Automatic Stream Ribbon Seeding

Dylan Geraint Rees

849119

September 2016

Abstract

Stream ribbons are ribbon representations that run at a tangent to the velocity at any point in a vector field. Twisting of the ribbon highlights helicity in a vector field. The purpose of this project is to create a novel automatic stream ribbon seeding algorithm for static vector field flows. In addition to this, a software application capable of implementing the algorithm and to visualize the results is created using C++, Qt and OpenGL technologies. The algorithm proposed prioritises seeding points in order of greatest helicity magnitude. Stream ribbons must be a minimum distance from other ribbons and must meet a minimum length criteria set as a user option.

Project Dissertation submitted to Swansea University
in Partial Fulfilment for the Degree of Master of Science

Prifysgol Abertawe
Swansea University

Department of Computer Science
Swansea University

Declaration

This work has not previously been accepted in substance for any degree and is not being currently submitted for any degree.

September 29, 2016

Signed:

Statement 1

This dissertation is being submitted in partial fulfilment of the requirements for the degree of a MSc in Computer Science.

September 29, 2016

Signed:

Statement 2

This dissertation is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by giving explicit references. A bibliography is appended.

September 29, 2016

Signed:

Statement 3

I hereby give consent for my dissertation to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

September 29, 2016

Signed:

Contents

1 Introduction

The way in which objects interact with fluids is becoming increasingly important as new technological barriers are being broken and efficiencies are becoming more important. For example aerodynamic efficiencies are extremely important in the automotive industry with a call to reduce fuel consumption in road vehicles and the requirement for improved performance in motorsport. The flow of liquids, gasses and fuel through an internal combustion engine are also relevant to the efficiency of the engine. Similarly in the aeronautical industry, the way an aircraft interacts with the air is fundamental to the amount of lift produced and the responsiveness which the pilot feels to their inputs. Many industries rely on knowledge of how objects interact with fluids; marine, energy, space, meteorology and even the medical industry where the flow of fluids around a body is yet to be fully understood.

To understand the fluid interaction, physical prototypes can be created and effects of the fluid interaction can be monitored or measured. However, the measuring and monitoring of fluids in itself is not a trivial matter. To measure the velocity of a fluid flow, a probe would have to be used which itself will interfere with the flow. Similarly using smoke in air or dye in a liquid to visualize the flow changes some of the properties of the fluid itself. The creation of prototypes can also be extremely expensive when the need to test many design iterations requiring many prototypes being produced which can be extremely expensive especially with more complex objects such as an internal combustion engine or a gas turbine.

The most cost effective way to understand fluid interaction is to use computational simulation. Virtual models of objects can be digitally constructed using solid modelling techniques and placed in virtual fluid flows. Computer fluid dynamics (CFD) is the simulation of the flow of fluids, and the way they interact with objects by using numerical analysis. CFD produces large amount of data which can be difficult to interpret. This data represents fluid features for all points of a mesh covering the area analysed. The mesh used can be unstructured and have different resolutions while the data contains many different fluid properties such as vector information, density, pressure and others. In order for engineers and researchers to gain the best insights of the data produced from CFD simulations the visualizing of the data is vitally important. Computer aided visualization is a field of computer science that is involved with the study and creation of visual representations of data. Within this field there are many subtopics, categorized by the type of data to be visualized. Flow visualization is the subtopic that visualizes vector data such as the data produced from CFD. Flow visualization is a scientific visualization, the geometry of the data is known and the data represent values in a physical or temporal space.

Many CFD packages have their own visualization packages however these are not always able to provide the user with the adequate insight into the data generated. Often the most useful insights in flow visualization come around unstructured flows such as vortices. These are created from interaction with objects or when two differing flows collide. Vortices are the cause of aerodynamic inefficiency with energy being lost in generating the vortex. There are many visualization techniques that can show vortices, the most effective being stream objects. Streamlines are the simplest and perhaps the most popular stream object. Streamlines are lines drawn at a tangent to the direction of a flow from a seeding point in a time independent flow. They visualize the theoretical path of a massless particle through the flow. Streamlines are calculated by integrating the vector field over a time step to form a series of points which are connected to form the streamline. Figure 1 shows an example of streamlines as they pass a marine turbine.

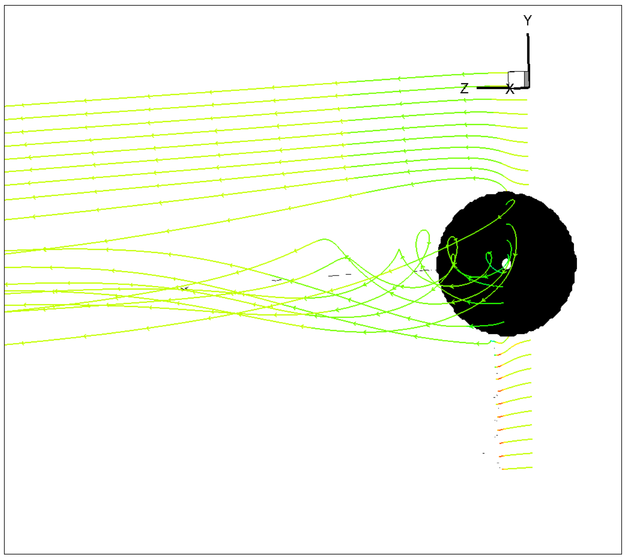


Figure 1: Manually placed streamlines flowing past a marine turbine (black disk). Image created by author using Tecplot, a commercial CFD visualization product.

An expansion on streamlines in three dimensional vector fields are stream ribbons. Stream ribbons are infinitely thin surfaces or ribbons attached to the streamline. Figure 2 show an image of stream ribbons as they flow past a marine turbine, these can be contrast with the streamlines visualized in figure 1. Stream ribbons allow for the visualization of the twisting of streamlines along their length, highlighting the helicity of a flow. Helicity is the measure of the knottedness and how tangled a vector flow is. This can be thought of as the

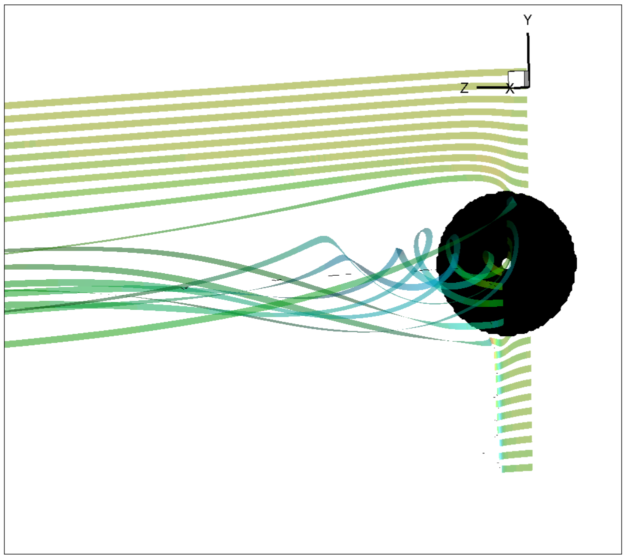


Figure 2: Manually placed stream ribbons flowing past a marine turbine (black disk). Image created by author using Tecplot, a commercial CFD visualization product.

tendency of a streamline to rotate or the amount of corkscrew motion along its length. Figure 3 shows an image of a delta wing in a water tunnel with dye injected near its apex. The resulting dye stream shows a good example of the helicity of the flow.

To achieve the best visualizations and therefore to gain the most knowledge from a flow visualization, the placement of stream objects is crucial. This is highlighted in figures 1 and 2 where the streamlines and stream objects offer little insight into the flow other than to highlight its laminar nature. The stream objects displayed from the centre of the representation of the turbine blade however offer valuable insights into the flow and its turbulent nature. An experienced engineer or researcher may know where to place stream objects to best represent flow features, however certain aspects may be missed. Having too many stream objects is also a problem, with occlusion becoming an issue. An example of occlusion is when stream objects in the front of an image or aspect block the viewer from seeing details behind. Automatic streamline algorithms exist to locate streamlines in order to achieve the most insightful visualizations and to minimizing occlusion, however no algorithm exists for the seeding of stream ribbons. The aim of this project is therefore to develop a novel stream ribbon seeding algorithm and a software tool to display them. In achieving this aim an important objective would be to learn to use a specific tool set for creating the visualizations.

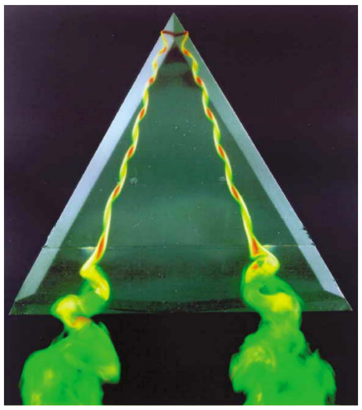


Figure 3: A delta wing in a water tunnel with dye injected near its apex [Tsi].

The rest of this paper is structured as set out by Laramee [Lar10]. Sections 1 through 4 are modified versions of work previously presented by the author in [Ree16].The following section highlights some background research performed; Section 3 details the specification of the project and the features of the software to be produced, along with a discussion of the tools to be used. Section 4 details the approach to the project including timings and risks to the project. An overview of the project design is presented in section 5 and the implementation applied in section 6. Section 7 highlights testing and evaluation of the project. A conclusion is drawn in section 8 and details of further work presented in section 9.

2 Background Research

2.1 Literature Review

This section discusses principles and techniques currently used to visualize data similar to the data that will be used in this project.

