on the screen. 'If we are presenting a

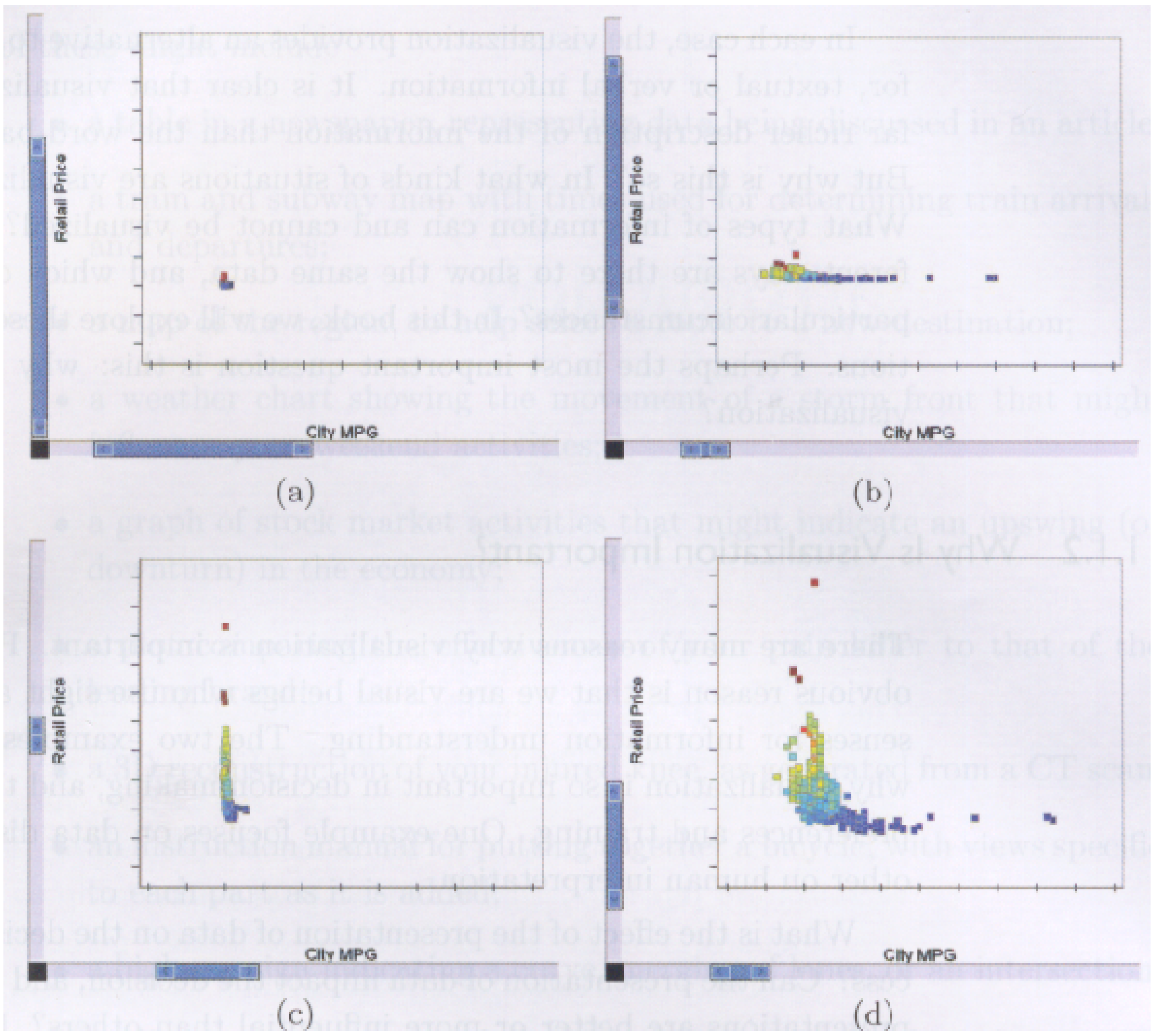


Figure 13: Plot (a) shows a large scale in the X and Y axes. Plot (b) shows a large scale in the Y axes. Plot (c) shows a large scale in the X axes and Plot (d) shows a range determined by the range of X and Y data values [11].

conclusion, we do not want ambiguities such as Shepard's many-legged elephant' [11] in figure 15. By knowing how the visual system works, it is possible to avoid such problems occurring.

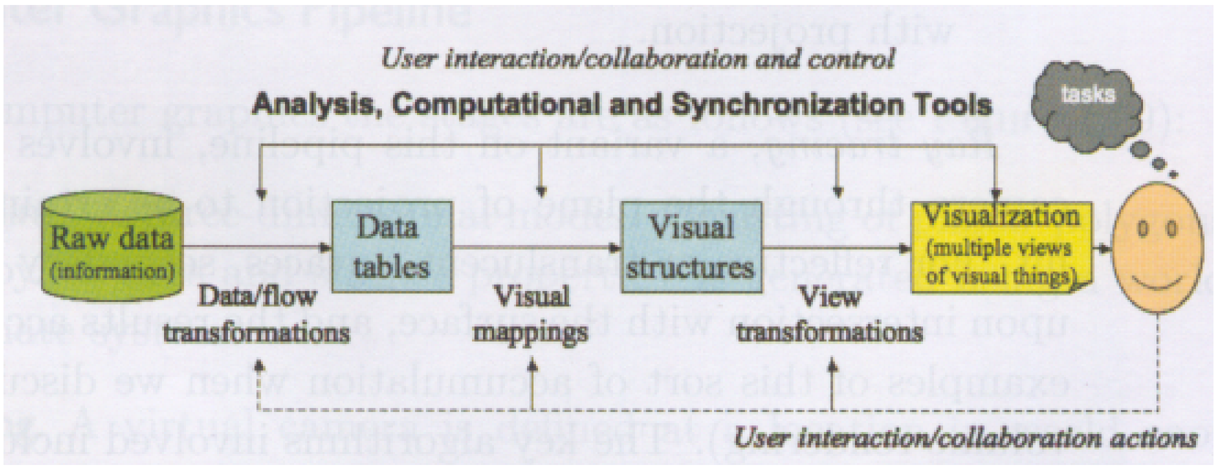

User interaction/collaboration and control
Analysis, Computational and Synchronization Tools
tasks
Raw data (information)
Data tables
Visual structures
Visualization (multiple views of visual things)
Data/flow transformations
Visual mappings
View transformations
User interaction/collaboration actions

Figure 14: The data visualisation pipeline [11].

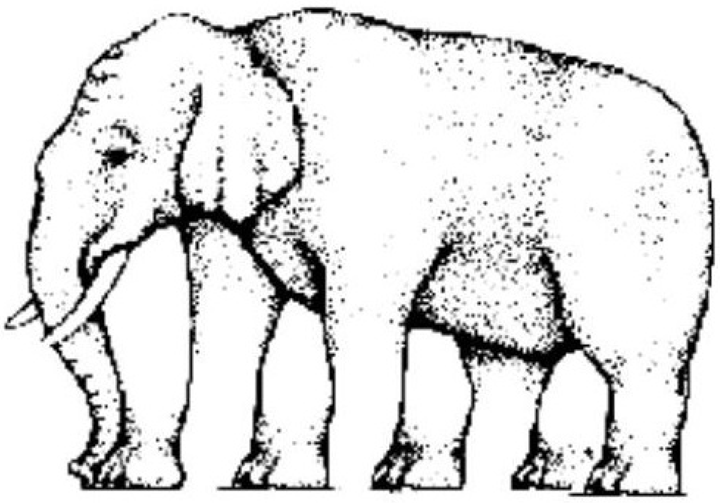


Figure 15: How many legs does this elephant have? Image from http://www.ilusa.com/gallery /elephant-illusion.jpg [11]

2.1.5 Graphical Perception of Multiple Time Series [7] - W Javed, B McDonnel and N Elmqvist

The paper "Graphical Perception of Multiple Time Series" introduces techniques for visualising two dimensional time series data. Visualisation techniques presented throughout the paper are: Simple Line graphs, Stacked Graphs, Braided Graphs, Small Multiples and Horizon Graphs. Each of these is evaluated for user performance by slow and discrimination tasks to discover which techniques are better and in what scenarios they should be used.

Daily Diary devices collect multiple attributes of time dependant data, the main focus of the paper. By introducing different methods of visualising recorded data on time intensity plots there is an opportunity to improve the current 2D plots used to visualise the Daily Diary data.

William Playfair invented the line graph in 1786, its purpose being to assist people in analysing time series data. Line graphs are today one of the most common types of statistical data graphs and are used to visualise a vast array of data, such as: finance, medicine and of course, animal behaviour. Line graphs are simple enough to use with one dimensional data, however, as we have seen with the current visualisations of Daily Diary data, time dependant data often involves many concurrent plots, which poses a problem. For example, to analyse Daily Diary data and indentify an animals activity we have to look across each time series at each sensor measurement. W Javed et al aim to address these problems by providing knowledge of when different graphing techniques are beneficial to the end user. This will be done by "rigorously evaluating graphical perception (Users ability to comprehend the visual encoding and there by decode the information presented in the graph) for different tasks involving multiple time series through controlled laboratory experiments" [7].

Each of the plotting techniques will now be summarised before outlining the evaluation criteria created and user study presented.

Line graphs are the most basic form of graphs. "Time is mapped to the horizontal (X) axis, and the value is mapped to the vertical (Y) axis." [7] To display the time series data, we map time to the vertical axis and plot the data, points are then connected using lines. "Adding multiple time series is easy, just assign each series a unique graphics property, such as colour or a line style and add them to the shared space." [7] Figure 16 shows a line graph visualisation with a shared space of 4 time series.

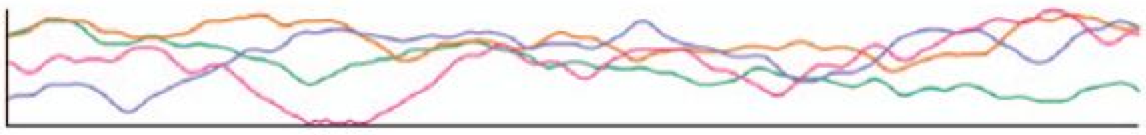


Figure 16: Basic Line Graph visualising 4 time series in shared space [7].

Small Multiples are similar to the basic line graphs discussed above. However, instead of adding each time series to the shared space, they each have their own separate graph. Commonly on each graph we "turn the line into a colour filled area to ease identification" [7]. Figure 17 shows a small multiples graph visualisation for 4 time series. This is similar to the technique currently used to visualise Daily Diary data.

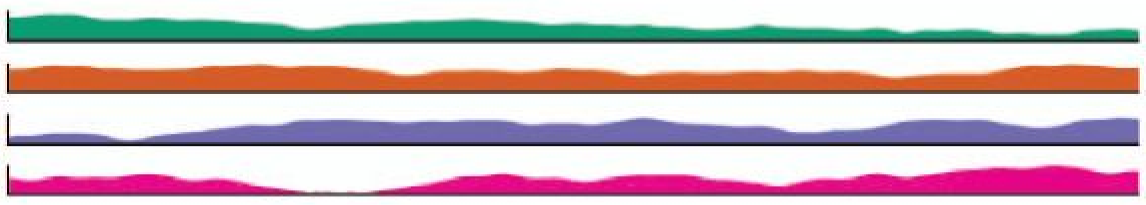


Figure 17: Small Multiples Graph displaying 4 time series in split space [7].

Stacked Graphs also turn the line into a colour filled area, but unlike Small Multiples they are a shared space technique, stacking the graphs on top of each other sequentially. Figure 18 shows a stacked graph visualisation with a shared space of 4 time series.

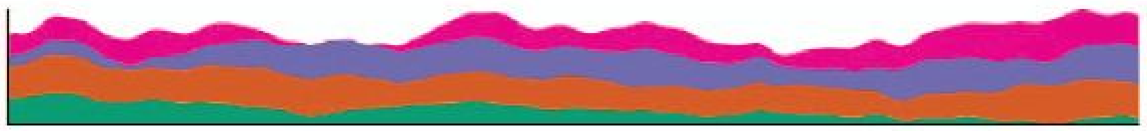


Figure 18: Stacked Graph displaying 4 time series in shared space [7].

Horizon Graphs are different in comparison to the previous three visualisations. They are space efficient by wrapping negative values in the positive axis. Different colours are allocated for positive and negative curves. Also the curve colour values are split into a discrete range, and mirrored above the baseline. "This virtual resolution and wrapping of negative values means that more space can be allocated for each individual time series" [7], so visual clutter is low. Figure 19 shows a Horizon Graph visualisation with of 4 time series, blue is used for positive values and red for negative.

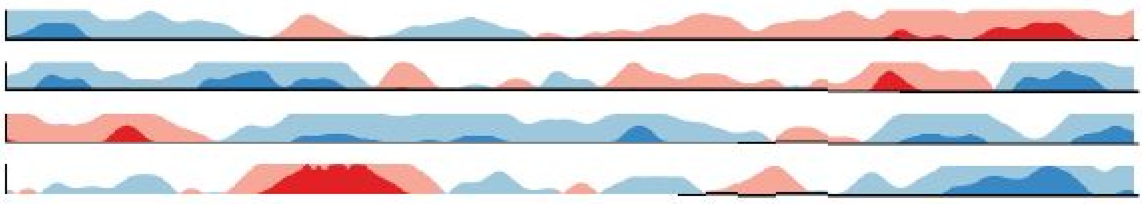


Figure 19: Horizon Graph displaying 4 time series in split space. Negative values are colour coded red and positive values blue [7].

Braided Graphs try to combine shared space and split space layout techniques. So the user can gain the advantages from both these layouts. Individual advantages are "that (i) shared space layout benefits comparison across time series over split space, but that (ii) split space layout makes identification easier." [7] In order to achieve this Braided Graphs use a similar concept to stacked graphs by filling the area under a curve will colour. Instead of stacking the graphs on top of each other they are cut into segments at intersection points to give the appearance of a continuous line through the time series graph. "This technique maintains a common baseline while using area curves to aid identification." [7] However, for large numbers of time series on the same graph there is a potential for high visual clutter. Figure 20 shows a Braided Graph visualisation for 4 time series.

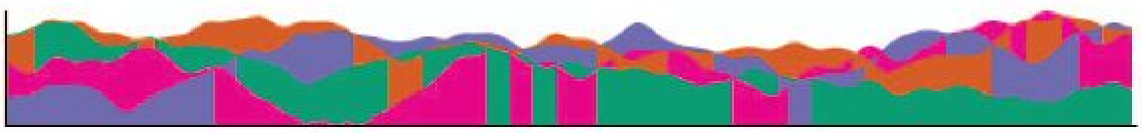


Figure 20: Braided graph displaying 4 time series in shared space. Each data series is seperated by colour [7].

Hypotheses are given as to what the authors expect the results to be from the study. The authors expected that shared space techniques will perform better for tasks with local visual span (Braided and simple graphs), Split space techniques will perform better for tasks with dispersed visual span (horizon and small multiples), many concurrent time and small display space will cause decreased performance. These will be proven or disproven using a user study.

Sixteen participants with a good graph reading ability were set a series of tasks to answer questions using the graphing techniques discussed. The time taken to answer each question for each graph was recorded and used for comparison. Three tasks include: Local maximum task - "Finding the time series with the highest value at a specific point" [7], dispersed Discrimination task - "Finding the time series with the highest increase during the whole displayed time series" [7] and a slope task - "Determining which time series had the highest value at a point specific to each series." [7]

The results confirmed the hypothesis, of which. "Shared space techniques were faster than split space techniques for local maximum task. Split space was faster than shared space techniques for dispersed discrimination task. Finally, Small multiples and Simple Graphs were fastest for the slope task." [7]

2.1.6 Stacking Graphic Elements to Avoid Over-Plotting [3] - Tuan Nhon Dang, Leland Wilkinson, and Anushka Anand

The paper "Stacking Graphics Elements to Avoid Over-Plotting discusses the problem with over-plotting which occurs when visualising a large abstract data set. A solution of stacking elements is proposed where "the height of the stack shows the density in the neighbourhood of a data value" [3]. One of the challenges of this project is the size of the data sets to be visualised. Over-plotting is serious concern and must be addressed. This paper assists in providing solutions to such a problem.

Previous work in solving over-plotting exists and several solutions are presented by the author. The first solution, "aggregate cases in local neighbourhoods and to use area to represent density" [3]. Instead of just displaying the plotted elements they are also placed in bins and a "count is presented in proportion to the size of a circle" [3], placed in the region the bin is located. This introduces more problems as these new representations can overlap and cause the same effects as over-plotting.

Second solution, "partition a space, aggregate within partitions and adjust the size of partitions in order to represent counts" [3]. This will prevent overlapping although can "lead to distortion of density locations" [3].

Other solutions which the paper describes include, colour mapping, random displacement of partitions (jitter) and using transparency to represent density. The problem will all of these solutions is they "break mapping between data points and locations on screen" [3], that is, they dont keep the objects in there spatial domain and instead manipulate or combine points together.

A proposed solution is to stack dots or lines in an additional dimension to represent the count of points at that region without distorting any plot positions. Stacking dots solves over-plotting by stacking dot elements which occur in the same region above each other. Aggregation of dots represents the density of points in the neighbourhood of the data value. Figure 21 shows an example of stacking dots. For each electricity and water amounts with more than one value the same, a dot is stacked above the original data point, in the extra dimension added to the visualisation.

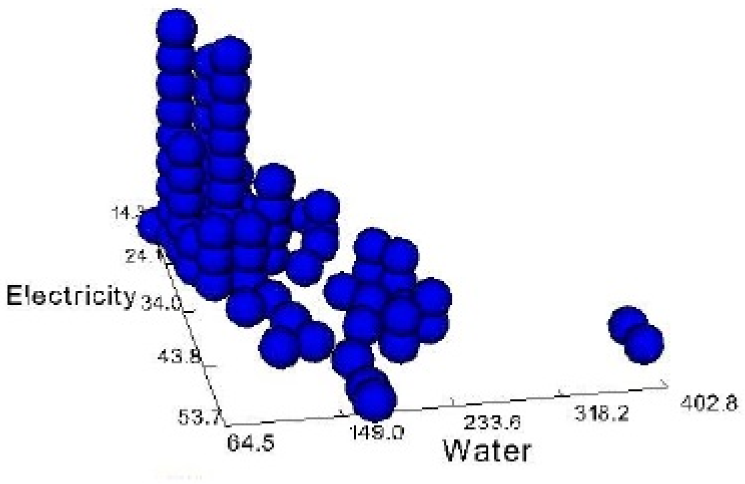


Figure 21: Dot Plot displaying Electricity against Water. When data values overlap they are stacked above each other in the extra dimension added [3].

Parallel Coordinates also suffer from over-crowding when utilised with large data sets. The problem with Parallel Coordinates is that the axes are visually separated making it hard to analyse data attributes which are not next to each other on the plot. It is also difficult to see the amount of points on each axis where data items share the same value. Figure 22 displays a standard Parallel Coordinates visualisation, it is obvious to see the problems listed above in the plot.

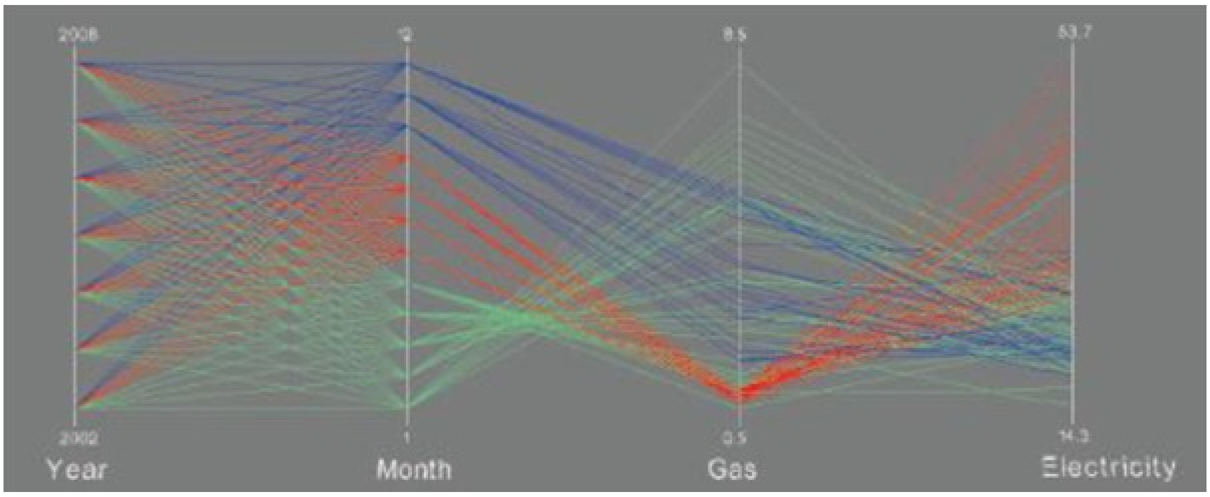


Figure 22: A standard Parallel Coordinates plot [3]. Key problems which limit data analysis: How can we compare the month to electricty relationship? How many data items are in January?

Authors of the paper have come up with two solutions to this. First solution, apply the same principles used in the dot plot algorithm to the axes of Parallel Coordinates (Figure 23). At each axes dots are stacked on top of each other to represent the count of data items at that point on the axis. "Polylines are colours by the season categorical variable" [3] and are also used in the dot plot to give the user a clear indication of the breakdown of where lines originated from.

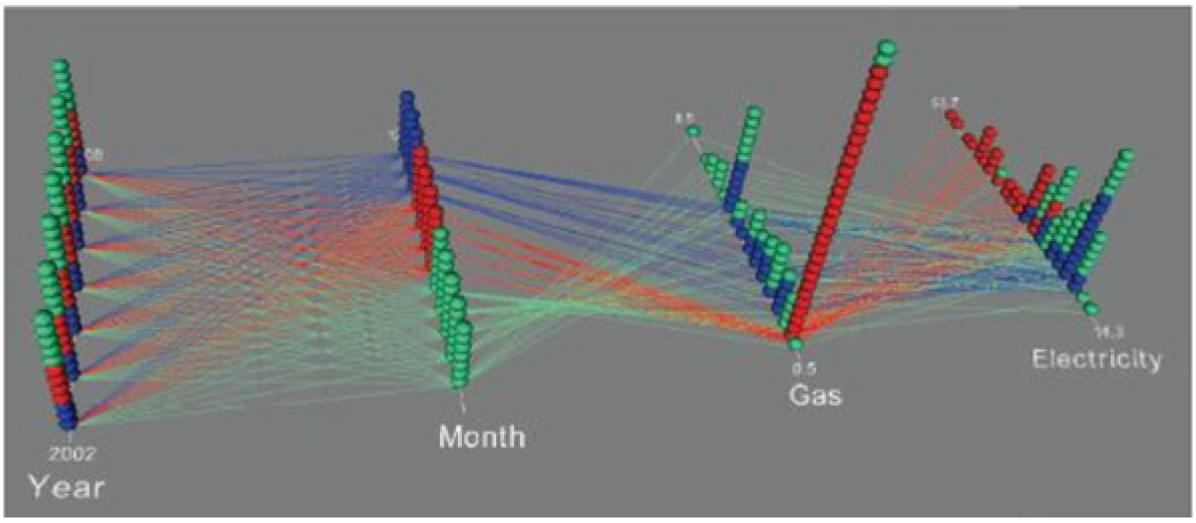


Figure 23: A Parallel Coordinates plot with Dot Plot combined [3]. It is now possible to determine how many data items are in January.

The Second approach is to stack lines instead of dots (Figure 24). Authors have extended the dot algorithm to work with lines. Lines are stacked on top of each other for the same mapping between two axes. Stacks are colour based on their height. This technique lowers visual clutter and is easier to see relationships in comparison to the first solution.

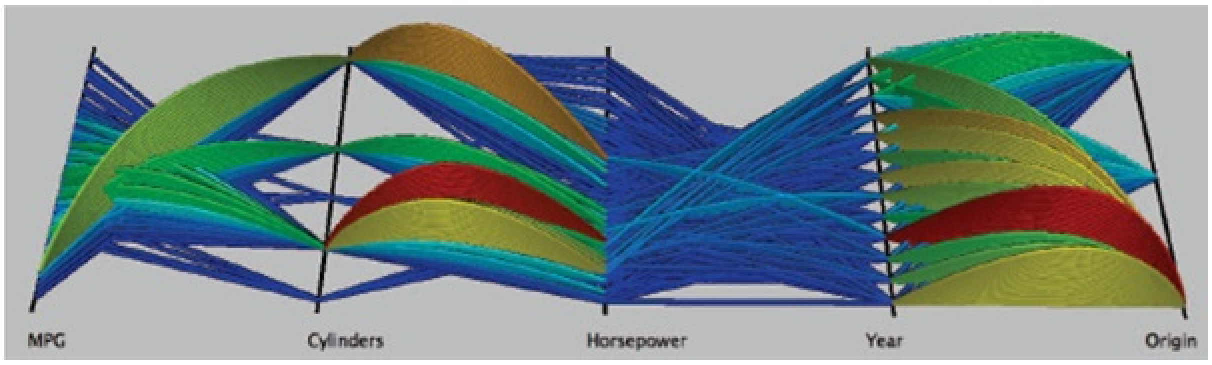


Figure 24: Parallel Coordinates plot with line stacking [3].

Both stacking dot plots and lines each have there own advantages. Stacking dots enables the user to see number of data items along each axes. Stacking lines means a user can see items with the same mapping between axes. Finally the two approaches are combined together (Figure 25) to add more clarity to the visualisaiton and better safeguard parallel coordinates against over crowding.

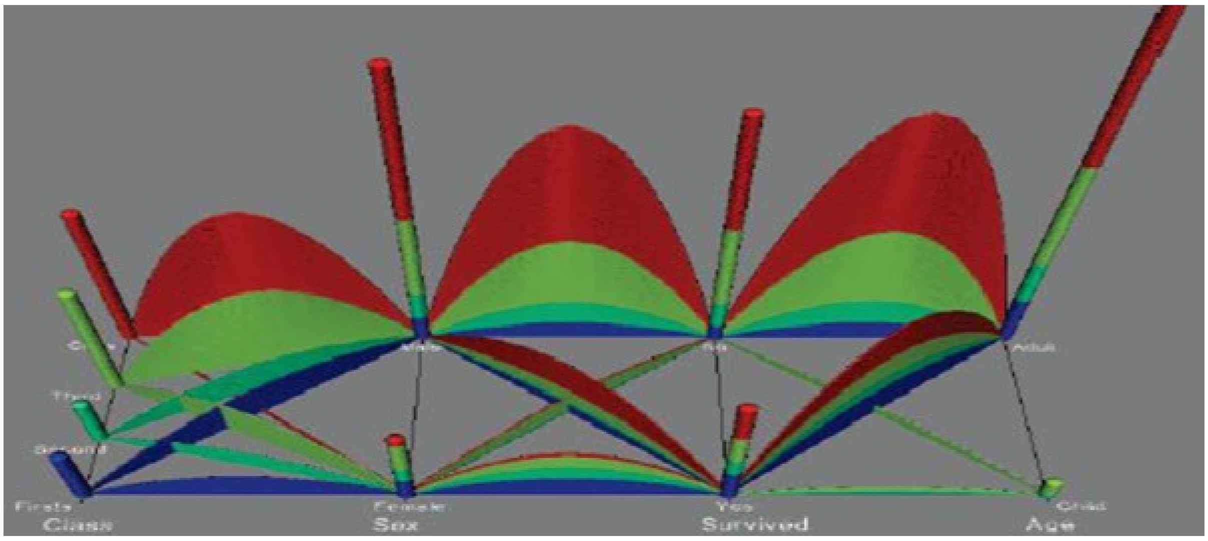


Figure 25: A Parallel Coordinates plot with line stacking and dot plots combined [3].

To conclude, the paper has introduced techniques to solve overcro2wding problems in visualisations without losing any information. This is done by stacking points which overlap using the dot plot algorithm. As we have seen with the Parallel Coordinates visualisation this technique can be ported to other visualisations to solve over-crowding.

2.1.7 Using Visualization to Debug Visualization Software [10] - Robert S. Laramee

One of the challenges of writing visualisation software is the ability to verify the correctness of results and find / resolve errors. "Traditional debugging software techniques are not applicable to the visualisation domain, such as printing results and using breakpoints which de-couple information from there spatial domain" [10]. Laramee provides a series of guidelines to debug visualisation software, developed over the years from his own software engineering and data visualisation experience. The key behind these guidelines is to use the strength of visualisation itself to display data structures or break algorithms down into their most basic components to see where / if bugs are occurring. This is especially useful to us as we do not want to provide the biologists with a tool which produces incorrect results.

Laramee describes why debugging is particularly hard for visualisation software. Often algorithms are very complicated and consist of a high complexity, simply viewing variable values will often not be enough to gain any insight into where results are incorrect. During visualisation we often want to visualise a large amount of data, up to gigabytes in size. 'Attempting to print out and analyse millions of primitives is not feasible'. Large complex data structures are also often utilised causing similar problems for debugging.

Twelve guidelines are presented by Laramee on how to effectively debug visualisation software. A few of these have been selected to be discussed which are relevant to the project.

Number 2: Visualize Data Structure Traversal and Evolution - Rendering the data structure as it is being created. This enables the software developer to see the data structure evolve and verify its correctness. Figure 26 displays a 'quadtree used to store adaptive resolution data from a height field together with a costal land map'. Example data structures include: Adaptive meshes and binary trees.

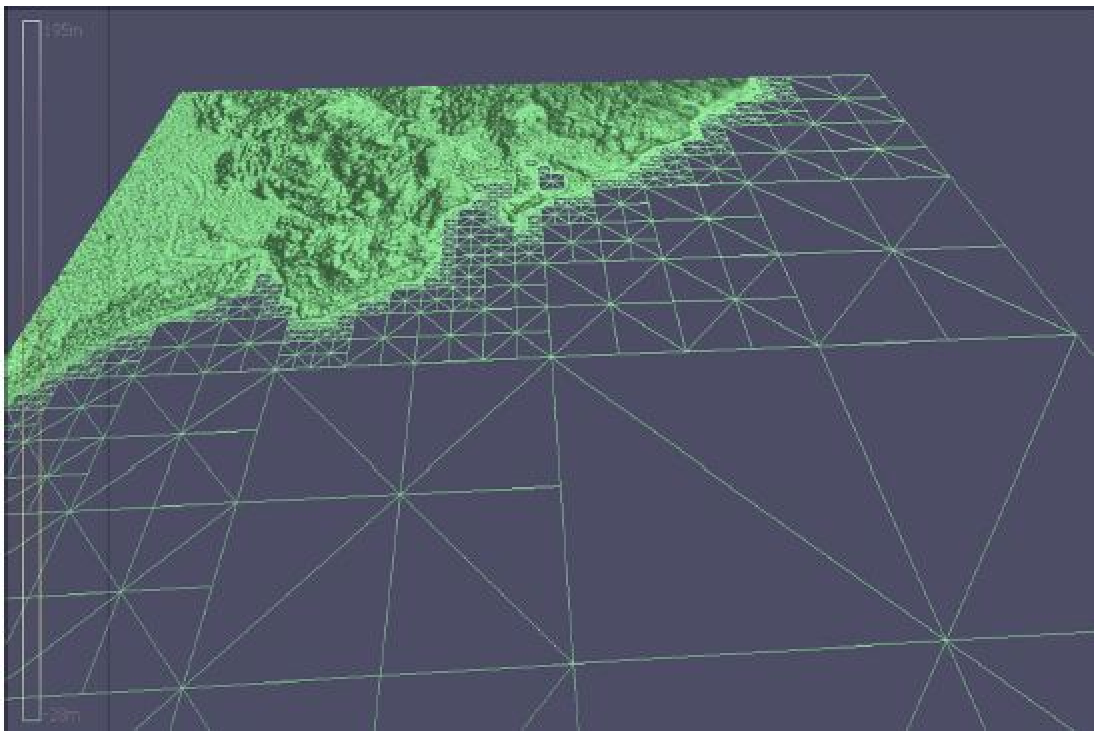


Figure 26: An adaptive quadtree data structure painted for debugging purposes [10].

Number 3: Classify and Colour Map - Geometric primitives can be colour mapped according to characteristics or categories they fall under to provide useful debugging information. It should provide a simple and intuitive step to the debugger if a primitive is incorrect.

Number 4: Incorporate Algorithm Parameters into User Interface - Parameters for algorithms are not usually known and as such should be incorporated into the user interface. The value may change for different data sets and therefore may prove to be useful during debugging.

Number 8: Test Algorithms on a Variety of Data Sets - Testing an algorithm on a variety of large and small data sets gives the developer more confidence there algorithm is correct. 'Believing that an algorithm works after only have tested a few small, simple data sets is a common mistake'.

Number 9: Exploit and Compare with Previous Literature - Often the data set being visualised has often been visualised before or a data set exists which can be used to test results output from the visualisation software created.

Number 10: Make Exclusive Use of Accessor Methods - All class variables should be accessed through accessor methods. This forces encapsulation and allows us to perform error and bounds tests on assignments. Using accessor methods leads to robust code.

Number 11: Follow Coding Conventions - Another paper by Laramee, "Bobs Concise Coding Conventions' outlines a series of coding conventions to follow when developing software. Following coding conventions enabled code to be more legible, leading to a lower amount of bugs and maximises code reuse.

In conclusion, this paper provides a nice set of rules to be followed to ensure a quality software product is delivered. During the project this paper provides a useful resource for fixing bugs and verifying visualisations are correct.

2.2 Previous Systems

Biologists currently plot data collected on 2D time series plots and analyse the data from this. Software used to produce these plots is commercial and as such provide generic visualisations for use across a wide range of different data sets. This is not ideal for the biologists as they need bespoke visualisations which are specifically developed for visualising Daily Diary data. The commercial software products used by the biologists are now discussed.

2.2.1 Origin Pro - OriginLab

Origin Pro is a commercial graphing and data analysis application produced by OriginLabs. Figure 27 shows an image of a line plot of Daily Diary data produced using the Origin Pro software. Origin runs only on the Windows platform and is available on a trial basis for 21 days. The official web page from which Origin Pro can be downloaded is: `http://www.originlab.com/`.

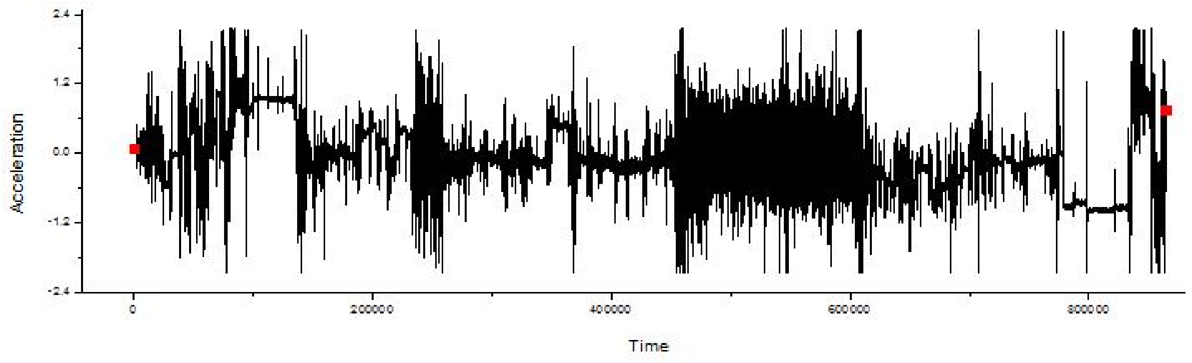


Figure 27: Heave Acceleration verses Time graph produced in Origion of a whole data set.

Origin pro was created with the intention of being used for data analysis and creating graphs from data input. Supporting a large variety of different file formats, the files produced from Daily Diaries can be imported very easily. Large data sets are also very well supported; the software is more than capable of handling a large Daily Diary data set containing over a million items and plotting them to a graph. Origin can also produce 3D data plots in the form of: 3D scatter plots, wire frame surfaces and much more, although it is not clear if the biologists currently utilise any of these techniques.

2.2.2 Microsoft Excel - Microsoft Corporation

Microsoft Excel is a commercial spreadsheet application produced by the Microsoft Corporation. Excel runs on both Windows and MacOS platforms and is commercial, although a free trial is available from the Microsoft website which lasts for 60 days. The official web page of Microsoft Excel is: `http://office.microsoft.com/en-us/excel/`.

Excel was created for use as a spreadsheet application, aimed to organize data and perform arithmetic operations. Excel can also be used to create graphs and has several different graphing options including: bar charts, scatter graphs, pie charts, histograms and many more basic techniques. Excels main user focus is on people who need to manipulate data and/or display it in a graphical form.

Figure 28 shows an example curve plot created in Excel of data recorded from a Daily Diary device Note this is the same data set as used in Figure 27 (Created using Origin Pro) but they are producing different results. The reason these graphs look different is excel has limitations when being used with large data sets. Excel can only hold up to 1,048,576 rows of data and can only plot graphs containing 32,000 data items - some of the data sets recorded from Daily Diaries can contain well over this amount of data items. Only a sub-sample of the biologists data sets can be input into excels at one time. Also Excel can only plot 2D representations of the data as unfortunately there are no 3D visualisations available.

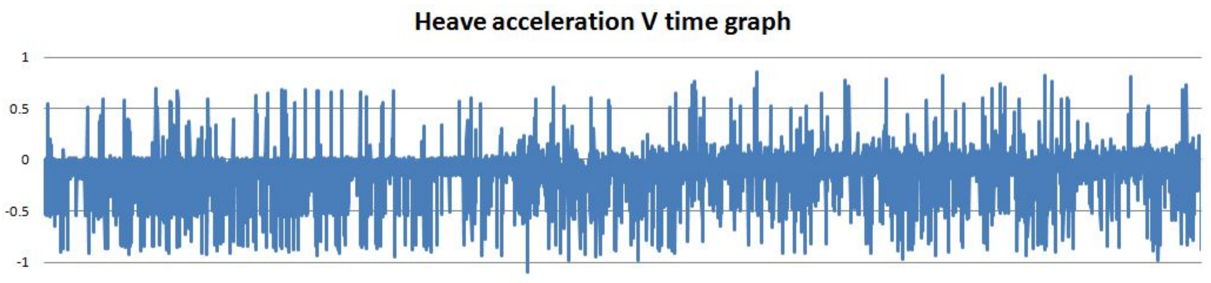


Figure 28: Heave Acceleration verses Time graph produced in Microsoft Excel of 32,000 data items.

2.3 Data Characteristics

Data is the driving point of any visualisation project. As a result, it is especially important to understand the data to be able to produce any kind of visualisation or tool which assists in data analysis. This section outlines information about the data output from the Daily Diary devices.

We will first look at the nature of the data and answer some key questions, such as where does the data come from and what insight the biologists aim to receive from collecting such data. We will then move on to looking at the data produced, including: which sensors are attached to the devices and typical values for these, finally, the grammar of the data files is defined.

2.3.1 Nature of Data

Data collected by the Daily Diaries is abstract (multiple dimensionality) and does not exist in an inherit spatial domain. Therefore collected data falls into the category of Information Visualisation. Typically under this field of Data Visualisation we are interested in the exploration of data, that is, we have no previous knowledge of the characteristics of the data (What animal the data was collected from, what activity the animal was undertaking, etc) and therefore we want to explore the data to discover information.

Data is collected at a set amount of times per a second (frequency), this means the data is time dependant (varies over time). A function f(t) returns a data subset for the time step 't'. Each valid time step will return a record of all the recordings of each sensor at that time.

Due to the data being time dependant, sizes of the data vary quite a lot and is dependent on how long the device was recording for and how many times a second measurements were taken. Typical sizes from the data sets provided from the Biology department at Swansea University range from 200kb (Probably a broken down data set) up to 290mb which is typically from 5 minutes of recording up to approximately 5 days.