Telea discusses many techniques currently used in the visualization of vector field data in his book Data Visualization Practices and principles [Tel08]. The book introduces the concept of divergence and vorticity. Divergence is a scalar quantity and indicates if the flow is towards or away from a point. A positive divergence is flow moving away from a point while a negative divergence indicates flow towards a point. Positive divergence points are called sources while negative points are known as sinks. Meanwhile vorticity is a measure of local rotation within a vector field and is a vector quantity. Both of these concepts are important in the description of the flow within a vector field.

Vector glyphs are most commonly used to visualize a vector field. These glyphs are usually in the shape of arrows that represent the vector field, with a glyph for each sample point. There is a trade off, however, of sampling density against the number of attributes that can be displayed. Often arrows are used to display vector direction and magnitude while colour is mapped to another attribute such as temperature or pressure. CFD derived data, such as the data for this particular project, is often difficult to visualize due to the high number of attributes at each dataset point. Telea points out that glyphs with 10 degrees of freedom have been designed and used in the visualization of fluid flows albeit with limited effectiveness. A particular problem that Telea notes with 3D visualizations is occlusion, where glyphs that are in the fore obscure glyphs behind. This prevents the viewer from gaining insights to the flow behaviour beyond the foreground. This can be mitigated against by using transparent glyphs and by subsampling the dataset, although the latter leads to loss of local detail. Another technique Telea discusses, vector colour coding, is a continuous vector display method for surfaces. The colour hue and saturation give the direction and magnitude of the velocity across the surface. Because this is a surface technique, its application is limited within a 3D flow environment to isosurfaces.

Displacement plots are another visualization technique that Telea highlights. These are usually slices within a 3D vector field that have been distorted according to the velocity direction and magnitude at the slice. Figure 4 shows an example of this. These surfaces can also be coloured to better define the distortion or to represent another attribute.

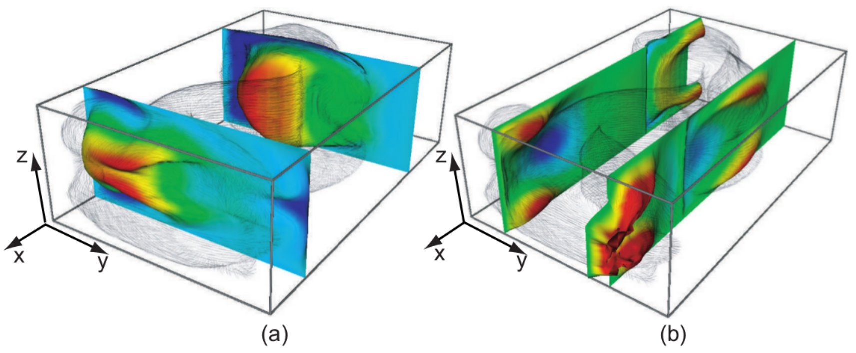


Figure 4: Displacement plot of a planar surface in a 3D vector field [Tel08].

Stream objects are perhaps the most intuitive and useful technique used in flow visualizations. They extend the idea of vector glyphs by creating a representation of the flow trajectory at a tangent to the flow. The simplest stream object is a streamline which is simply a line that indicates the flow in a static time field. This differs from streaklines and pathlines which are stream objects in time-dependent fields which are not covered in detail within the book. The book discusses streamline calculation methods, in particular the Euler integration and the Runge-Kutta method, and the trade-off between computation time and accuracy by varying the time step (Δ t). Location and the number of streamline seed points are another important consideration of streamlines; failure to address this could lead to a clutter of streamlines or an undersampled area of the vector field. Stream tubes are similar to streamlines but instead of lines use a sweeping circular cross-section, orthogonal to the streamline tangent to create a tube structure. An example of stream tubes can be seen in figure 5. Glyphs are often used to cap the tube ends and to show direction of flow.

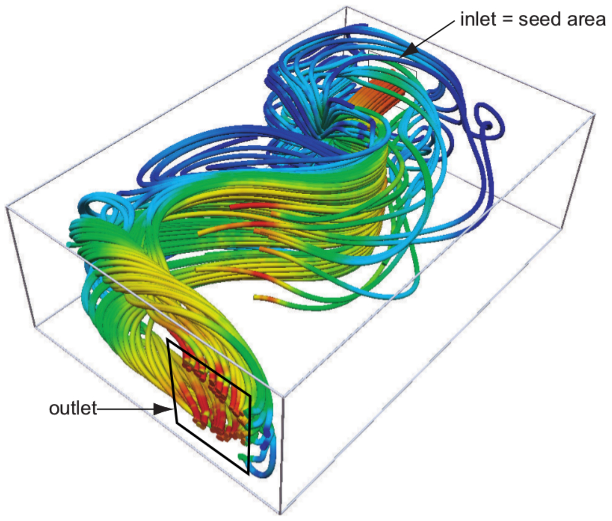


Figure 5: Stream tubes traced from a seed area at the flow inlet [Tel08].

Another stream object technique is stream ribbons. These are created by drawing a surface between two streamlines seeded close to one another. These are particularly useful to highlight high vorticity within a 3D flow field, showing as a twisting of the stream ribbon. Stream objects are often applied by visualization packages that produce flow visualizations, however the placement of the streamlines are vitally important to highlight areas of interest such as vortices. The placement of stream objects can be improved due to feature detection methods which can be done analytically or by using automated functions. However Telea points out that these functions do have some problems such as defining precise numeric criteria and features at different scales. Further examples of automated stream objects are discussed later in this paper.

Another technique that Telea introduces is texture-based vector visualization. This involves the blurring of a random black and white static image along streamlines, to create an effective continuous visualization of the flow field. Examples of texture based flow visualisations discussed are the line integral convolution (LIC) and the image based flow visualization (IBFV). LIC calculates a streamline forwards and backwards, for a predetermined length, from each pixel on the static noise image. A convolution of the grayscale colours along the streamline is then used to produce the final colour of that pixel. IBFV uses a static grayscale image as a basis overlaid with a grid structure which is then warped according to the vector field. Texture based visualizations can be animated to further enhance the visualization or to display time-dependent data.

Telea and Wijk further demonstrate the use of IBFV in 3D in their paper [TW03]. The results of their work is an animated 3D stream tubes that can be easily rendered on consumer hardware. Occlusion is however an issue with the 3D IBFV techniques as with the 3D glyphs

In order to prevent occlusion when using streamlines to represent a flow field and to help show flow direction Fuhrmann et al. introduces dashtubes in [FG98]. Dashtubes are described as animated, opacity-mapped streamlines which are used in 3D vector fields. These dashtubes are sparsely and evenly spaced throughout the volume to avoid any occlusion but to provide detail throughout the volume. The authors also present a magic lens that can be used to hover over areas of the volume to show more detail in that specific region. The magic lenses can then be expanded to form magic boxes for volumetric spaces.

McLoughlin et al. highlight and summarize some streamline placement algorithms in their survey paper Over two decades of integration-based, geometric flow visualization [McLL+09].

One of the earliest and most popular streamline seeding algorithm is presented by Jobard and Lefer in a paper titled 'Creating evenly-spaced streamlines of arbitrary density' [JL97]. Their algorithm applies to 2D steady flow fields and defines a minimum separation distance between streamlines. This was built on previous work carried out by Max et al. in [MCG94]. They proposed that lines should grow backwards and forwards from a seeding point until either the line exceed the boundary of the flow, the line had reached a predetermined length, a sink or source had been reached or the line came too close to another streamline. The algorithm was originally proposed for a 3D surface by Max et al. and extended for 2D cases by Jobard and Lefer.

Ye et al. developed a strategy presented in their paper strategy for seeding 3D streamlines [YKP05]. The strategy represents an expansion of another

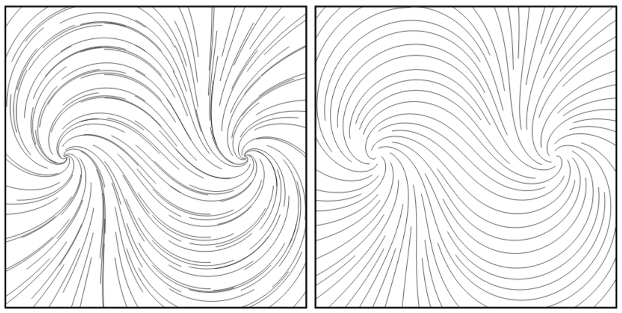


Figure 6: Long streamlines with seed points placed on a regular grid (left); same low field computed using the Jobard and Lefer eveny spaced streamlines algorithm [JL97].

2D strategy into 3D. Streamlines are seeded in areas where critical points are identified, ensuring all features are covered. Filtering then occurs to remove streamlines that are too similar and that are too straight to indicate any area of interest.

Chen et al. present an adaptive streamline placement algorithm [CCK07] that uses a similarity index to form a set of candidate seed points. The similarity index takes into account Euclidean distance between streamlines as well as streamline direction and shape. The authors claim that the method produces streamlines that naturally accentuate regions of geometric interest. The technique is relevant in both 2D and 3D vector fields but only for static data. The authors also present an error evaluation on how correct the streamline is in relation to the rest of the vector field.

Other stream objects that have had an automatic seeding algorithm applied to them are stream surfaces. Stream surfaces are surfaces drawn between two streamlines to create a surface. Edmunds et al. present an automatic stream surface seeding in their paper Automatic Stream Surface Seeding [EMcLL+11]. Surfaces are seeded at the domain boundary from isolines derived from a scalar field. The surfaces are then terminated when a predetermined length is reached, the surface becomes too close to another surface or when the surface becomes too close to the domain extremities.

Roth analyses many existing automatic feature detection algorithms to establish their underlying definition of a vortex in [Roth00]. He also proposes a new definition for vortex cores based on a second-order derivation. The paper also details the mathematical functions used to define key flow field parameters such as helicity.

$$Helicity = (\nabla \times v) \bullet v \tag{1}$$

Helicity is defined mathematically in equation 1 where v is the velocity field and $()\nabla \times v)$ is the vorticity field. The ∇ symbol denotes the gradient of the vector field.

An important consideration when presenting data for visualization is the colours that are used to represent the data. Telea [Tel08] indicates five goals for the use of an effective colourmap showing scalar values.

1. Absolute value - To be able to tell the absolute scalar value in a dataset
2. Value ordering - To be able to tell which value is higher from two given data points .
3. Value difference - To be able to identify the value difference between two given data points.
4. Selected values - To be able to identify data with the same scalar value.
5. Value change - To be able to see the rate of change in the scalar value.

Telea goes on to say that to meet all these goals is a difficult task. Telea discusses the use of the rainbow map, a colour map ranging from blue through green, yellow and orange to red similar to a rainbow. The limitations of the rainbow colourmap are discussed, such as that warm colours attract more attention then colder colours and that luminance values differ over the colour range. Despite it's limitations, Borland and Taylor highlight the prevalent use of the rainbow map as the default colourmap in many commercial visualization applications in [BT07].

Moreland also highlights the inadequacies of the rainbow colourmap in [Mor09] and comments "Know thy enemy" of it. Moreland continues to discuss the development of a diverging colourmap from blue to red through white. The colour is interpolated using magnitude saturation and hue and can be seen in figure 7. Because 3D surface visualizations use lighting and shading effects to give clues to the surface shape, the change in brightness of the colourmap should be avoided. The colourmap developed was set as the default choice in ParaView [Kit00] and received positive user feedback.

Another colourmap is proposed in a paper by Kindlmann et al. in [KRC02]. The paper focuses on user feedback of facial recognition with different colours applied. The colourmap produced is based on the rainbow colourmap with the luminance adjusted so that it monotonically increases making it more perceptual.

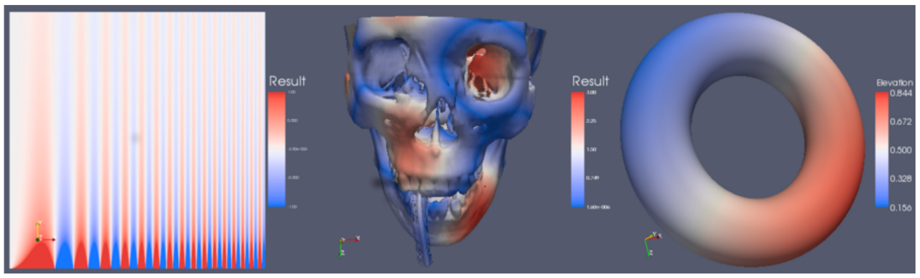


Figure 7: Three visualizations highlighting the use of Moreland's blue to red diverging MSH colour map [Para07].

Because of the novelty of Qt, OpenGL and C++ to the author, literature guides on the use of the languages are an important consideration. A C++ reference book by Stroustrup is to be used [Str15] throughput the project as guidance to the language. Sellers et al. introduce OpenGL in [SWH15]. Simple OpenGL principles and programs are introduced as well as guidance on 3D mathamatics and more in-depth topics.

Blanchette and Summerfield introduce the Qt application framework in [BS08]. The book details tutorials that will be useful while using the Qt framework. Tutorials range from the simple 'Hello World' program to advanced full feature GUIs. Chapters of particular interest for this project are the 2D graphics and the 3D graphics chapters, where an introduction to creating 2D and 3D graphics is supplied. Another chapter of interest is the container classes chapter. This details some container classes specific to Qt such as the QVector class, which allows the storing and direct access of other container classes, and the similar QMap class which allows storage and access to a container object through the use of a key.

2.2 Previous Systems

SimVis, developed by VRVis Research Center, is an application designed with the specific task of analysing large, complex flow data produced from CFD simulations. Doleisch, one of the developers, introduces the application in his paper [Dol07]. The application is designed specifically for analysis by a flow expert who is able to analyse the data extract the relevant information. The application is based on a feature-based flow visualization approach. It however introduces a novel approach by allowing the user, an expert, to interact with the data to find specific features rather than relying on automated feature detection. The application features multiple linked views which can show different aspects of the data or different representations of the same aspects. As with most applications that visualize CFD datasets, there is a scientific 3D layout of the view. SimVis allows this to be supplemented with other views such as 2D scatterplots, parallel coordinate plots or time-dependent histograms. A screenshot of the application can be seen in 8.

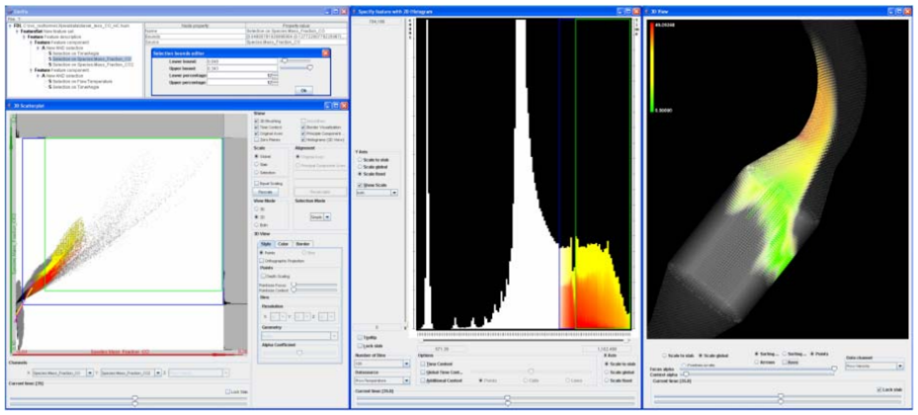


Figure 8: A screenshot of a SimVis scenario from [Dol07]. The image shows simulated flow through an automotive diesel particulate filter. The left frame shows the brushing of data from a scatter plot which is shown in the middle plane, a histogram. The data in the histogram is then brushed to give the final 3D scientific view with focus and context in the right frame.

Another important feature is the user interaction with the data. The user can select, or brush, specific data points of interest. This then updates other linked views with the brushed data emphasised in the context of the other data points, e.g. arrow glyphs representing the flow in a 3D spatial domain become transparent and grey, while brushed data is opaque and coloured. SimVis also incorporates a Fuzzy Classification which uses a gradient between the brushed data and non brushed data rather than a distinct contrast.

Doleisch presents some real-world use of the SimVis software, the majority of which are for automotive applications [LJH03][DMG+04], but not exclusively[DMH04].

Peng et al. of Swansea University present a generic framework for visualization of high-dimensional CFD flow simulation data in their paper [PGL12]. They implemented the visualization framework to developed a software tool designed to visualize data specifically for marine turbine applications. The software is presented by Peng et al. in [PGN+14] and can be seen in 9. Like the SimVis application, this application is an interactive, multiple coordinated view program. This application does however provide some novel views not seen in the SimVis application. These additional views are a histogram table showing all data attributes, a spherical histogram plot and a streamline plot.

The CFD simulation data is presented as an interactive histogram table providing a multi-dimensional overview of the data. The table consists of a stack

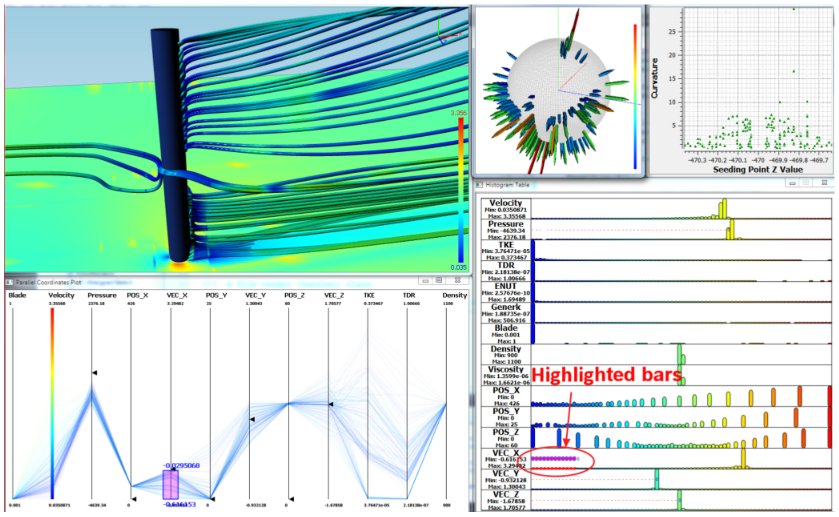
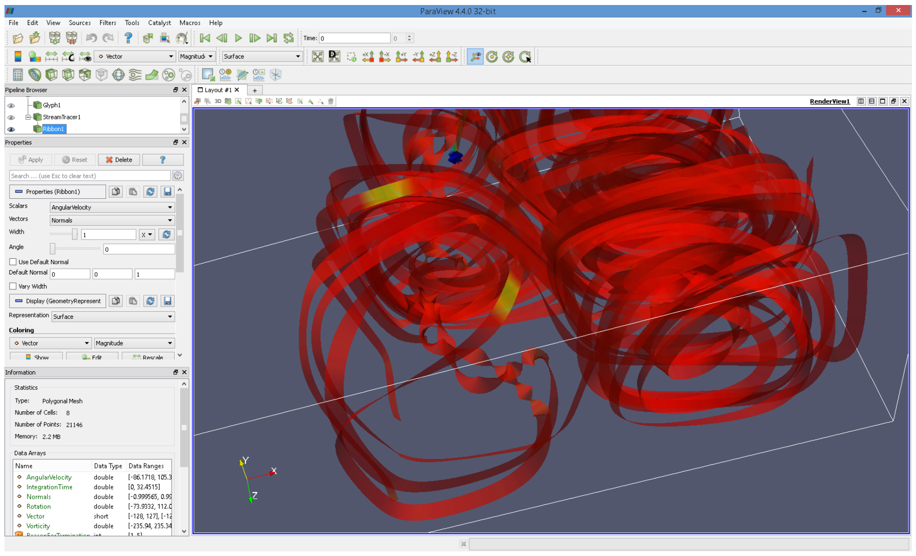