As we have already seen, data is collected from Daily Diary devices attached to animals. Devices have up to 13 sensors attached, which are discussed in closer detail in the next section. Daily Diaries can be attached to a wide variety of animals, the only limiting factor being the weight and size of the device, as it must not affect the animals movement.

Rory Wilson, head of the Biology Department at Swansea University was impelled to create a device which could record the day to day activity of animals. In an interview upon receiving a Rolex Award [2], Rory told the story of how the Daily Diary device idea came about. On his first day of studying Penguins off the South African Coast, he watched them go into the sea to supposedly hunt fish and then come back some time later. Rory wanted to know what the Penguins were doing during their time underwater. To do so he created a series of devices and prototypes for recording animal movement, eventually arriving at the tri-axel accelerometer Daily Diary device which we are studying in this project. However, this is not just a problem with Penguins, but all submersible animals. This demonstrates the importance of the Daily Diary devices throughout the field of biological research and how important it is to provide the biologists with the visualisation tool they so desperately need.

Another animal of interest is the Whale Shark (Figure 29). Whale Sharks are the worlds biggest fish and can live for months underwater at a time. Researchers simply did not know what Whale Sharks were doing under water, until attaching a Daily Diary device to one. Data collected from a device gave insight into the life cycle of Whale Sharks and also revealed a unique diving pattern as reported in a news report by Science Alert [1].

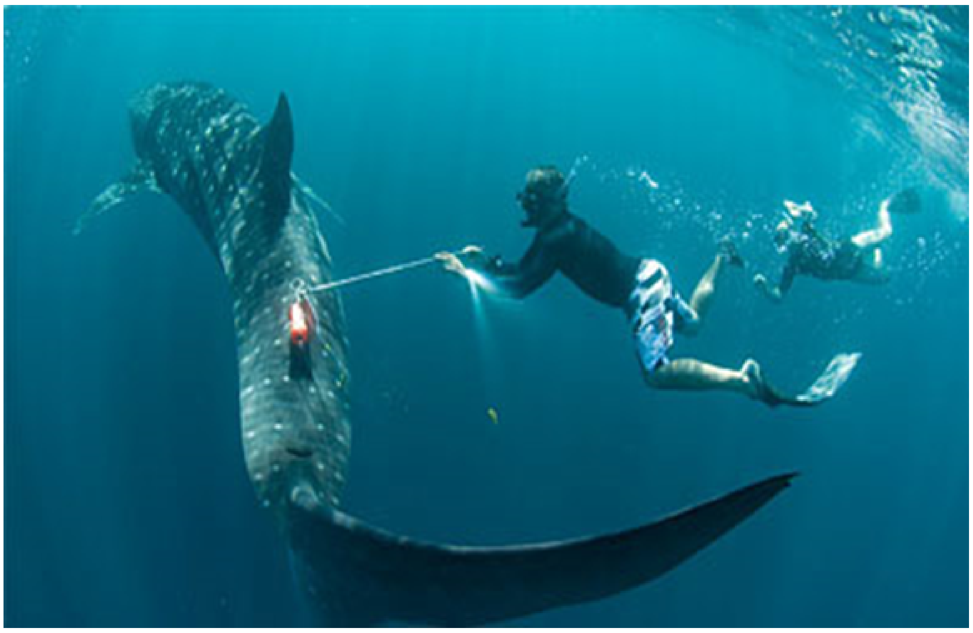


Figure 29: Daily Diary device being attatched to a Whale Shark [2].

2.3.2 Sensors

Up to 13 sensors can be attached to the Daily Diary devices. Listed below are the typical sensors attached and a brief explanation of what each sensor records:

- Accelerometers in X, Y and Z axes.
  Accelerometers measure acceleration forces (g). They do this by reacting to changes in the earths gravitational field. The recorded acceleration value is given as a total acceleration value, consisting of a dynamic and static acceleration component.

- Hall
  Hall is a measure of the proximity of a nearby magnet (mV). In 'Measuring the state of consciousness in a free-living diving sea turtle"' [8] Houghtona et al. described how a Hall sensor was used on turtles, "By positioning the Hall sensor on the upper mandible of the turtle and a neodymium boron magnet on the lower mandible, we recorded when the turtle opened its mouth".

- Pressure
  Pressure (mBar) is a measure of the force per unit area perpendicular to the sensor. Pressure is useful for identifying the locale environment of an animal. For example, in water the pressure is different to that above water level and as such can be used to determine if an animal is in water or above land.

- Infrared - Commonly abbreviated to IR
  Infrared is a measure of electromagnetic radiation (mV). In "Prying into the intimate details of animal lives: use of a daily diary on animals" Wilson et al. discusses how a highly flexible paddle is placed on the aquatic versions of Daily Diary devices. As water flows over the paddle it is bent backwards indicating the relative speed of an animal in

water. Using an IR sensor it is possible to determine the angle of the paddle and therefore the relative speed.

- Compass in X, Y and Z axes
  Compasses are used as measurements in relation to the earths gravitational field (mV). This can then be used to determine an animal headings and/or orientation. In "Prying into the intimate details of animal lives: use of a daily diary on animals" Wilson et al. discusses using this data to create a dead reckoning system.

- Temperature - Commonly abbreviated to Temp
  Temperature (C) can be set to record the internal body temperature of the animal the device is attached to or the external locale temperature.

- Light Intensity - Commonly abbreviated to Lux
  Lux is a measure of the light intensity (Lux) in the surroundings of a device. Sensor is useful for deriving information about the local environment of an animal. Could be used for instance to determine if an animal is a dark place (e.g. a cave) or a light place (e.g. in the open).

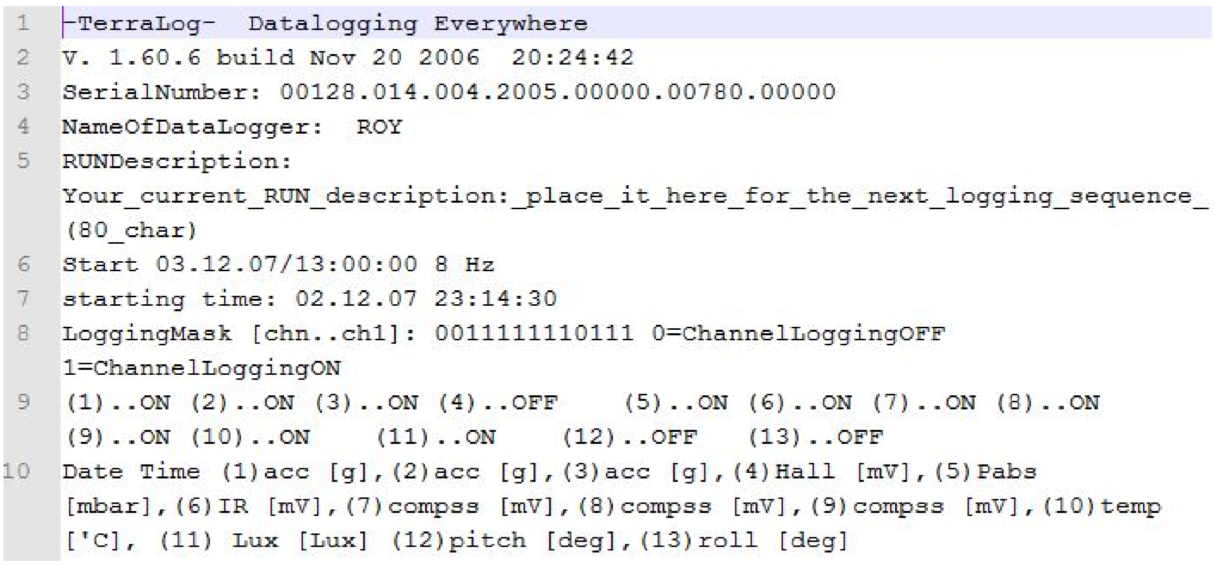
- Pitch
  Pitch is a measure of the body pitch of the device in degrees.

- Roll
  Pitch is a measure of the body roll of the device in degrees.

2.3.3 Specification of a Data File

One data file contains information recorded from a Daily Diary device. Data files are in ASCI format and are saved with a ".ASC" extension. A data file consists of a header (containing Meta data) and records (data recorded from the devices sensors).

The header section is split into 10 lines and specifies the characteristics of the recorded data. Figure 30 shows a header file from a data set.



```
1  -TerraLog-  Datalogging Everywhere
2  V. 1.60.6 build Nov 20 2006  20:24:42
3  SerialNumber: 00128.014.004.2005.00000.00780.00000
4  NameOfDataLogger:  ROY
5  RUNDescription:
   Your_current_RUN_description:_place_it_here_for_the_next_logging_sequence_
   (80_char)
6  Start 03.12.07/13:00:00 8 Hz
7  starting time: 02.12.07 23:14:30
8  LoggingMask [chn..ch1]: 0011111110111 0=ChannelLoggingOFF
   1=ChannelLoggingON
9  (1)..ON (2)..ON (3)..ON (4)..OFF    (5)..ON (6)..ON (7)..ON (8)..ON
   (9)..ON (10)..ON    (11)..ON    (12)..OFF   (13)..OFF
10 Date Time (1)acc [g],(2)acc [g],(3)acc [g],(4)Hall [mV],(5)Pabs
   [mbar],(6)IR [mV],(7)compss [mV],(8)compss [mV],(9)compss [mV],(10)temp
   ['C],  (11) Lux [Lux] (12)pitch [deg],(13)roll [deg]
```

Figure 30: Header section of a data set recorded from a Penguin.

A line by line breakdown of the header section now follows:

- **Line 1:** A static string consiting of the software the device is running.
  Typical value: "-TerraLog- Datalogging Everywhere"

- **Line 2:** A String consisting of the version of the device.
  Example value: "V. 1.60.6 build Nov 20 2006 20:24:42"

- **Line 3:** Contains a serial number which is unique to each data file.
  In the form: "SerialNumber: xxxxx.xxxxx.xxxxx.xxxxx.xxxxx.xxxxx.xxxxx" where x is an integer value. This should be unique to each data file.
  Example value: "SerialNumber: 00128.014.004.2005.00000.00780.00000"

- **Line 4:** Name of the data logging device.
  In the form: "NameOfDataLogger: " followed by a name.
  Example value: "NameOfDataLogger: Roy"

- **Line 5:** Description of data recorded.
  In the form: "NameOfDataLogger: " followed by a description up to 80 characters in length.
  Example value: "RUNDescription: Recording of a penguin in the Atlantic"

- **Line 6:** Contains the start time of data recording and recording frequency.
  In the form: "Start " followed by a date and time in the format: "day.month.year/hour:minutes:seconds". A space then separates this from the frequency. The frequency is an integer followed by " Hz".
  Example value: "Start 03.12.07/13:00:00 8 Hz"
  Note 1: Start time should unify with the first data recording line in the data set.
  Note 2: Frequency stated should also be the same as the frequency represented in the

data. So in the example string given above, 8 Hz means 8 lines of data should be shown for each second.

- **Line 7:** Starting time of recording.
  Example value: ”starting time: 02.12.07 23:14:30”

- **Line 8:** States which devices are being recorded.
  In the form: ”LoggingMask [chn..ch1] :” followed by a string of 13 binary integers, indicating which sensors are being logged. 1 represnts a sensor as being on, 0 represents off.
  A breakdown of the integers little endian (right to left) first:

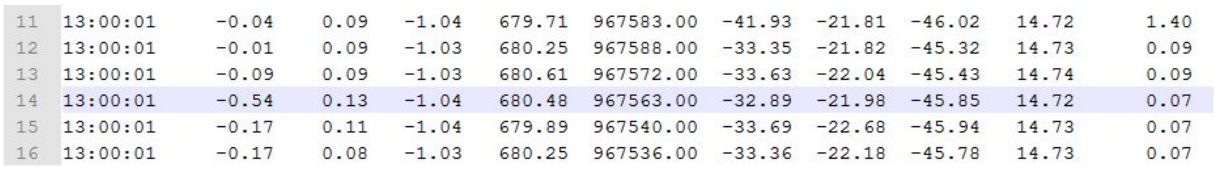
  1. Accelerometer X
  2. Accelerometer Y
  3. Accelerometer Z
  4. Hall
  5. Pabs - Pressure
  6. IR - Infer-Red
  7. Compass X
  8. Compass Y
  9. Compass Z
  10. Temp - Temperature
  11. Lux - Light Intensity
  12. Pitch
  13. Roll

  This is then followed by the string ” 0=ChannelLoggingOFF 1=ChannelLoggingON”
  Example String: ”LoggingMask [chn..ch1]: 0011111110111 0=ChannelLoggingOFF 1=ChannelLoggingON”
  This string would translate to: Accelerometer X, Accelerometer Y, Accelerometer Z, Pabs, IR, Compass X, Compass Y, Compass Z, Temp and Lux sensors being logged. Pitch, Roll and Hall sensors are not being logged.

- **Line 9:** Summarises the data shown on the previous line.
  This represents the data shown on the previous line in a more human readable representation. The Data is read from right to left and is then written with the position of the integer in brackets followed by ”..ON” or ”..OFF”.
  The data on line 8 would lead to the string ”(1)..ON (2)..ON (3)..ON (4)..OFF (5)..ON (6)..ON (7)..ON (8)..ON (9)..ON (10)..ON (11)..ON (12)..OFF (13)..OFF”

- **Line 10:** A Static string indicating what the numbers in the previous line correspond to.
  Static string: ”Date Time (1)acc [g],(2)acc [g],(3)acc [g],(4)Hall [mV],(5)Pabs [mbar],(6)IR [mV],(7)compss [mV],(8)compss [mV],(9)compss [mV],(10)temp ['C], (11) Lux [Lux] (12)pitch [deg],(13)roll [deg]”

Moving swiftly on to the data recordings section. Figure 31 shows a subset of an original data set, taken from the same data set as the header section in Figure 30. A Data recording can consist of anywhere from 100,000 lines up to 2 million lines of data. The format of a single line of data follows:



```
13:00:01    -0.04    0.09    -1.04    679.71    967583.00    -41.93    -21.81    -46.02    14.72    1.40
13:00:01    -0.01    0.09    -1.03    680.25    967588.00    -33.35    -21.82    -45.32    14.73    0.09
13:00:01    -0.09    0.09    -1.03    680.61    967572.00    -33.63    -22.04    -45.43    14.74    0.09
13:00:01    -0.54    0.13    -1.04    680.48    967563.00    -32.89    -21.98    -45.85    14.72    0.07
13:00:01    -0.17    0.11    -1.04    679.89    967540.00    -33.69    -22.68    -45.94    14.73    0.07
13:00:01    -0.17    0.08    -1.03    680.25    967536.00    -33.36    -22.18    -45.78    14.73    0.07
```

Figure 31: Data file of a Sub-set of 6 logged data measurements. This data is from a device attached to a Penguin.

A line of data consists of several attributes corresponding to the sensors being logged. For this task it is assumed that all the sensors are turned on. However if a sensor is not turned on the data is effectively skipped and the next sensor being recorded is written to the file. Data attributes are separated by a tab character.

- Attribute 1: Time (always displayed in data)
  Format: "HH:MM:SS"

- Attribute 2: Acceleration X
  Format: Floating point number in the range -6.00 to 6.00 [13].

- Attribute 3: Acceleration Y
  Same as Attribute 2.

- Attribute 4: Acceleration Z
  Same as Attribute 2.

- Attribute 5: Hall
  Format: Floating point number. Range not known.

- Attribute 6: Pabs (Pressure)
  Format: Floating point number in range 100.00 to 200000.00 [13].

- Attribute 7: IR (Infrared)
  Format: Floating point number in range 0.00 to 1000000.00 [13].

- Attribute 8: Compass X
  Format: Floating point number. Measurement range to maximum of earths magnetic field [13].

- Attribute 9: Compass Y
  Same as Attribute 8.

- Attribute 10: Compass Z
  Same as Attribute 8.

- Attribute 11: Temp (Temperature)
  Format: Floating point number in range -20.00 to 60.00 [13].
- Attribute 12: Lux (Light Intensity)
  Format: Floating point number in range 0.00 to 100000.00 [13].
- Attribute 13: Pitch
  Format: Floating point number in range -180.00 to 180.00.
- Attribute 14: Roll
  Format: Floating point number in range -180.00 to 180.00.

An example data line would be as follows:
Line 8: "LoggingMask [chn..ch1]: 0011111110111 0=ChannelLoggingOFF 1=ChannelLoggingON"
Recorded Line: "13:00:01 -0.04 0.09 -1.04 679.71 967583.00 -41.93 -21.81 -46.02 14.72 1.40"

2.4 Separating Dynamic and Static Components of Acceleration

Acceleration values recorded by Daily Diary devices are output as total acceleration values. Biologists are more interested in the separated dynamic and static components. As described in "Identification of Animal Movement Patterns Using Tri-axial Accelerometry" by Shepard et al. A static acceleration value can be found by parsing an appropriate smoothing filter over the data set. Once the static acceleration value has been found, the dynamic component can be found by subtracting the static component away from the total acceleration value. In this project we use two smoothing filter techniques: Moving Average filter and Savitzky-Golay Filter.

A Moving Average filter works on the principle of continuously creating a serious of averages from a specified window size throughout a data set. The window starts at the beginning of the data set and computes the average for the data item in the center of the window. After this value is computed the window shifts by one to the right and the process continues, repeating until the end of the data set is reached. Many different variations of the Moving Average filter exist, however, we are interested in implementing a central average filter, using previous and future data items. The pseudo-code below displays the filter for deriving the moving average.

```
// Iterate over data set. Note  Must take window size into account when
// setting loop bounds.
For (int p = (WindowSize / 2); p < DataSet.Size() - (WindowSize / 2); i++) {
    // Holds summation of values through window
    Float total = 0;

    // Iterate over window
    For (int i = -(WindowSize / 2); i < (WindowSize / 2); i++){
        total += DataSet[p+i];
    }

    // Store moving average some where
    MovingAverage[p] = total / windowSize;
}
```

Figure 32 displays the results for using the Moving Average filter over the total acceleration values. The Red line represents the total acceleration value and the green line displays the derived static component found using the Moving Average filter with a window size of 32. As can be seen the moving average creates a smooth line and displays a clear average.

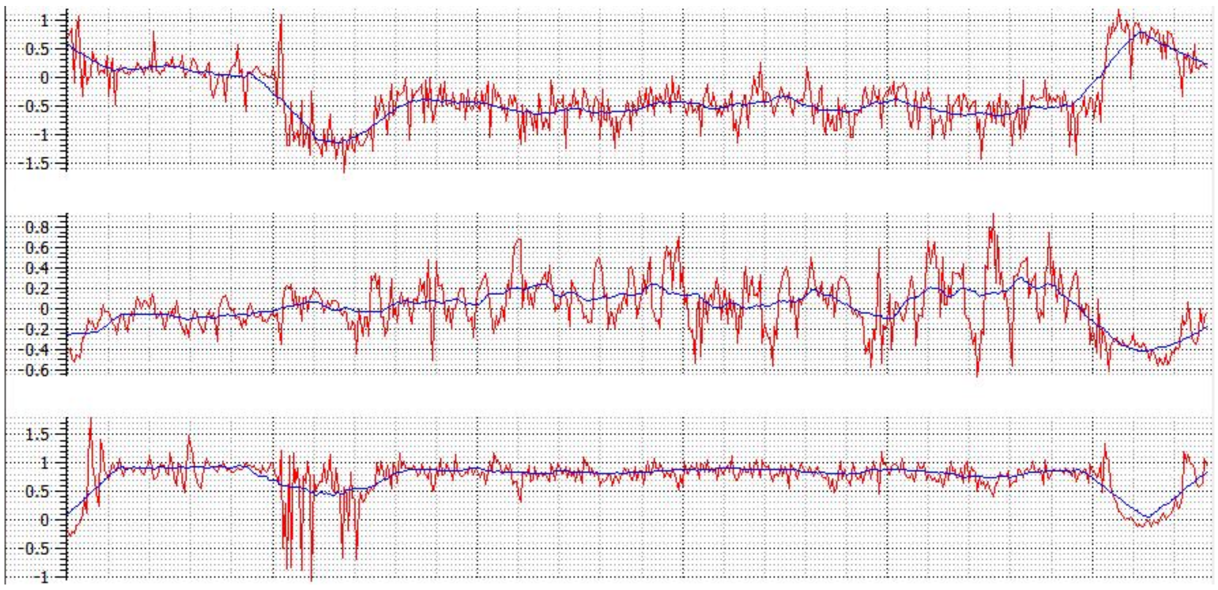


Figure 32: Stacked graph displaying tri-axel accelerometer data. The red curve displays total acceleration and the blue curve displays static acceleration derived using a windowed Moving Average filter.

Different window sizes affect the accuracy of the derived components. A larger window size will give a more accurate result for the average, however, it will also dampen any peaks in the total acceleration, meaning detail is lost. This is a trade off between accuracy and detail, clearly a huge disadvantage of the Moving Average filter.

The Savitzky-Golay filter offers the same benefits as the Moving Average filter except aims to preserve peeks and troths which were no longer existent after applying the Moving Average filter. The Savitzky-Golay filter works on the basis of a polynomial function from which filter coefficients are found which preserve higher moments[cite]. It is possible to adjust the orders of the polynomial function used although commonly it is of either quadtratic or quartic orders. Figure 33 displays the results of using a Savitzy-Golay filter using a quartic function with a window size of 32 data items. In comparison to the Moving Average filter it is clear that the static component found has maintained more features of the total acceleration curves while still computing a smoothed average curve.

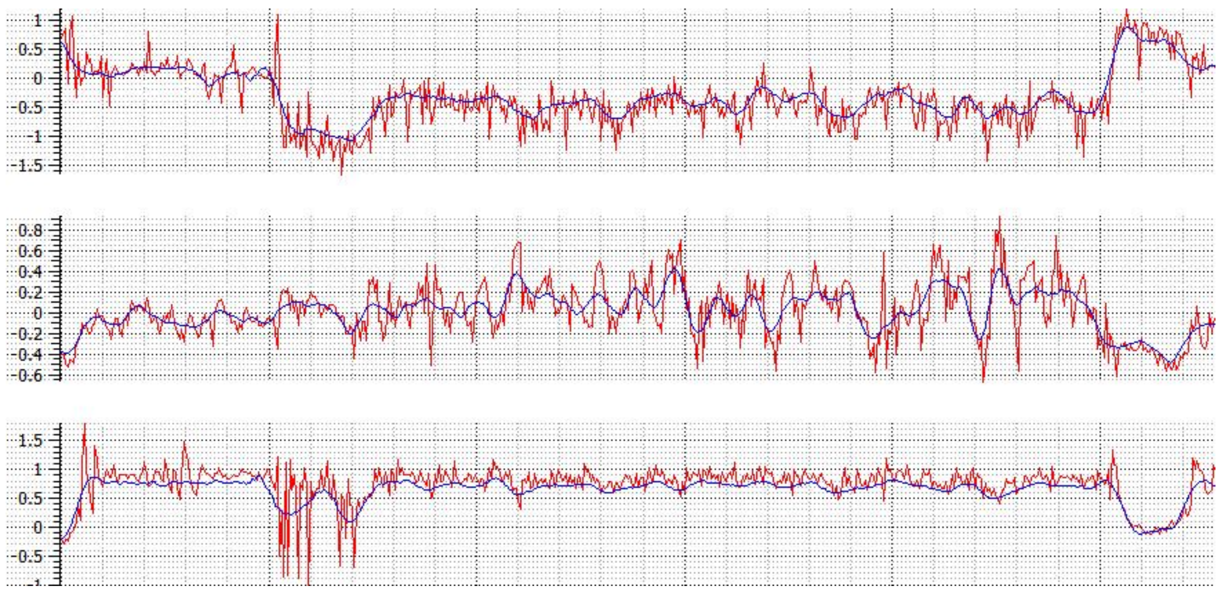


Figure 33: Stacked graph displaying tri-axel accelerometer data. The red curve displays total acceleration and the blue curve displays static acceleration derived using the Savitzy-Golay filter.

To conclude, Savityy-golay is a much better algorithm for our purposes in preserving the detail from total acceleration while creating a good average. However, the Moving Average filter is much faster to compute and is a technique already commonly used by the Biologists. It is therefore necessary to have both of these filters as a user option.

3 Project Specification

3.1 Feature Specification

The main objective of the project was to create a tool that can be used to visualise data from Daily Diary devices. In order to achieve this, the specification was broken down into a number of sub-goals to give clear aims.

Before creating the software requirements, it was necessary to compute a list of objectives which would make data analysis better and more knowledge rich for the biologists. This list can then be used to create a list of software requirements stating how goals will be achieved. We aim to do this by:

- **Making data analysis less reliant on skill of biologists.** Currently it requires many years of experience in data analysis to be able to retrieve knowledge from the 2D time intensity plots. Visualisations should be easy to understand and provide an intuitive step to the biologists as to what is being displayed.
- **Combining multiple attributes together into one visualisation.** Current visualisation techniques are limited to plotting one dimension of data. Combining data attributes together into one visualisation will make data analysis faster and visualisations more knowledge rich.

- **Enabling pattern finding capabilities.** Currently it is not possible to extract patterns from the data, as the only visualisation techniques used are the 2D time intensity plots. It should be possible to derive more knowledge from the data such as, common animal postures.
- **Quicker data analysis.** One of the biggest problems with the current visualisation techniques is the time biologists spend analysing data. Data analysis should be fast and have methods of automatically finding areas of interest.

Project requirements are split into two sections; required features and optional features. Required features consist of aims to bring the program up to the standard of the current visualisation tools biologists use. Optional features will take the application upwards and beyond any existing applications for visualising the data and aim to solve the objectives outlined for the biologists.

Requirements have changed dramatically throughout the lifespan of the project, since the initial and interim documentation periods. In the initial stages when the first initial document was produced there was not a great enough understanding of the project and data. This lead to an incomplete list of requirements. During the interim documentation stages there was a greater understanding of the project and it was possible to produce a more complete overview of the requirements. However, the requirements have since changed again, this is due to the nature of the project. Ideas for visualisation techniques often come about with time and experimentation with data, as such more visualisations were added to the optional requirements.

Required Features

1. File Reader
   Data from the Daily Diaries needs to be uploaded into the system. From this the visualisations and 2D data plotter can utilise the data for their own uses.
2. Data Processing
   Acceleration components should be processed using an appropriate smoothing filter as discussed earlier, this can then be used to separate the static and dynamic components of acceleration. Users should then be able to decide if they want to visualise static, dynamic or total acceleration.
3. Two Dimensional time intensity data plot
   Data is to be displayed in a standard recognisable format to the biologists, in an intensity verses time graph. A view displaying what the biologists are used to seeing will enable the biologists to use the existing visualisation method to aide in analysis alongside new visualisations created.
4. Context and Focus views
   Users should be able to select samples of data from the 2D data plots, for example picking data between 12:00 and 12:30. A whole data set will provide too much information to be reviewed at once, therefore by selecting a sub-set from the data and displaying it on a different 2D plot it is possible to see an overview and a focus of the sub-set of data, from which the user can closer inspect the graphs and label sections of animal behaviour.

Optional Features

1. Binary file writer/reader
   Due to the slow performance of reading ASCI data, a binary file reader/writer would potentially increase the speed of data processing. The original files could then have the option of being converted into binary for future input into the system.

2. Playback Mode
   A user option to play back through the data set in real time adds clarity to visualisations created and enables a user to view visualisations progress as the data set is played through.

3. 3D visualisations
   One of the challenges of this project is creating visualisations which achieve the objectives previously outlined. Many different visualisation techniques exist for displaying abstract data. Only visualisations that assist in data analysis of the animal tracking data should be chosen as a requirement and implemented in the application.

   The following visualisation techniques are optional requirements:

   (a) Scatter Plot
   A three dimensional scatter plot of acceleration or compass data will show the distribution of a subset of data selected. This can then be used to show anomalies and clusters of similar data.

   (b) Spherical Scatter Plot
   Acceleration and compass data is recorded as a vector quantity with a magnitude and direction. A spherical scatter plot removes the magnitude of any points and maps them to a fixed radius. It is necessary to normalise results as the strength of signals is not important, only the direction. This can then be used to show patterns in the data, such as, common animal orientations.

   (c) Spherical Histogram
   A Spherical Histogram represents the number of points in a segment (bin) of a sphere and projects the height in proportion to the number of points in that region. This visualisation is useful for showing patterns in the data. For example, visualising static accelerometer data will show common animal orientations and in compass data will show common animal directions. Although this visualisation has similar aims to a spherical scatter plot, points may be over crowded in a spherical scatter plot so the user may be deceived to the actual number of points in an area. A Spherical Histogram offers many more advantages, such as bin sizes can be altered and it shows more clearly the number of points in a specific region.

   (d) Triangle Fan
   A Triangle Fan visualises future and previous time steps during playback. A triangle fan segment is made up of 3 vertexes: one being the origin and the other two vertexes are data items. With a user option to set the amount of time steps being visualised this should display the transition between data items and assist users in perceiving more than one time step which would otherwise be seen during playback.

   (e) Central Glyphs
   Glyphs can be utilised to show the current acceleration and compass directions at a specific time step during playback. At the origin of coordinate space, the current data item being plotted would be shown as a vector. This increases the clarity during

play back, so the user can easily see the current vector. For example, the current orientation (acceleration) or the current heading of an animal (compass) .

(f) Animal Posture Diagrams
An Animal Posture Diagram visualises animal movement. Using the static component of acceleration it is possible to derive the pitch and roll of an animal at a specific time step. This enables the recreation of animal movement.

4. Assistance in indentifying areas of interest in the data
Instead of having to scan through the whole data set which could potentially take days, a user has some means of taking them to the next area of interest. For example, a high change in accelerometer value, potentially linking to an interesting animal movement or activity.

5. Colour Mapping
Mapping colour to visualisations will strengthen boundaries and can be used to display more attributes of data in a visualisation.

6. Interactive Techniques
Interactive visualisations which the user can interact with enable a user to make the visualisations more specific to their needs. Examples of such interaction techniques include: moving the visualisations around via rotation, zooming in on areas of interest and changing colour mappings.

7. Dead reckoning system
Linking compass and accelerometer data together could produce a dead reckoning system Inferring the location of an animal by taking in a number of factors such as direction and speed. A similar idea is used on submarines to a high accuracy.

8. Automatic identification of behaviours
Once an activity has been identified, the signature pattern for it could be stored in a database. Next time a dataset is uploaded to the program, segments of a plot could be automatically labelled using automatic pattern matching techniques.

3.2 Technology Choices

Technology choices for this project were of a high importance. Ideally the tools being used needed to be best suited to implement the requirements of the project and aid in creating a successful product with ease. In the initial and interim documentation stages, a wide variety of technology choices were discussed and compared to ensure the right choices were made. Here a summary of the technology choices used throughout the project are presented.

3.2.1 Programming Language

The programming language to be used throughout the project was required to be imperative and object orientated, to maximise code reuse and maintainability. C++ was found to meet both of these requirements and has therefore been used to implement our application.

C++ is the industry leading programming language, created by Bjarne Stroustrup in 1979 [14]. C++ code is compiled directly into machine language making C++ a very high performance language. The language combines high-level and low-level language features. C++ does not provide a garbage collector for memory, instead the programmer is in charge of allocating and releasing memory throughout the programs life span, if not done correctly this can cause memory leeks, a problem in many applications.

C++ has proved to be a good choice, despite the lack of garbage collector, memory leaks have been avoided by using various memory tracking programs. C++ has also offered good support for OpenGL, the graphics library used throughout the project. This has assisted in adding to the success of this project.

3.2.2 GUI Library

A Graphical User Interface (GUI) Library was required to go alongside C++ to provide a means of creating a graphical user interface, allowing a user to interact with our program. Qt 4 was selected for this purpose.

Qt 4 is a cross-platform application and User Interface framework [18]. Qt is open source and is for use with C++, although can be used with other languages, such as: Java and Python. Qt provides a GUI library as well as a relatively large generic library of code. Code written in Qt can be compiled and run natively on Windows, Linux, Mac OS and embedded Linux without any source code changes [6].

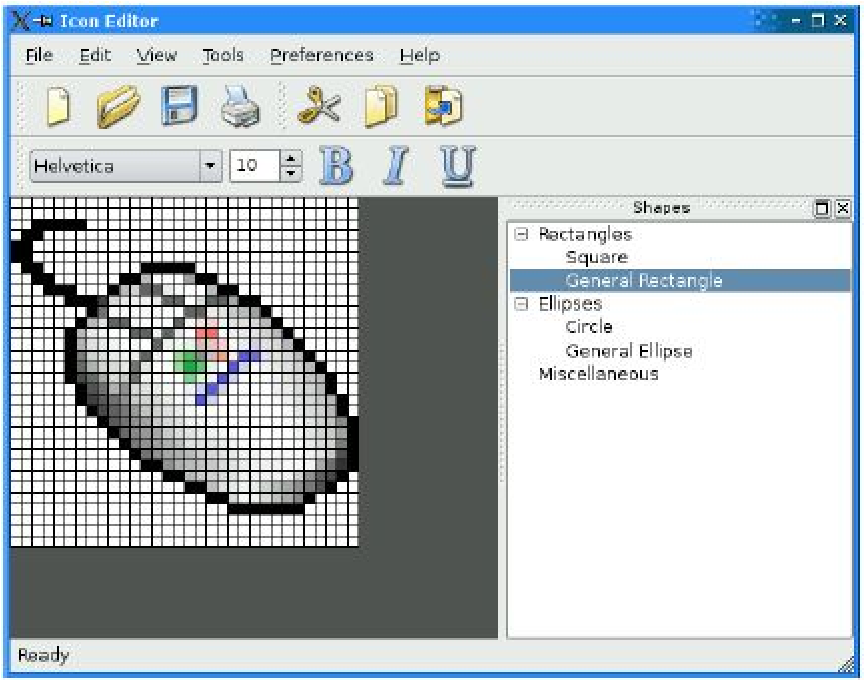


Figure 34: An applicaiton written using the Qt 4 GUI library. [6]

Qt 4 has provided many features which have proven beneficial to the project. The library code included with Qt 4 has saved a vast amount of time and has made programming in C++ a real pleasure. Qt has also provided great support when problems have arisen. The official QT forum and mailing lists have both been used for support when code issues have arisen, this has allowed problems to be overcome quickly without falling behind on deadlines.

3.2.3 2D Plotting

In the project requirements section, a must have feature was to keep the existing visual aids, the 2D time intensity plots. In order to do this a 2D data plotting library was required. Qwt a plotting tool compatible with QT was chosen for this.

Qwt is a library that contains 2D plotting tools and widgets for plotting data [16]. Qwt can plot many different types of graphs; Scatter plots, histograms, curve plots and many more. Qwt can also be used to plot large data sets and has caching features to improve performance. Included with the installation is a large amount of example programs, demonstrating how to use the library and tools.

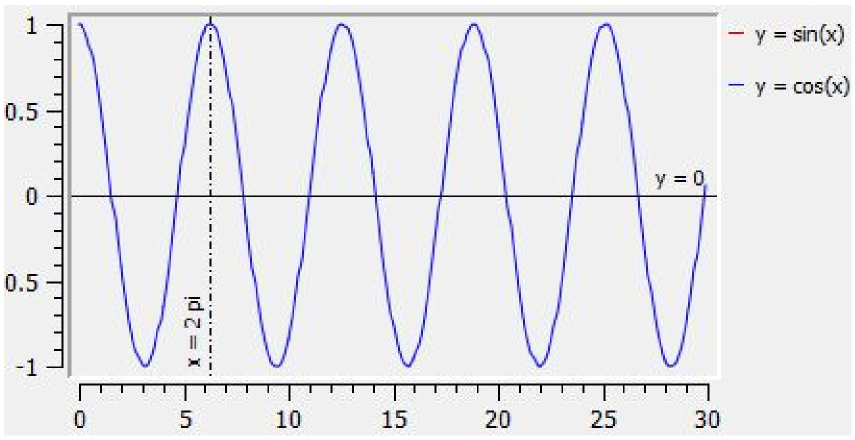


Figure 35: A curve plot created using Qwt.

Qwt is an already produced solution for plotting data and has therefore allowed the project to continue on more pressing matters such as creating advanced visualisation instead of re-inventing the wheel by creating a plotting tool. The performance Qwt has brought has been beneficial, large data sets containing over a million data items are painted quickly and can be cached to enable a better experience for a user using the system.

3.2.4 Graphics Library

A Graphics Library is needed to produce 3D computer graphics, this enables us to map data items into a spatial domain. OpenGL was the graphics environment chosen for this.

OpenGL is an open source cross platform graphics environment and is one of the most common graphics libraries used across the graphics domain. OpenGL offers a high performance 3D renderer, which can be used to render millions of primitives at a high speed. QT4 offers good support for OpenGL, a class exists called QGLWidget which provides functionality for displaying OpenGL graphics integrated into a Qt application.

OpenGL has proven to be a good choice for creating our visualisations. Qt 4 has made it simple to integrate OpenGL windows into the application. It is easy to control environment conditions such as camera coordinates or rotation by the use of functions. Objects can be created with ease by using primitives or by using predefined objects that are available in toolkits which come integrated with OpenGL. Because of OpenGLs open source nature it has a large community of users who can be contacted for assistance via forums and mailing lists, proving to be another useful resource during implementation.

4 Project Design

A clear object-oriented structure has been designed to make the implementation stages quicker and code more maintainable. A good design has been developed which allows for additional features to be added with ease in the future.

4.1 Classes and Descriptions

Each class will now be stated and its purpose in the program will be discussed.

4.1.1 File Handler

The following classes are for use in the data management aspect of the program. This includes file reading and data storage in memory.

- **Class: DataItem**
  A trivial class which stores a row of data and is used for accessing and setting recordings from logged sensors. More information on what fields the object contains can be found in the Data Characteristics section. On top of this data, it is also used to store pre-processed data such as static and dynamic components of acceleration.
- **Class: DataSet**
  A class which stores a complete data set made up of DataItem classes. The class also stores various meta-data which can be found in the header of a data file. A data set is stored in a list of DataItem classes, from which these can be accessed and utilised for manipulation in other classes.
- **Class: FileHandler**
  A generic class for reading in a data file. It implements basic file reading functionality, such as opening and closing a file, getting the size of a file etc. This class is inherited by classes implementing reading daily diary data functionality.
- **Class: ASCIDataHandler**
  A class used for reading data files output by Daily Diary devices. Files are read line by line and then stored in a DataItem class. All of the combined DataItem classes are stored in the DataSet class for future usage by other classes.
- **Class: BinaryDataHandler**
  A class used for reading and writing Binary data files. Due to the slow reading speeds of ASCI files a Binary file reader/writer is needed to make file reading faster. A data set stored in the DataSet class can be written to a Binary Data file using this class. Binary files can then be read into the DataSet class when required.

4.1.2 GUI

The following classes are Graphical User Interface (GUI) classes, used for enabling the user to interact with the system.

- **Class: MainWindow**
  A class for creating a new window component, containing the main GUI for the application. It is responsible for communication between GUI components, such as the 3D viewer and context + focus windows as well as allowing a user to interact with the system.

- **Class: SettingsDialog**
  A class which provides a GUI to the Settings class. Responsible for allowing a user to change various program settings via a user interface, such as sampling rate and number of sphere segments.

4.1.3 2D Plotter

The following classes define the structure for a 2D plotting widget used for plotting time intensity graphs.

- **Class: Plot**
  A virtual class which constructs a 2D time intensity graph. Must be extended to define how data plot curves are attached. For example, the MultiplePlotWidget and SinglePlotWidget classes extend from this class. The MultiplePlotWidget class constructs stacked graphs where as the SinglePlotWidget class plots a single attribute of data.

- **Class: MultiplePlotWidget** A Class which constructs and displays a 2D time intensity graph. Allows multiple attributes of data to be painted on top of each other for direct comparison. Each plot curve is given a unique colour for identification.

- **Class: SinglePlotWidget** A class which constructs and displays a 2D time intensity graph for one attribute of data. Attributes being displayed can be changed. A plot curve is mapped to a colour mapping function defined in the AmplitudePlotCurve class.