Figure 9: A screenshot of the application developed by Peng et al. of Swansea University taken from the paper [PGN+14]. The image shows multiple views of flow data past a marine turbine. Data highlighted in the histogram table (bottom right) is displayed in the parallel coordinate plot (bottom left) and spherical histogram. Data brushed in the parallel coordinates is then used to generate the streamlines in the spatial view (top left). A scatterplot shows the curvature along the lengths of the streamlines generated.

of histograms with each histogram showing a particular data attribute. Data can be brushed from the histogram table by the user to update linked views. A spherical histogram is available that displays an intuitive description of the velocity direction of the brushed data. The user is able to interact with this by rotating it and by customizing the bin resolution. A parallel coordinates plot allows for investigation of the relationships between the various data attributes. The parallel axis can be rearranged by the user to allow the comparison of particular attributes. Data brushed from the parallel coordinates plot generate streamlines in a rotatable and zoom-able spatial view. The streamlines displayed are represented in a scatter plot of calculated streamline curvature against their length. This allows for identification of streamlines with particular relevant swirl characteristics. Individually selected streamlines from the scatter plot are rendered in the spatial view for further exploration. The paper highlights some particular unexpected insights gained from the visualizations along with positive feedback from domain experts.

Another more developed and widely known visualization software is ParaView, developed by Kitware Inc. [Kit00]. ParaView is an open source 3D scientific visualization software and is capable of generating many types of visualizations including vector glyphs, iso-surfaces, streamlines and stream ribbons.

Figure 10 shows a screenshot of ParaView visualizing stream ribbons. It is a multiplatform Qt based system designed to run on multiple cluster machines, however it can also be run on a single computer. It allows full 3D interaction with the visualization on screen. Although there are no automatic feature detection methods built in with the software, users are able to perform calculations on the imported dataset.



Properties (Ribbon1)
Scalars: AngularVelocity
Vectors: Normals
Width: 1
Angle: 0
Use Default Normal
Default Normal: 0 0 1
Vary Width
Display (GeometryRepresent
Representation: Surface
Coloring: Vector, Magnitude

Information
Statistics
Type: Polygonal Mesh
Number of Cells: 8
Number of Points: 21146
Memory: 2.2 MB

Data Arrays

Name	Data Type	Data Ranges
AngularVelocity	double	[-86.1718, 105.3
IntegrationTime	double	[0, 32.4515]
Normals	double	[-0.999565, 0.99
Rotation	double	[-73.9332, 112.0
Vector	short	[-128, 127], [-12
Vorticity	double	[-235.94, 235.34
ReasonForTermination	int	[1, 5]

Figure 10: Screenshot of ParaView with zoomed image of ribbon in a RayleighBnard flow.

Tecplot 360 is another commercial visualization tool developed by Tecplot Inc. [Tec81]. It is designed as a post-processing tool for CFD derived data including producing visualizations. It is also capable of producing stream ribbon visualizations as can be seen in figures 1 and 2.

MATLAB from MathWorks [Mat84] is a powerful numerical computing environment. It has many inbuilt functions to calculate a variety of vector field properties such as vorticity, curl and divergence. It also has the ability to easily visualize basic stream ribbons in a 3D plot using simple commands. An example of this can be seen in figure 11. Lighting and shading effects are also included and the ability to interact with the rendering by rotating and zooming. This allows for quick and convenient analysis and visualization of vector fields.

2.3 Data Characteristics

Vector field data is required for both the development of an algorithm and for development of the software product. Three different data files will be used for

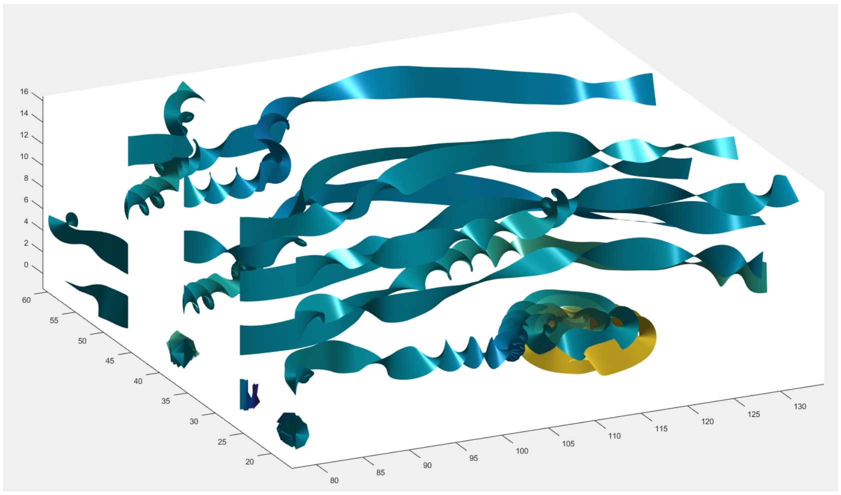


Figure 11: Visualization of stream ribbons in wind data produced using MATLAB.

the development and to testing of the product. All datasets are static velocity fields, i.e. they do not vary with time, and all datasets use a 3D Cartesian coordinate system. The first dataset is a tornado simulation given the name Sally. Sally is an analytical data set produced from a procedural function on a regular Cartesian grid. The size of the Sally dataset can be adjusted by specifying the dimensions of the dataset, in this case a 128^3 node model will be used. Figure 12 shows a representation of Sally using a single stream ribbon, produced in ParaView. The twisting of the ribbon highlights features that would not be seen with a simple streamline.

The second dataset to be used is a sampled vector field of a RayleighBnard convection flow. The dataset provided has a resolution of 128 x 32 x 64 and is provided in a binary .raw file format. Figure 13 shows a multiple ribbon visualization of the flow produced in ParaView.

The final dataset is a ArnoldBeltramiChildress (ABC) flow which represents an exact solution to Euler's equations in fluid dynamics. It is another flow field that is produced from a procedural function as seen in equation 2 .

$$v(x) = \begin{pmatrix} Asin(z) + Bcos(y) \\ Bsin(x) + Ccos(z) \\ Csin(y) + Acos(x) \end{pmatrix}, \qquad x \in [0, 2\pi]^3 \tag{2}$$

A, B and C represent constants that can be adjusted to create different flows. For this project the following parameters shall be used: A = 0.1, B = 0.5 and C = 0.5. The size of the flow field can also be adjusted as in the tornado

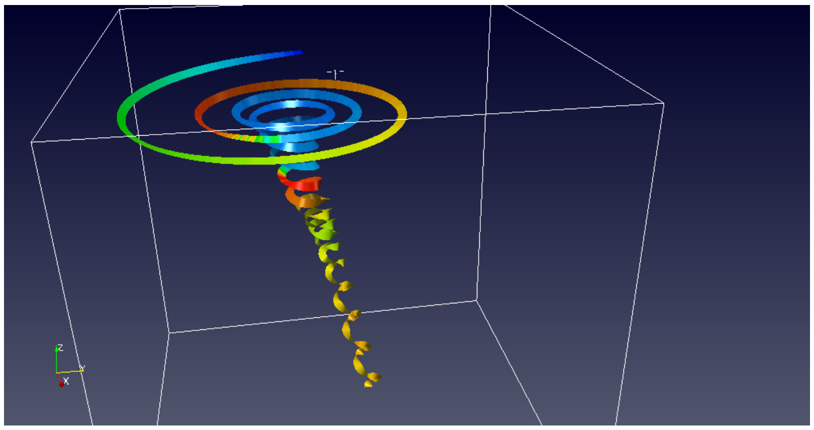


Figure 12: A 128^3 node tornado simulation visualized using a single stream ribbon, produced in ParaView by the author. The colour of the ribbon is mapped to the velocity magnitude.

dataset, for the purposes of this project a 128^3 will be used. Figure 14 shows a visualization of the dataset.

3 Project Specification

The primary aims of this project is to develop an automatic stream ribbon seeding algorithm and to develop a software tool capable of showing it. The software tool should be capable of:

- Reading a vector field data file
- Analysing the vector field to determine stream ribbon placement
- Visualizing the vector field using the stream ribbons

The most important aspect of the project is the personal development of the author. Therefore a key objective is to increase understanding of the tools and processes required to produce scientific data visualization. The project will be limited to steady vector field data using 3D Cartesian coordinates.

3.1 Software Feature Specification

The software will include the following features:

1. A graphical user interface
2. A 3D visualization pane
3. The ability to read data from binary files

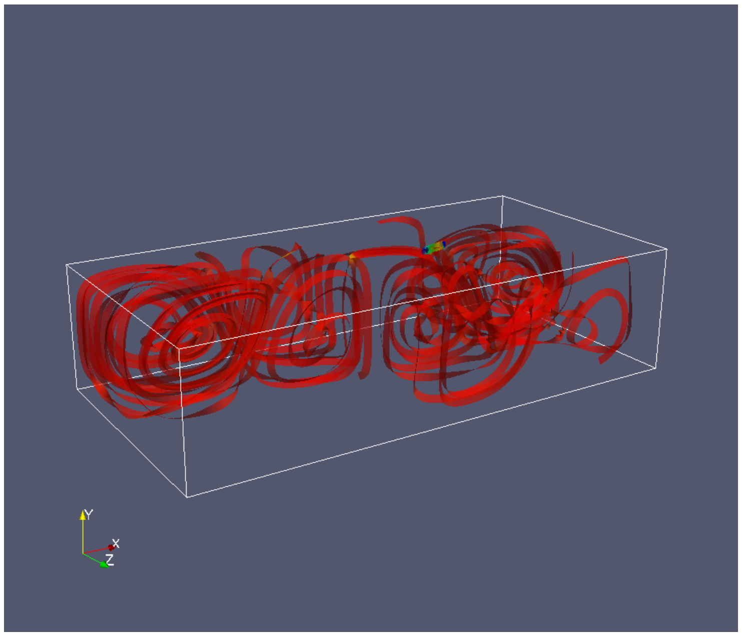


Figure 13: A multiple stream ribbon visualization of a RayleighBnard convection flow produced in ParaView by the author.

4. A mathematical function for calculating streamlines
5. An ability to visualize stream ribbons
6. A mathematical function for calculating stream ribbon seeding locations

Additional features to be implemented in the software are:

1. An interactive 3D visualization pane allowing rotation, translation and zooming
2. Lighting and shading effects in the visualization pane
3. Filtering options for stream ribbons displayed
4. The ability to represent data fields using arrow glyphs

3.2 Technology Choices

C++ using a Qt [NCE91] GUI has been decided upon as the languages to construct the software tool for this project. Both are commonly used in the visualization field and have the advantage of being multi platform, freely available and well documented. Blanchette and Summerfield describe Qt as a C++ application development framework for creating cross platform GUI applications

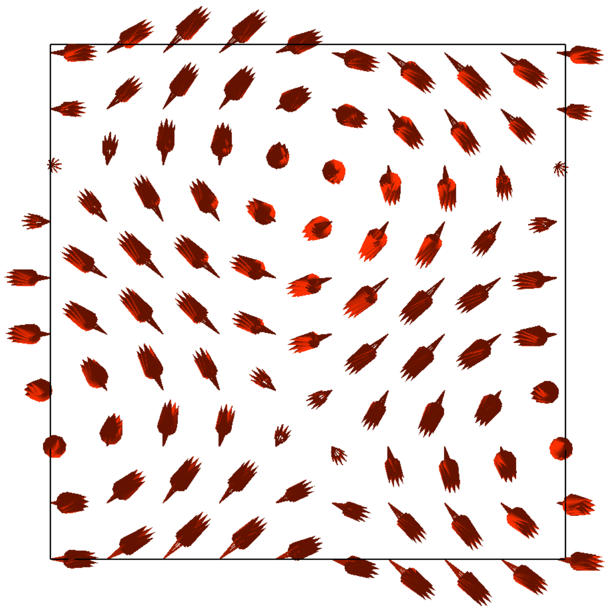
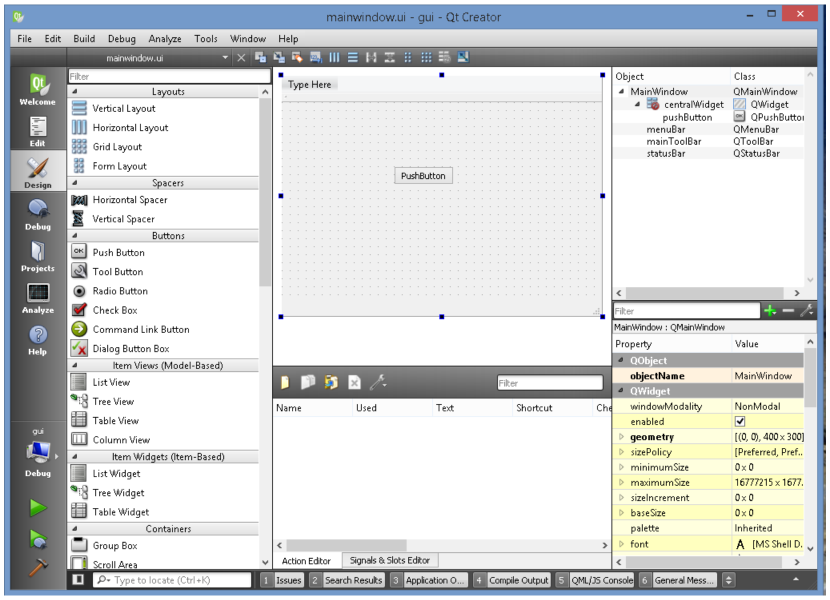


Figure 14: A hedgehog plot visualization of an ABC flow viewed from the x-y face.

[BS08]. The Java programming language was also considered. Both Java and C++ are popular languages, well documented, free to use and cross platform. Java also has the advantage of running on differing platforms without the need to recompile, unlike C++. However the use of C++ is more common in the visualization field and so it was decided to use C++ for the benefit of the personal development of the author.

Similarly other GUI libraries were considered, FLTK [Spi98], JUCE [Sto04] and in particular wxWidgets [Sma94]. All are cross platform and are free to use however Qt appears to have more support options with a number of forums as well as a mailing list. Furthermore Qt documentation is more readily available, with a number of publications in the university library dedicated to the technology and far fewer dedicated to other GUI technologies. Qt also offers its own interface development environment in Qt Creator, a screenshot of Qt creator can be seen in figure 15.

The use of Qt is also more prevalent in the visualization field so it would be more beneficial in the development of the author. To produce a 3D rendering of the flow domain, Qt can utilise OpenGL libraries using QtOpenGL module. OpenGL is a application programming interface for rendering graphics, utilising GPU hardware acceleration. It has powerful in built libraries that enable such effects as lighting and shading. These are important components in allowing



Object	Class
MainWindow	QMainWindow
centralWidget	QWidget
pushButton	QPushButton
menuBar	QMenuBar
mainToolBar	QToolBar
statusBar	QStatusBar

MainWindow : QMainWindow

Property	Value
QObject	
objectName	MainWindow
QWidget	
windowModality	NonModal
enabled	✓
geometry	[(0, 0), 400 x 300]
sizePolicy	[Preferred, Pref...
minimumSize	0 x 0
maximumSize	16777215 x 1677...
sizeIncrement	0 x 0
baseSize	0 x 0
palette	Inherited
font	A [MS Shell D...

Figure 15: Screenshot of Qt creator GUI creation interface.

the viewer to interpret a surface such as a ribbon and the way it twists and turns. An alternative to OpenGL would be DirectX however this is limited to Microsoft based platforms.

Some specific C++ libraries identified as being required for the software are iostream, stdio.h and fstream, to allow for data files to be read. The Math.h library will be particularly important when applying the seeding algorithm. An example of a use of the library are of the functions sqr and sqrt to square and root vector components in order to get a scalar value. Some refinement can be achieved using the stdlib.h library and the 'malloc' function to allocate memory dynamically for increased efficiency. The OpenGL utility library (GLUT) is an OpenGL library that contains some basic primitive shapes such as spheres and cones. FreeGLUT is an open source alternative that perform many of the same functions that can be used in this project.

Qt has many inbuilt libraries which can also be used such as 'QMouseEvent' which allows for easy interpretation of mouse based inputs. The Qt container class libraries will be of particular use. These include 'QVector3D', which is an object designed to contain a 3D vector, 'QVector' a class for containing other classes and 'QMultiMap' Other useful Qt libraries are 'QMessageBox', which displays pop-up message boxes that can be used to communicate with the user, and 'QFileDialog' which implements a dialog for finding and opening files within the operating system.

A GitHub repository will be used to track changes in the code as it is being developed and also used as backup in case of computer malfunction. The use of GitHub will also allow for development access across multiple computers.

Any modern consumer grade PC with a dedicated graphics processing unit should be able to run the product produced. To be able to visualize large sets of data however, a large amount of RAM memory may be required. The use of memory allocation techniques has been considered in the design of the software which will help reduce memory requirements. The software development will be performed on a PC with a minimum of 8GB of RAM to accommodate testing with larger datasets.

The software application will be developed and tested on a PC with 16GB of RAM, an Intel i5-6500 using the Intel B150 Express chipset. The PC's operating system will be Linux, specifically Ubuntu 16.04. Qt creator 4.0 IDE will be used for code development using Qt version 5.6.0, OpenGL version 3.3 and compiled using GCC 4.9.1.

4 Project Plan and Timetable

4.1 Development Approach

Due to the novel nature of this project and the authors inexperience with the technologies to be used, an agile software development cycle has been decided upon to manage this project. More specifically the adaptive software development (ASD) has been selected. This approach allows for continuous evolution of the software and promotes a learning phase as part of the cycle, which will be crucial to the development of the software along with the personal development of the author.

This differs from the traditional waterfall model of software development where more emphasis is put on the design phase. The waterfall methodology was adapted from a hardware oriented design principles, where the implementation phase would involve costs in the physical construction of the product, so the design once, implement once methodology was important. In software development however, the only cost involved in the implementation phase is time; therefore a try it and see methodology promoted by the adaptive software development is acceptable. The adaptive approach also matches the author's natural tendency for development. A graphical representation of the ASD cycle can be seen in figure 16.