- **Class: PlotAreaWidget**
  A generic class to create and manage multiple plots (Generic plot classes). Plots can be added or removed providing a user with a visualisation displaying stacked time intensity graphs, aligned vertically for direct comparison.

- **Class: ContextPlotArea**
  A class to create and manage multiple plots. Plots can be added or removed and are painted to the screen, this functionality is inherited from PlotAreaWidget class. This class is intended to display the whole data set providing a user with an overview of the data set. Functionality exists so sub sets of data can be selected by the user. Note - this is the same as plotareawidget but with ability to select sub sets of data by user interaction with the widget.

- **Class: FocusPlotArea**
  A class to create and manage multiple plots. Plots can be added or removed and are painted to the screen, this functionality is inherited from PlotAreaWidget class. This

class is intended to display the subset of data selection from the ContextPlotArea and as a result has functions to change selection of data being painted. On top of this there is additional functionality to paint a line as to where the playback is positioned while the user is playing through the data set.

- **Class: AmplitudePlotCurve**
  A class to draw a line on a 2D plot of input data utilised in the Plot class. This class allows for different colour lines to be drawn depending on the colour mapping technique choosen using the ColourManager class.

- **Class: PlotSelection**
  A trivial Class to manage the area of data selected. Holds information on where lines should be painted on the FocusPlotArea and converts screen coordinates mapping to plot coordinates.

4.1.4 Primitives

The following classes are trivial and used for defining basic primitive objects to be used during the rendering process and mapping data to primtives.

- **Class: PointPrimitive**
  A trivial class for defining a point in 3D space with additional functionality for converting to spherical coordinates.

- **Class: QuadPrimitive**
  A trivial class for defining a quad in 3D space with additional functionality to determine a surface normal.

- **Class: CubePrimitive**
  A trivial class for defining a cube in 3D space with additional functionality to project a quad using a surface normal vector.

4.1.5 3D Viewer

The following classes define the usage of the 3D viewer package used to visualise acceleration and / or compass data.

- **Class: ViewerWidget**
  A class for displaying 3D data visualisations. This class enables various visualisations to be rendered to the screen using OpenGL. Visualisations are treated as layers and so this class manages which visualisations should be displayed and which should not. This class also communicates with the visualisations during play back to render them up to the appropriate time step.

- **Class: SphericalPlot**
  A class for rendering a spherical plot (In spherical coordinates) using the data set in the class DataSet of either acceleration or compass data. During playback the visualisation is rendered up to a certain time step.

- **Class: SphericalHistogram**
  A class for rendering a spherical histogram plot using the data set in the class DataSet of either acceleration or compass data. During playback the visualisation is rendered up to a certain time step.

- **Class: CentralGlyph**
  A class for rendering a central glyph showing the current static acceleration vector or compass vector, dependant on what data type is being displayed. A glyph is only painted during playback of the current vector.

- **Class: AnimalOrientation**
  A class for creating an animal orientation animation. Utilises the central glyph and spherical plot components.

- **Class: Lighting**
  A class for controling the light sources in an OpenGL scene. A user can control lights by turning them on or off.

- **Class: PenguinModel**
  A class for rendering a 3D geometric Penguin in OpenGL.

- **Class: RawPlot**
  A class for rendering a raw plot in OpenGL of 3D compass or 3D acceleration data.

- **Class: TriangleFan**
  A class for rendering a triangle fan in OpenGL of acceleration data.

- **Class: ViewerScene**
  A class for encapsulating the rendering of the default scene for the 3D visualisation windows. Renders the central sphere and axis. Along with handling user interaction, such as: synchronised rotation and zooming.

- **Class: ViewerVisualisation**
  A class for defining the basic interface of a visualisation. Implements basic functionality such as mapping data values to a colour.

4.1.6 Global

The following classes are 'global' and are used by more then one package in the application.

- **Class: Settings**
  A trivial static class for storing global application settings.

- **Class: PlayBack**
  A class used for defining a timer to play back through the data set. The rate each time step is itterated over is user defined, although, by default this is the same speed the data was recorded at.

- **Class: ColourManager**
  A class for mapping a data attribute to a rainbow colour value.

4.2 Process Diagrams

Process diagrams show how data flows through the system and the outputs expected. Below is the process diagram for the proposed application.

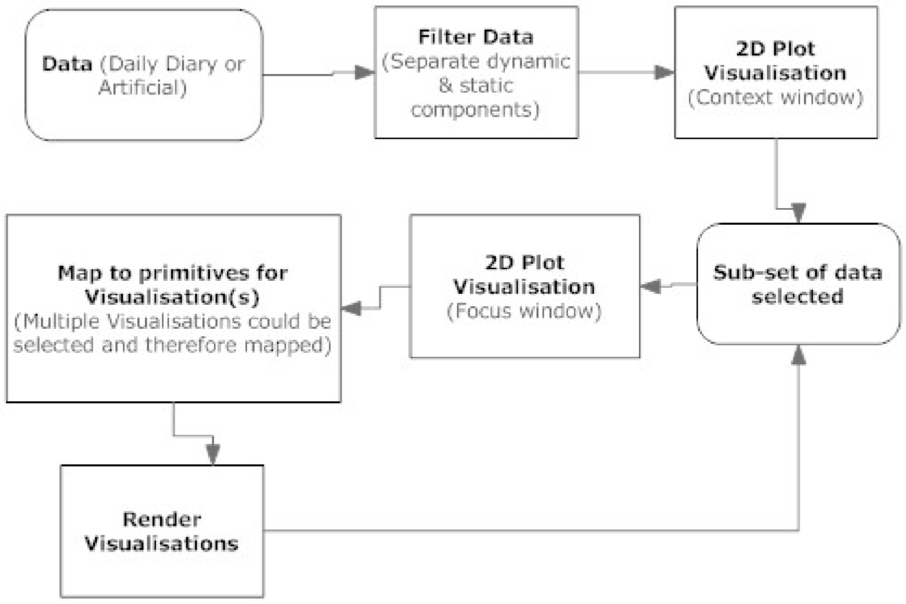

Data (Daily Diary or Artificial)
Filter Data (Separate dynamic & static components)
2D Plot Visualisation (Context window)
Sub-set of data selected
2D Plot Visualisation (Focus window)
Map to primitives for Visualisation(s) (Multiple Visualisations could be selected and therefore mapped)
Render Visualisations

Figure 36: A Process Diagram displaying how data flows through the system. Rounded rectangles display input/output data and square rectangles display processing stages.

4.3 Class Hierarchy

Class hierarchy diagrams show the structure of the design classes and display their relationships between other classes such as the use of inheritance or interfaces. Outlined below are the various hierarchies of classes. Only classes which are inherited from other classes are shown, all other classes are flat and therefore do not extend from any other classes.

Figure 37 shows the class hierarchy of the file handler options. ASCI or Binary data files can be read into the system.

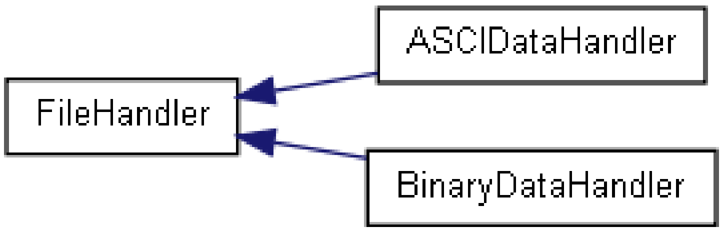


Figure 37: A class hierarchy diagram of the file handler options.

Figure 38 shows the class hierarchy of the plot area options. In the application focus and context windows are used to display data in 2D time intensity plots.

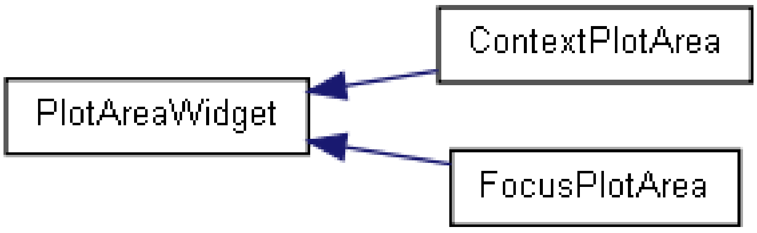


Figure 38: A class hierarchy diagram of the plot area options.

Figure 39 shows the class hierarchy of the Viewer Widget . This class is used to render visualisations.

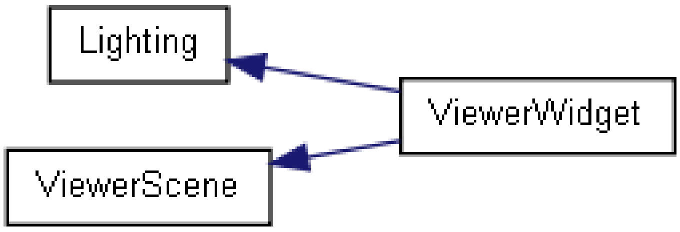


Figure 39: A class hierarchy diagram of the viewer widget class, used to render visualisations.

Figure 40 shows the class hierarchy of the visualisation sub-systems. ViewerVisualisation class is used to define visualisations and enable generic options such as colour options. This class allows visualisations to be rendered in the ViewerWidget class.

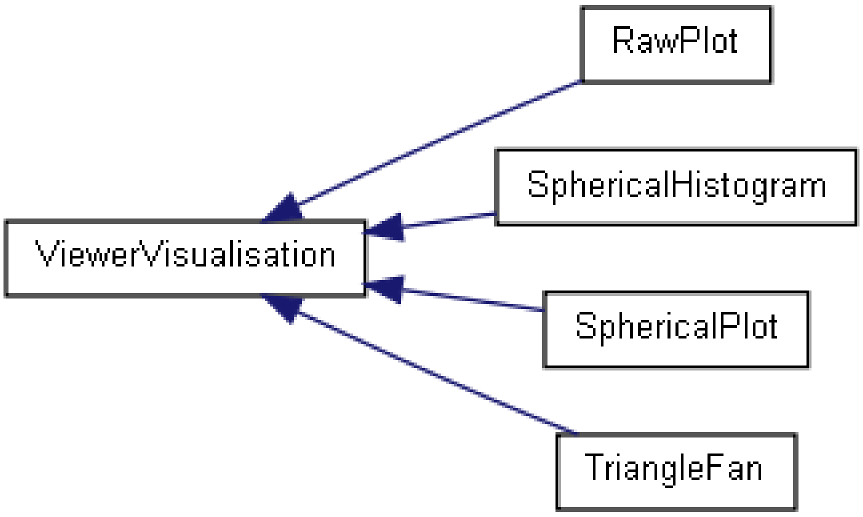


Figure 40: A class hierarchy diagram of the visualisation sub-systems used to define the behaviour of visualisations created in the application.

All of the class hierarchy diagrams have been created using Doxygen and are available to view online at the following URL:

`http://cs-sol.swan.ac.uk/~cs521866/Doxygen`

4.4 Collaboration Diagrams

Collaboration diagrams show class relationship between components in the system. They graphically represent an is-part-of relationship between classes, that is, they show the relationship between classes that rely on other classes to work correctly and perform their tasks.

To minimise the size of the document, only a selection of collaboration diagrams are shown. This is because some of the diagrams are large and are often repetitive. All of the class collaboration diagrams, along with the class hierarchy diagrams can be found at the following URL:
`http://cs-sol.swan.ac.uk/~cs521866/Doxygen`

Figure 41 shows the collaboration diagram for the ASCIDataHandler class. This class is responsible for reading in ASCI data files produced from the Daily Diary devices.

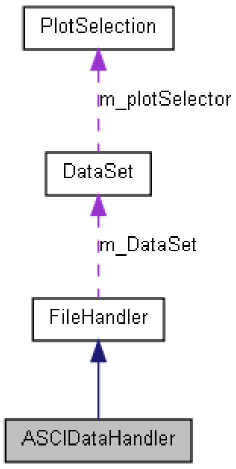


Figure 41: A collaboration diagram for the ASCIDataHandler class.

Figure 42 shows the collaboration diagram for the AnimalOrientation class. This class is responsible for replaying animal orientation. As can be seen this class collaborates with the glyph class, penguin model class and scatter plot to produce the visualisation required.

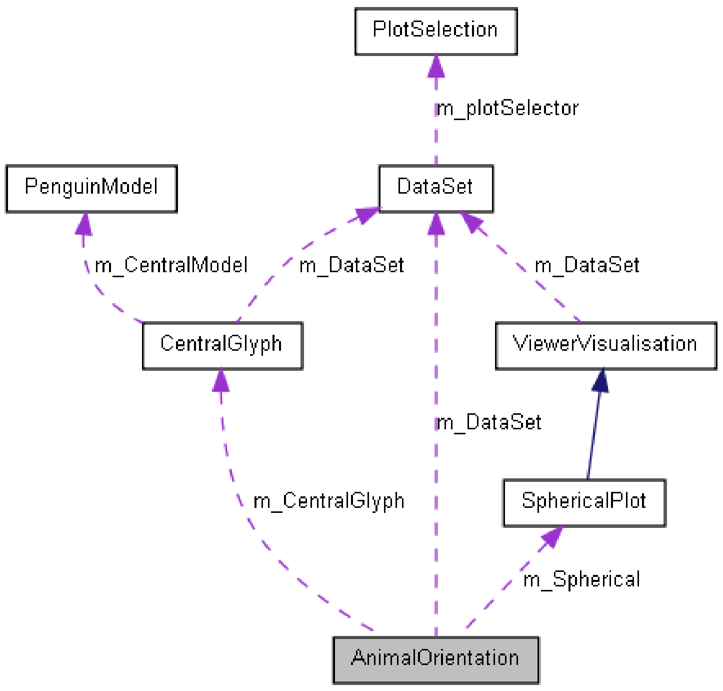


Figure 42: A collaboration diagram for the AnimalOrientation class.

Figure 43 shows the collaboration diagram for the 'CentralGlyph' class. The Central glyph is used to show the current acceleration or compass vector during playback.

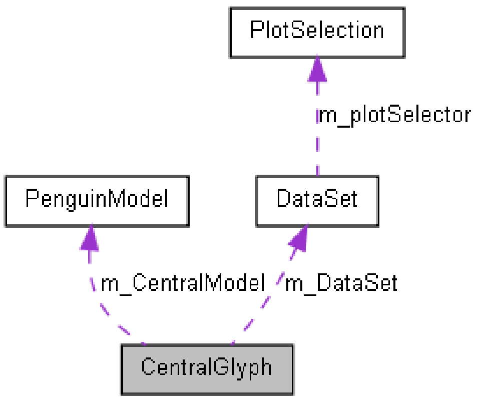


Figure 43: A collaboration diagram for the 'CentralGlyph' class.

Figure 44 shows the collaboration diagram for the 'SphericalHistogram' class.

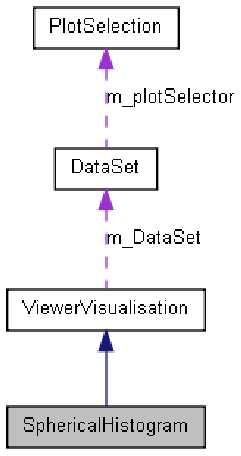


Figure 44: A collaboration diagram for the 'SphericalHistogram' class.

Figure 45 shows the collaboration diagram for the 'PlotAreaWidget' class. The Plot area widget class is used to produce 2D time intensity plots of logged data.

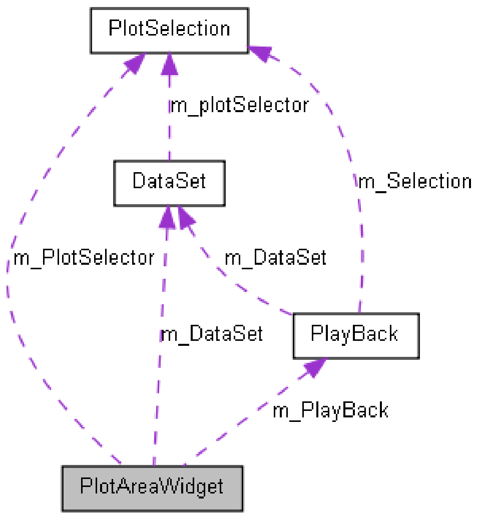


Figure 45: A collaboration diagram for the 'PlotAreaWidget' class.

5 Project Plan and Timetable

5.1 Software Development Model

A Software development model is a defined structure for developing software, consisting of multiple stages. Each stage defines a process which must be completed in order to continue to the next stage in the model (e.g. design would lead to implementation in the waterfall model), this is repeated until the project is complete. Many development models exist, each with its own advantages and disadvantages. The choice of development model is 'based on the nature of the project, application, the methods and tools to be used, controls and deliverables that are required [12]. A selection of software models were investigated during the initial and interim documentation stages. After discussing the advantages and disadvantages of a wide variety of models, a conclusion was made to the use the Incremental Model.

The Incremental Model is built on the same concepts as the Waterfall model. The waterfall model consists of the following states: analysis, design, coding and testing. Each state directly flows onto the next once completed. Instead of just one project cycle the Incremental Model consists of multiple Waterfall models which are applied incrementally.

Figure 46 shows the incremental model, At every linear sequence a new deliverable is produced, allowing for each additional project deliverable to be tested as a beta at each stage. For example, a 'word-processing software developed using the incremental model might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.' [12]

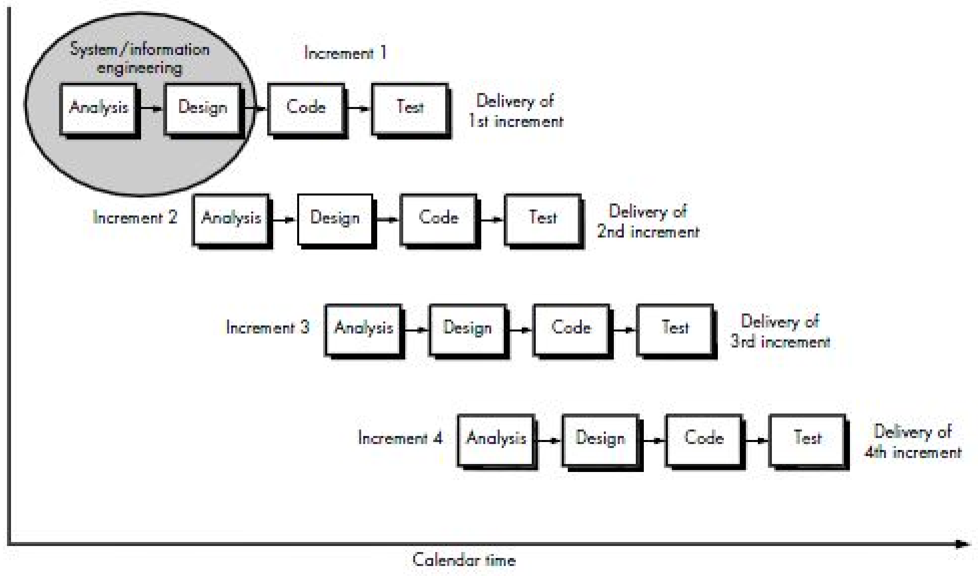

System/information engineering
Increment 1
Analysis
Design
Code
Test
Delivery of 1st increment
Increment 2
Delivery of 2nd increment
Increment 3
Delivery of 3rd increment
Increment 4
Delivery of 4th increment
Calendar time

Figure 46: The Incremental software development model [12].

The incremental model allows for maximum code re-use, as the previous code is always being built on to continually add more functionality until the product is complete. Requirements can change and be easily dealt with. Each component can also be tested individually before being built upon so errors can be spotted early on. If the requirements are not clear and already decided upon this model can pose problems with regards to adding new features when the requirements are complete. The software architecture may not allow for the new features to be integrated without a major system rewrite an expensive and time consuming process.

Throughout the projects development the incremental model has been used. Compared to the other models investigated, the incremental model offered the advantageous feature of being able to create a working product early on in the software life cycle - A nice feature which has enabled visualisations to be tested as soon as a new visualisation is implemented. It was also unclear during the initial documentation period of the complete requirements of the software product. Using this software model, the requirements have been added to as a greater understanding of the project has been acquired, other software models would not have allowed for this.

5.2 Coding Conventions

Bobs Concise Coding Conventions (C^3) [9] have been used throughout the implementation stages of the project. Bob's conventions outline a series of rules to follow while writing code in regards to the format of code. The use of such conventions makes code easier to read and modify if/when the code needs to be updated, contributing to the success of the software product.

5.3 Commenting Code

Doxygen is a source code documentation generator for various programming languages including C++. Doxygen has been used throughout the implementation stages of the project to comment code. It is important to comment code for future reference, instead of having to read code to know how a function or class works to understand what it does, commenting code allows us to abstract away from the code and state what a function does without having to know how. Doxygen also creates various diagrams of the structure of a software product, allowing for a future developer to instantly see how the system works without having to read thousands of lines of source code.

All source code documentation is available at the following URL:
`http://cs-sol.swan.ac.uk/~cs521866/Doxygen`

5.4 Timetable

The timetable shows the time plan for the project. Milestones are listed with their expected date of completion and actual date of completion to show the progress made throughout the projects life span.

Project Timetable

	Description	**Expected Date**	**Completed Date**
1.	Initial project document deadline	25 Oct 10	25 Oct 10
2.	File Reader	30 Oct 10	1 Oct 10
3.	2D Plotter - Context Window	6 Nov 10	19 Oct 10
4.	Binary File Reader	13 Nov 10	23 Nov 10
5.	Initial Project Presentation (Gregynog)	15 Nov 10	15 Nov 10
6.	2D Plotter - Focus Window	15 Nov 10	2 Nov 10
7.	Playback Mode	20 Nov 10	5 Nov 10
8.	3D Visualisation Windows	27 Nov 10	9 Nov 10
9.	3D Scatter Plot	27 Nov 10	9 Nov 10
10.	3D Spherical Plot	10 Dec 10	12 Nov 10
11.	Data Processing - Moving Average	5 Dec 10	5 Dec 10
12.	Project Review	7 Dec 10	7 Dec 10
13.	Colour Mapper	15 Dec 10	15 Dec 10
14.	Spherical Histogram	10 Feb 11	1 Feb 11
15.	Interim document deadline	14 Feb 11	14 Feb 11
16.	Glyphs (Acceleration + Compass)	15 Feb 11	22 Feb 11
17.	Colour Legend	22 Feb 11	22 Feb 11
18.	Interaction - Syncornised Orientation	1 Mar 11	1 Mar 11
19.	Triangle fan	8 Mar 11	15 Mar 11
20.	Data Processing - Savitzky-Golay	15 Mar 11	15 Mar 11
21.	Dissertation Outline deadline	29 Mar 11	29 Mar 11
22.	Animal Posture Diagrams	1 April 11	1 April 11
22.1.	Compute pitch and roll	23 Mar 11	23 Mar 11
22.2.	Animal Geometry	1 April 11	1 April 11
23.	Demonstration and viva deadline	18 May 11	18 May 11

6 Implementation

6.1 Overview of Implemented Features

Listed below is a basic summary of the features implemented in regards to the requirement features outlined in the feature specification section. A tick is placed next to items that have been completed and a cross next to requirements which have not been met.

Required Features:

1. File Reader ✔
2. Data Processing ✔
3. Two Dimensional time intensity data plot ✔
4. Context and Focus views ✔

Optional Features

1. Binary file writer/reader ✔
2. Playback Mode ✔
3. 3D visualisations ✔
   (a) Scatter Plot ✔
   (b) Spherical Scatter Plot ✔
   (c) Spherical Histogram ✔
   (d) Triangle Fan ✔
   (e) Central Glyphs ✔
   (f) Animal Posture Diagrams ✔
4. Assistance in indentifying areas of interest in the data ✔
5. Colour Mapping ✔
6. Interactive Techniques ✔
7. Dead reckoning system ✗
8. Automatic identification of behaviours ✗

6.2 Description of Implemented Features

The Implemented features will now be discussed. Each of the implemented requirements will be described and explained to give the user an understanding of how the visualisation software created works. The Incremental software model has been used throughout the implementation stages and as such the implementation process will be described in this way, adding additional functionality until the application is complete.

6.2.1 Basic Features

1. **ASCI File Reader**
   An ASCI file reader is required to read into the system the raw data files produced by the Daily Diary devices. Once read in, data can then be utilised by other sub-systems for their tasks, after all the data is the driving force behind the application.

   QT (the framework being used in conjunction with C++) provides various libraries for reading data from files. The two libraries being used are QFile and QTextStream. QFile is a class that provides methods for reading from and writing to files. QTextStream is an interface for reading and writing text from a QFile object. QTextStream is used with QFile to conveniently fetch data line by line. Reading the data in this way means the header section gets processed first. On a line by line basis the header file is split using regular expressions to extract logged data, discarding any static data which serves no purpose in the application.

   One of the challenges of reading the header section of the ASCI data files is that they do not conform to the same grammar specification. Many different Daily Diary devices have been produced and as a result they output different formatted files. Whilst it is possible to produce many different lexical analysers which read in the data files, this is not the aim of the project. A decision was made to focus on one Daily Diary version, specifically, version 1.60.6. It is important to note the system has been designed in such a way that more file readers could be added which have functionality to read in other versions if it is required at a later date.

The data section of the data files which logs measurements from the sensors is defined on a record per line basis. Because of this each line read in by the file reader can be stored as a record. Each line of data needs to be split to be able to extract each of the logged sensor measurements. Fortunately, each sensor measurement is separated by a tab; this makes data extraction much simpler than if values were separated by non constants. It is not required to have different regular expressions for each attribute and instead attributes can be extracted by using a string tokenizer. A string tokenizer ignores white space and returns a list of extracted values which were previously separated by while space. Extracted values can then be stored in a QList, allowing all of the data items to be placed in main system memory for future access.

A problem encountered during a performance analysis of the ASCI file reader was the amount of time taken to read a data file, taking around 60 seconds (system dependant) to read a 74mb file containing 864,316 data items. Reading and extracting ASCI text requires a lot of processing, especially with such large data files. A quicker option is to produce a binary data file writer/reader of which a user has the option of converting the .ASC files to binary format and then reading from these files in future use.

A GUI has been created to allow the user to interact with the ASCI file handler. A user can interact via a File menu from which files can be opened. Figure 47 shows this.

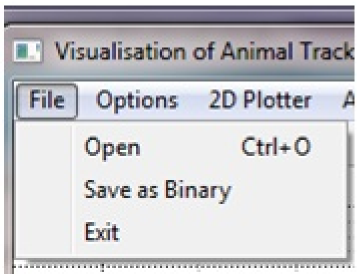


Figure 47: File Menu for opening data files.

When the files are being processed a progress bar displays how much of the file has been processed so far. Figure 48 shows this.

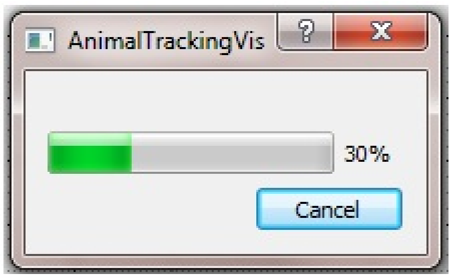


Figure 48: Progress bar shows 30% progress in processing a data file.

2. **Two Dimensional data plot**
The purpose of the two dimensional data plotter is to utilise the data read in from the file reader and plot it to an intensity verses time graph. A user should be able to select which data attributes to plot. This basic functionality will eventually be utilised to create the focus and context plotters.

   In the technology choices section it was discussed how the Qwt library will be used to plot graphs. Plotting on a basic level is easy using Qwt. Data to be plotted is converted to a two dimensional array and then uploaded into a QwtPlotCurve to be displayed on a QwtPlot. Figure 49 illustrates the result of a basic line plot using data from a Daily Diary device.

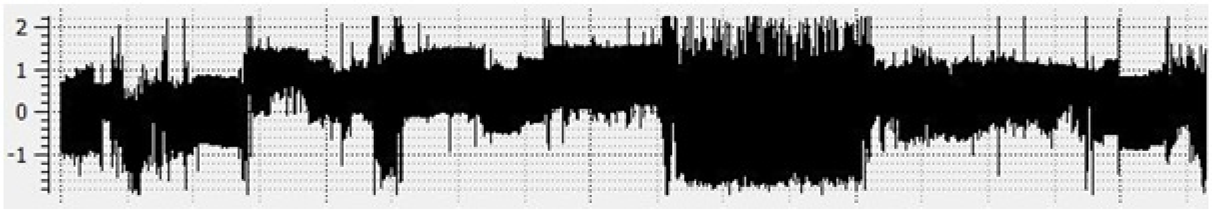


Figure 49: Initial plot created using Qwt.

   Although this is a nice starting point, the Qwt API code needs to be extended in order to achieve the requirements set out. To make achieving this simpler to implement the requirements are broken down into sub problems. These are listed below:

   - Stacking Graphs on top of each other.
   - Adding / removing attributes of data.
   - Colour mapping data.
   - Adding / removing line plots of acceleration in graphs.

   Stacking graphs on top of each other requires a visual appearance of the graphs being linked together. On first thought this sounds like a job for a QT layout manager, placing several plotting widgets on top of each other in the layout. However, this is not ideal, later on in the project it will be required to paint over the graphs to add functionality for the current playback position to be displayed, which is not possible using the standard functionality of a layout manager. Instead a custom widget is required which implements its own painter function and paints the plots being displayed inside of it. By doing this we are required to implement our own functionality for adjusting the size of plots contained when plots are removed or added. Figure 50 displays the plots using a custom painted widget.

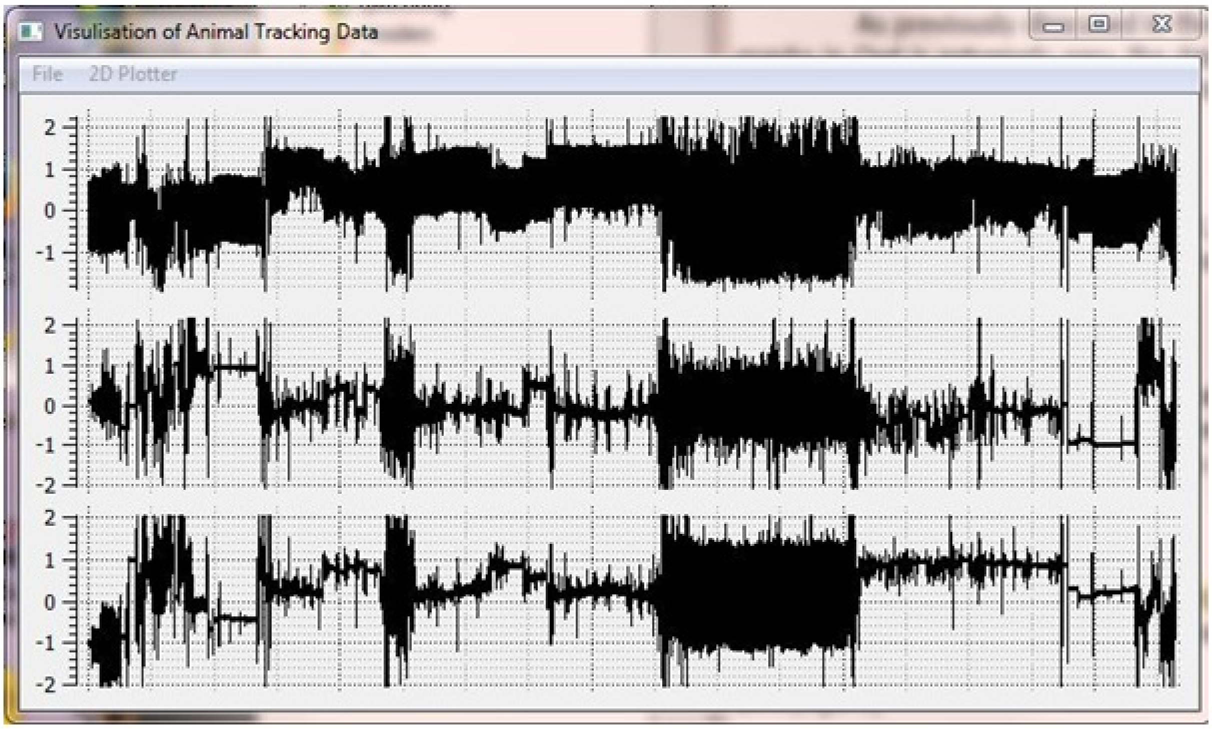


Figure 50: Stacked plots using a custom layout manager.

Adding and removing plots of data requires a class to be created which manages this process. The custom layout class for the widgets can be extended for this. By storing plots in a list in the layout class, plots can be simply added or removed. In order to paint plots stored in the list to the screen, they need to be iterated over, calling the appropriate paint function on each plot. Additional measures need to be put in place to ensure graphs cannot be added which are not logged by the Daily Diary device or ensure plots can be added which are already existent in the layout and the same for the remove function.

To create an interface from which multiple plots can be displayed toolbars were created with options where plots can be added or removed (Figure 51 shows the various options). Only data recorded by the device can be used as an option to produce a graph.

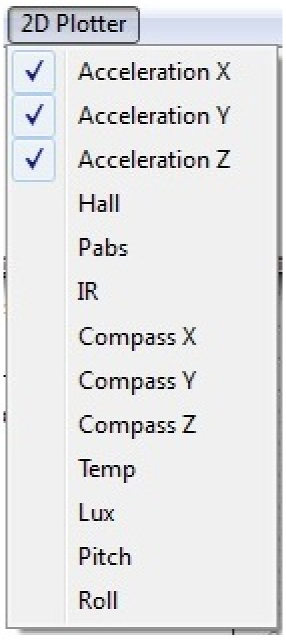


Figure 51: Options allow plots to be added or removed.

Basic functionality is extended by adding colour to the plot curves. QwtPlotCurve is in charge of painting line plots in Qwt and must be extended to add colour to lines. Before making a rainbow line plot, a basic version was created which paints uniform colours, Red for X acceleration, green for Y acceleration and blue for Z acceleration. Figure 52 shows such a graph.

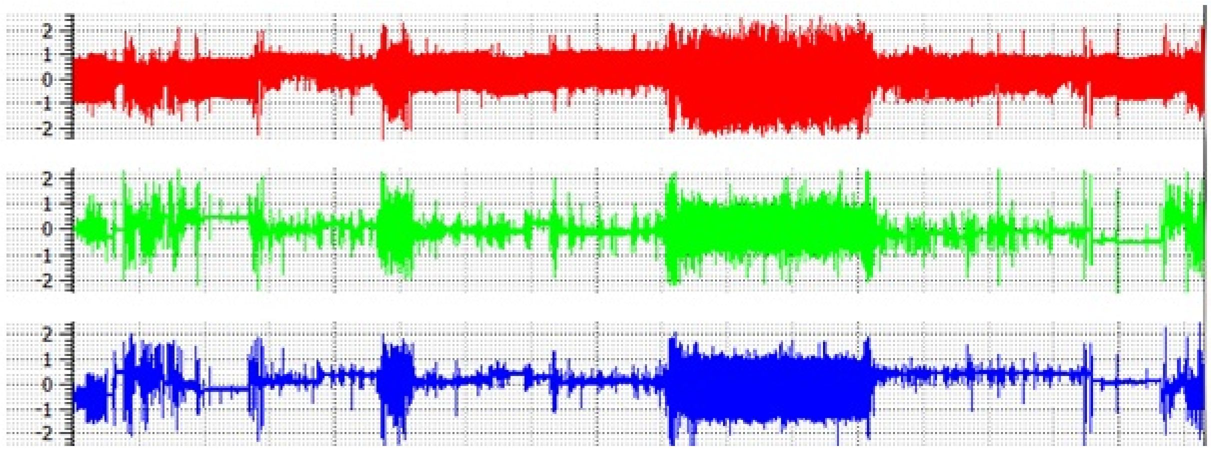


Figure 52: Standard colour mapping option for painting graphs. Red for X acceleration, green for Y acceleration and blue for Z acceleration.

A more advanced option was then created, when the acceleration values went above a threshold they were painted blue and red for below. Figure 53 illustrates the result of a graph produced using threshold colour mapping.

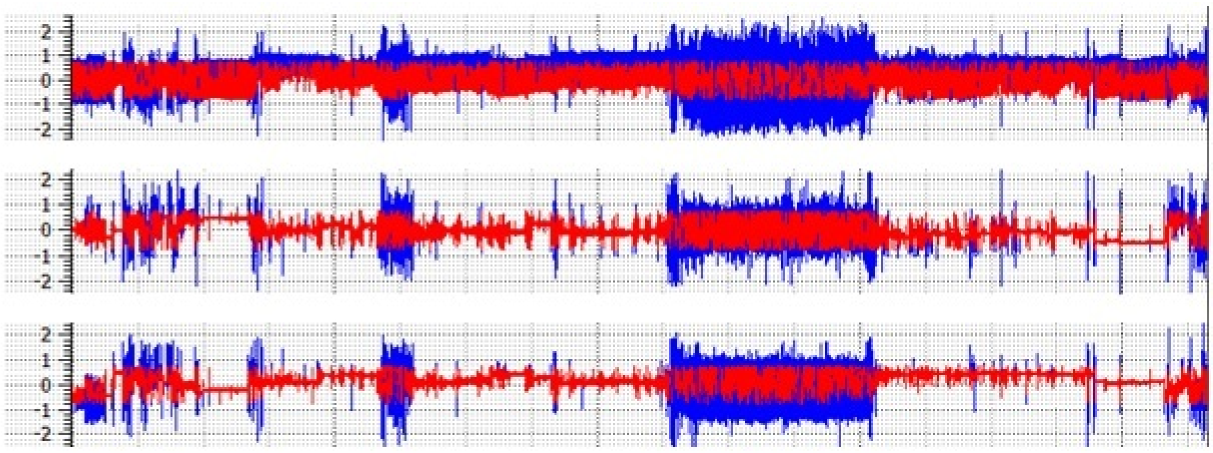


Figure 53: Threshold colour mapping option for painting graphs. When the acceleration values went above a threshold they were painted blue and red for below.

Finally, a rainbow colour plot, where data values are directly linked to a colour. A rainbow colour plot will assist the user in being able to distinguish lines in the plot and to make it easier to visualise large data sets where there is a risk of overcrowding. The algorithm for doing this is described in the Colour Mapping section. In order to achieve this effect, a line colour is mapped to the acceleration value colour. Figure 54 displays the plots produced of the 3 acceleration plots.

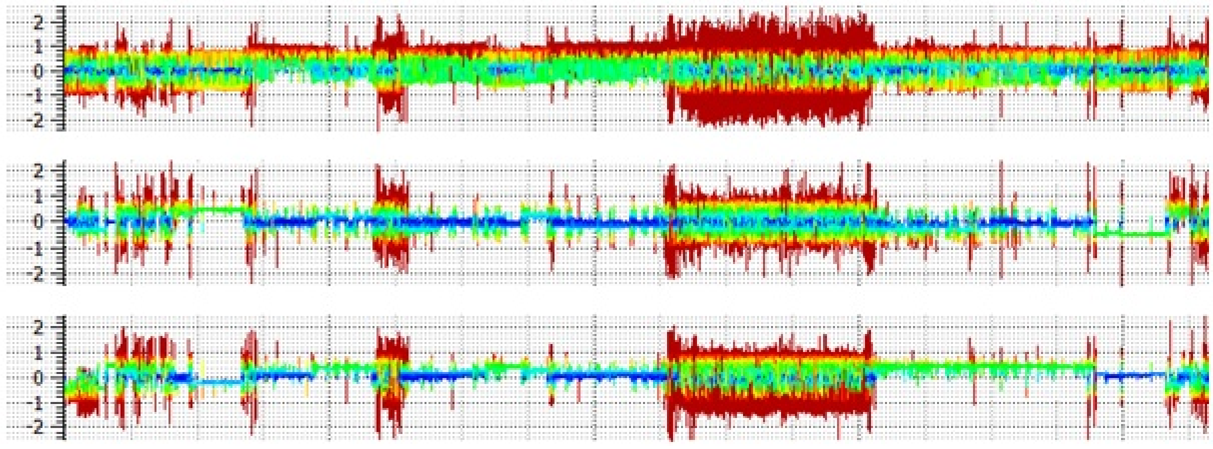


Figure 54: Rainbow colour mapping option for painting graphs.

Adding and removing lines of acceleration to a shared space plot is required to show a direct comparison between the three types of acceleration: static, dynamic and total. In our implementation we treat the addition of acceleration lines as global, that is, adding a line for dynamic acceleration will add a dynamic acceleration line to all the acceleration plots. Achieving this requires an additional QwtPlotCurve to be added to a plot and adding a method to manage the addition and removal of curves. Each component of acceleration line is given the following colour scheme: Total acceleration - red, dynamic acceleration - green, static acceleration - blue. This allows a user to depict the components from a graph. Figure 55 displays a plot of acceleration data showing a shared space plot of static and total acceleration components.

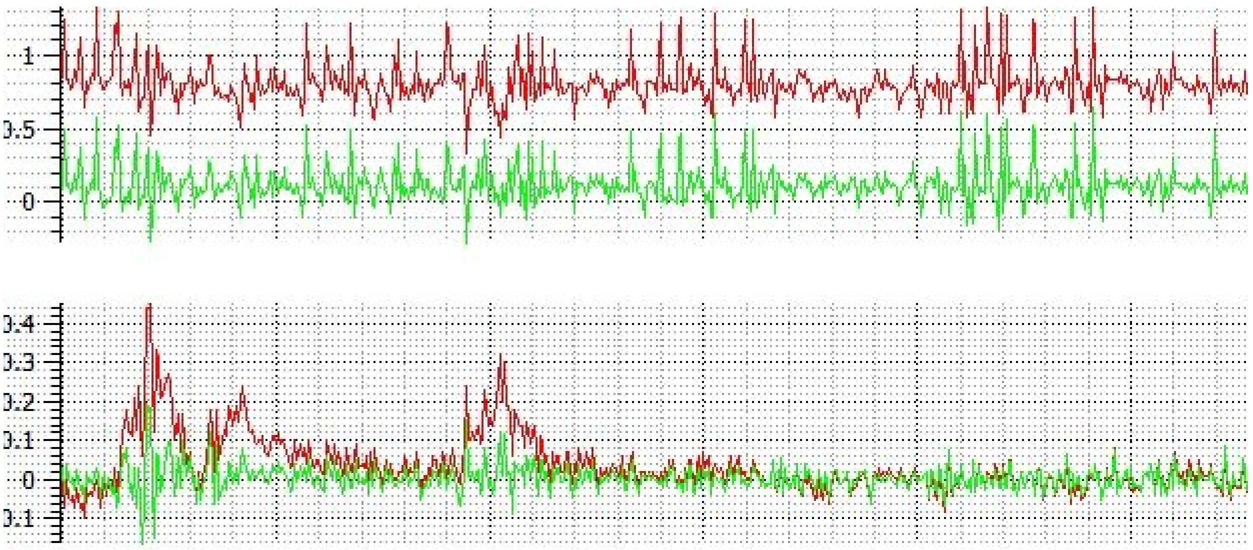


Figure 55: A shared space plot showing total and static acceleration components.

3. **Context and Focus Windows**
   Basic functionality has been achieved for the 2D plotter, the next step is to extend this functionality to create the focus and context windows. Additional functionality needed from the context window is the ability to select regions from plots. A selected region should then be regarded as a subset of the original data set and plotted on the focus window.

   In order to select data from the context window, it must be possible to retrieve data items back off the graph using the mouse, that is, when a user clicks on the graph we want to know the data item at that specific location. This is necessary to know where the user is highlighting the subset to be displayed. A class PlotSelection has been created to map and switch between viewing coordinates and graph coordinates. This same class is also used to hold the information about the region selected. i.e. the boundary values of a selection.

   The Context window is linked to the focus window via signals and slots (event listeners) created in the main GUI window. When a new region is selected the focus area is repainted with this selection of the data set, made available by the class PlotSelection. Figure 56 shows the result of this.

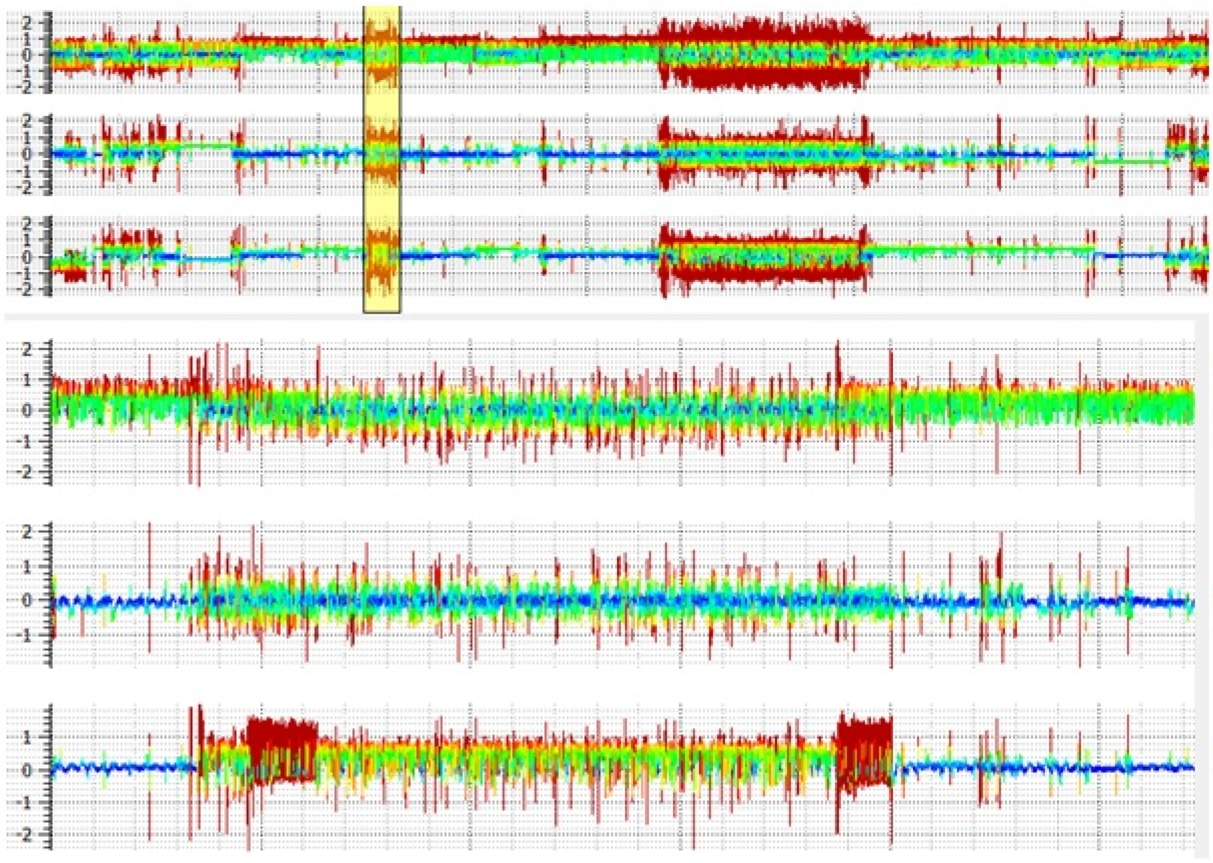


Figure 56: Context window (top) displayed with a focus window (bottom). Data rendered in the focus window can be seen selected in the context window.

There were performance issues with the context and focus windows, when selecting regions from the context plot the whole plot was repainted and recalculated making the application very slow. This is an unnecessary repaint as we only want to repaint a selection on top of the graphs. In order to facilitate repainting, a technique called double buffering was used. Double buffering paints the graphs to a cache in the form of a QPixMap, this is then painted to the screen every time, allowing to the program to be much more responsive. Instead of repainting up to a million lines (potentially up to 4 seconds), the application now paints a pixel map to the screen (constant time). When the graphs need to be updated they can be re-cached, giving the programmer control over repaints.

6.2.2 Enhancements

1. **Binary File Reader / Writer**
   The ASCI file handler was slow at reading and processing data files. To speed up data processing, data files need to be converted to a format from which they can be directly read into main memory, without the added overhead of splitting strings using regular expressions.

Reading data directly into memory from a file requires the data stored in the file to be serialized. The first step in this process is to convert existing ASCI data files to the binary format. QT offers a helping hand in implementing serializing data. The QDataStream class serializes primitive types to a binary format which can be directly written using QFile. To be able to convert files to binary, the binary file handler needs to work closely with the ASCI data handler. Once a file has been processed and stored it in main memory, it can then be directly written from memory to a Binary file by serialization. This is a fairly trivial process, we iterate over the whole data set, serializing the types and writing to a file.

Next, we need to add the ability to read the binary format files which the ASCI files have been converted to. This is also a simple task, files can be directly processed without the use of regular expressions. Instead files are read directly into memory using QDataStream .

A Binary file reader offers a significant speed increase over the ASCI file reader. Instead of taking 60 seconds to process a 74mb file, this can now be processed in 8 seconds. This is 7.5 times faster, which is a great increase in performance for the end user.

2. **Playback Mode**
   A playback function is required to allow the user to play back through the sub-set of data displayed in the focus window. Each visualisation technique will have its own methodology for how they play back through the data. As a result of this, a playback class is needed to make playback synchronised throughout all the visualisations.

   To implement a playback system a method of iterating through each time step is required, spending a set amount of time at each time step before jumping to the next. In the design section it was discussed how this class is to be used by all other visualisations during playback, and will be used to synchronise them all to the same time step. Therefore, there should be a method to update components which require to be synchronised during playback.

   A QTimer class in QT provides a repetitive timer which sets off a signal in the system at a user defined interval. Data in our application is stored in a global QList data structure by an integer index which corresponds to its time step. To move to the next time step the current index in the 'QList' object needs to be incremented to the next element in the list. As a consequence of this the signals and slots mechanisms provided in QT (event listeners) are used to connect the playback class and appropriate visualisation classes. When the timer times out, the program then emits the position of the next time step in the QList, this is repeated until playback is paused or stopped. It is important to test if the time step is within the bounds of the data set to avoid reading values not defined. Visualisations can then use this to get the index of the current time step during play back and repaint accordingly.

   A GUI has been added to create Play, Pause and Stop menus for the user to interact with and control the playback state. (Figure 57) shows the GUI.

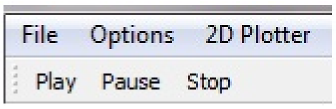


Figure 57: Menu for controlling the playback state throughout the application.

Playback functionality has also been added to the focus window. This can be seen in Figure 58, the line located down the centre of the plot corresponds to the current play back position. When the playback reaches the end of the data set playback is automatically stopped.

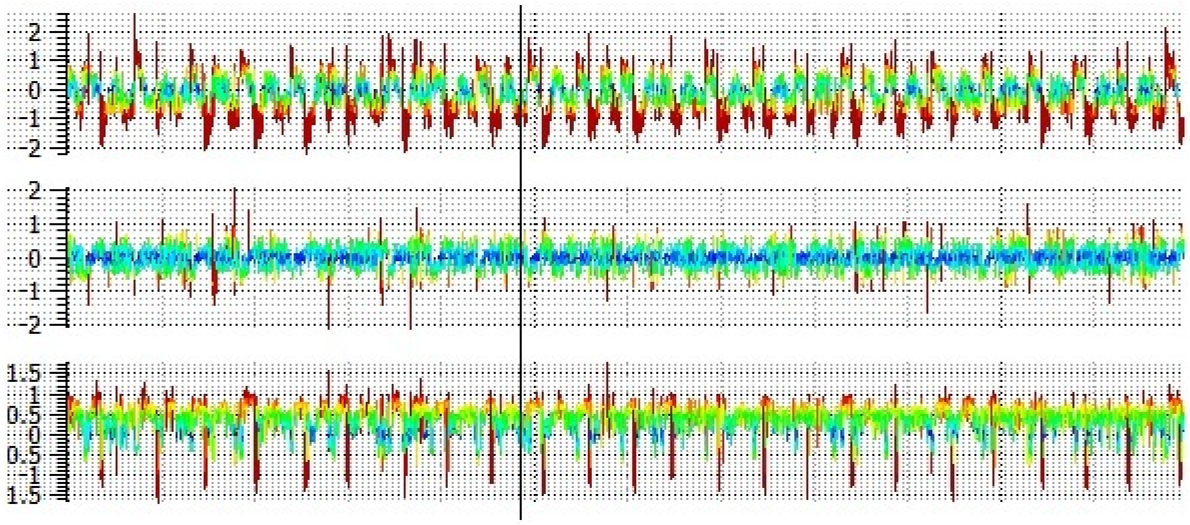


Figure 58: Playback enabled in the focus window. The vertical line at the center of the graph indicates the current playback position.

The moving average window can also be selected to be displayed during playback. Figure 59 shows this, the area marked in yellow represents the window used for the smoothing filter. The line in the center of the window represents the current play back position.

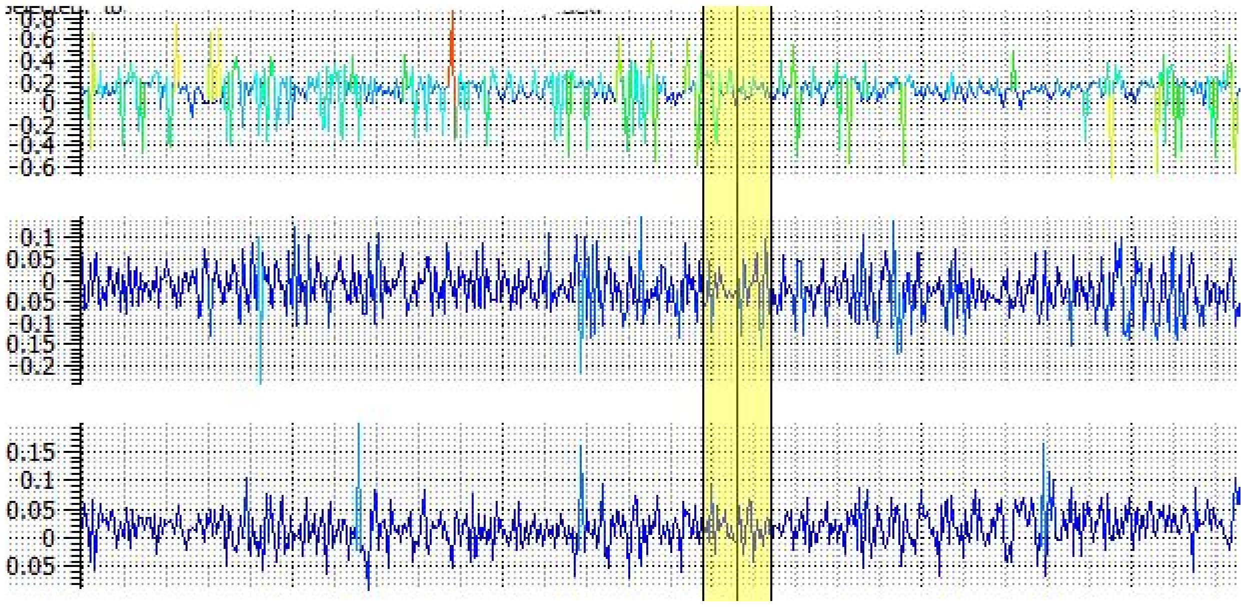


Figure 59: Visual aide showing the window size used to derive static acceleration. The window is shown during playback highlighting the area used to calculate the average for a particular data item.

3. **Colour Mapping**
   A colour mapping function maps data values to a unique colour value. Any data quantity can be represented by a unique colour in a visualisation. An algorithm for creating a rainbow colour map is described by Telea in Data Visualization [17]. An example output from the algorithm is reproduced in Figure 60. Here low data values are mapped to a dark blue colour and high values are mapped to red. This is exactly what we aimed to achieve in the visualisations created throughout the project.

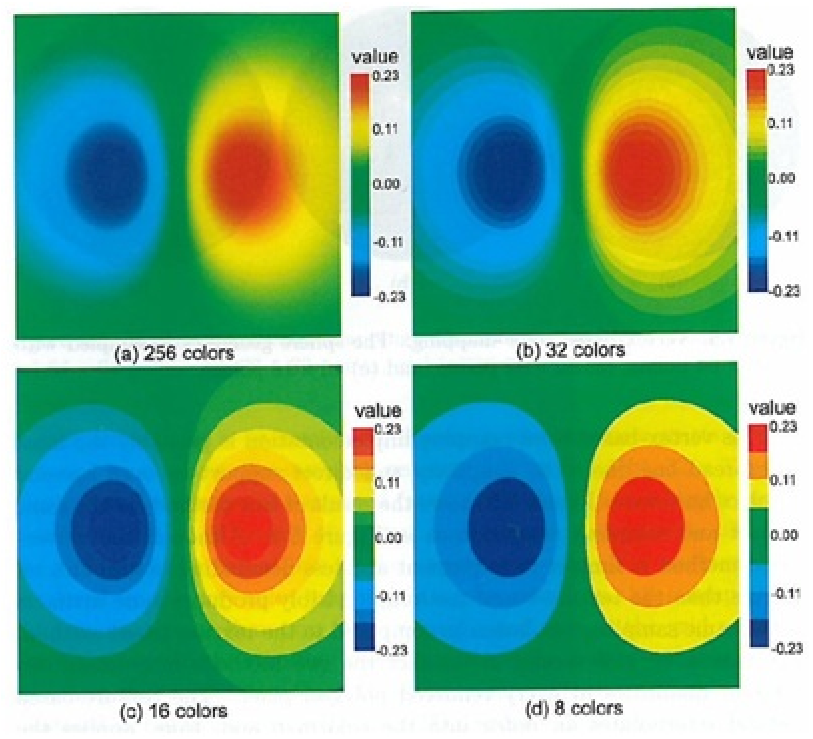


Figure 60: An example visualisation using a rainbow colour mapping function. Here low data values are mapped to a dark blue colour and high values are mapped to red [17].

The function works by assigning a colour with every scalar value input, using effectively what is a transfer function, combing values of red, blue and green. Figure 61 shows how a transfer function for a rainbow colour mapping function is produced.

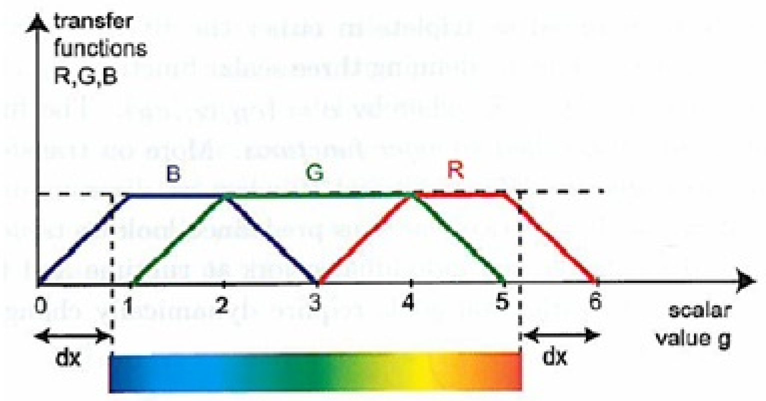
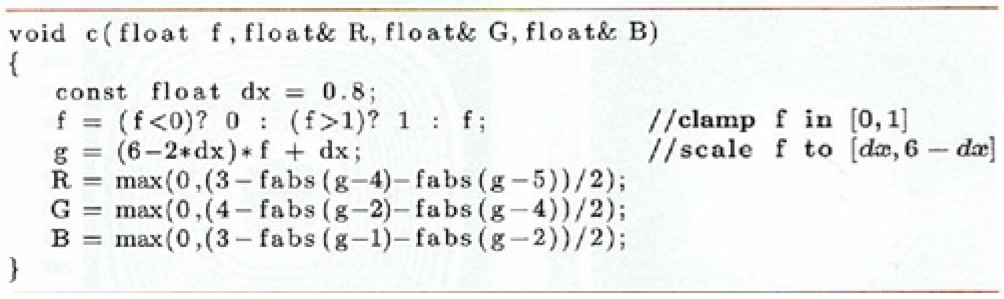


Figure 61: Construction of rainbow colourmap [17].

As presented by Telea the algorithm for this rainbow colour map is displayed in figure 62:



```
void c(float f,float& R,float& G,float& B)
{
   const float dx = 0.8;
   f = (f<0)? 0 : (f>1)? 1 : f;          //clamp f in [0,1]
   g = (6-2*dx)*f + dx;                  //scale f to [dx,6 - dx]
   R = max(0,(3-fabs(g-4)-fabs(g-5))/2);
   G = max(0,(4-fabs(g-2)-fabs(g-4))/2);
   B = max(0,(3-fabs(g-1)-fabs(g-2))/2);
}
```

Figure 62: Algorithm for the rainbow colour map [17].

In order to use this function a variable f is passed as one of its arguments, this is the data value from which a colour value is created. For colour mapping functions to work correctly, data values must be normalised, this means all values of f should be in the range 0 to 1. To do this the following normalisation formula is used:

$$NormalisedValue = \frac{(I - min)}{MAX(1, max - min)}$$

max = Maximum data value in the data set.
min = Mimimum value in the data set.
I = Current data value to be normalised.
MAX(x,y) is a function which returns the maximum value of x and y. This safeguard has been put in place so its not possible to divide by zero or end up with a normalised value bigger than the input value.

Three other variables are passed into the function, R (red value), G (green value) and B (blue value), these are assigned to the colour values computed by the function.

4. **3D Visualisation Windows and Interaction Techniques**
Two windows are required for the 3D visualisations: one for visualising compass data and the other for visualising acceleration data. A generic component can be created which can be reused for these two attributes of data. The viewer window initiates the standard settings for the 3D visualisation components being painted with in it. In this section we will implement a scene consisting of the following components for use with the visualisations:

- Central sphere.
- Axis - The X, Y and Z axes used as a reference point for data plotted.
- Colour legend - Colour legend providing a user with a legend to see how data is mapped to colour.
- Lighting - Defining lighting in the scene for visualisations which choose to use shading.
- Individual rotation and synchronisation - Rotation can be interacted with between the two components individually or synchronised.

OpenGL is incorporated with a toolkit called glut. In this package there are various functions for making implementation easier when creating scenes in OpenGL. A function called gluSphere is of particular use and can be used to draw a sphere. The function call is shown below:

gluSphere(quad,radius,slices,stacks)

In the arguments of the function "quad" is the quadrilateral that contains the sphere. "radius" is the radius of the sphere, "slices" and "stacks" define the number of longitudes and latitudes that construct the sphere. An example sphere in figure 63 is made up of 2500 quadrilaterals (quads), the number of horizontal segments define the stacks and the number of vertical segments define the slices. The number of slices and stacks will be a user defined option in the application. This also allows the bin sizes to be adjusted in the Histogram visualisation, we will talk about this later.

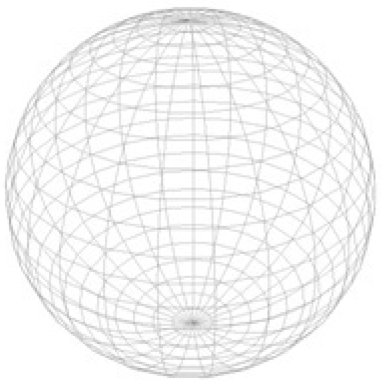


Figure 63: A sphere created using the gluSphere function in OpenGL.

Code to draw a sphere in the application is given below. A radius of 1 is chosen for simplicity in deciding on suitable values for other 3D components.

```
GLUquadric *quad = gluNewQuadric();
gluSphere(quad, 1.0f, userDefined, userDefined);
```

Next, the three axes need to drawn which correspond to the X, Y and Z axis of visualisations. Lines are drawn in OpenGL using the function "glBegin(GL_LINES)" and are defined between this statement and a closing statement "glEnd()". A line can then be defined using two points, using the function "glVertex3f(x,y,z)". Axes are given a colour to make them stand out and recognisable. A standard colouring system is adopted of red for the X axis, blue for the Z axis and green for the Y axis. Lines are given colour via the function "glColor3f(r,g,b)". The following code is used to draw the axis:

```
glBegin(GL_LINES);
glColor3f (1,0,0); // X axis - red
glVertex3f(-1.4f, 0.0f, 0.0f);
glVertex3f(1.4f,0.0f,0.0f);
```

```
glColor3f (0,1,0); // Y axis - green
glVertex3f(0.0f, -1.4f, 0.0f);
glVertex3f(0.0f,1.4f,0.0f);
glColor3f (0,0,1); // Z axis - blue
glVertex3f(0.0f, 0.0f, -1.4f);
glVertex3f(0.0f,0.0f,1.4f);
gllEnd();
```

Figure 64 displays the result of both the axes and sphere rendered in OpenGL.

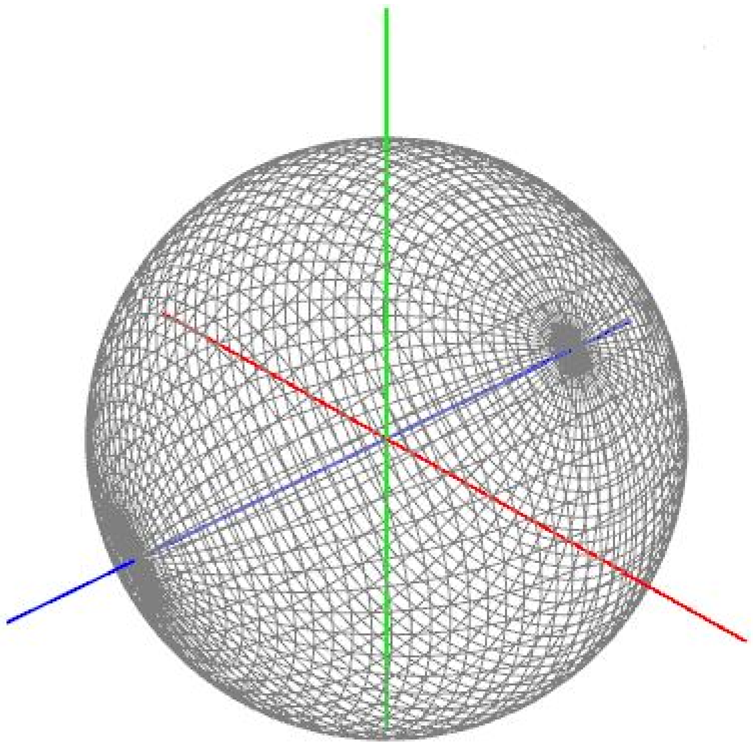


Figure 64: A sphere rendered with axes.

A colour legend is usually not simple to implement, requiring the use of a technique called over painting. As we dont want to define the legend in a 3D space we must paint a 2D image on top of the 3D scene. QT has great support for OpenGL and as a result has good support for integrating a QPainter with OpenGL too. In fact, QPainter is defined using OpenGL commands. Overpainting is simple with QT and essentially requires overriding the paint device of a widget. Using a QLinearGradient mapped to the rainbow colour map and applying it to a QRect it is possible to create a colour legend. Figure 65 shows the results of this. The windows are also over painted with a string indentifying which window is used for visualising compass or acceleration data.

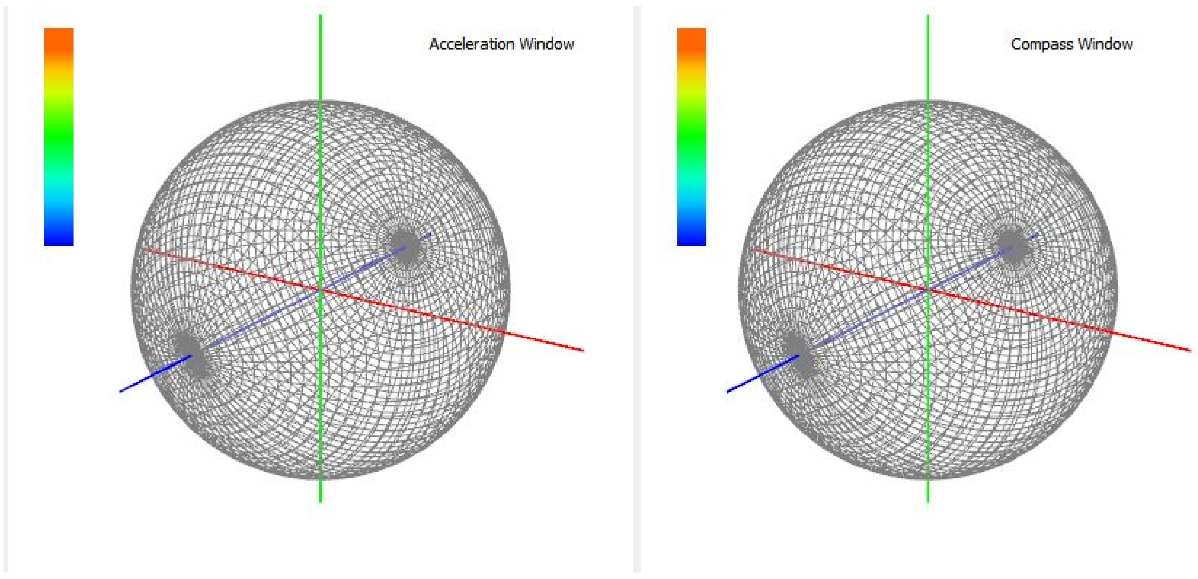


Figure 65: Acceleration and compass visualisation windows shown side by side. Windows are over painted with an identifier (top right) and colour mapping legend (top left).

In terms of lighting the current set up (figure 64) is not using a lighting model. By default, OpenGL colours primitives according to the settings specified in the glColor command. This is fine for the basic setup where lighting will always be turned off when rendering these elements. However, some of the visualisations use shading and as such a lighting mechanism is required. OpenGL has 7 built in lighting sources, by default these are all disabled. In our system we want to give the user total control over light sources. The 7 light sources are distributed in the corners of the scene and are accessible by a lighting menu (Figure 66). This enables the user to turn any of the 6 light sources on or off to create the visualisation set up which best suits their needs.

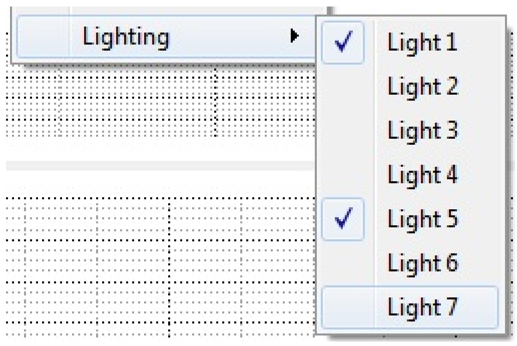
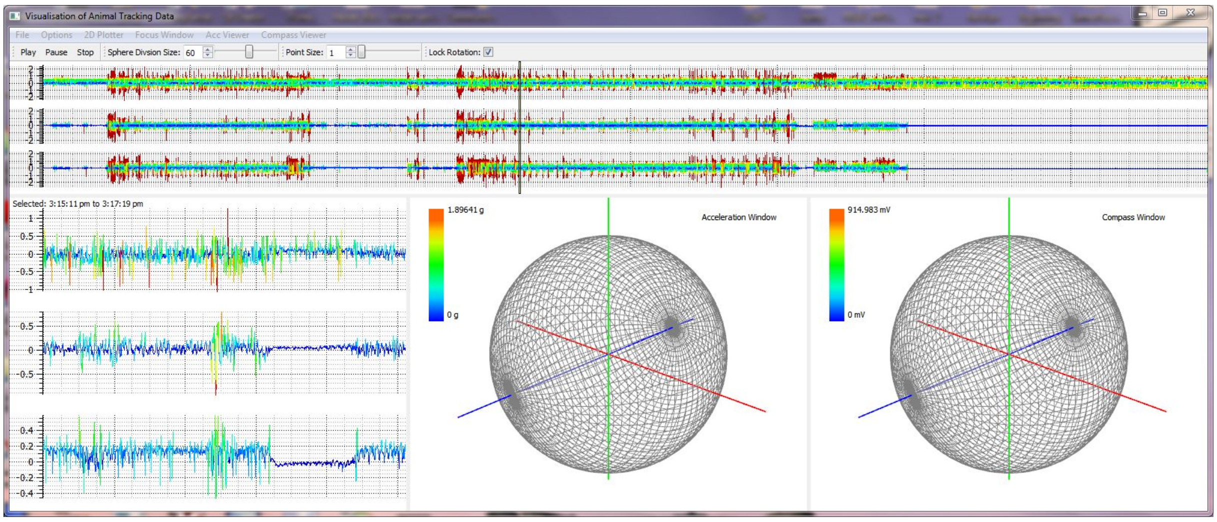


Figure 66: A toolbar showing the various lighting options a user can turn on and off.

The final result of the two 3D windows can be seen in Figure 67. This is now the basic layout for the program created. There is a context window (top), a focus window (bottom left), an Acceleration window (bottom centre) and a Compass window (bottom right).



Sphere Divsion Size: 60
Point Size: 1
Lock Rotation: ✓

Figure 67: This figure shows the GUI layout for the application. There is a context window (top), a focus window (bottom left), an Acceleration window (bottom centre) and a Compass window (bottom right)

5. **Scatter Plot**
A raw scatter plot is produced by directly mapping acceleration and compass data to a 3D spatial domain. All visualisations produced are rendered in the 3D viewer window which comprises of the scene defined above. Visualisations are treated by layers and as such multiple visualisations can be displayed at any one time.

Each data item is mapped directly to a raw 3D space in the viewer widgets. To do this the data set is iterated over with each data item being plotted as a point. Below shows a code snippet used to plot acceleration data to the screen. The size of the points can also be changed by a user, when there is a dense area of points the user may require a small point size to be able to see detail. When there is a sparse density of points the user may want a larger point size to be able to see detail. For these reasons the point size is a user defiend option.

```
glBegin(GL_POINTS);
     for(int i =0; i< m_DataSet->GetSampledData()->size(); i++){
     DataItem *item = m_DataSet->GetSampledData()->at(i);
     glVertex3f(GLfloat(item->GetAccX()),
         GLfloat(item->GetAccY()),
         GLfloat(item->GetAccZ())
             );
     }
 glEnd();
```

A Rainbow colour map is used to strengthen borders or add an extra attribute of data to the plot. The Colour Mapping section describes this algorithm. To strengthen borders an average of the data attributes being plotted is computed and then mapped to a colour value. Figure 68 shows the result of this visualisation. Otherwise a user can select any data attribute to map colour to in order to add an extra dimension.

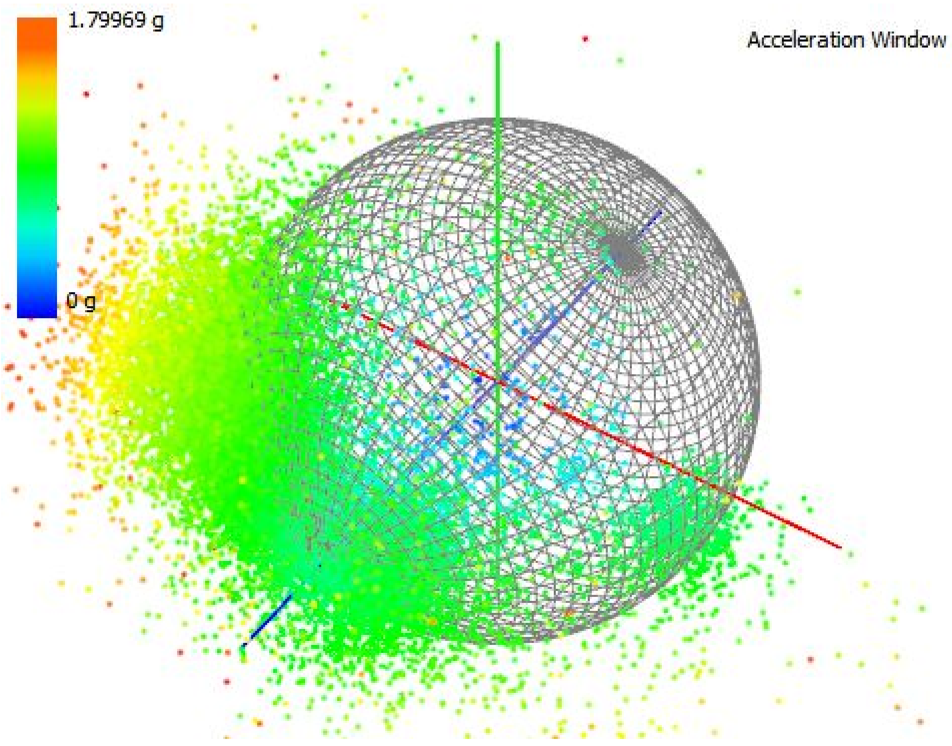


Figure 68: A raw scatter plot of acceleration data with rainbow colour mapping to strengthen borders.

During play back every data value is iterated up to the current time step plotting all the points iterated over. As playback continues the plot is built up to the next time step and repeated until complete.

6. **Spherical Scatter Plot**

   In the Spherical Scatter Plot visualisation the aim is to translate points so they are mapped on to the central sphere surface, this removes magnitude of data items while maintaining the direction component.

   In the raw scatter plot Cartesian Coordinates were used, which place a point in a three dimensional plane using three coordinates, x, y and z. Spherical Coordinates use a different system for plotting points in a 3D space, points are defined in terms of a sphere. Three control points can define any location in a 3D space: A Radius, Thi (horizontal angle) and Theta (vertical angle). By adjusting the radius to the radius of the sphere, points can be mapped to the surface of the central sphere. Figure 69 displays this in a visual format.

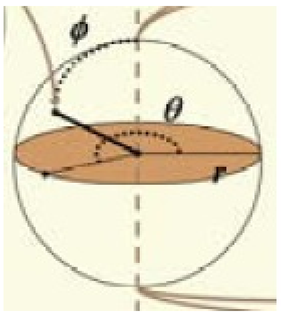


Figure 69: Diagram displaying spherical coordinates and how a point can be defined in terms of Thi (φ), Theta (θ) and the radius (r).

To create the visualisation all points must first be translated to Spherical Coordinates, their radius can then be changed to that of the sphere, 1 and then converted back to Cartesian Coordinates and plotted directly in 3D space.

Formulas for translating Cartesian Coordinates to the Spherical Coordinate system:

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = cos^{-1}\left(\frac{z}{r}\right)$$

$$\varphi = tan^{-1}\left(\frac{y}{x}\right)$$

Formulas for converting Spherical Coordinates back to Cartesian Coordinates:

$$x = r * sin\theta * cos\varphi$$

$$y = r * sin\theta * sin\varphi$$

$$z = r * cos\theta$$

Points can then be plotted using the same method described in the Scatter Plot section above, by plotting converted points directly as 'GL_POINTS'.

It is computationally expensive to keep computing converted points during playback and interaction with visualisations (e.g. rotation). Instead, when the points are computed they are stored in memory for future access when rendering the visualisation. Points are only re-calculated when a new sub-set of data is selected.

The rest of the visualisation is very similar to that of the raw plot. When played back the points are iterated up to the current time step, and colour mapping remains the same.

Figure 70 shows a side by side visualisation produced of the same data in a Raw Plot (left) and a Spherical Plot (right) with points normalised to the radius.