The cycle starts with the speculate phase where initial ideas of how to implement a software feature are sought based on the requirements and constraints of the feature. The term speculate is used as the idea selected may not be a workable solution but allows for a try it and see approach. The collaborate

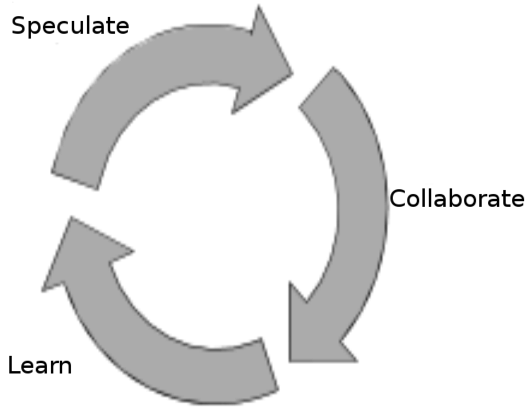


Figure 16: Adaptive software cycle framework.

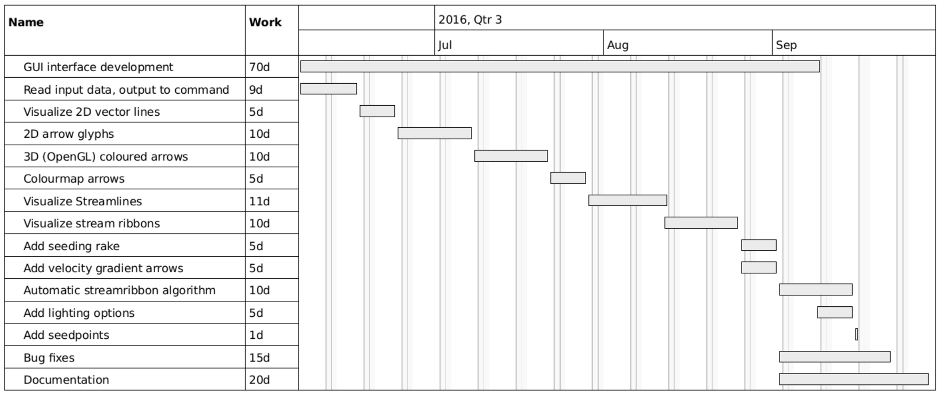
phase is essentially the implementation phase of the software feature. The final learning phase allows for assessment and testing of the solution implemented before either rejecting the solution and returning to the speculate phase, or moving to the next phase. The ASD promotes rapid iterations of this cycle in order to find a workable solution.

This implementation works on many levels of the software development, from trying individual functions within libraries, through implementing complete features within the software, to the complete program. There are however some disadvantages to this methodology. The software development could be in danger of being stuck in a continuous loop trying to implement one specific feature; therefore careful management of the time constraints are important and an emphasis must be made to try to stick to the timing plan, as presented in the next subsection.

4.2 Project Timing

This section discusses the projected timing for completion of various stages of the project. Figure 17 shows a Gantt chart summarizing the timings.

The project is due to commence in the second week of June and proceeds until the final deadline on the 30th of September. The project can be separated into three components, the development of an automatic stream ribbon seeding algorithm, the production of a software tool for visualizing the stream ribbons and the writing of a report detailing the experience. The development of the algorithm and the software will be done in parallel before integrating the algorithm into the software.



Name	Work
GUI interface development	70d
Read input data, output to command	9d
Visualize 2D vector lines	5d
2D arrow glyphs	10d
3D (OpenGL) coloured arrows	10d
Colourmap arrows	5d
Visualize Streamlines	11d
Visualize stream ribbons	10d
Add seeding rake	5d
Add velocity gradient arrows	5d
Automatic streamribbon algorithm	10d
Add lighting options	5d
Add seedpoints	1d
Bug fixes	15d
Documentation	20d

Figure 17: Gantt chart of project timings.

The development of the user interface is to be evolved throughout the project with modifications to be made to include new features as they are developed. Because the development will require learning how to use the C++, Qt and OpenGL resources, the tasks associated with completing the project are ordered in order of increasing complexity. The first task is to input or generate a dataset. Once a dataset is available within the program, it can be visualized. Easier 2D visualizations will be created first before more complex 3D OpenGL visualizations are created. Vector arrows shall be visualized first before more complex stream objects. Lighting options and aesthetic improvements will be implemented last.

Four weeks have been dedicated to the writing of the report at the end of the project. If however other sections of the project were to overrun, as a contingency, some final software development may be done in parallel with the commencing of the document. Bug fixes will also be performed as they are found.

4.3 Risk Analysis

The analysis of risks to a project before commencing is an important process so that the most likely and most devastating risks can be mitigated against. This section discusses risks associated with this particular project.

Arnuphaptrairong discusses common risks associated with software engineering as stated in different studies in his paper [Arn11]. He lists the different top ten risks stated in 12 different works from 1981 to 2003 to determine which are the most agreed upon. He concludes that the risks most frequently identified are:

- Misinterpreting project requirements

- Poor management support
- Poor end user input
- No end user commitment
- End user expectation too high
- Changing project requirements

Because of the primary aim of this project is to develop an automatic seeding algorithm for stream ribbons some of the above risks do not apply. The project requirements are set by the author working alone on the project, therefore misinterpreting the requirements is unlikely. The poor management support is interpreted as supervisor support in this instance and has been included. There is no specific end user in this instance however consultation with a domain expert would be required to specify what stream ribbon visualizations benefit them the most and to provide feedback on results produced. The main risks can be seen in table 1.

The first risk identified is that the combination of technologies identified for the development are not adequate for the production of the desired software product. The impact this would have on the project would be high as the goals would not be achievable. It has however been given a low probability due to the fact that these are mature technologies that have a proven record in producing software of a similar nature.

The second risk identified is due to the author of the project being unfamiliar with the technologies identified to carry out the project. This has been considered as having a high impact on the project, without the skill to use the tools, the software cannot be created. The probability has been judged as medium however as there is enough time scope to learn the skills required. The choice of popular, well documented technologies helps mitigate this problem with many support forums and mailing lists available for aid. There are also a plethora of tutorials available online, as well as support available from the experienced project supervisor.

The project being too complex is the third risk identified given a medium probability and a medium impact. Due to the novel nature of the project a degree of complexity is to be expected. Similarly to the previous risk, support and guidance from an experienced supervisor will be helpful. Similar seeding algorithms such as streamlines and stream surfaces can be consulted.

The fourth risk was identified from the paper by Arnuphaptrairong [Arn11] mentioned above, having poor supervisor support. This has been deemed as a low probability with a high impact. Because of the novelty of the project,

Table 1: Risk Analysis Table

Risk	**Probability**	**Impact**	**Precautions taken**
The software cannot be created with the specified tools	Low	High	A proven language is to be used along with powerful libraries
A lack of knowledge or skill to successfully execute the project	Medium	High	Well documented language and interfaces are to be used
The project is too complex	Medium	Medium	Supervisor to advise on project simplifications
Poor supervisor support	Low	High	Weekly meetings with supervisor
Poor/no domain expert input	Medium	Low	Can visit university engineering department
Software generated contains errors	Medium	Medium	Time allowed for debugging
Personal illness	Low	High	Wellbeing and doctors available on university campus
Underestimation of the time required for sections of the project	Medium	Medium	Scheduled writing time can be shared with project finalisation
Equipment failure	Low	Medium	Computer equipment available on campus
Data loss	Low	High	Ensure backups are taken regularly

experienced guidance is necessary. With weekly face to face meetings scheduled this risk should be mitigated.

Guidance from a domain expert is important to the project to help suggest stream ribbon placement and to give feedback on the results produced. The unavailability of a domain expert has been identified as a medium probability, low impact risk. A visit to the engineering department within the university should help locate an appropriate person to consult. However without a domain expert the project can still be completed.

Another risk identified is that the code contains errors. This has been classed as a medium probability risk due to the inexperience of the author in using the technologies, with a medium impact. Again the technologies are well documented and supported which helps mitigate this. The code will be written to standards set according to Laramees paper Bobs Concise Coding Conventions (C3) [Lar10a], allowing easier debugging and future maintenance.

The seventh risk identified is due to personal illness of the author. This has been classed as a low probability risk with a high potential impact. Student welfare services are available on campus in case of stress and health care support is available.

The underestimation of time for each section of the project is the next risk identified. This has been classed as a medium risk as the author has little experience in knowing the time required to use the technologies specified. There is however time available within the project schedule for contingency, weekends are not specified as working days but could be utilised. Some parallelisation of writing the final report while completing other aspects is also possible.

Equipment failure is another risk identified and given a low probability. Potential impacts are the loss of data, problems in producing a hard copy of the final report and having a platform to write code or the report on. Alternative equipment is available for use in this project helping to mitigate the risk.

The final risk identified is from data loss. This has been given a low probability, but with a high potential impact. Precautions taken are to make multiple backups of work completed so far.

5 Project Design

Table 2 lists the software classes and their responsibilities .

Figure 18 shows an interaction diagram between the software classes. There are five different types of classes in the design. The first type is the MainWindow class which is inherited from the inbuilt Qt 'QMainWindow' class. This class manages the user interaction with the main user interface. The second type of class are those that inherit from the inbuilt Qt 'QOpenGLWidget'. These classes are the OpenGL windows responsible for displaying information to the user.

The third type of class are those that inherit from the Qt 'QDialog'. These classes are responsible for managing user interaction with pop-up dialogs. The fourth class type is the ColourMap class which is simply a container holding the information for the colourmaps. The final class type are container classes for flow domain data such as the vector information and helicity at each vertex.

Figure 19 shows the dependency diagram for the MainWindow class. Further examples of Doxygen generated diagrams are available online at http://cs-ugrad.swan.ac.uk/849119/.