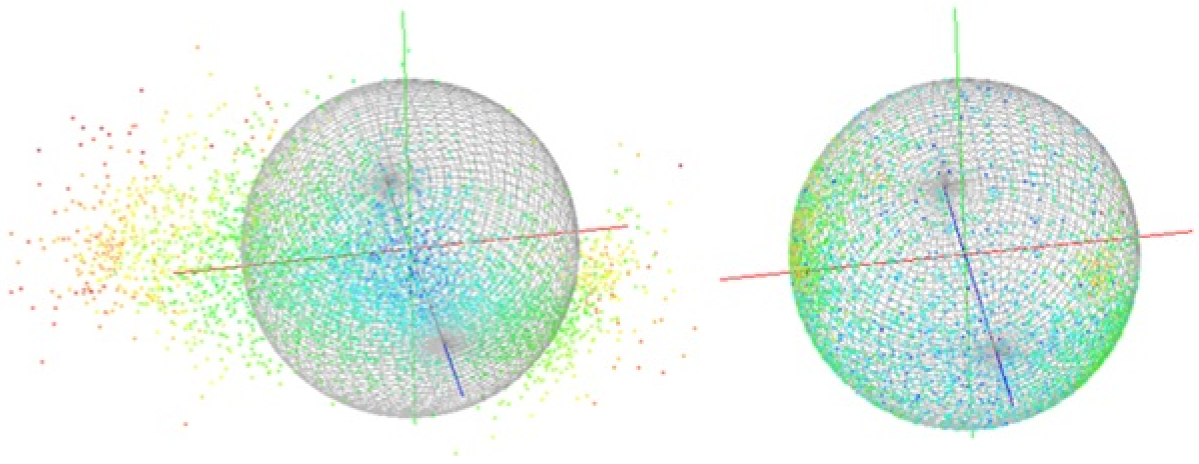


Figure 70: A side by side visualisation produced of the same data in a Raw Plot (left) and a Spherical Plot (right) with points normalised to the radius

An option for points to be joined together by a line has been implemented. This allows a user to see the connectivity between data points. This is especially useful when looking at static acceleration or compass data, as these should both produce smooth lines. To do this instead of rendering the data as 'GL_POINTS', GL_LINE_STRIP is used instead. Figure 71 displays a plot produced with this option turned on.

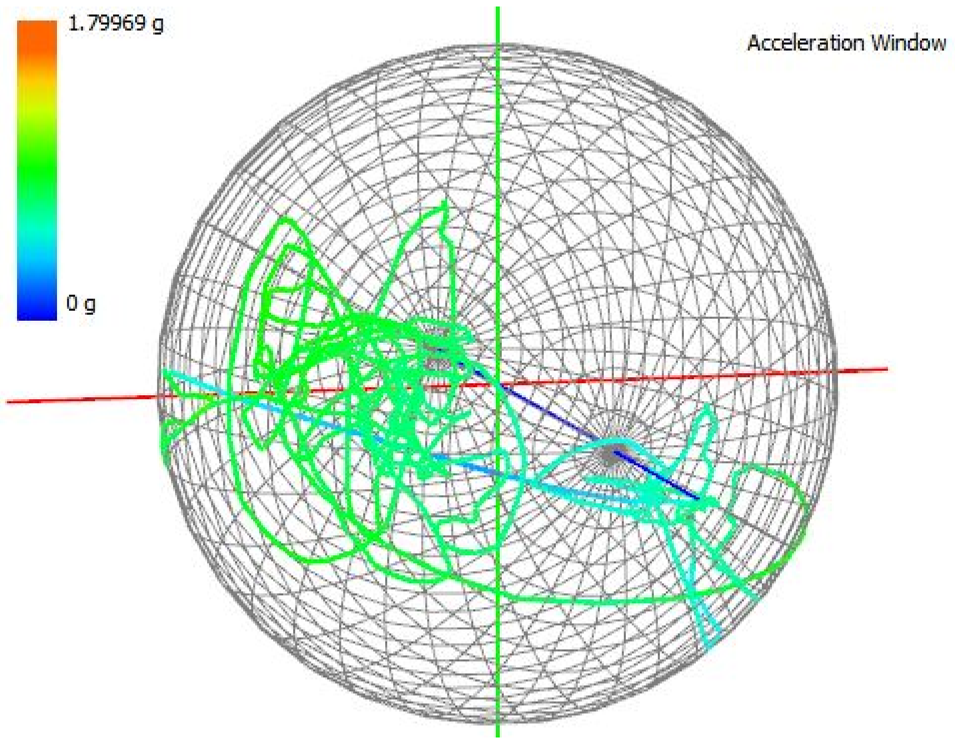


Figure 71: A spherical plot of static acceleration data plotted as lines between data points.

7. **Spherical Histogram**
A Spherical Histogram visualisation is the most complex and challenging technique so far in the project and requires significantly more time and effort to implement than the raw and spherical plots. The main concept behind a spherical histogram is to partition a sphere into a series of bins. The number of points inside each bin is aggregated to give a value for each bin. This value is then normalised and used to define the height of cubes extruded outwards from each bin.

To make the visualisation simpler to implement it is broken down into the following sub problems and built upwards to produce full functionality. These are listed below:

- Partition the spatial region (Create bins)
- Determine which bin a data item (point) is in.
- Summate points inside each bin.
- Normalise summation (count).
- Extrude bins outwards in relation to count.
- Colour mapping
- Contours

The first implementation step is to create the bins. Bins should be mapped to the same positions to that of the sphere created in the Viewer Window. To do this an algorithm has been made which calculates the coordinates of bin vertexes. This is similar to what the gluSphere function mentioned earlier does, however, instead of painting primitives to the screen they are stored for later use in determining which bin a point is in. The algorithm for doing this can be seen below. Bins are calculated in spherical coordinates and by using two nested for loops it is effectively winding around the sphere travelling upwards to create all the bins. Bins are stored in a flat array with their position corresponding to the amount of iterations taken through the for loops.

```
for(int i =0; i< Settings::GetLats() - 1; i++){
    double thiGap = FULL_ROTATION_ANGLE / Settings::GetLats();
    double thi = thiGap * i;
    double thi2 = thiGap * (i + 1);

    for(int j = 0; j<(Settings::GetLongs() - 1); j++){
        double thetaGap = HALF_ROTATION_ANGLE / Settings::GetLongs();
        double theta = thetaGap * j;
        double theta2 = thetaGap * (j + 1);

        //First latitude points
        PointPrimative *bottomLeft = new PointPrimative;
        bottomLeft->SetX(r * sin(theta / DEGREES) * cos(thi / DEGREES));
        bottomLeft->SetY(r * sin(theta / DEGREES) * sin(thi / DEGREES));
        bottomLeft->SetZ(r * cos(theta / DEGREES));
        PointPrimative *bottomRight = new PointPrimative;
        bottomRight->SetX(r * sin(theta / DEGREES) * cos(thi2 / DEGREES));
        bottomRight->SetY(r * sin(theta / DEGREES) * sin(thi2 / DEGREES));
        bottomRight->SetZ(r * cos(theta / DEGREES));

        //next latitude points
        PointPrimative *topLeft = new PointPrimative;
        topLeft->SetX(r * sin(theta2 / DEGREES) * cos(thi2 / DEGREES));
        topLeft->SetY(r * sin(theta2 / DEGREES) * sin(thi2 / DEGREES));
        topLeft->SetZ(r * cos(theta2 / DEGREES));
        PointPrimative *topRight = new PointPrimative;
        topRight->SetX(r * sin(theta2 / DEGREES) * cos(thi / DEGREES));
        topRight->SetY(r * sin(theta2 / DEGREES) * sin(thi / DEGREES));
        topRight->SetZ(r * cos(theta2 / DEGREES));

        QuadPrimative *quad = new QuadPrimative;
        quad->SetBottomLeft(bottomLeft);
        quad->SetBottomRight(bottomRight);
        quad->SetTopLeft(topRight);
        quad->SetTopRight(topLeft);
        m_BinCoordinates[(i * Settings::GetLats()) + j] = quad;
    }
}
```

Next it is necessary to calculate which bin a point is residing in. To do this we must do something clever. An obvious solution would be to use a brute force approach and traverse every bin to find out if a point is in this region. However, this computational expensive when the sphere is made up of a large amount of latitudes and longitudes, making rendering very slow. Instead the algorithm for creating bins can be reversed, calculating which bin in the array which contains the point. The algorithm for intersection is displayed below. To test which bin a point intersects with, a point is first converted to spherical coordinates and then reversed to calculate which region it occupies by calculating the index of the bin it resides in.

```
int intersect(double x, double y, double z)
{
    //Convert to spherical
    double r = sqrt((x*x) + (y*y) + (z*z));
    double thei = atan2(y , x) * DEGREES;
    double theta = acos(z / r) * DEGREES;

    if(thei < 0)
        thei = thei + FULL_ROTATION_ANGLE;

    if(theta < 0)
        theta = theta + HALF_ROTATION_ANGLE;

    //reverse formula to find out what region its in
    double thetaGap = HALF_ROTATION_ANGLE / Settings::GetLongs();;
    double thiGap = FULL_ROTATION_ANGLE / Settings::GetLats();;

    int lats = int((thei / thiGap));
    int longs = int((theta / thetaGap));

    double pos = (lats * Settings::GetLats()) + longs;
    return pos;
}
```

The code generated so far allows a point to discover which bin it is residing in. Next, all the points inside each bin must be summated. When testing for intersections an array is maintained containing the amount of points inside a bin. Simply, the array is incremented by one when a bin is found to contain a point. Once every point has been calculated the summated bins needs to be normalised so they are in the region 0 to 0.5. This is because the summation will be directly linked to the height of cubes. Normalising values will keep the heights relative to each other. 0.5 is a good height for the maximum bin as it is half the radius of the sphere. To normalise the heights, the formula previous discussed in the Colour Mapping section is used.

Bins need to be extruded outwards to form the histograms. Surface normals are calculated to project the bins (stored as quads) into cubes using the normalised height obtained previously. A surface normal is a vector which is orthogonal to the plane of a surface, in our case, a quad. After calculating the surface normal it is then possible to calculate the top quad of the cube by multiplying the normal by the height and adding it to each vertex of the bin. From this it is now possible to construct a cube as we have the top and bottom quads.

Cubes are mapped to a colour using the rainbow colour mapping function previously discussed. In this visualisation it is only possible to map the height of cubes to a colour value. As the heights are already normalised between 0 to 0.5, they can simply be doubled to put them in the range 0 to 1, needed for the rainbow colour plot. A Side by side visualisation of a spherical scatter plot and a spherical histogram produced can be seen in Figure 72.

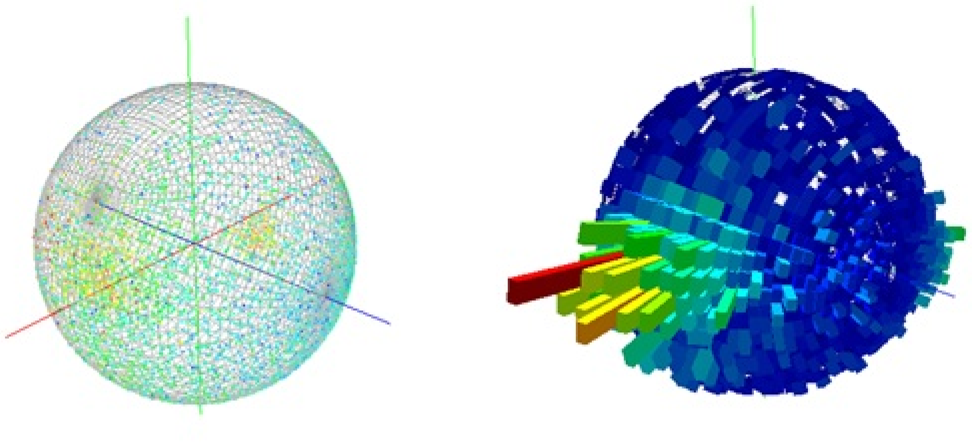


Figure 72: A side by side view of a sperical scatter plot visualisation and a spherical histogram visualisation of the same data set.

It is hard to see any of the detail in the histogram as the cubes appear to merge together, especially in regions of a similar colour. Edges are accentuated by adding lines to them. This makes the edges more defined and easier for someone viewing the visualisation to see. The histogram can be rendered again in 'GL_LINE' mode to produce a wireframe around the cube, a polygon offset has to be set to avoid a stitching effect occurring. Figure 73 shows the results of adding contours.

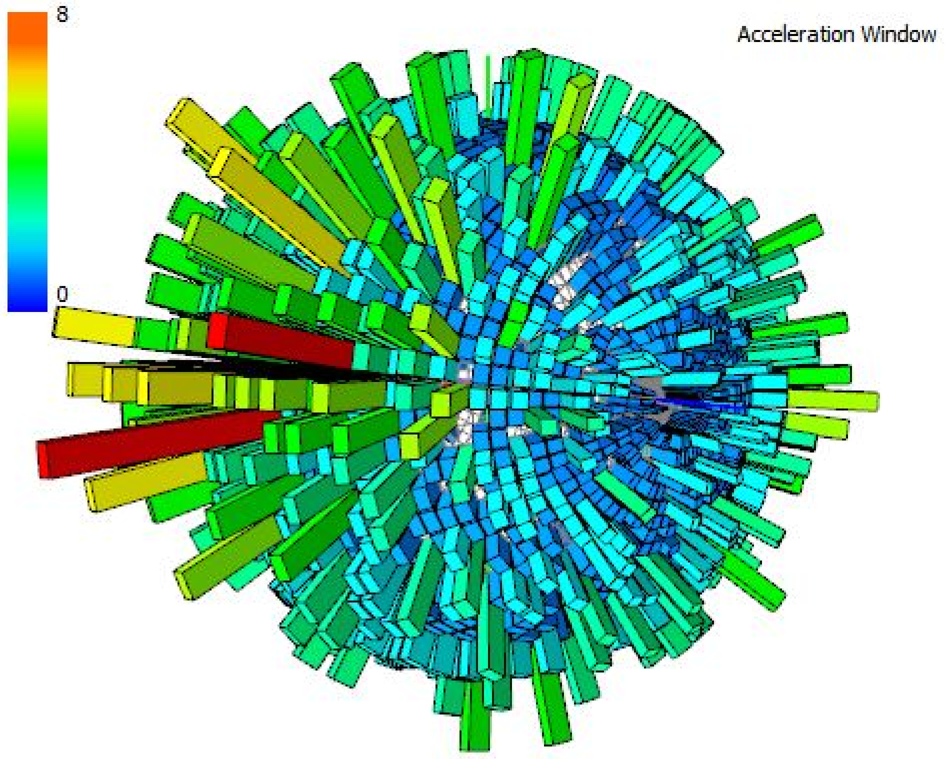


Figure 73: Spherical histogram of dynamic acceleration data rendered with contours to accentuate edges.

8. **Triangle Fan**
   A triangle fan visualisation is used to perceive future and previous time steps during playback. Its called a triangle fan as it appears like the shape of a fan when being used to visualise data. To describe this visualisation first we will look at the 2D case.

   Using figure 74 as a reference, we are visualising three time steps into the future using two triangles. A triangle consists of three vertexes: the first vertex is the origin (V1), this is the same for all triangles in the triangle fan. The next two vertexes are data points mapped onto the surface of the sphere, for example, in the 2D case, T0 is time step one, and T1 the 2nd time step and T2 the third time step. To visualise two triangle steps into the future the first triangle is made up of vertexes: v1, T0 and T1. The next triangle is made up of: v1, T1 and T2. This creates a fan like appearance. Essentially each triangle consists of the vertexes: v1, Tn,Tn+1 where n is the value of the previous time step to be visualised.

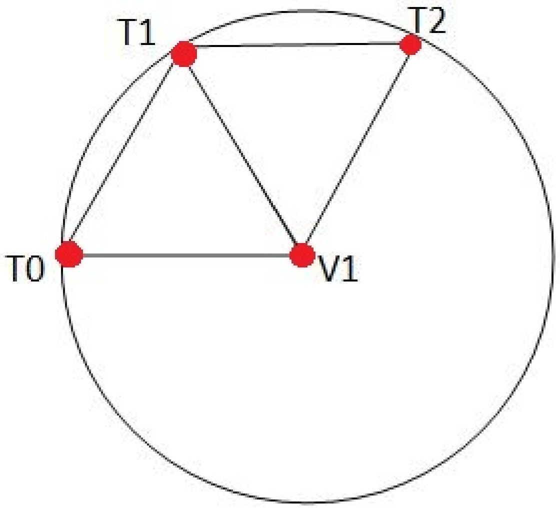


Figure 74: A diagram explaining the triangle fan visualisation. The example shown in the diagram is made up of 2 triangles, visualising 3 time steps into the future. Each triangle shares the vertex V1, the origin. T0, T1 and T2 refer to time steps. The two triangles consist of the following vertexes: triangle one - T0, T1, V1 and triangle two T1, T2 and V1.

Extending the 2D case to 3D is trivial, the algorithm still only requires 3 vertexes to make up a triangle, instead, a Z depth component as to be added when specifying vertexes. The data acceleration being visualised is of three dimensions so these values can be directly mapped to the triangles once normalised to the radius of the sphere. "GL_TRIANGLES" is used to draw the triangles in the fan. The algorithm for plotting points in the future is outlined below. The 'm_PlayBackSteps' variable represents the number of time steps to visualise into the future:

```
glBegin(GL_TRIANGLES);
// Loop over time steps to visualise
for (int i = currentTimeStep; (i < (currentTimeStep + m_PlayBackSteps))
            && ((i + 3) < dataSubSet->size()); i++)
{
    // Get data items
    DataItem *item = dataSubSet->at(i);

    PointPrimitive *vertex1 = PointPrimitive::normaliseRadius(
                item->GetAccX(),
                item->GetAccY(),
                item->GetAccZ(),
                1
            );

    item = dataSubSet->at(i+1);
    PointPrimitive *vertex2 = PointPrimitive::normaliseRadius(
                item->GetAccX(),
                item->GetAccY(),
                item->GetAccZ(),
                1
            );

    // Three vertexs which make up triangle
    glVertex3f(0.0f,0.0f,0.0f);
    glVertex3f(vertex1->GetX(),vertex1->GetY(),vertex1->GetZ());
    glVertex3f(vertex2->GetX(),vertex2->GetY(),vertex2->GetZ());
}
glEnd();
```

Contours are added to the edges of the triangle fan to make edges more visible, similar to that as used in the histogram previously. Figure 75 shows a triangle fan visualisation displaying 30 time steps into the future.

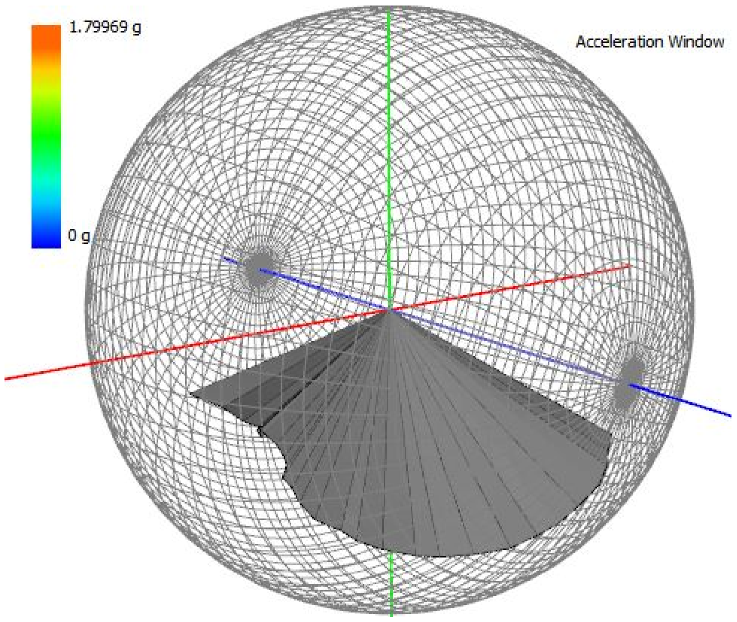


Figure 75: A Triangle fan visualisation, displaying static acceleration data of 30 time steps into the future.

9. **Central Glyphs**
   Glyphs are used to show the current acceleration or compass vector during play back. Given a time step, a glyph plots the vector for that point in time. Glyphs created reside in the center of the central sphere, this allows the user to see the vectors direction changes over time.

   A glyph is constructed to look like an arrow, so a user can see which direction it is pointing in clearly. In order to create this in OpenGL the arrow is broken down into two sections: A head and shaft (Figure 76).

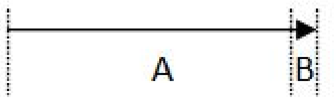


Figure 76: A diagram displaying the individual components of an arrow. (A) refers the arrow shaft. (B) refers to the arrows head.

To create both of these sections, OpenGLs toolkit, glut can be used. Glut has functionality to draw cylinders, which can be utilised to draw both the head and shaft components. The gluCylinder function, used for drawing cylinders in glut is shown below:

```
gluCylinder (   GLUquadric* quad
              , GLdouble base
              , GLdouble top
              , GLdouble height
              , GLint slices
              , GLint stacks
              );
```

In the arguments of the function: "quad" is the quadrilateral that contains the cylinder, base is the radius of the cylinder at z=0, "top" is the radius of the top of the cylinder, "height" is the height of the cylinder and "slices" and "stacks" make up the number of subdivisions the cylinder is made up of. The shaft of the arrow needs to have the same radius at the top as the base. The head should have a pointy end and overlap the shaft. To achieve this, the head should have a radius larger than that of the shaft at the base, and a radius equal to zero at the top to achieve a pointy end. The code used to draw glyphs is shown below:

```
/** Geometric attributes of the glyphs */
const int GLPYH_STACKS = 32; /* Number of slices */
const int GLPYH_SLICES = 32; /* Number of stacks */

/** Geometric attributes of shaft part of glyph */
const float SHAFT_BASE = 0.05f; /* Base diameter */
const float SHAFT_TOP = 0.05f; /* Top diamemeter */
const float SHAFT_HEIGHT = 0.7f; /* Height of HEAD */

/** Geometric attributes of head part of glyph */
const float HEAD_BASE = 0.10f; /* Base diameter */
const float HEAD_TOP = 0.0f; /* Top diameter */
const float HEAD_HEIGHT = 0.2f; /* Height of HEAD */

gluCylinder(quadratic,SHAFT_BASE,SHAFT_TOP,SHAFT_HEIGHT,GLPYH_SLICES,GLPYH_STACKS);
glTranslatef(0.0f,0.0f,SHAFT_HEIGHT);
gluCylinder(quadratic2,HEAD_BASE,HEAD_TOP,HEAD_HEIGHT,GLPYH_SLICES,GLPYH_STACKS);
```

Next, the glyph needs to be rotated to the current heading. In order to do this points must first be translated to spherical coordinates (Discussed in Spherical Scatter Plot section) to get the phi and theta values which are needed to rotate the glyph to the correct heading. Once theta and phi have been calculated the glyph can be rotated to the correct position using glRotate. Code for this is shown below:

```
glRotatef(theta,0.0f,0.0f,1.0f);  // rotation in z
glRotatef(phi,0.0f,1.0f,0.0f); // rotation in y
```

Figure 77 displays the completed glyph. Shading and colouring has been added to make it easier to see its location in a 3D environment.

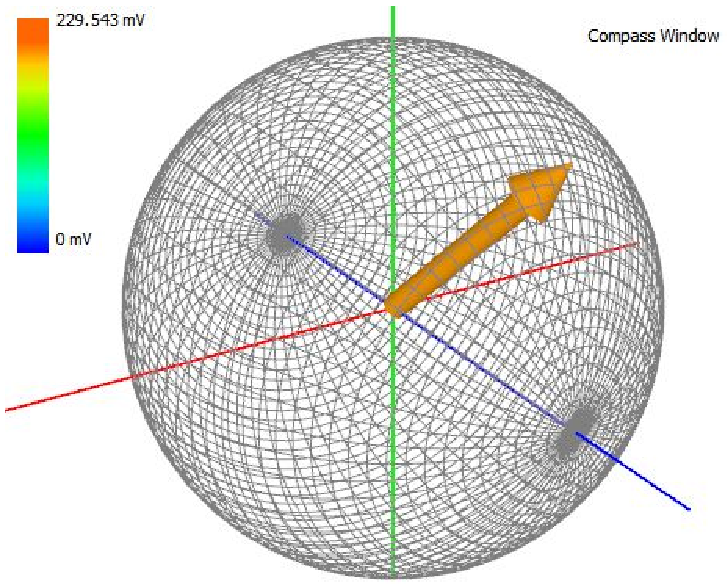
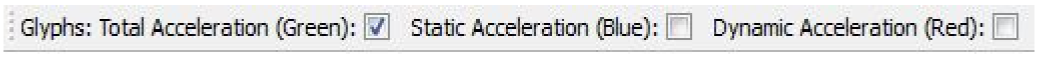


Figure 77: A glyph visualisation displaying the current compass heading.

This same process is used for each of the acceleration and compass glyph components. A GUI is added so the user can add multiple acceleration glyphs (Figure 78), each glyph is treated as a layer so multiple glyphs can be displayed at any one time. Each glyph is assigned a unique colour to avoid identification problems. Compass data only consists of one component and therefore only one glyph is required. A simple toolbar menu has been added to allow a user to add a compass glyph visualisation.



Glyphs: Total Acceleration (Green): [✓] Static Acceleration (Blue): [ ] Dynamic Acceleration (Red): [ ]

Figure 78: A GUI for adding and removing acceleration glyphs.

A user option has been added to enable individual components of the vectors which make up the glyphs to be plotted separately, as raw line segments. This assists a user to see the strength of the individual components. Each line segment is colour coded using the rainbow colour map. Figure 79 shows a glyph visualisation with line segments turned on.

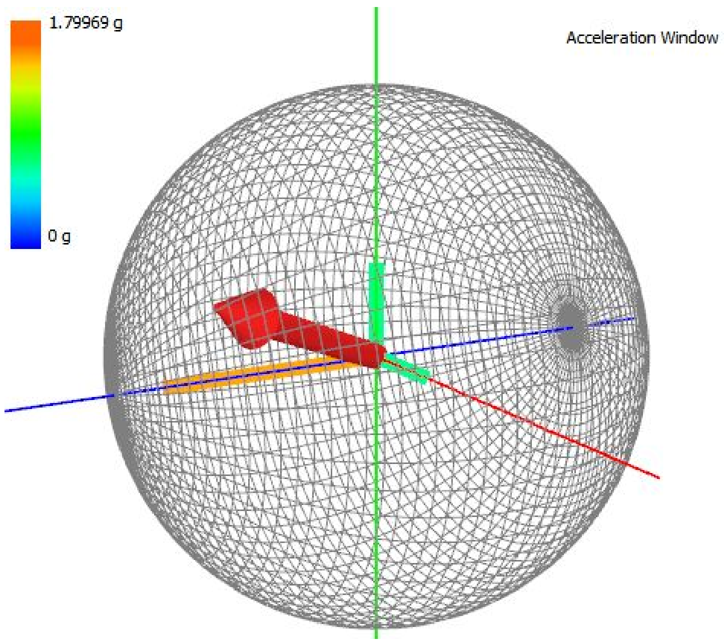


Figure 79: A glyph visualisations with the line segements option turned on.

As a final step to assist in making the glyphs more intuitive an animal geometry can be stuck on the end of the glyph. A geometry of a penguin (thanks to `http://artist-3d.com/free_3d_models/dnm/model_disp.php?uid=615` for the model) has been imported into OpenGL. The animal geometry is rotated so it is orthogonal to the glyph and then rotated again with the theta and phi components. Figure 80 shows the results of this.

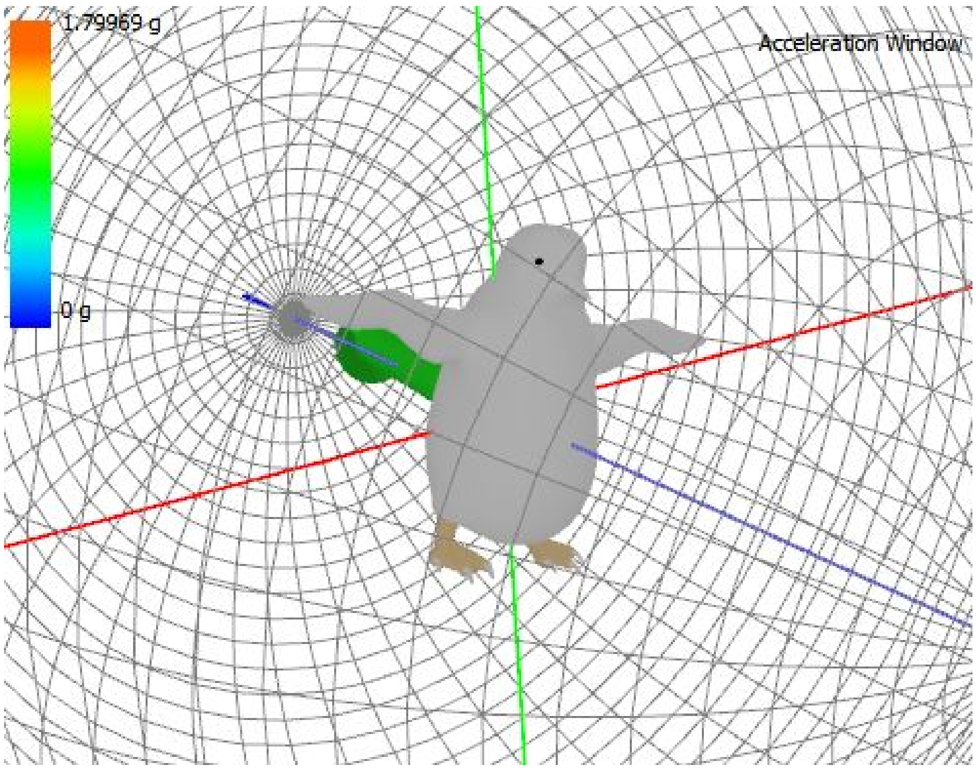


Figure 80: A penguin geometry applied to the glyph visualisation.

10. **Animal Posture Diagrams**
An animal posture diagram consists of recreating the movement of an animal. To do this the pitch and roll components must be computed and then applied to the animal geometry and glyphs. In addition to this a spherical plot should also be created of the path the animal has taken.

To compute the pitch and roll, Sheppard et al. in 'Identification of Animal Movement Patterns Using Tri-axial Accelerometry' state the pitch is equal to the arcsine of the static heave (X component) acceleration component and the roll is equal to the arcsine of the static sway (Y component) component. Code for this is displayed below:

```
double pitch = asin(m_DataSet->GetSampledData()->at(pos)->GetMovingAvgX());
double roll = asin(m_DataSet->GetSampledData()->at(pos)->GetMovingAvgY());
```

All we do then is convert the pitch and roll to degrees and rotate the pitch around the x axis and the roll around the z axis:

```
glRotatef(roll,0.0f,0.0f,1.0f);  // rotation in z for roll
glRotatef(pitch,1.0f,0.0f,0.0f); // rotation in x for pitch
```

Figure 81 displayes the resulting visualisation:

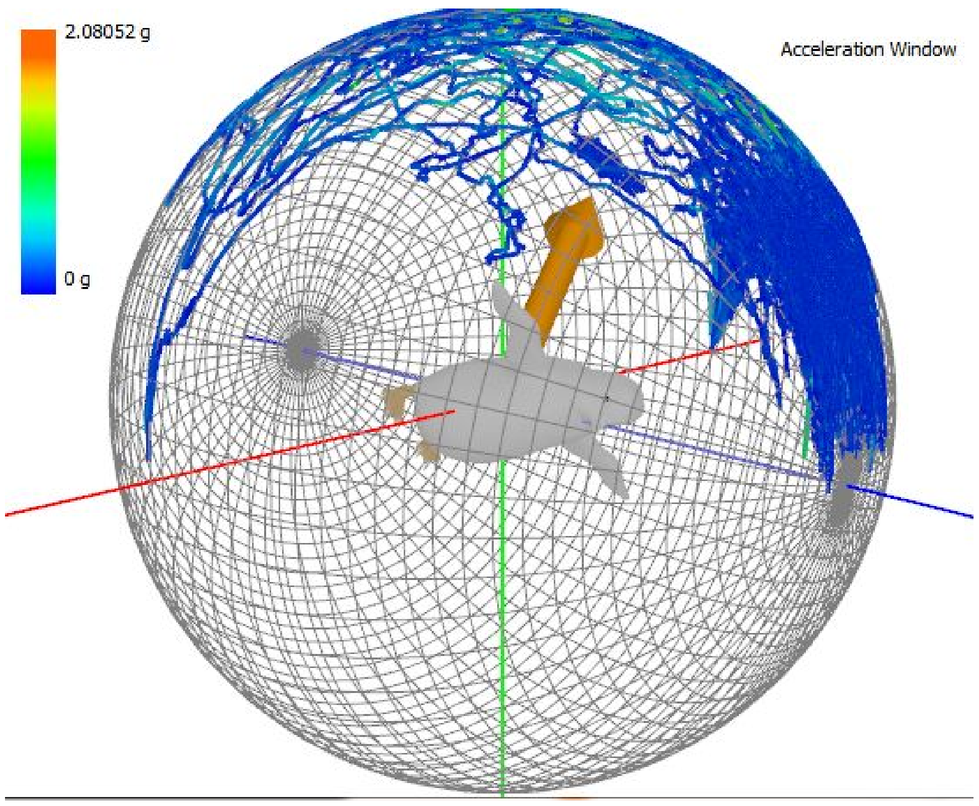


Figure 81: An animal posture diagram of a Penguin showing the current orientation of the animal. Also included is the path taken by the animal shown in spherical line format on the sphere.

6.3 Documentation of Software Model Used

During all of the meetings with the projects supervisor Dr Robert S. Laramee minutes of meeting are taken. These show the week by week progress made of the project and as a result also show proof of the incremental model used throughout the project. All of the minutes of meeting are available at the following URL:

`http://cs-sol.swan.ac.uk/~cs530379/Year3Project/Minutes/`

7 Testing and Evaluation

7.1 Testing

Insuring a quality software product is delivered to the biologists is vital. In order to guarantee this each of the visualisations must be proven to be correct. In 'Using Visualization to Debug Visualization Software' Laramee presents the idea of debugging visualisation by visualisation. Using this idea, each of our visualisations will be tested by proving their correctness through visualisation.

Each visualisation will be tested on a variety of data set sizes, the data sets to be utilised during testing are identified below. It is important to note that each of these data sets will be tested on all of the test cases to ensure a wide variety of data sets are used.

1. Large: 041207_Lindt_corm.asc - Device attatched to an Imperial Cormorant, consisting of 32 hours of data.
2. Medium: 041207_Roy.asc - Device attatched to a Magellanic Penguin, consisting of 9 hours of data.
3. Small: Rhabarber_Kuchen_(cut).asc - Device attatched to a Leatherback Turtle, consisting of 4 hours of data.

The test cases are now outlined:

1. **2D Plots**
   Test Case:
   Using one of the previous systems used by the biologists, Origin Pro, a 2D plot of a data set can be produced. Comparing the plot output to the 2D plotting visualisation tool we have created, the plotting tool can be verified for correctness.

   Expected Result:
   It is expected the plots produced by Origin Pro and our visualisation application will look the same.

   Result:
   Figure 82 shows a resulting 2D plot using the data set 041207_Roy.asc of the heave acceleration axis produced using origin pro.

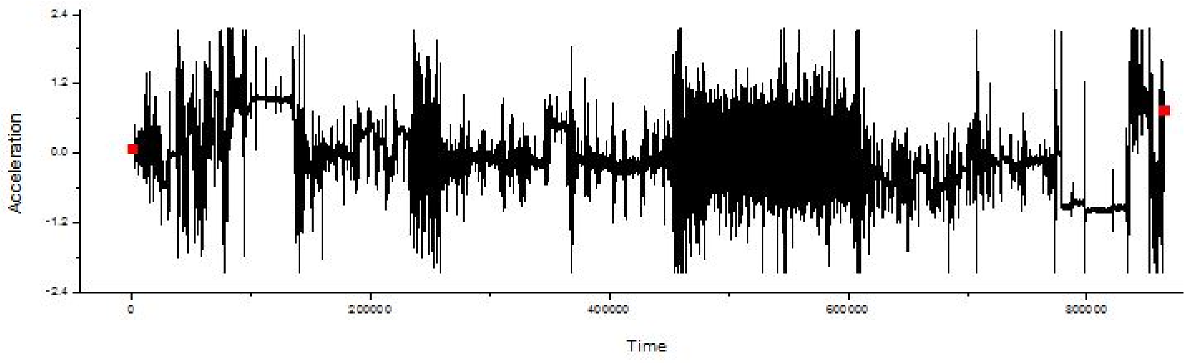


Figure 82: A heave acceleration plot produced using Origin Pro.

   Figure 83 shows the 2D plot of the same data produced using our visualisation software. By directly comparing these two plots, it is immediately obvious that both applications have produced the same plot.

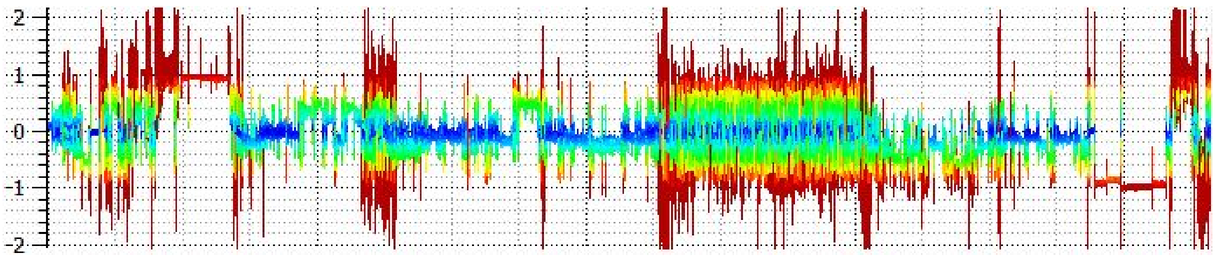


Figure 83: A heave acceleration plot produced using our visualisation software.

After testing the application with the other data sets and a variety of data attributes, all the plots produced correct results.
Pass / Fail: Pass ✔

2. **Derived Static Component of Acceleration**
Test Case:
The static component of acceleration is found by passing a smoothing filter over the data set. A smoothing filter essentially produces an average of a curve within a specified window. By directly comparing the total and static acceleration values it is possible to see if the static component is derived correctly. Two smoothing filters have been implemented in the application, Savitzky-Golay and the Moving Average filter. These must both be tested for correctness.

Expected Result:
It is expected the static component will be a smoother version of the total acceleration curve. The Moving Average filter should produce a flat curve which does not preserve peeks and troths where as Savitzky-Golay should produce a curve which preserves some of these features.

Result:
Figure 84 shows a 2D shared space time verses intensity plot of static and total acceleration data. A sub-sample of approximately 1 minute of data from the data set "111207_Lindt_penguin.asc' has been visualised. The static component of acceleration has been derived using the Moving Average filter. It is clear from using the window painted on the plot as a guide that the static component has created a correct average of the total acceleration curve. Using the window painted on top of the plot it is also clear the average has been computed correctly.

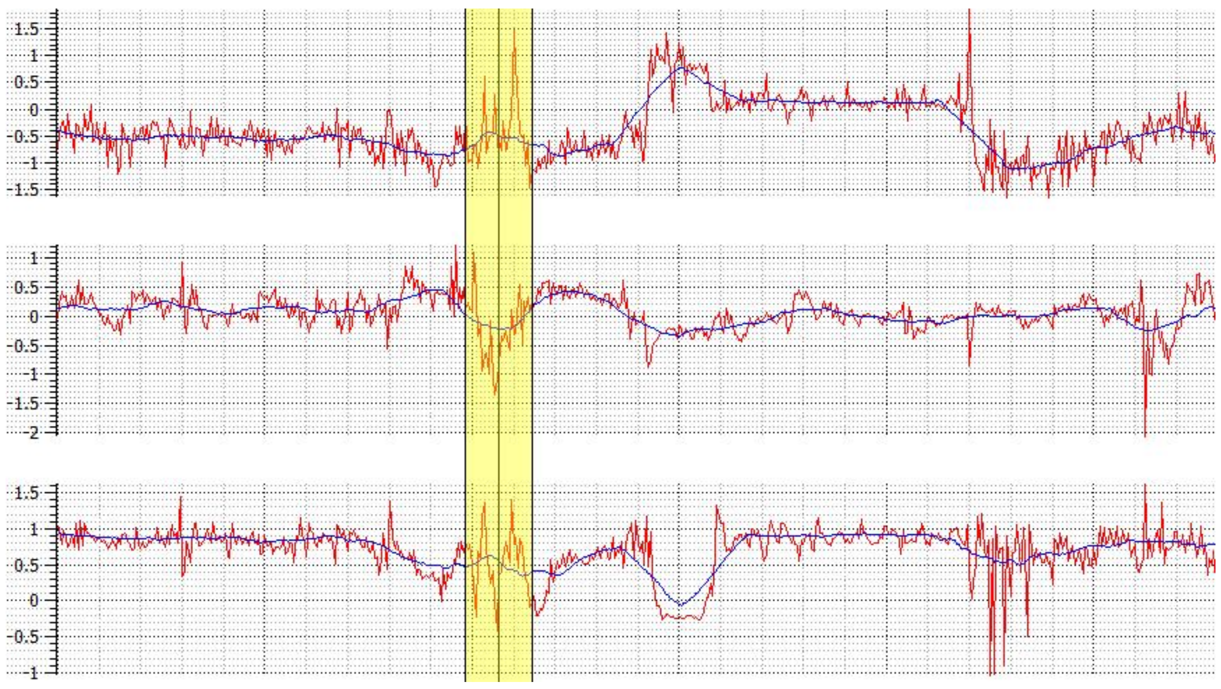


Figure 84: Shared space time intensity plot of static acceleration (blue), calculated using a windowed Moving Average filter and total acceleration (red). Window size used for computing static component is highlighted.