Table 2: Class Table

Class name	Description
chooseMap	Manages the dialog for selecting a colour map.
clearPoint	Keeps track of co-ordinates availability as new seed point. A temporary set of points is used while constructing a ribbon. The temporary set of points get converted to permanent on completion of an individual ribbon.
ColourMap	Colourmap rgb values.
GLWidget	Draws in the main 3D pane using OpenGL instructions. Handles mouse interaction. Handles lighting effects.
legGLWidget	Draws colourmap on legend. Communicates map number to MainWindow.
load	Manages the dialog for reading or generating data. Read datafile or generate dataset. Communicates availability of data.
MainWindow	Manages main GUI. Performs vector field calculations and calculates stream objects. Keeps track of data objects.
Ordered	Container class to order and store co-ordinates in order of (helicity) key. Calculates the highest and lowest value to be used for the colour legend.
OrieGLWidget	Draws orientation pane. Resets zoom and orientation.
scalarField	Container class for storing 3D scalar values. Interpolates scalar values between vertices.
vecField	Container class for storing 3D vector values. Interpolates vector values between vertices. Calculates field gradients. Calculates field curl.

6 Implementation

This section describes the features implemented in the project. It has been split into two distinct parts, one for the implementation of the software and one for the implementation of the automatic stream ribbon seeding algorithm.

6.1 Software Implementation

This subsection describes the software developed for this project. The features of the software are explained along with details of how they are implemented.

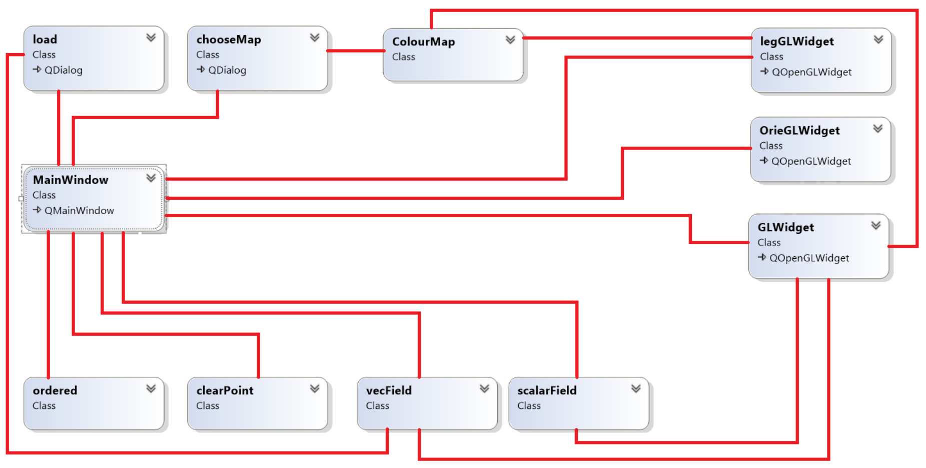


Figure 18: Class interaction diagram.

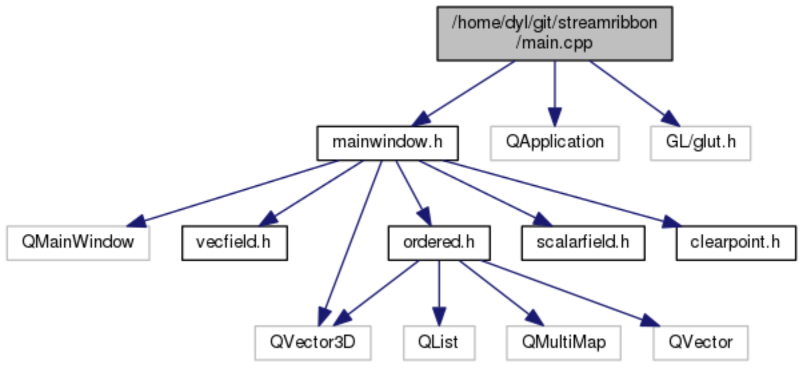
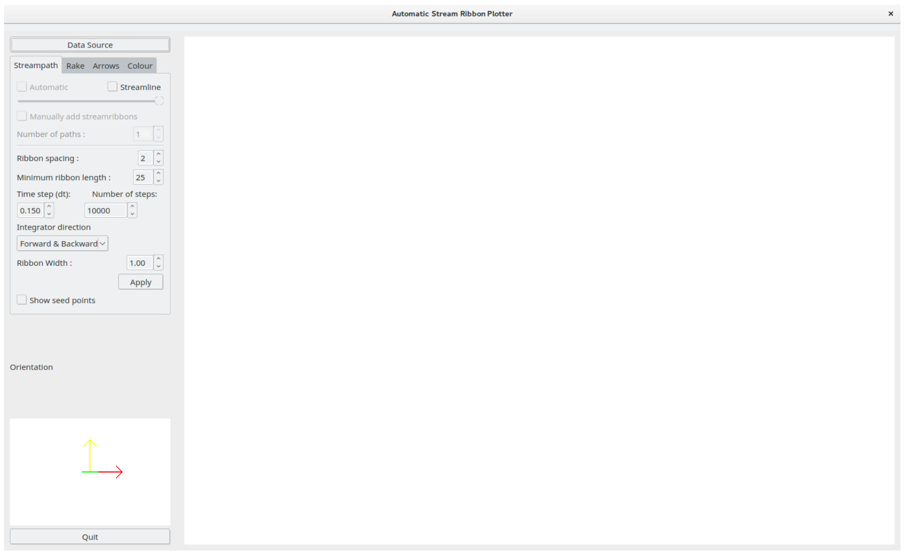


Figure 19: Doxygen generated dependency graph for the MainWindow class.

6.1.1 Graphical User Interface

Figure 20 shows the graphical user interface when the program is run. There are three distinct areas that can be mediately identified, the bulk of the interface is taken by the 3D visualization pane which is blank when the program is initiated. On the bottom right of the interface there is an orientation pane which shows the orientation of the dataset in the 3D visualization pane. Finally there are the user interaction options.

To operate the program a dataset needs to be loaded. This is done by clicking the 'Data Source' button on the top left of the user interface. This action launches the 'Load Data' dialog, passing a vecField class named mData to the dialog, which is to be populated with the velocities of the flow domain.



Automatic

Streamline

Manually add streamribbons

Number of paths : 1

Ribbon spacing : 2

Minimum ribbon length : 25

Time step (dt): 0.150

Number of steps: 10000

Integrator direction: Forward & Backward

Ribbon Width : 1.00

Show seed points

Figure 20: Stream ribbon software when initially opened.

The vecField class is the main data structure used in the software. It is a class designed to hold the three velocity vectors at each vertex. The main data structure within the class is a 2D array consisting of three floating point numbers for each vertex within the flow domain. Memory is allocated for the vecField object using the 'malloc' function from the 'stdlib' library. An alternative data storage method would have been to use the Qt 'QList' container along with the 'Q3DVector' container. Before data can be input into the vecData class, the size of the flow domain must first be input using the 'setSize' function. Vector data is input into the class using either 'setX', 'setY' or 'setZ' function as appropriate.

The vecField class is able to process the data to return different properties of the vector field. These are accessible by using the get methods of the class. The getX, getY and getZ functions return the x, y and z vector components respectively for a given x, y and z coordinate. The vecField class also has a function that interpolates the data between vertices to return vector components. The getSpeed function returns the scalar speed value for given coordinates while the angV function returns the angular velocity about the flow tangent. Six velocity gradients are also retrievable, the x velocity gradient in the y and z direction, the y velocity gradient in the x and z direction and the z velocity gradient in the x and y direction. These are accessed using the UYgrad, UZgrad, VXgrad, VZgrad, WXgrad and WYgrad respectively.

6.1.2 Loading Data

A 'Load Data' dialog box is the main interface for the user wishing to load data into the program. A screen shot of the dialog can be seen in figure 21. The dialog has two initial choices for the user. The first choice is to be able to load data from a file and the second is to generate a dataset from inbuilt functions. This choice is given in the form of a Radio Button, allowing only one option to be selected at a time.

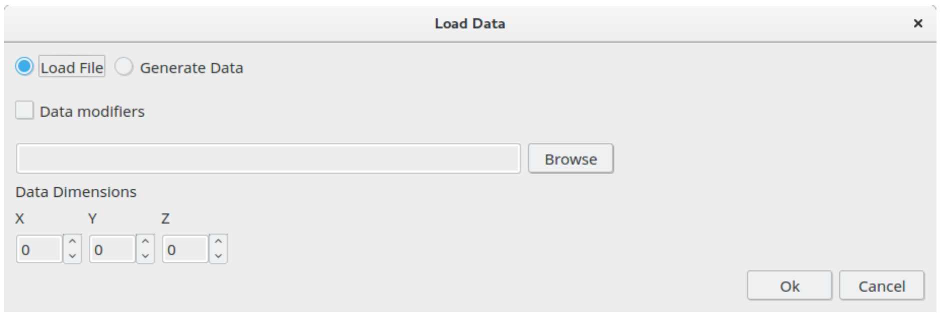
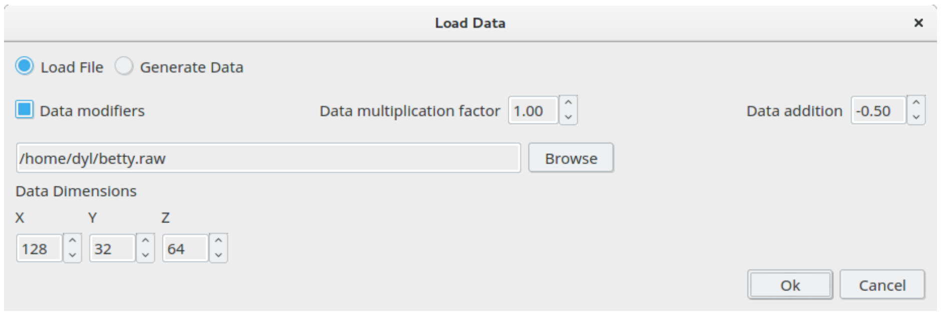


Figure 21: Load Data dialog box when initially opened.