Figure 85 shows the same data plotted with static acceleration derived using the Savitzy-Golay filter. This curve preserves the features of the total acceleration plot, and it is also clear with guidance from the painted window that a correct average has been found.

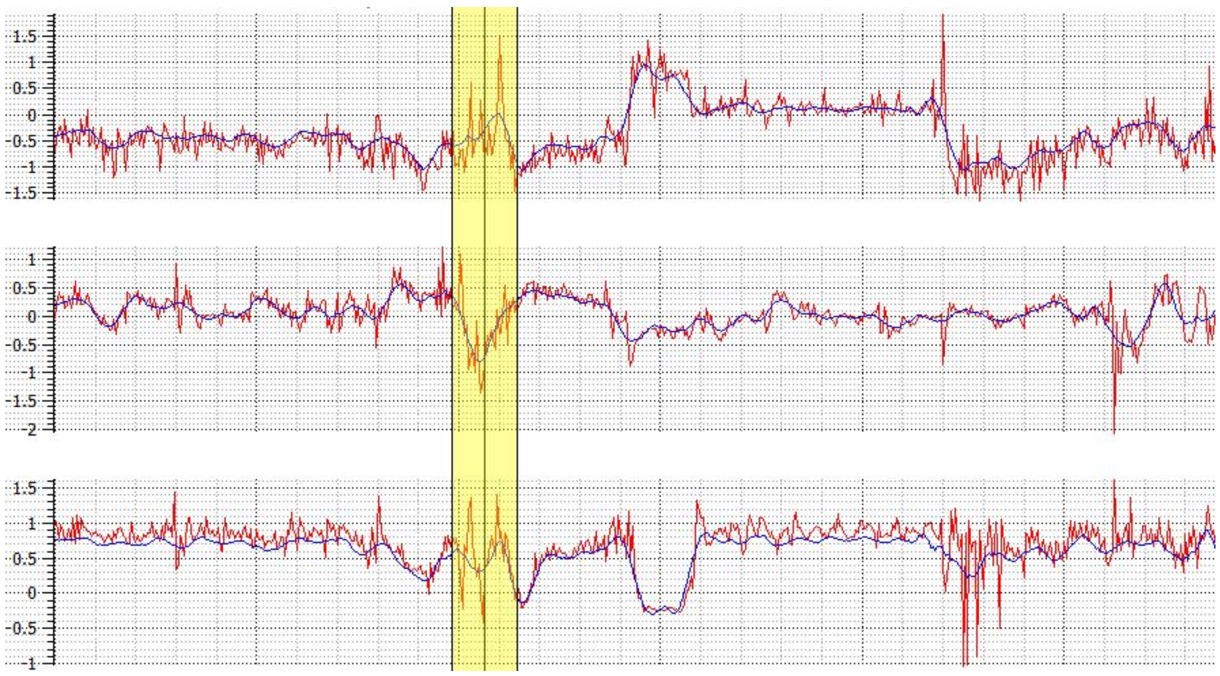


Figure 85: Shared space time intensity plot of static acceleration (blue), calculated using a Savitzy-Golay filter and total acceleration (red). Window size used for computing static component is highlighted.

The test was then applied to a variety of sub-sets of data and different data sets. Window sizes were also adjusted to safeguard the results of these two filters is correct. All graphs plotted produced correct results.
Pass / Fail: Pass ✔

3. **Derived Dynamic Component of Acceleration**
   Test Case:
   Dynamic acceleration is computed by subtracting static acceleration away from total acceleration. The static component has been proven correct in the previous test case, it is therefore now possible to test the dynamic component has also been derived correctly. By plotting the total, static and dynamic components of acceleration in a shared space graph it is possible to visualise if the components are subtracted correctly. Graphs can then be manually inspected to test the derived dynamic component is correct.

   Expected Result:
   It is expected the dynamic acceleration curve will be the static acceleration curve subtracted away from the total acceleration curve.

   Result:
   Figure 86 shows the individual acceleration components plotted on shared space 2D time intensity plots. A sub-sample of approximately 2 minutes of data from the data set "041207_Roy.asc' has been visualised. The Static acceleration component is computed using a moving average filter. Using a moving average filter means the static acceleration curve is less tightly bound to the total acceleration curve, this makes it easier to check

for correctness of subtraction. We can see the components are subtracted correctly by picking points on graph and verifying correctness. For example in figure 86, at point A, Total acceleration is equal to 1 and static acceleration is equal to 0.6. Using this data it is predicted the derived dynamic component is equal to 0.4. Using the graph we can see the dynamic component is equal to 0.4, the dynamic component is calculated correctly at this point.

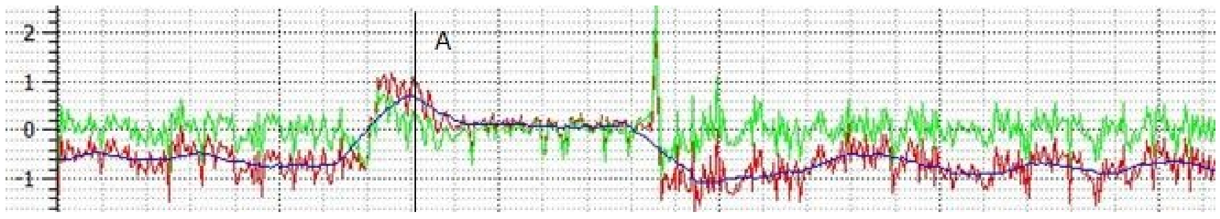


Figure 86: Shared space time intensity plot of static acceleration (blue), dynamic acceleration (green) and total acceleration (red) in the heave axis.

This test was then applied to acceleration in the other axes, a variety of different data sets, sub-sets of data and sampling locations. All graphs plotted produced correct results.
Pass / Fail: Pass ✔

4. **Scatter Plot**
   Test Case:
   The scatter plot visualisation consists of a direct mapping of acceleration and compass data to a 3D space. It is possible to test this visualisation by depicting patterns in the 2D plots. Finding patterns in combined plots it will then be possible to test if the same patterns still occur when visualised as a 3D scatter plot. This will tell us if the data items are being plotted correctly.

   Expected Result:
   It is expected the same patterns which exist in the combined 2D plots will still exist in the 3D scatter plot.

   Result:
   Figure 87 shows the three 2D time plots of raw acceleration in the X, Y and Z axis. From visual inspection of the plots it is clear the most common data items exist in the following domain: In the X axis most data items have a value below zero, In the Y axis the values fluctuate above and below zero and in the Z axis the majority of data values are above zero with a average value of approximately 0.8g.

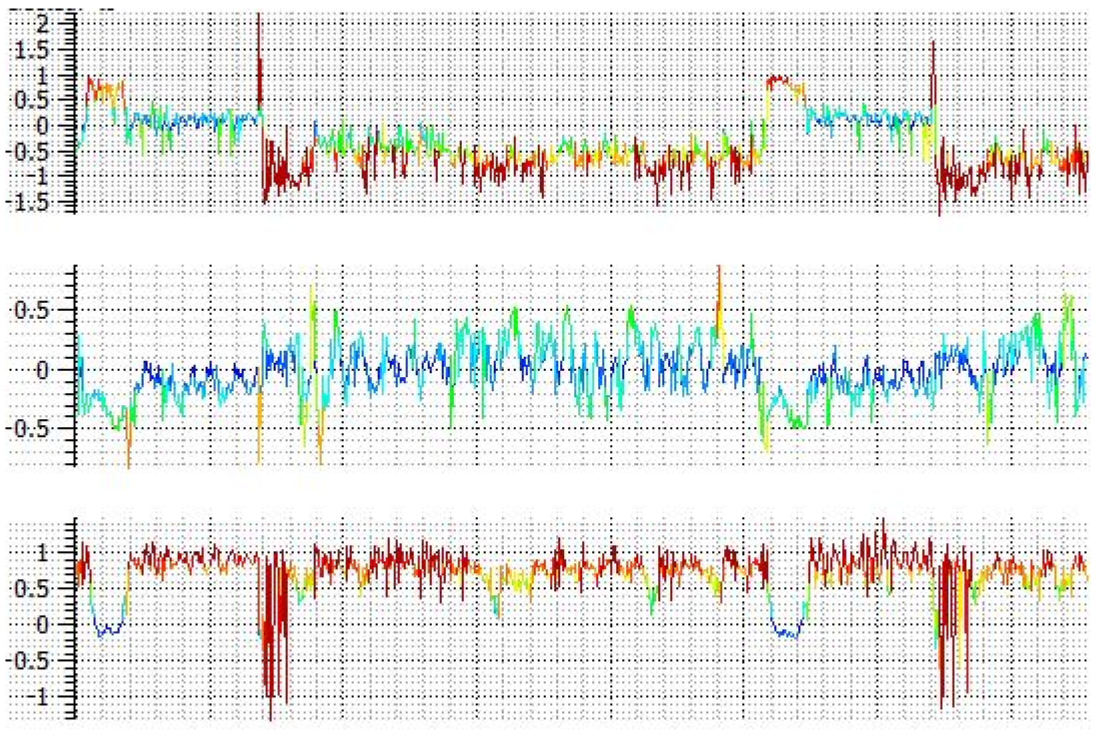


Figure 87: A time intensity plot of dynamic acceleration in the heave, surge and sway acceleration axes.

Figure 88 shows a scatter plot visualisation of this same data described above. It is clear the same patterns occur with the data existing the in the same domain outlined. The majority of data items in the X axis (red) have a value below zero (left size). Data items in the Y axis (green) are above and below zero. In the Z axis the majority of data values are above zero (closest to the screen) and are around the 0.8g mark (about of the length along the z axis lines).

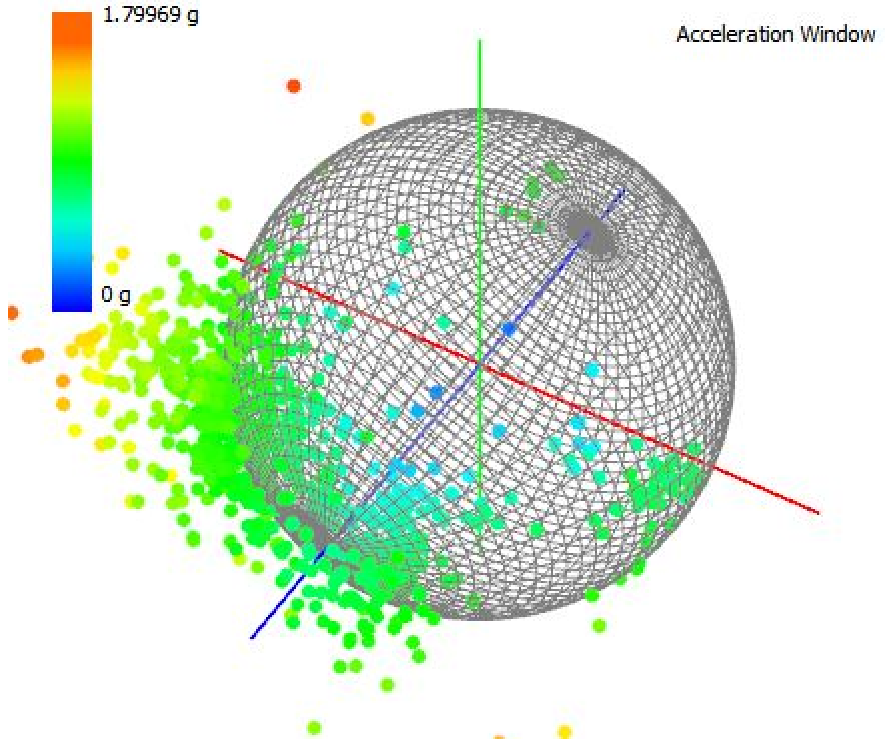


Figure 88: A raw scatter plot visualisation of dynamic acceleration data.

In order to verify this visualisation is correct it must be tested using the same technique for compass data as well as the other components of acceleration. It is also vital to test this on a variety of different data set sizes showing different patterns.

Pass / Fail: Pass ✔

5. **Spherical Scatter Plot**
   Test Case:
   The spherical plot visualisation maps raw data items onto the surface of the central sphere. It is possible to test this visualisation using the previously tested raw scatter plot visualisation. When a line is drawn from the origin to a spherical plot data item, it should also pass through the raw data item. When mapping to the sphere surface, the magnitude of a point is changed to the radius of the sphere, the direction still remains constant. This enables us to show the mapping is correct by testing a line originating from the origin to the spherical plot data item passes through the raw data item.

   Expected Result:
   It is expected the raw data values will lie on the line from the origin to the spherical line plot.

   Result
   During playback each point is inspected for correctness, a step function is used to go to the next time step. Figure 89 displays a tested point of total acceleration. A line originates from the origin passing through both the raw (a) and spherical (b) values. The mapping displayed here from raw to the central sphere is correct.

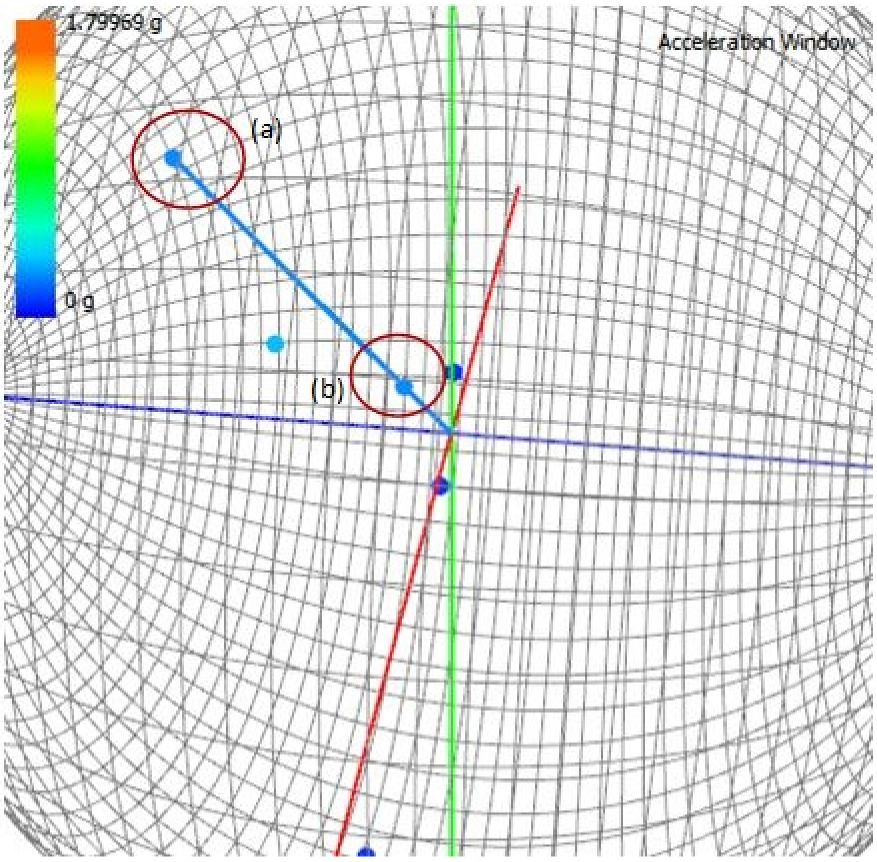


Figure 89: A spherical scatter plot and raw scatter plot visualisation of total acceleration data. A line is projected through a spherical point from the origin to show the mapping from raw (a) to spherical (b) is correct.

This test was also applied to several subsets of data from a variety of data sets. All the results produced were correct.

Pass / Fail: Pass ✔

6. **Histogram**
   The histogram visualisation algorithm is more computationally complex than any of the other visualisations. To test this visualisation the histogram algorithm is broken down into smaller components. Each step of the algorithm can then be tested individually to ensure the algorithm is correct. Test cases follow:

   (a) Histogram sub-test 1
   Test Case:
   The first step of the histogram algorithm is to discover which bin a point resides in. Painting bins which have data items residing in them a different colour makes it possible to visualise if the algorithm is computing the mapping from points to bins correctly. Each bin with at least one point residing inside will be painted red to make the selected bins stand out.

Expected Result:
It is expected the data structure will only paint bins with points residing in them. No other bins should be painted in red.

Result:
Figure 90 shows the result of the bin data structure. As can be seen the bins painted red are the only ones with data items residing inside of them.

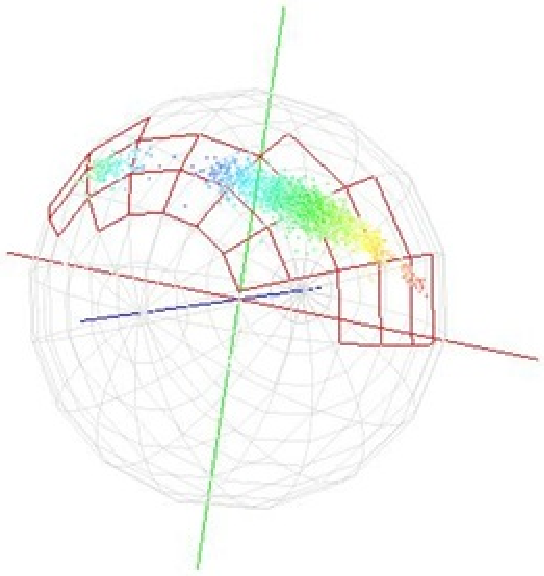


Figure 90: The data structure for the histogram is rendered to the screen. Bins which have points residing inside of them are painted red.

This test was then applied to a variety of different data sets with a variety of different patterns existing in the data. Each test passed and was successful.

Pass / Fail: Pass ✔

(b) Histogram sub-test 2

Test Case:
A step function has been implemented for testing purposes. As described by Laramee in 'Using Visualization to Debug Visualization Software', a step function sets playback to the next time step each time it is pressed. Using the play back step function and glyph visualisation, it is possible to watch the histogram visualisation grow step by step. At each step it can be ensured the correct bins are growing by using the glyph visualisation. The glyph visualisation shows the current vector being plotted and gives a visual aide by pointing at the bin which has been grown (incremented) at the time step.

Expected Result:
Each bin being pointed at by the glyph visualisation should be incremented by one. Visually this means, a bins height and colour should increase if it is not already the current largest bin density. Otherwise the rest of the bins should decrease in size leading to a reduction in colour mapping due to the normalisation process.

Result:
Figure 91 displays three spherical histograms applied with a glyph visualisation, each visualisation displayed is of consecutive time steps. It can be seen that each bin pointed at by the glyph visualisation is creating a cube. Each cube is displaying an intensity of one point residing in each bin, this can be seen by the green colouring of cubes and the height is half of that of the maximum red cubes which contain 2 points.

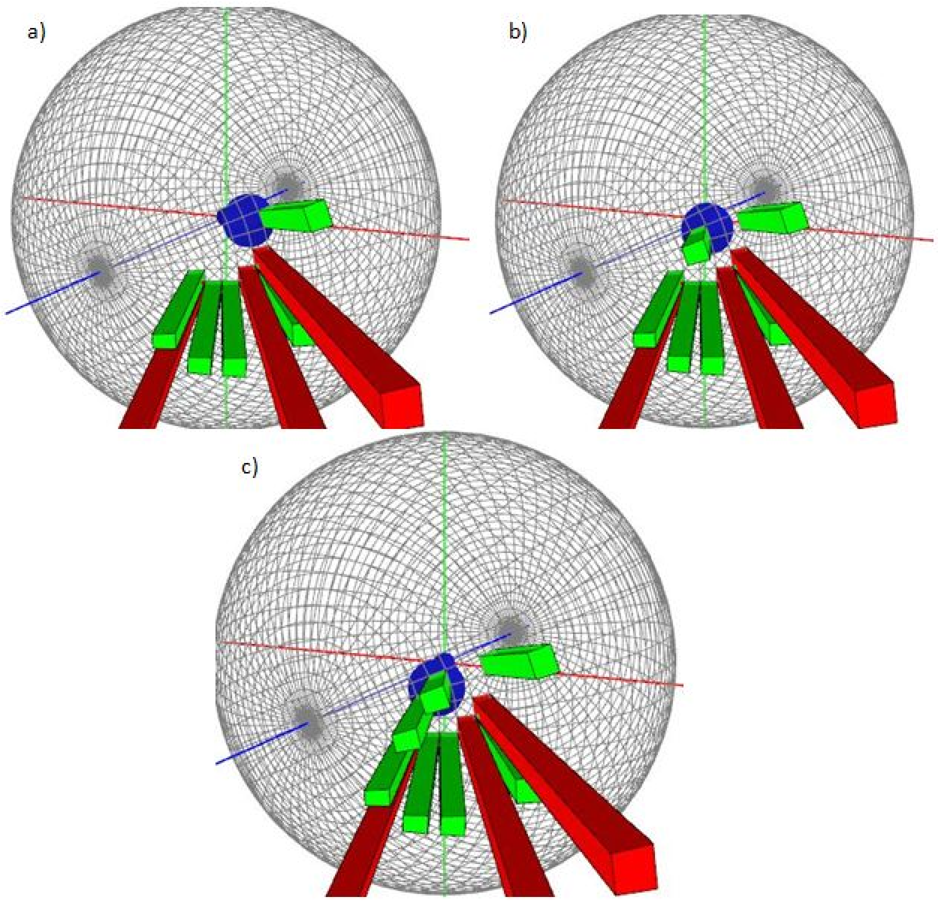


Figure 91: This image shows: Three spherical histogram visualisations with a glyph visualisation added of static acceleration data. Visualisations (a), (b) and (c) are all consecutive time steps, directly after each other. Each visualisation displays a bin being incremented from zero to one.

Figure 92 displays two spherical histograms, as before they are also applied with the glyph visualisation and are of consecutive time steps. In the second time step visualisation, the glyph points at a bin which already has one point residing in it. The cube pointed at by the glyph is therefore increased in height and the colour is changed to red, indicating the bin now contains two points.

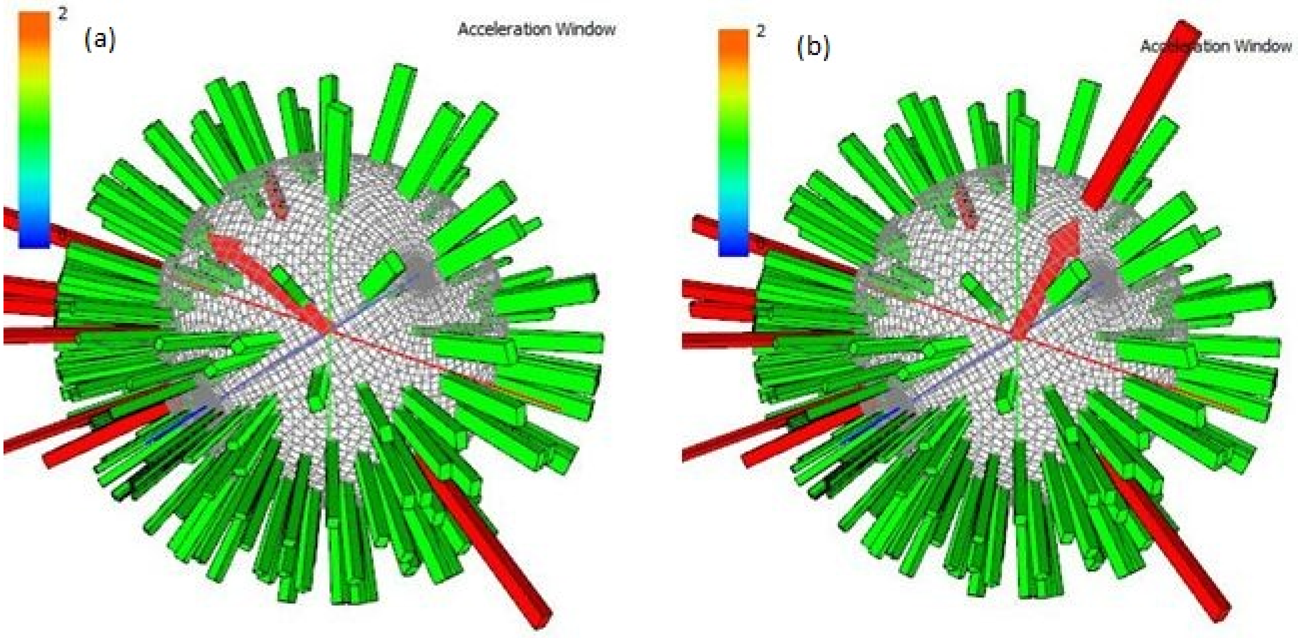


Figure 92: This image shows: Three spherical histogram visualisations with a glyph visualisation added of static acceleration data. Visualisations (a) and (b) are consecutive time steps, directly after each other. In visualisation (b) a bin is incremented which in visualisation (a) contains one point residing in it. It can be seen in (b) that the bin has been incremented and the bin now contains two points.

Finally, figure 93 displays two spherical histograms in the same set up as previously discussed. This time a glyph points at a bin which is the largest sized bin in the histogram. The bin visually stays the same, instead all other bins are reduced in size and their colour map reduced as well. The colour mapis now showing 3 data items instead of 2, another indication the bin has been incremented.

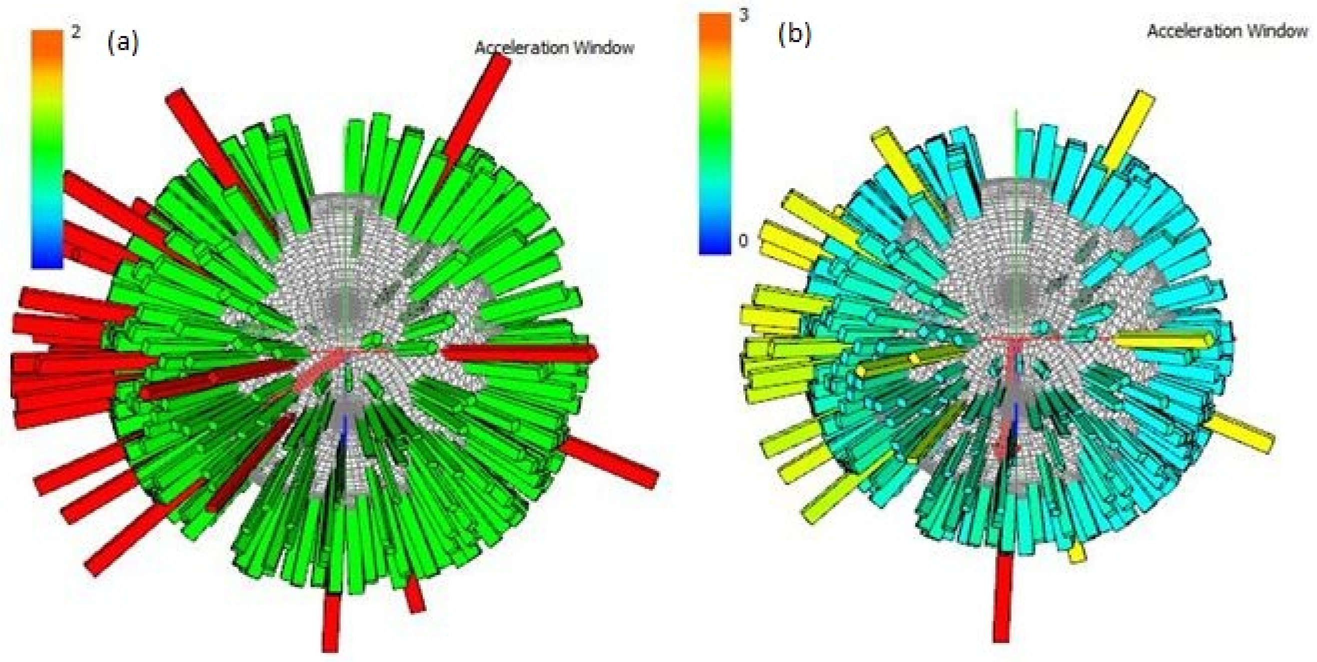


Figure 93: This image shows: Three spherical histogram visualisations with a glyph visualisation added of static acceleration data. Visualisations (a) and (b) are consecutive time steps, directly after each other. In visualisation (b), the maximum bin is incremented by one. In comparison to visualisation (a), the previous state of the histogram, all bins other then the bin being incremented have been reduced in size due to normalisation.

These results discussed show what was expected and therefore the tests have been passed. Other data sets and attributes have also been tested and have been found to also produce correct results.
Pass / Fail: Pass ✔

7. **Triangle Fan**
   The triangle fan visualisation plots future and previous time steps up to user defined amount. Both of the future and previous plotting options need to be tested separately for correctness. Each of these individual tests follows:

   (a) Triangle Fan sub-test 1
   Test Case:
   In this test the triangle fan for visualising previous time steps is tested. Using the spherical scatter plot visualisation it is possible to see if the outside vertexes of the triangles in the triangle fan match those of the points in the spherical plot.
   Expected Result:
   Each triangles outer vertex in the triangle fan visualisation should match up to a point in a spherical scatter plot. Points will have already been plotted in the spherical scatter plot and as the triangle fan is plotting previous time steps should correspond to a spherical point.
   Result:

Figure 94 shows the result of this test applied to static acceleration data. In part (a) an overview of the triangle fan visualisation is displayed, showing 20 time steps into the past. Part (b) shows the same result as part (a) zoomed in for clarity. Each green point in the visualisation is a point plotted by the spherical scatter plot visualisation. As expected each of the green points match up to a vertex in the triangle fan.

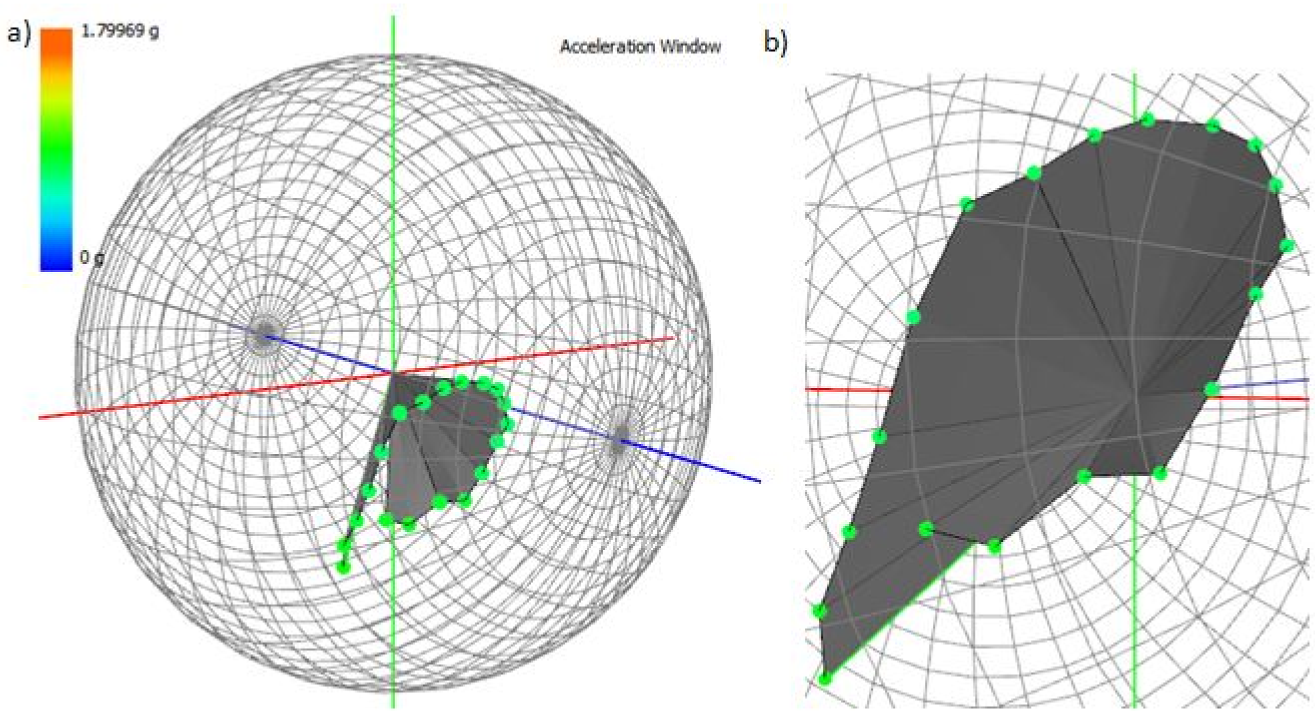


Figure 94: A triangle fan visualisation combined with a spherical plot visualisation of static acceleration data. Image (a) displays the resulting visualisation. Part (b) shows this same visualisation zoomed in.

The same test procedure was applied to other previous time step values and other data sets. All test cases were found to produce correct results.

Pass / Fail: Pass ✔

(b) Triangle Fan sub-test 2

Test Case:

In this test case the triangle fan visualisation is tested for visualising future time steps. This is slightly more challenging to test as it comprises of visualising time steps which have not yet been perceived in visualisations, creating effectively a look-ahead option for users. To test this visualisation the glyph visualisation will be used to trace the triangle fan and test it goes along the same path outlined by the triangle fan. Using the step playback function it is possible to test this on a step by step basis and make sure the glyph goes to the next triangle vertex.

Expected Result:

It is expected the glyph will follow the path outlined by the triangle fan. The triangle fan path should remain static and not change path while playing back through the data set.

Results:

Figure 95 shows three triangle fan visualisations of static accelerations, visualising 20 time steps into the future. Each visualisation sequentially shows a gap of 10 time steps during playback. At each time gap the glyph follows the path outlined in the triangle fan. The triangle fan also maintains its structure throughout and does not shift or produce snake like effects.

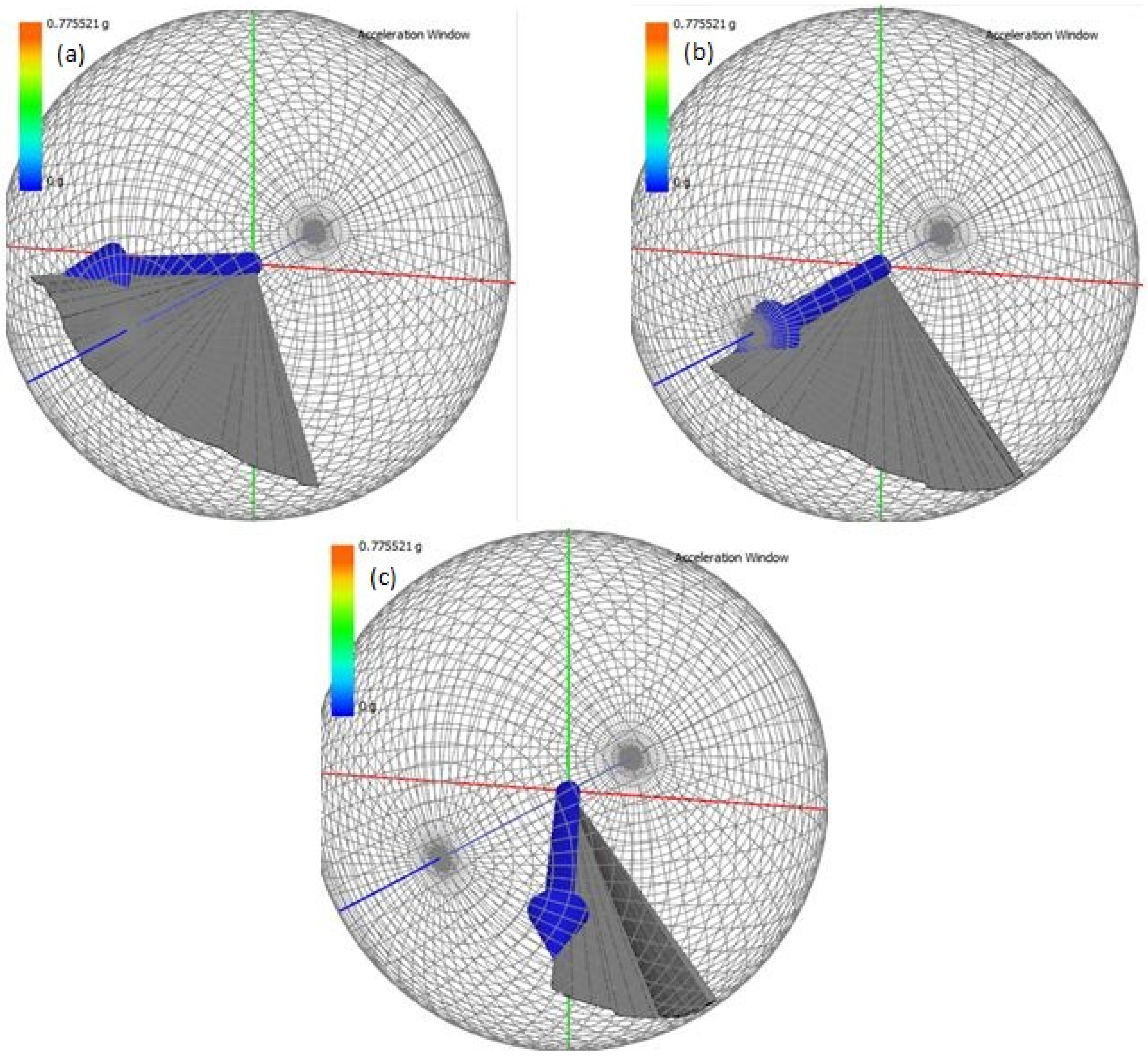


Figure 95: This image shows: Three triangle fan and glyph visualisations. Each visualisation (a), (b) and (c) are seperated by 10 time steps.

The results of this test were as expected. The test case was also applied to other future time step values as well as other datasets. Every data set applied to the visualisation produced correct results.

Pass / Fail: Pass ✔

8. **Glyphs**

   Test Case:

   In this test case the glyph visualisation is tested. Glyphs are a graphical representation of the current 3D compass or acceleration vector at a specified time step. A vector is the summation of each of the 1D data items in there respected axes. To test this visualisation each of the individual data items can be visualised as separate line segments. Visualising the line segments separately will allow us to visually combine the line segments together to verify the 3D glyph has been plotted correctly.

   Expected Result:

   It is expected the combination of all of the line segments plotted will produce the glyph as visualised.

   Results

   Figure 96 shows a glyph visualisation of static acceleration data plotted along with each of its three separate line segments. It is obvious by looking at the individual line segement components and combining them that the glyph has been plotted correctly.

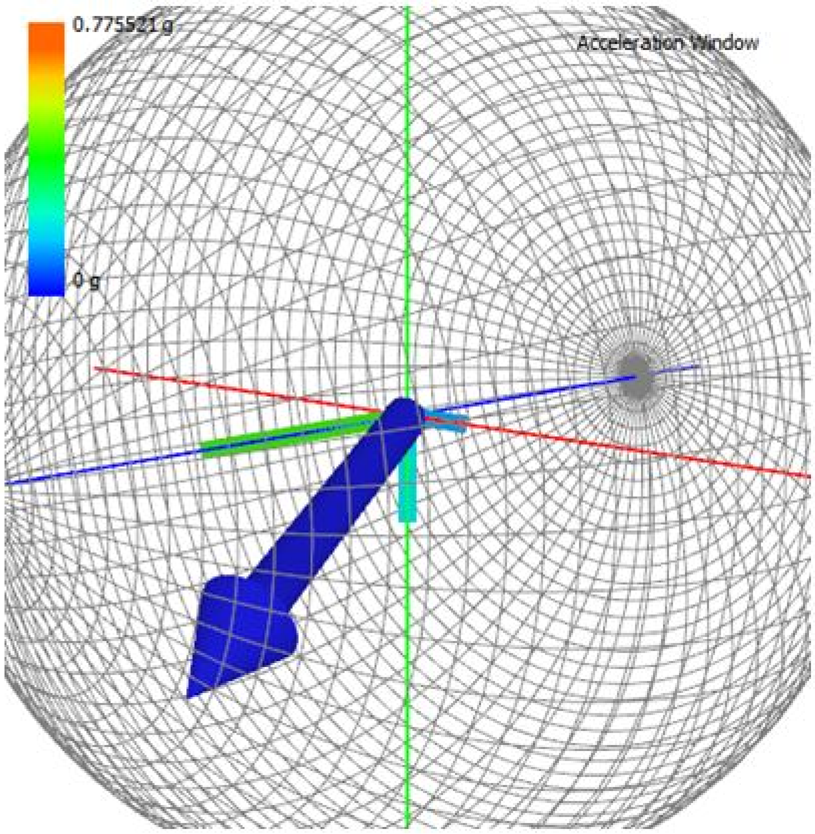


Figure 96: A glyph visualisation of acceleration data plotted along with each of its individual line segments.

This test has also been applied to glyph visualisations of the compass data and all data attributes of acceleration using a variety of data sets. All of the tests have produced correct results.
Pass / Fail: Pass ✔

9. **Animal Geometry**
   Test Case:
   The animal geometry diagram plots the orientation of an animal during play back. This test case is to test this visualisation is plotted correctly. Animal geometry is produced from a series of rotations on a 3D geometric penguin and glyph. The rotations applied are in the form of a pitch and roll which are applied around the X and Z axis respectively. The pitch and roll values will be printed out to the terminal and then visually inspected for correctness in the visualisation.

   Expected Result:
   It is expected the pitch and roll components printed to the terminal will match those as applied to the glyph and animal geometry.

   Result:
   Figure 97 displays the resulting animal geometry visualisation when applied with a pitch of 321 degrees and roll of 347 degrees. OpenGL uses the right hand rule for rotation and as such we can see that the pitch rotation of 321 degrees is approximately correct. It is also possible to see the roll is little to none and as such the model is almost aligned to the x axes. This shows the correctness of the visualisation.

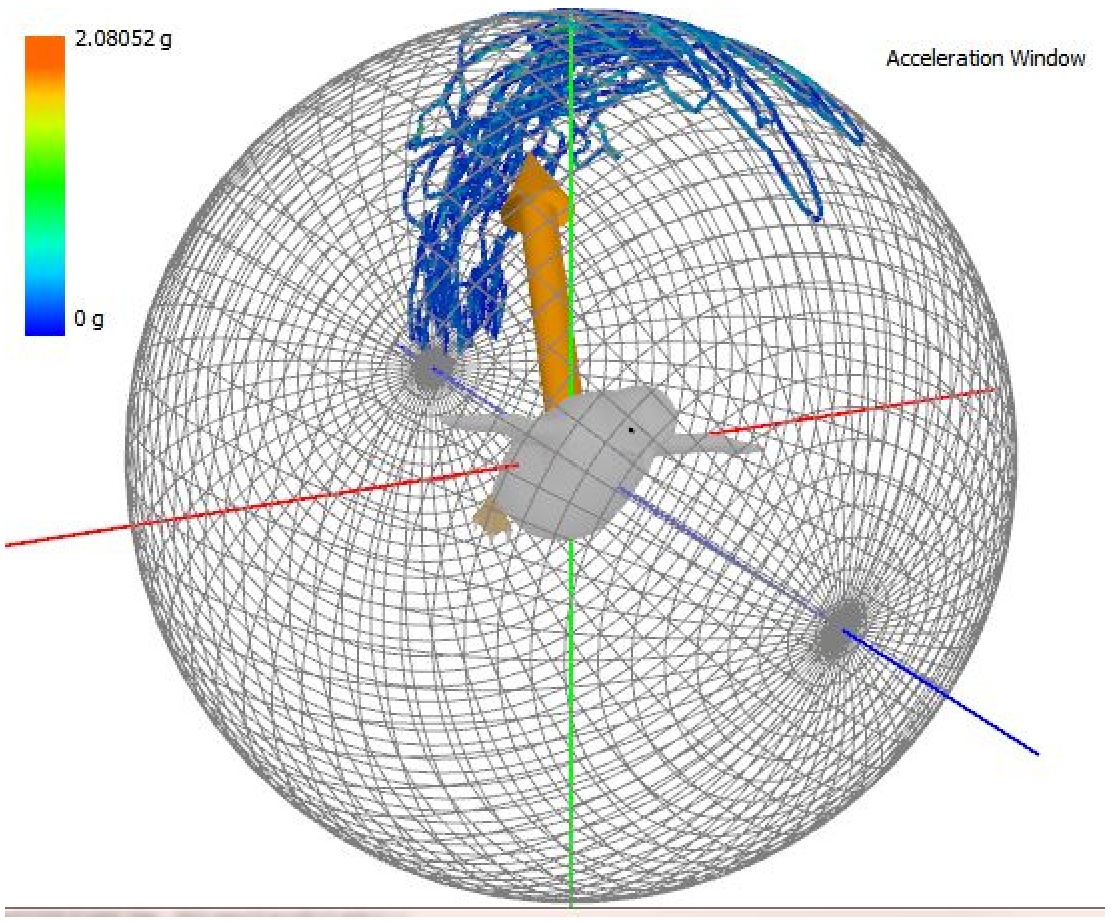


Figure 97: An animal posture diagram displaying a pitch of 321 degrees and a roll of 347 degrees

The visualisation has been tested under the same test case with a variety of other data sets of diving, flying and upright animal orientations.

Pass / Fail: Pass ✔

7.2 Results

Each of the visualisations produced during the project will now be presented. The insight being gained from each of the visualisations will be described to enable the benefits our visualisations to be seen.

7.2.1 Raw Scatter Plot

The raw scatter plot visualisation can be used to visualise attributes of static, dynamic or total acceleration as well as compass data. Data is directly mapped into a three dimensional spatial domain using the values logged in the data files. This visualisation does not give an intuitive step to derive behavioural patterns but instead produces a more knowledge rich visualisation in comparison to the time intensity plots. Instead of the biologists having to manually combine three plots of acceleration or compass data, it is now possible to see this visual mapping immediately in a 3D space.

Figure 98 shows a raw scatter plot produced using data from a Magellanic Penguin. 13 minutes and 2 seconds of data is visualised of Total dynamic acceleration data (left) and compass data (right). Using this visualisation it is now possible to see the spatial distribution of the data. Compass data is clearly clustered together showing that the animal has not changed heading much over this time period. Acceleration data is sparsely distributed, potentially showing an animal activity of interest is occurring.

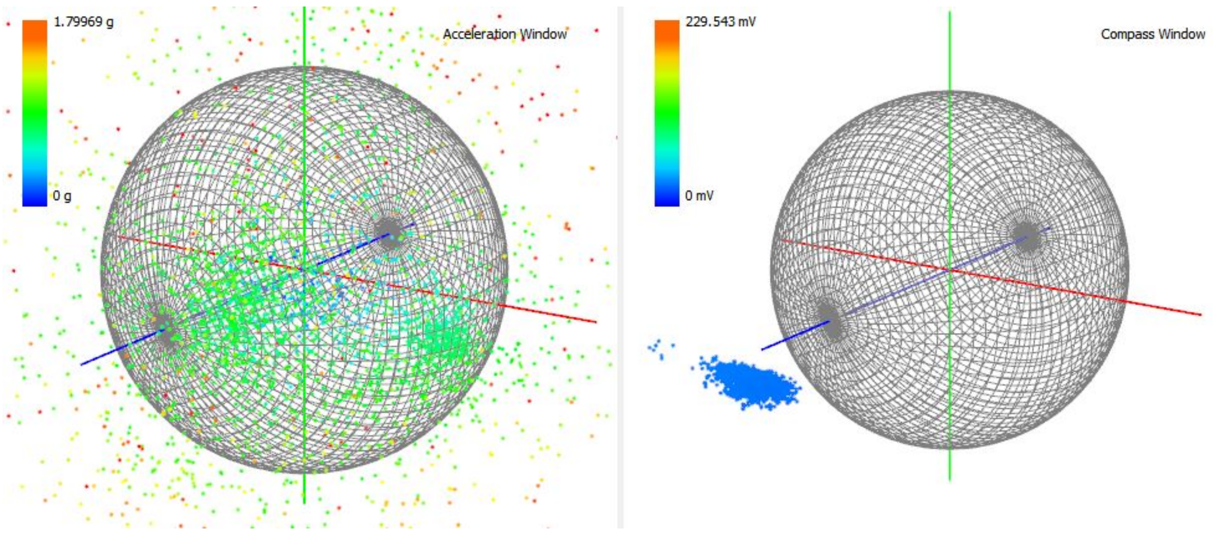


Figure 98: A raw scatter plot produced using data from a Magellanic Penguin. 13 minutes and 2 seconds of data is visualised of Total dynamic acceleration data (left) and compass data (right).

Using the aims outlined in the requirements, this visualisation achieves the following goals:

- **Combining multiple attributes together into one visualisation** - Three data attributes have been combined together into one visual aide.
- **Quicker Data analysis** - This visualisation ultimately saves time for the biologists. Instead of having to directly combine plots together in their mind, the data attributes are now directly mapped into a 3D space.

7.2.2 Spherical Scatter Plot

The spherical scatter plot visualisation can also be used to visualise attributes of static, dynamic or total acceleration as well as compass data. Data is effectively normalised to the central sphere hiding any magnitude information from users. Typically biologists are not interested in the magnitude information, if an animal is in a certain orientation it does not matter what the strength of the signal is but that the animal is that specific orientation. Similarly the same can be said about dynamic acceleration movement and compass headings.

Figure 99 shows a spherical scatter plot visualisation of static acceleration data (left) and compass data (right) collected from a device attached to an Imperial Cormorant. A sub-set of 3 minutes 4 seconds of data is visualised. This visualisation shows common data distributions. For example, in figure 99, on the left, common animal orientations are shown and on the right, common compass headings are visualised.

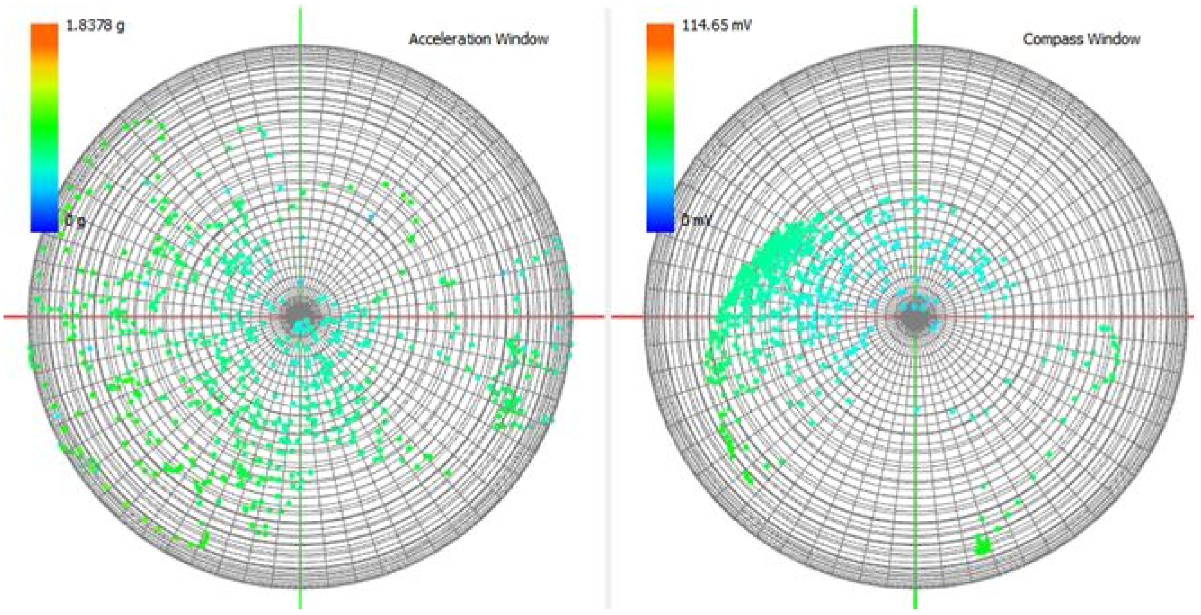


Figure 99: A spherical scatter plot visualisation of data from a device attatched to an Imperial comorant. 3 minutes and 4 seconds of static acceleration data (left) and compass data (right) are both visualised.

A problem occurs when visualising large data sets using this visualisation. Plots can easily become over crowded when selecting large sub-sets of data. Figure 100 shows an example of this when 1 hour 9 minutes of data is selected. It is hard to extract much information from this plot as it is so overcrowded.

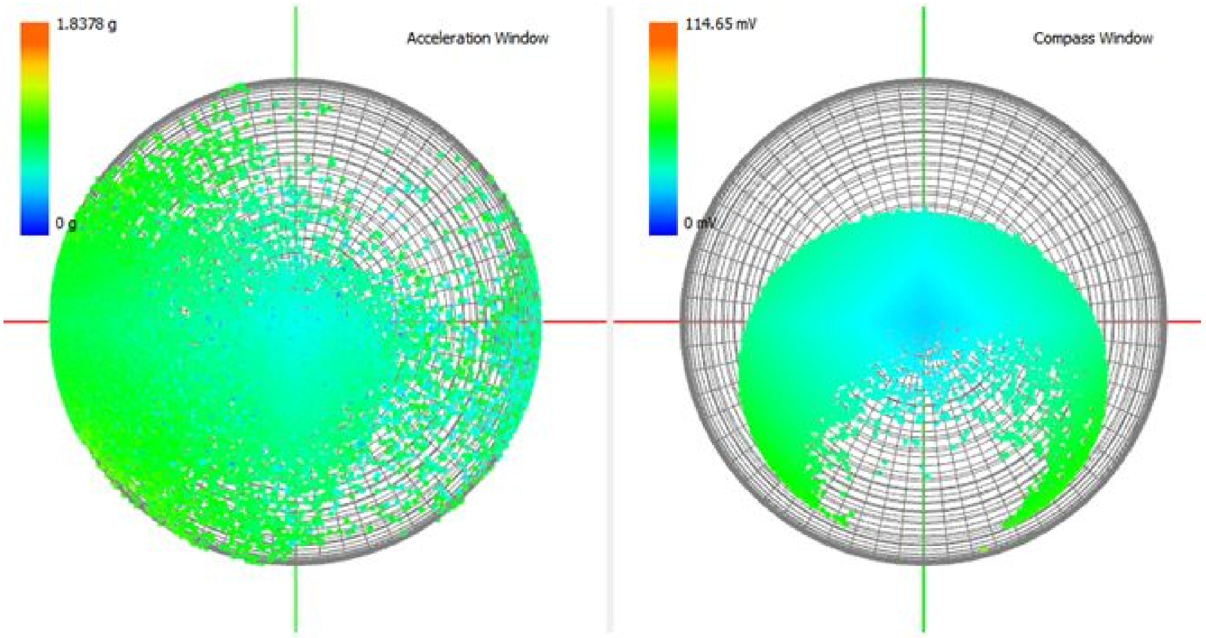


Figure 100: A overcrowded spherical scatter plot showing 1 hour and 9 minutes of static acceleration data (left) and compass data (right).

Outlined below are the requirements achieved by this visualisation:

- **Enabling pattern finding capabilities** - Finding common patterns in data sets would not have been possible to biologists, using the time intensity plots. Now, finding common data distributions of compass headings, total, static and dynamic acceleration is possible.

7.2.3 Spherical Histogram

The spherical scatter plot has issues with overcrowding when visualising large data sets. The histogram visualisation can be utilised to visualise the same types of data as the spherical scatter plot. The histogram assists in solving overcrowding by counting the number of points inside each sphere segment and then showing this count in terms of bin height and colour. This aids in showing common data patterns with large and small data sets, adding additional clarity in comparison to the scatter plot visualisation.

Figure 101 displays this visualisation technique using data captured from a Leatherback Turtle. The visualisation displays a sub-set of approximately 15 minutes of data. On left hand visualisation common animal orientations (static acceleration) is shown. The right hand visualisation is displaying common compass headings. The highest bin, painted red, displays the most common data attribute for each of the sub-sets of data visualised.

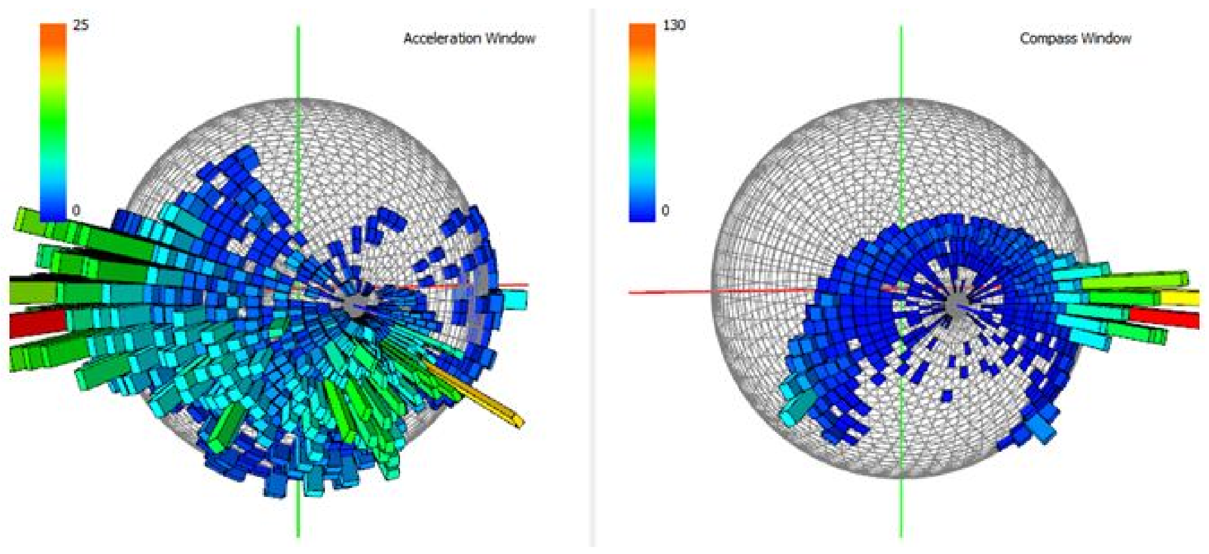


Figure 101: A spherical histogram visualisation of data captured from a Leatherback Turtle. 15 minutes of static acceleration data (left) and compass data (right) is visualised.

Outlined below are the requirements achieved by this visualisation:

- **Enabling pattern finding capabilities** - As mentioned in the spherical scatter plot visualisation.
- **Quicker data analysis** - It is immediately obvious which are the most common data patterns in the data set.
- **Making data analysis less reliant on skill of biologists** - The visualisation makes it intuative to see which is the most common data pattern in the data set and does not require any expert knowledge to derive this.

7.2.4 Triangle Fan

The triangle fan visualisation is used to perceive future and previous time steps of dynamic, static and total acceleration data. This visualisation is used as an aide a user to perceive future and past time steps. This allows for a more knowledge rich experience during data analysis.

Figure 102 shows a triangle fan visualisation of orientation data (static acceleration) of a device attached to a Magelinc Penguin. This particular visualisation shows 26 steps into the future during playback. The fan created is wide and spread out indicating a large amount of movement is occurring over the visualised time steps. If the fan is relatively closed and doesnt spread out it is a clear indication of not much movement in the animal. This same strategy can also be applied to movement data (dynamic acceleration) and total acceleration.

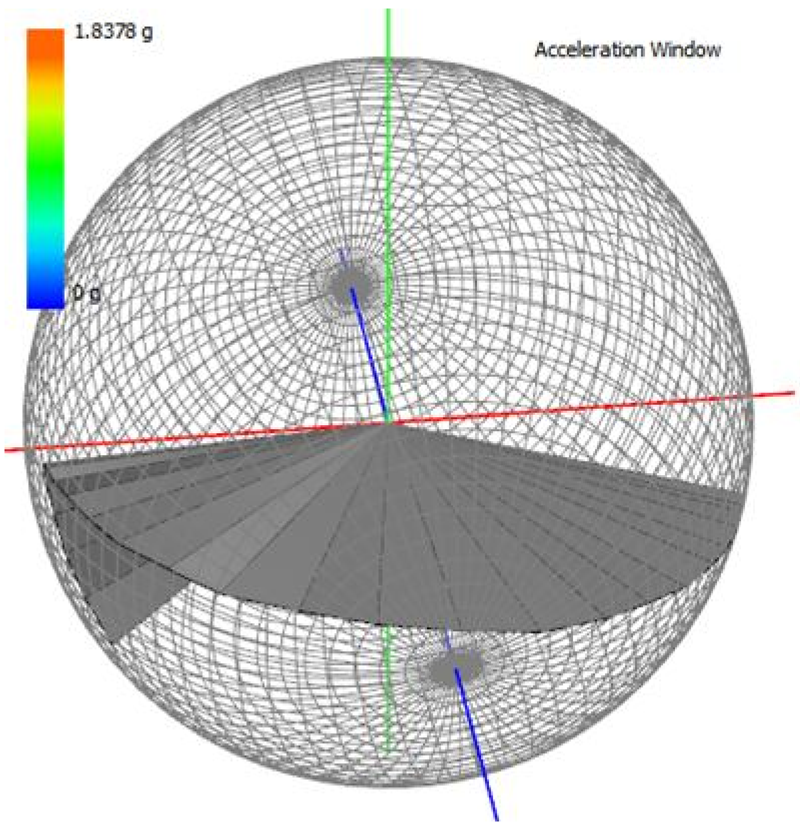


Figure 102: A triangle fan visualisation of data captured from a Magelinc Penguin. 26 time future steps are visualised of static acceleration data.

Outlined below are the requirements achieved by this visualisation:

- **Making data analysis less reliant on skill of biologists** - Users do not have to remember the time steps they have visualised and instead can view them in the visualisation for direct comparison between time steps.

7.2.5 Central Glyphs

Central glyphs show the current movement vector for static, dynamic and total acceleration data as well as the current compass heading. This enables a user to perceive the current directional vector during play back.

Figure 103 shows glyph visualisations of both acceleration and compass data. On the left hand side glyphs of acceleration data are shown. Blue represents the static acceleration vector, green total acceleration and red dynamic acceleration. Each of these can be turned off by the user to reduce the amount of glyphs on display. On the right hand visualisation a glyph of the current compass heading is shown.

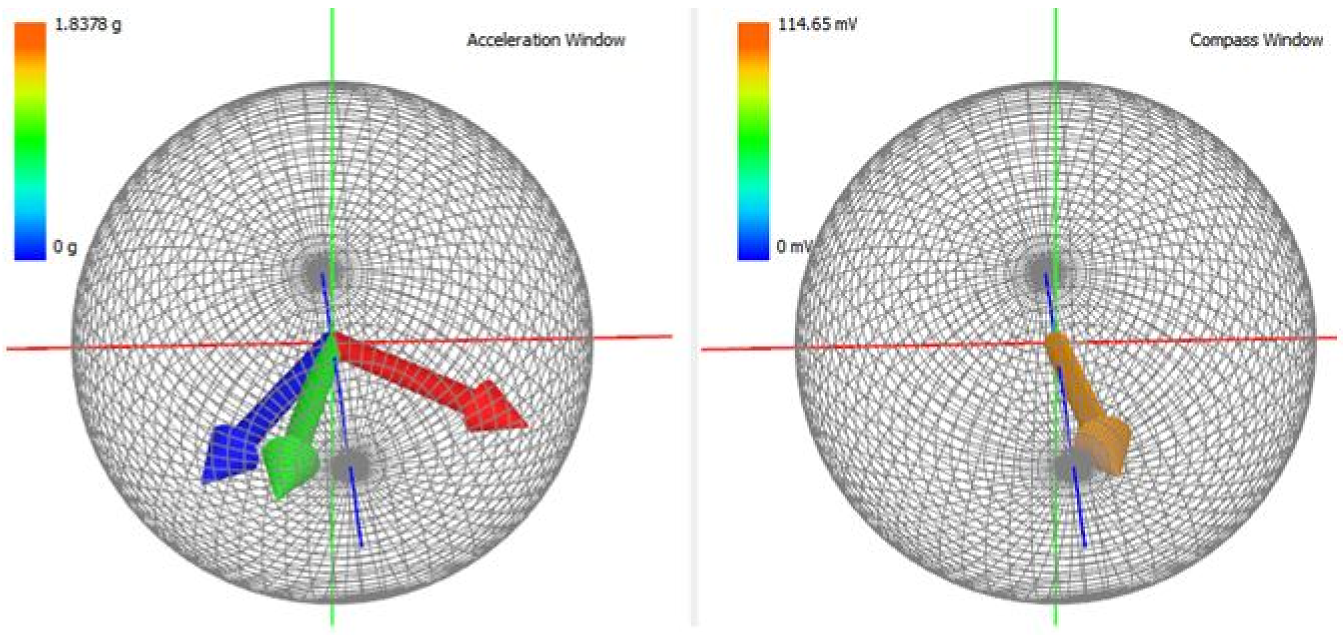


Figure 103: A glyph visualisation of data collected from a device attatched to an Imperial Cormorant. On the left hand side glyphs of acceleration data are shown. Blue represents the static acceleration vector, green total acceleration and red dynamic acceleration. On the right hand visualisation a glyph of the current compass heading is shown.

Outlined below are the requirements achieved by this visualisation:

- **Making data analysis less reliant on skill of biologists** - Previously it would have taken a lot of knowledge of the 2D plots to determine an animals compass heading or movement vector. It is now possible to instantly see the direction vector during playback.
- **Quicker Data analysis**

7.2.6 Animal Posture Diagrams

The final visualisation produced recreates actual animal movement. During playback the pitch and roll of the animal are applied to a geometric model of a penguin. This is an intuitive visualisation to assist a user in seeing what an animals orientation is at the time of play back.

Figure 104 shows a data set of an unidentified species of penguin. The playback has been paused as the penguin is ascending upwards in water. It is possible to tell this by looking at the orientation of the animal, in this example the animal is tilted upwards. Combined with a plot of pressure (bottom plot in the focus window) it is possible to see the animal is going into a medium less dense, above water.

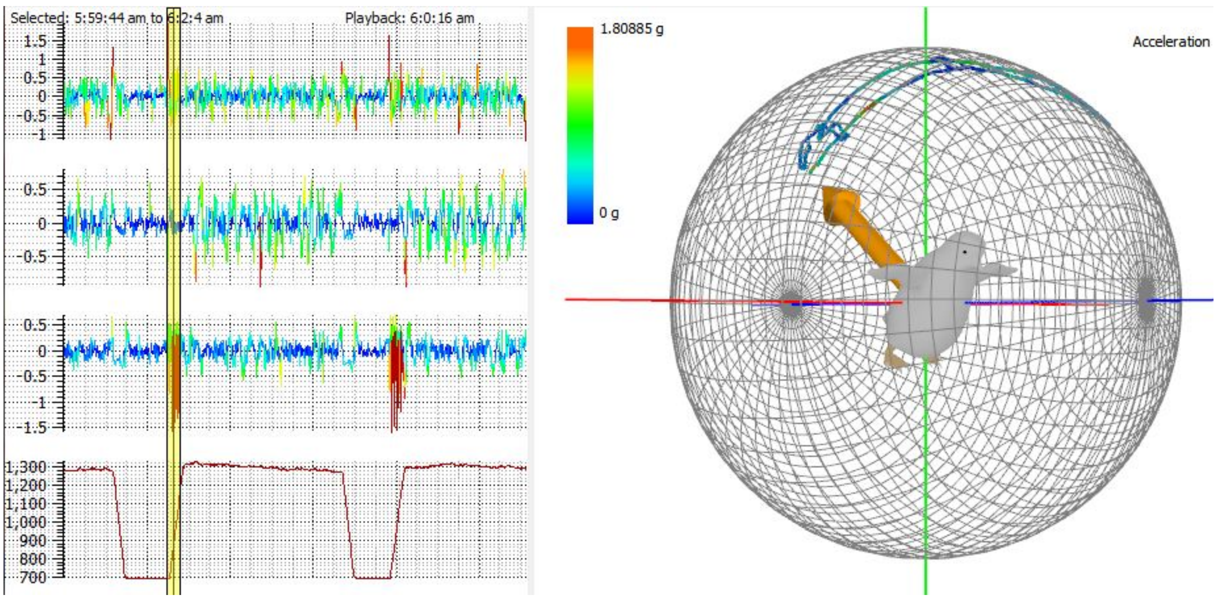


Figure 104: An animal posture diagram (right) alongside the focus window (left) of data collected from an unidentified species of penguin. Here, the animal is shown ascending in water.

Figure 105 shows the penguin descending back under water, 1 minute after the previous visualisation shown above, which showed the penguin ascending in the water. Including this visualisation again with a plot of pressure it is possible to see that the animal has gone back under water.

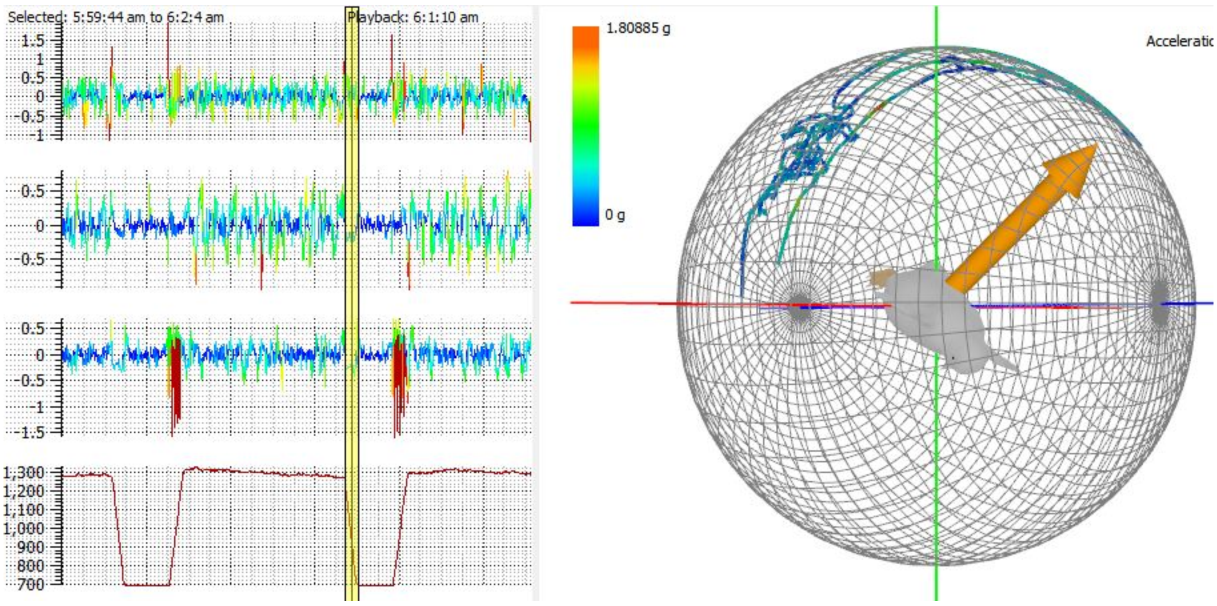


Figure 105: An animal posture diagram (right) alongside the focus window (left) of data collected from an unidentified species of penguin. Here, the animal is shown descending in water.

Using the aims outlined in the requirements, this visualisation achieves the following goals:

- **Quicker Data analysis** - Previously it would have taken a lot of interpretation effort to derive what orientation an animal is at. Now the data can be played back in real time displaying an animals current orientation.
- **Making data analysis less reliant on skill of biologists** - The ultimate goal of this requirement is to recreate animal movement, that is, what the animal was doing at the time of the recording. This has been partially achieved by creating a visualisation which plays back through the animals orientation positions. This provides an intuitive visualisation of which makes it immediately obvious to anyone looking at the visualisation what orientation the animal is at.

8 Conclusions

The outcome of this project has been a great success, achieving the goals set out in the requirements section. Previously, data was analysed on 2D time intensity plots, this was a time consuming and error prone process. The visualisations we have created combat these problems and provide the biologists with a more pleasurable data analysis experience.

Visualisations produced in this project allow for a more knowledge rich environment during data analysis. Visualisations are much more intuitive in comparison to the 2D time series plots. Previously, it would take a lot of interpretive effort and skill to be able to extract animal behaviour from these plots. Now, it is possible to instantly extract information by the use of intuitive knowledge rich visual representations of the data. For example, our animal posture diagram recreates animal movement making it possible to instantly see an animals orientation. We have also made visualisations which enable more knowledge than previously possible to be extracted from the data, such as common animal orientations, this would not have been possible using the old visualisations techniques.

A great software product has been produced which features many interaction techniques. A personal favourite of the authors is the play back feature, allowing users to play back through the data in real time, as the data was recorded. This adds additional clarity to the visualisations which only a custom bespoke solution could achieve.

The project has been managed well. All deadlines have been meet and often exceeded in most cases. The success of the project is put down to a good project design and choice of software model. Also where problems have occurred they have been rectified quickly, this is thanks to weekly meetings with the authors supervisor, Robert S Laramee and online forums provided by OpenGL and Qt, which have proved to be an invaluable resource.

Personally, this project has been a great learning experience for the author. Approximately seven months ago when the project started the author had no experience C++ or any graphics environments, let alone OpenGL. The author has defiantly come a long way since then and now feels comfortable programming using these tools. Taking a stand point and looking back at the project, it is obvious some things would be done differently, such as, implementing the histogram visualisation more efficiently, instead of recalculating every primitive during playback. Also the code produced in the earlier stages of the project when there was lack of experience is poor and often inefficient. However, the author is still very proud of what he has achieved.

To conclude, this has been a great project to work on and thoroughly enjoyable, most of the time! The author has a firm belief that as Daily Diary devices become more advanced they will be more prevalent in coming years and will become a key tool in monitoring animal behaviour. It is hopeful that visualisation techniques discussed and created will becoming a useful resource in discovering animal behaviour.

9 Future Work

This project has taken an analytical approach to visualising the Daily Diary data, briefly touching on recreating animal movement. Biologists at Swansea University are currently working on taking the next step towards creating a virtual environment system. The project, called Keyhole, involves making a system which they aim to recreate an animals actual behaviour in a real 3D environment, similar to that of Google Earth. Inside of this environment they hope to also recreate locale conditions, such as if it is sunny or raining. This would be a great leap forward in the visualisation of the daily diary data. I cant imagine a more intuitive visualisation than recreating the actual animal behaviour and movement as the animal was when it was wearing the device.

Figure 106 shows the current progress produced on the Keyhole project, displaying a whale shark in a 3D environment. A user can "swim underneath, behind or in front of a tag-fitted shark and choose to be very close or a more comforting farther away".

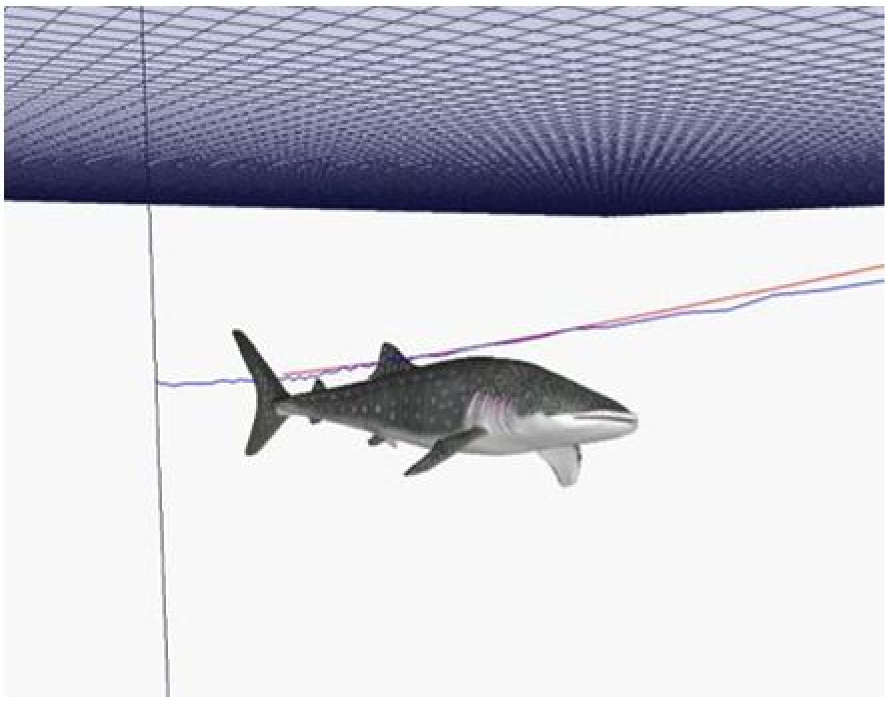


Figure 106: A screen capture of the current work being produced for the keyhole project. Displays a whale shark in a 3D environment following the path outlined in the image. Image Credit: http://www.swan.ac.uk/biosci/research/smart/smartsoftware/

10 Acknowledgments

Many thanks go to the authors project supervisor Bob (Robert S Laramee) for providing valuable discussions, feedback and guidance throughout the projects life span. Bob's paper 'Bobs BSc. Project Guidelines: Writing a Dissertation' has also proved to be an invaluable resource which has been followed meticulously throughout the project.

11 Reproduced Material

The following sections have been reproduced with minor corrections and additions in order to make this dissertation document complete:

- 2.1.1 Visualisation of Sensor Data from Animal Movement
- 2.1.2 Identification of Animal Movement Patterns Using Tri-axial Accelerometry
- 2.1.5 Graphical Perception of Multiple Time Series
- 2.3 Data Characteristics

All other work has been significantly edited or is original to this document.

References

[1] Science Alert. Whale shark diving discovered. `http://www.sciencealert.com.au/news/20081706-17506-2.html` [Last Access: February 2011].

[2] Rolex Awards. New technology to track animals. `http://rolexawards.com/en/the-laureates/rorywilson-the-project.jsp` [Last Access: February 2010], 2006.

[3] Tuan Nhon Dang, Leland Wilkinson, and Anushka Anand. Stacking graphic elements to avoid over-plotting. *IEEE Transactions on Visualization and Computer Graphics*, 16:1044–1052, 2010.

[4] Robert S. Laramee Rory P. Wilson Edward Grundy, Mark W. Jones and Emily L.C. Shepard. *Visualisation of Sensor Data from Animal Movement.* Swansea University, 2009.

[5] Flavio Quintana Agustina Gmez Laich Nikolai Liebsch1 Diego A. Albareda Lewis G. Halsey Adrian Gleiss David T. Morgan Andrew E. Myers Chris Newman David W. Macdonald Emily L. C. Shepard, Rory P. Wilson. *Identification of animal movement patterns using tri-axial accelerometry.* Brendan Godley, University of Exeter, Cornwall Campus, UK, 2008.

[6] Mark Summerfield Jasmin Blanchette. *C++ GUI Programming with Qt 4.* Prentice Hall, 2010.

[7] Waqas Javed, Bryan McDonnel, and Niklas Elmqvist. Graphical perception of multiple time series. *IEEE Transactions on Visualization and Computer Graphics*, 16:927–934, 2010.

[8] Andrew E. Myersa c Niko Liebscha Julian D. Metcalfed Jeanne A. Mortimere f Jonathan D.R. Houghtona, Allen Cedrasb and Graeme C. Hays. *Measuring the state of consciousness in a free-living diving sea turtle.* ELSEVIER, 2008.

[9] R. S. Laramee. Bob's Concise Coding Conventions (C^3). *Advances in Computer Science and Engineering (ACSE)*, 2009. (forthcoming, available online).

[10] Robert S. Laramee. Using visualization to debug visualization software. *IEEE Computer Graphics and Applications*, 30(6):67–73, 2010.

[11] Daniel Keim Matthew Ward, Georges Grinstein. *Interactive Data Visualization, Foundations, Techniques, And Applications.* A K Peters/CRC Press, 2010.

[12] Darrel Ince Roger S Pressman. *Software Engineering: A Practitioner's Approach - Fith Edition.* McGraw-Hill Higher Education, 2001.

[13] N. Liebsch Rory P. Wilson, E. L. C. Shepard. *Prying into the intimate details of animal lives: use of a daily diary on animals.* Endang Species Res, 2007.

[14] Herbert Schildt. *C++ A Beginner's Guide - Second Edition.* Osborne, 2004.

[15] Eric Sorensen. *Conservation Magazine.* Conservation Magazine, January-March 2007 (Vol. 8, No. 1).

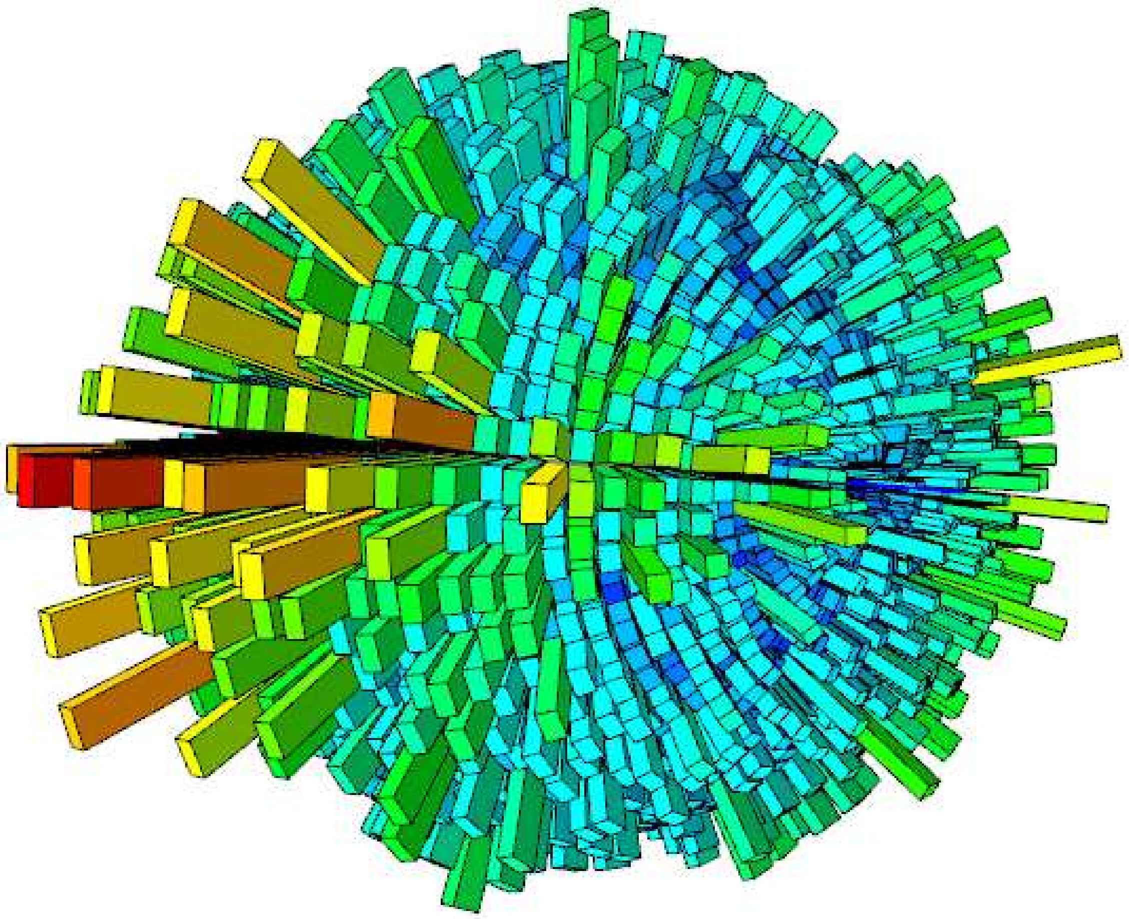
[16] The Qwt Team. Qwt. `http://qwt.sourceforge.net/` [Last Access: October 2010], 2010.

[17] Alexandru C. Telea. *Data Visualization.* CRC Press, 2007.

[18] Trolltech. Qt 4. `http://qt.nokia.com/products/qt` [Last Access: October 2010].

[19] Liebsch N Wilson RP, Shepard ELC. Prying into the intimate details of animal lives: use of a daily diary on animals. *Endang Species Res*, (4):123–137, 2008.

Animal Tracking Data Visualisation Tool



User Guide

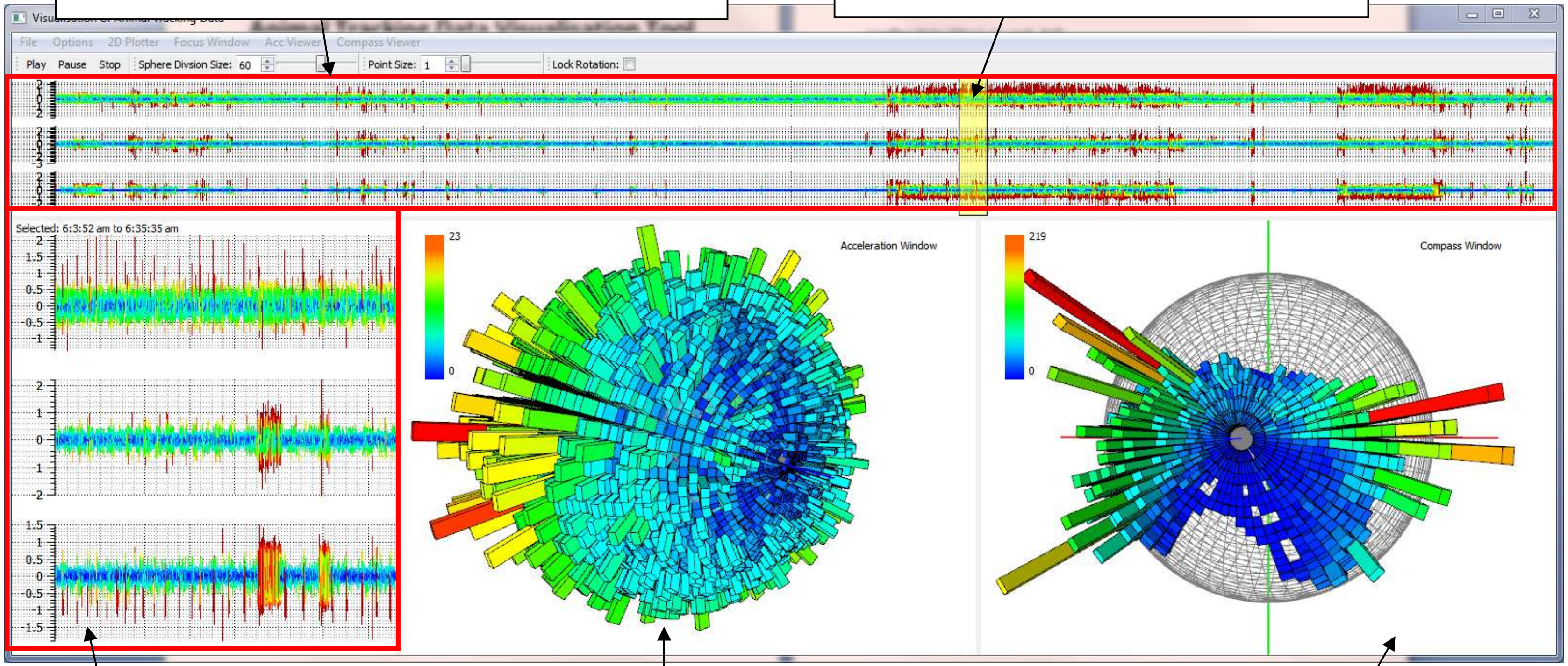
<u>Contents</u>

1.0.Overview of Application

The screen capture below shows an overview of the applications user interface. Areas of interest are labelled and described in detail.

Context Window – Shows an overview of the whole data set. By default, this plot displays three axes of acceleration data. More plots can be added (See section‘5.0 Adding and Removing 2D plots’).

Data Selection – Sub-sets of data can be selected from the context window (See section ‘4.0. Selecting Sub-sets of Data’). This data is plotted in the Focus Window.



Sphere Divsion Size: 60
Point Size: 1
Lock Rotation: ☐

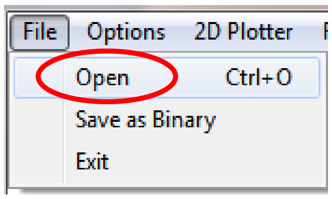
Focus Window – Shows the sub-set of data selected. By default this plot displays three axes of acceleration data. More plots of data attributes can be added (See section ‘5.0 Adding and Removing 2D plots’).

Acceleration Visualisation Window – window used for visualising acceleration data. See section ‘12.0. Adding Visualisations’ for adding visualisations.

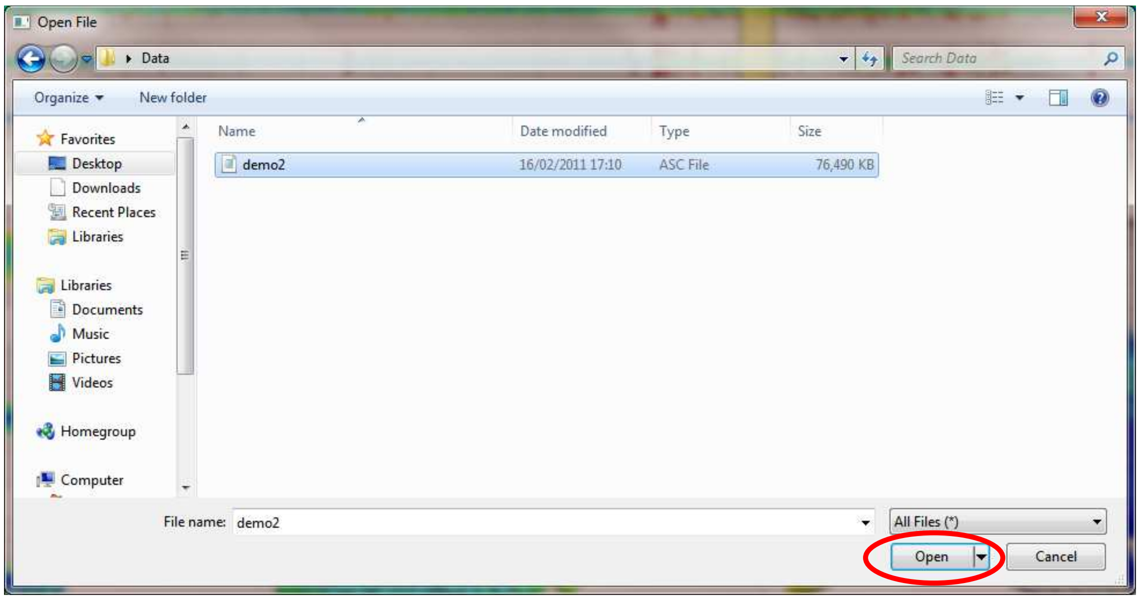
Compass Visualisation Window – window used for visualising compass data. See section ‘12.0. Adding Visualisations’ for adding visualisations.

2.0. Loading Data Files (.asc and .dat)

Two file formats are accepted by the application: The default ASCI files output by the Daily Diary devices in '.asc' format and binary files in '.dat' format. Files can be loaded into the application by locating to the 'File' menu at the top of the application and then selection the option 'Open'. Alternatively Ctrl-O can be used as a shortcut.



The 'Open File' dialog should now be displayed. From here, navigate to the data file, select it and then press open.



Name	Date modified	Type	Size
demo2	16/02/2011 17:10	ASC File	76,490 KB

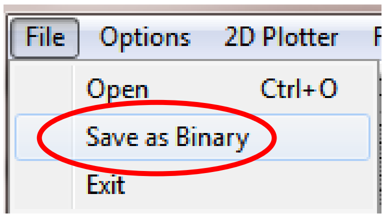
File name: demo2

The data file should immediately start to be loaded into the application, it is normal for this process to take a few minutes. Once loaded, the data will be plotted to the context window and is now ready for use in the application.

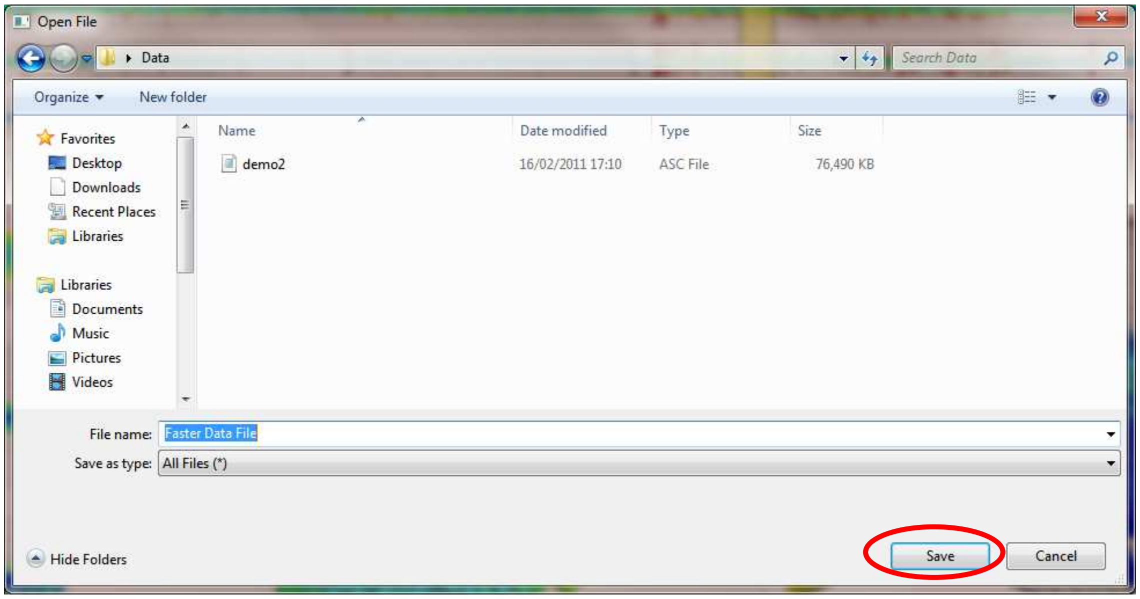
3.0. Converting Data Files to Binary Format

Using the raw file format output by the Daily Diary devices is slow and takes a long time to read into the application. A faster file format has been developed for the data files. Existing files can be converted to this format for future use.

To convert files to the new format is simple. Once a raw data file has been loaded into the application (See '2.0. Loading Data Files') it can be saved in the new file format. Navigate to 'File' and then select 'Save as Binary' to convert such a file.



A dialog box will soon appear requesting where you want to save the file on your computer. Navigate to the correct directory, give the file an appropriate name and then press save.



Name	Date modified	Type	Size
demo2	16/02/2011 17:10	ASC File	76,490 KB

File name: Faster Data File

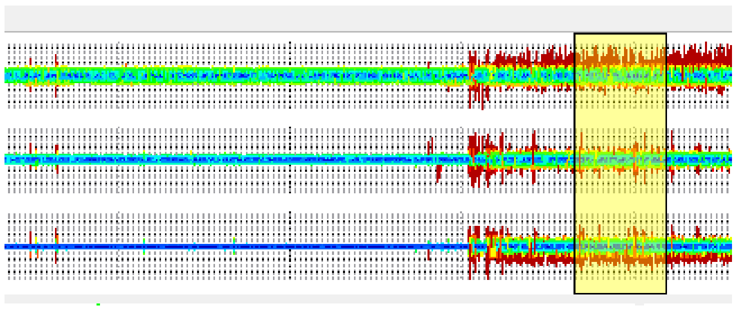
Save as type: All Files (*)

The file will then be converted and can now be used to load data files from now on.

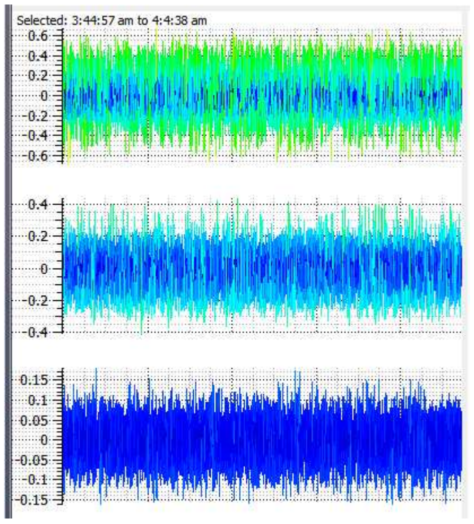
4.0. Selecting Sub-sets of Data

From the context window, which displays the whole data set in 2D time intensity plots, sub-sets of data can be selected. Selected data is then utilised in the focus window (think of this as a zoomed in time intensity plot of the selected data) and visualisation windows.

A sub-set of data can be selected by simply dragging the mouse across an area on the context plot area. A yellow transparent region will be selected to give you guidance over a selection made.

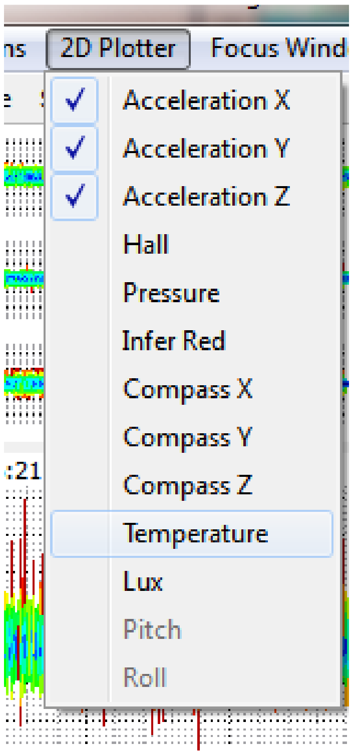


Once selected, the data is then instantly plotted in the focus window and visualisation windows. The data selected in the context window above is displayed in the focus window below:



5.0 Adding and Removing 2D plots

The default behaviour of the Context and Focus windows is to display acceleration data. Additional attributes of data can be plotted and added or removed using the ‘2D Plotter Menu’ (shown below). Here, a selection of data attributes which are logged by the data file being utilised can be added to both the windows.

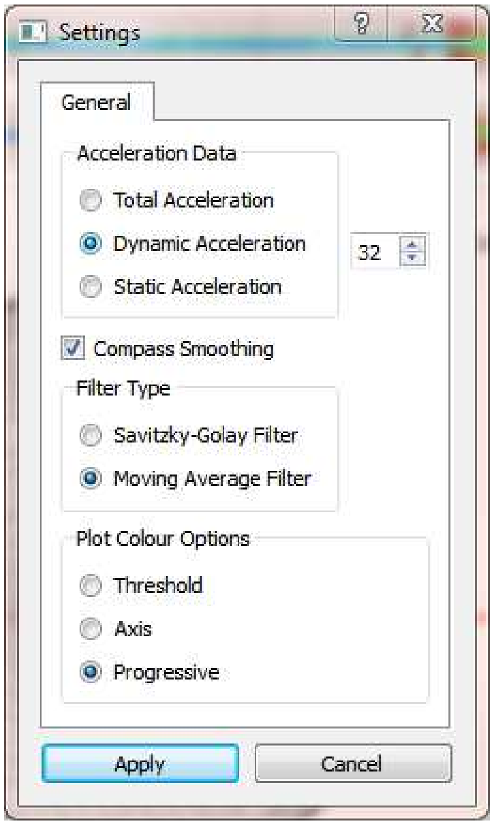


After selecting a data attribute to add to the focus and context windows or remove the plots are updated with this new information and re-plotted accordingly.

6.0. Changing Acceleration Attribute Being Visualised

The default behaviour of the application is to use the derived dynamic component of acceleration for visualisations which are use acceleration data. This can be changed to static or total acceleration by the setting dialog window.

To access the settings dialog window, navigate to the ‘Options’ menu at the top toolbar of the application and then select ‘Settings’. The settings dialog box should then appear, similar to that shown in the image below.



Settings

General

Acceleration Data

Total Acceleration

[✓ Dynamic Acceleration]

Static Acceleration

32

[✓ Compass Smoothing]

Filter Type

Savitzky-Golay Filter

[✓ Moving Average Filter]

Plot Colour Options

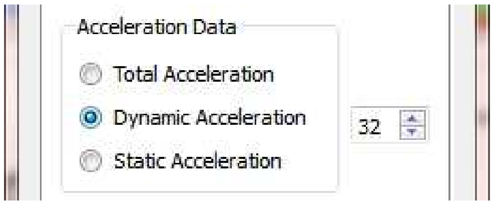
Threshold

Axis

[✓ Progressive]

Apply Cancel

A vast amount of settings can be changed using this dialog box, see section '8.0. Settings Dialog' for each of the settings individually explained. For this task, we are interested in the 'Acceleration Type' option, shown below:



Acceleration Data

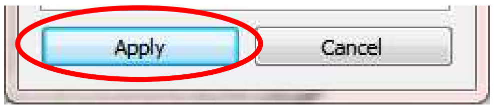
Total Acceleration

[✓ Dynamic Acceleration]

Static Acceleration

32

Use the combo box to select the acceleration to be visualised throughout the application. A choice of: total, dynamic or static acceleration should be chosen. Once the required component is selected, press apply.

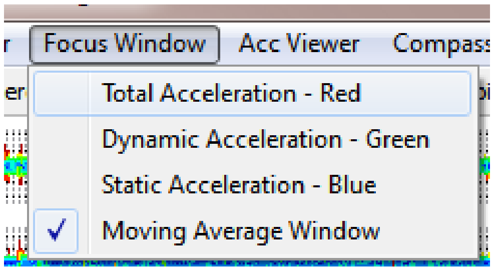


The visualisations and context + focus windows will now be re-plotted and will visualise the data attribute selected.

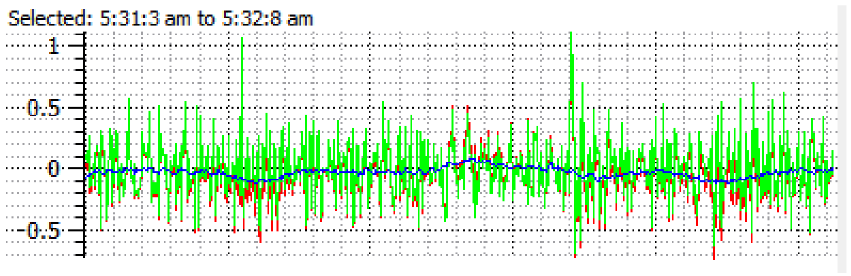
7.0. Adding Shared Space Acceleration Plots

Acceleration plots by default, show the acceleration attribute selected in the settings dialog window. Acceleration attributes can be directly compared using the shared space plotting technique integrated in the focus window of the application.

An overlay of total, dynamic or static acceleration can be added to acceleration plots. In order to add such an attribute, navigate to the 'Focus Window' toolbar (shown below). From this window the attributes can be selected.

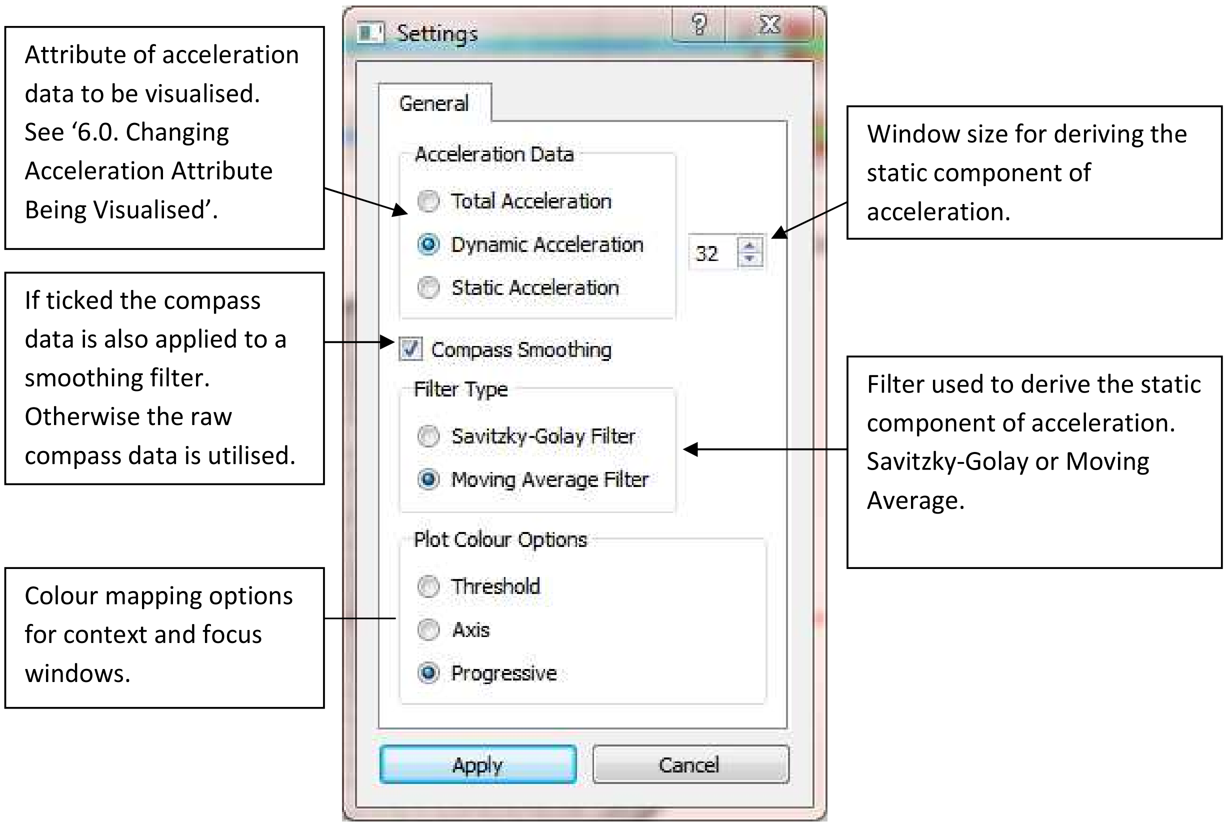


Each plot is given a unique colour. Total Acceleration – Red, Dynamic Acceleration – Green and Static Acceleration – Blue. The Image below shows a plot with all the components of acceleration added in a shared space layout.



8.0. Settings Dialog

The Settings dialog is used to change global settings in the application. Shown in the image below, each of the options available is described.



Attribute of acceleration data to be visualised. See '6.0. Changing Acceleration Attribute Being Visualised'.

If ticked the compass data is also applied to a smoothing filter. Otherwise the raw compass data is utilised.

Colour mapping options for context and focus windows.

Window size for deriving the static component of acceleration.

Filter used to derive the static component of acceleration. Savitzky-Golay or Moving Average.

Settings

General

Acceleration Data

- ○ Total Acceleration
- ● Dynamic Acceleration
- ○ Static Acceleration

32

[✓] Compass Smoothing

Filter Type

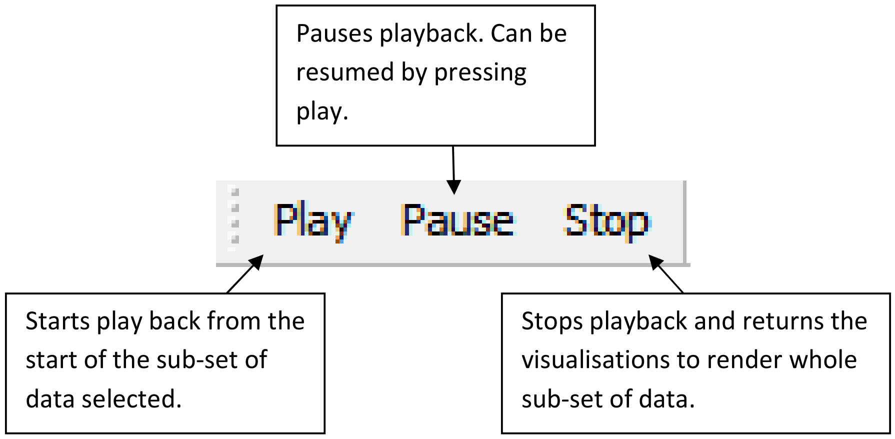
- ○ Savitzky-Golay Filter
- ● Moving Average Filter

Plot Colour Options

- ○ Threshold
- ○ Axis
- ● Progressive

9.0. Playing Back the Data Set

All visualisations allow for data to be played back in real time. This option is accessible by using the playback menu, located in the top toolbar of the application. This menu is explained in the image below.

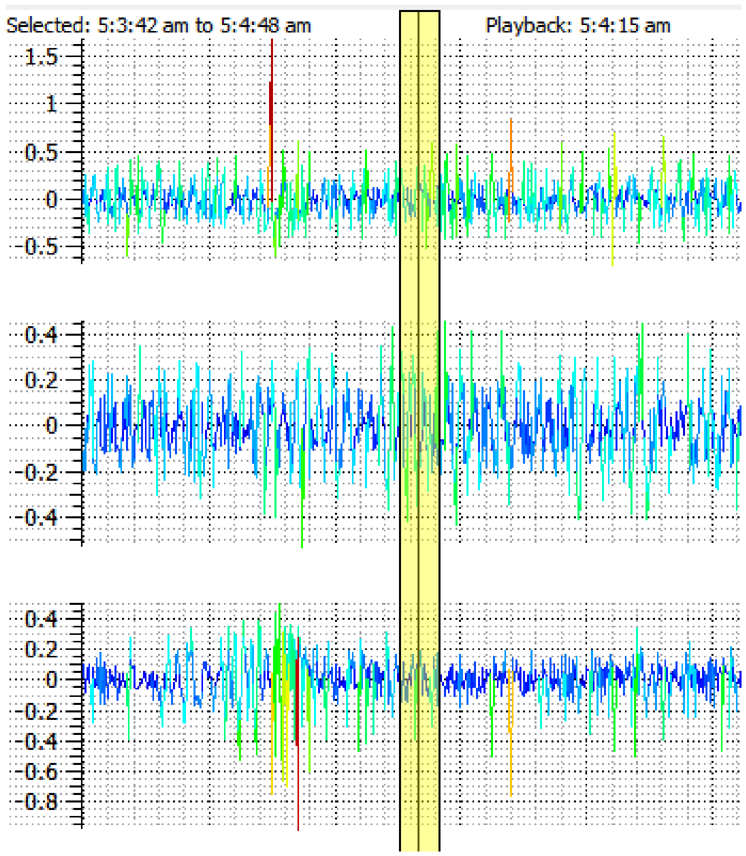


Pauses playback. Can be resumed by pressing play.

Starts play back from the start of the sub-set of data selected.

Stops playback and returns the visualisations to render whole sub-set of data.

When playback is enabled or is paused, the current playback position is displayed on the focus window. The yellow area outlines the window used for deriving the static component of acceleration. In addition to this the current playback time is also shown.

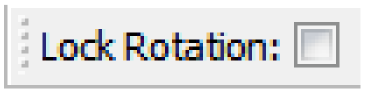


10.0. Interaction with Visualisation Window

The compass and acceleration visualisation windows can be interacted with by rotation and zooming. Windows can be zoomed in or out by using the scroll wheel on a mouse. When the mouse is hovering over a visualisation window, scrolling the mouse wheel upwards zooms in and scrolling the mouse wheel down zooms out.

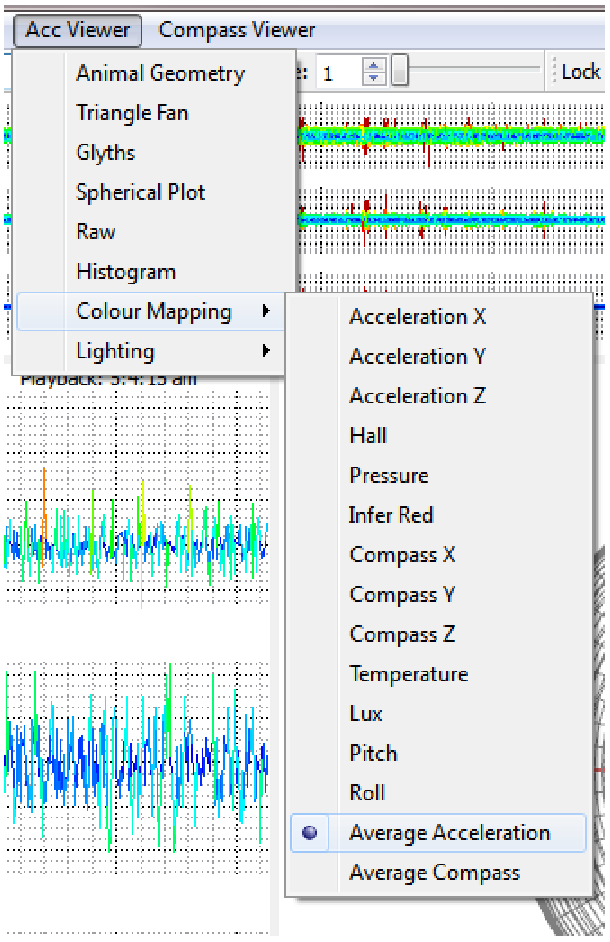
Rotation can be either synchronised or individual between the visualisation windows. By holding down the left button of the mouse over a visualisation window and moving the mouse the visualisations inside of the window are rotated.

Click the 'Lock Rotation' option (shown below) to synchronise the visualisation windows while rotating.



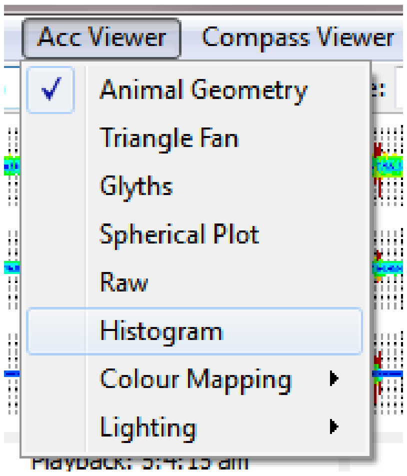
11.0. Colour Mapping Options

The Spherical Scatter Plot and Raw Scatter Plot visualisations by default map the colour of points plotted to the data attribute being plotted, either acceleration or compass data. This can however be changed to any attribute of data logged in the data file imported. To change the colour map, navigate to the appropriate visualisation window menu, either: 'Acc Viewer' – Acceleration visualisation window or 'Compass Viewer' – Compass visualisation window. From here, navigate to the 'Colour Mapping' sub-menu (Image below) and then select the appropriate colour mapping option.



12.0. Adding Visualisations

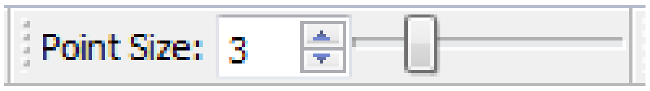
Visualisations are treated like layers, multiple visualisations can be added at any one time. Each window has its own menu to control the visualisations being shown. To add a visualisation, navigate to the appropriate visualisation window menu, either: 'Acc Viewer' – Acceleration visualisation window or 'Compass Viewer' – Compass visualisation window. From here, visualisations can be applied by selecting the appropriate menu items. A tick is placed next to visualisations being displayed in the window.



13.0. Raw Scatter Plot Visualisation

A Raw Scatter Plot visualisation can be applied using the appropriate visualisation window menu: compass or acceleration, see section '12.0. Adding Visualisations'.

An option exists to change the point size of points plotted in the visualisation (See image below). The text field accepts values in the range from 1 to 10. A value of 1 renders small points, where as a value of 10 produces large points.

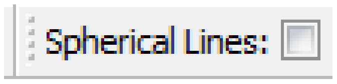


14.0. Spherical Plot Visualisation

A Spherical Plot visualisation can be applied using the appropriate visualisation window menu: compass or acceleration, see section '12.0. Adding Visualisations'.

Similar to that of the Raw Scatter Plot Visualisation discussed in section 12.0, the point's size of plotted items can be changed using the same menu.

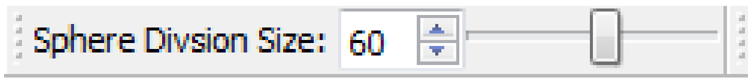
The default behaviour for items plotted is to plot them as points. Alternatively point's can be plotted with lines to show the connectivity between data items. To do this navigate to the top menu and find then Spherical Plot toolbar (Imaged below), select 'Spherical Lines' to turn on or off this feature.



15.0. Spherical Histogram

A Spherical Histogram visualisation can be applied using the appropriate visualisation window menu: compass or acceleration, see section ‘12.0. Adding Visualisations’.

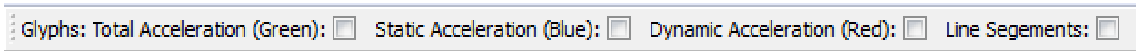
The number of Latitudinal and Longitudinal components of the central sphere can be defined, using the Spherical Histogram toolbar (Image below). This also changes the bin sizes in relation to the sphere. To change this, type in the number of latitude and longitude components the sphere is required to be made up of. In the example below, a value of 60 is chosen, this means the sphere will be made up of 3600 segments. A large value will produce small bins, where as a small value will produce large bins.


Sphere Divsion Size: 60

16.0. Glyphs

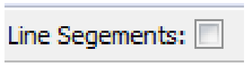
A Glyph visualisation can be applied using the appropriate visualisation window menu: compass or acceleration, see section ‘12.0. Adding Visualisations’.

Acceleration data consists of Total, Dynamic and Static components. Each of these components can be used for the glyph visualisation. Adding an acceleration glyph consists of the normal method of selecting the glyph visualisation from the acceleration window. After this a new toolbar will appear (See Image). From here, the acceleration components to visualise can be selected. Each glyph is uniquely colour coded: Total acceleration – Green, Static acceleration – Blue and Dynamic acceleration – Red.


Glyphs: Total Acceleration (Green): ☐ Static Acceleration (Blue): ☐ Dynamic Acceleration (Red): ☐ Line Segements: ☐

Compass data only consists of one component, the raw value logged by the Daily Diary devices. As such, the normal method of selecting ‘glyph’ from the compass window will make this visualisation appear.

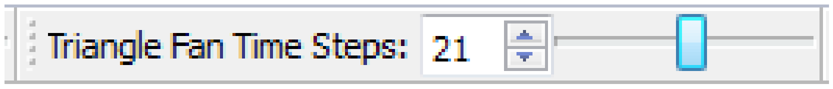
A user option is also to plot each of the individual line segments of the glyphs. This option can be enabled / disabled using the ‘Line Segments’ menu (Image below).


Line Segements: ☐

17.0 Triangle Fan

The Triangle Fan visualisation only works on acceleration data and is used to visualise future or previous time steps. A Triangle Fan visualisation can be applied using the acceleration menu, see section ‘12.0. Adding Visualisations’.

The number of perceived time steps to visualise in the future or past can be set by using the Triangle Fan menu (image below). Here, the number of time steps to visualise should be entered. A minus value is used to visualise previous time steps and a positive number for future time steps. In the example shown below a value of 21 is chosen. This means 21 time steps into the future will be visualised by the Triangle Fan.


Triangle Fan Time Steps: 21

18.0. Animal Geometry

The Animal Geometry visualisation shows the current orientation of an animal during play back. An Animal Geometry visualisation can be applied using the acceleration menu, see section ‘12.0. Adding Visualisations’. This visualisation is viewed in the Acceleration window.