While the 'Load File' radio button is selected the user is presented with a text field for the file path, a 'Browse' button and a Check Box to allow for data modifiers. On clicking the browse button a standard 'QFileDialog' dialog box appears that allows the user to browse and search for files on the PC. Files displayed in the file dialog are limited to the .raw file extension, the file extension used for the given RayleighBnard convection flow. The software is limited to only being able to read files in a binary unsigned character in little endian format with the .raw file extension. Once a file had been chosen, on pressing 'OK' within the file dialog, the full path and file name will be populated into the text field of the load data dialog.

The 'Data modifiers' check box allows for minor alterations to be made to the data read in from the .raw file. This option was created to accommodate a modification required to read in the RayleighBnard convection flow dataset. This modification required the subtraction of 0.5 from each value. Two 'Spin Boxes' and labels appear when the 'Data modifiers' box is selected, the first spin box, labelled 'Data multiplication factor' enables a multiplication factor to be applied to the dataset. The second spin box is labeled 'Data addition' and allows for the addition of a value to each dataset value. Figure 22 shows the 'Data modifiers' check box enabled with the 0.5 subtraction applied to the RayleighBnard convection flow dataset.


Load Data

Load File [✓] Generate Data

[✓] Data modifiers — Data multiplication factor 1.00 — Data addition -0.50

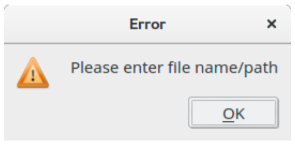
/home/dyl/betty.raw — Browse

Data Dimensions

X 128 — Y 32 — Z 64

Figure 22: Load Data dialog with file path populated, data modifiers enabled and dimensions of the dataset recorded.

Below the file path text field there are three additional 'Spin boxes' that allow for the dimensions of the dataset to be input, one for each of the x, y and z Cartesian coordinates. The final aspect of the 'Load Data' dialog are the 'OK' and 'Cancel' buttons. If the cancel button is pressed the dialog will close, cancelling all inputs entered. The OK button applies all the settings input and loads the appropriate dataset. If there are issues with the input data, such as no dimensions entered for the dataset or no filename chosen, a warning message will display with the issue. This can be seen in figure 23. The warning dialog is created using the unmodified Qt 'QMessageDialog' class.

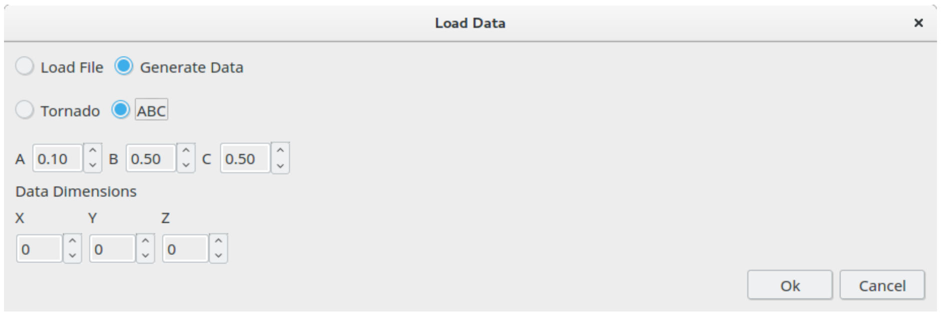

Error

Please enter file name/path

Figure 23: Warning Dialog received when no filename is entered.

If all the requirements have been applied and the 'Ok' button clicked the readFile function is applied. This function takes uses the fstream and iostream libraries to read in the file in the given file path. The function uses three nested for loops to read the x, y and z coordinate vectors. Data modifiers are the applied before the data gets stored in a vecField class object, using the set functions, which is then returned to the main window.

The 'Generate Data' radio box allows for the generation of the other two datasets used in the project. When the 'Generate Data' radio button is selected, the file path text field, the 'Browse button and the 'Data modifiers' check box disappear. In their place another two radio buttons will appear, one for the generation of the tornado dataset and one for the generation of the ABC flow. Only one of these dataset is able to be selected at a time. Selecting the 'ABC' radio button displays an additional three spin boxes to enable the user to set values for the A, B and C constants for generating the dataset. By default these values are set as A = 0.1, B = 0.5 and C = 0.5 as can be seen in figure 24. Selecting the Tornado dataset again hides the A, B and C spin boxes. For both the tornado and the ABC datasets, the dimensions of the flow domain must be input as for the file load option. On clicking Ok either the 'genABC' function or the 'genTornado' functions are used to populate a vecData class object with the appropriate vector field.



A 0.10 B 0.50 C 0.50

Data Dimensions

X 0 Y 0 Z 0

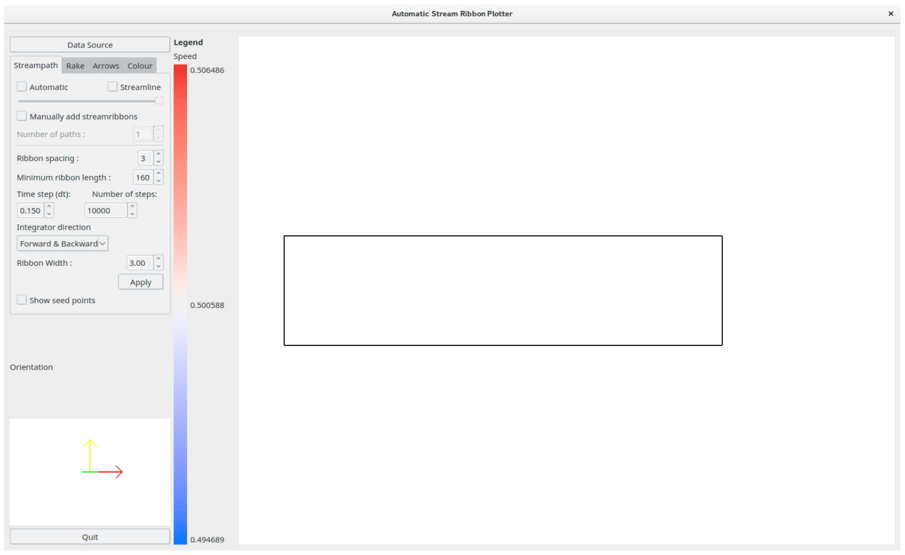
Figure 24: Load Data dialog with Generate Data and ABC radio boxes selected.

6.1.3 3D Visualization Pane

Once data has been loaded into the program, the 3D visualization pane will display a box representing the flow domain as seen in figure 25. The 3D visualization pane is managed by the GLWidget class. This class is inherited from the Qt 'QOpenGLWidget' class and uses the inherited initializeGL, paintGL and resizeGL functions. The initializeGL function is used to set the background colour of the pane and to enable some OpenGL functions such as lighting effects and the reflective qualities of surfaces. The resizeGL function is used to manage the perspective when the window is resized while the paintGL function contains the OpenGL instructions for the visualization displayed.

The paintGL function initially loads the colourMap to be used for the visualization. After this it assesses the size of the flow domain to set up the perspective in order for the whole flow domain to be displayed no matter what size. When initially developing the software it appeared as if arrows to be drawn were not being drawn. It was then discovered that the arrows were being drawn out of the field of view, this feature was added to counter this.

The positioning and zoom level of the flow domain are applied next, followed by the field rotation. The user is able to interact with the image in the 3D visualization pane by using the mouse. The mouse movements and button presses are tracked by the Qt 'QMouseEvent' library, if the mouse is moved



☐ Automatic ☐ Streamline

☐ Manually add streamribbons

Number of paths : 1

Ribbon spacing : 3

Minimum ribbon length : 160

Time step (dt): 0.150

Number of steps: 10000

Integrator direction: Forward & Backward

Ribbon Width : 3.00

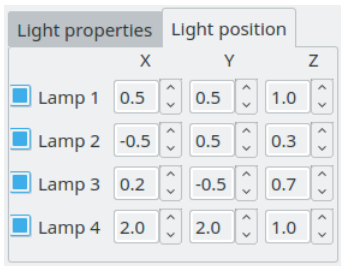
☐ Show seed points

Figure 25: The main graphical display after loading the RayleighBnard convection flow dataset.

while the left button is depressed, then a x and y positioning factor is adjusted. The mouse movements also call the update function which reruns the paintGL function. By depressing the right mouse button and moving the mouse, rotation of the scene is applied, vertical movement causing rotation about the x axis and horizontal movement causing rotation about the y axis. Rotation about the z axis is achieved by pressing the right mouse button along with the Ctrl key on the keyboard and moving the mouse in a horizontal direction. The rotation is achieved by adjusting a floating point number which is applied in a OpenGL command glRotatef within the paintGL function. The rotation factors are also sent to the orientation pane.

Zooming is available by using the Qt 'QWheelEvent' function. Rotating the wheel on the mouse changes a floating point zoom factor number. This number then adjusts the perspective allowing for zooming of the flow domain.

Once the paintGL has handled the rotation the setLight function is called. This function applies the light settings set by the user from the options panel, found in the tab labelled 'Colour'. These options are to turn on or off four separate light sources and to vary the positioning of the lights, these options can be seen in figure 26. Lights are switched on and off by using glEnable and glDisable functions, and their positions set using glLightfv function. The ambient, diffuse and specular light conditions can also be set by the user using horizontal sliders as seen in figure 27. These are also set using the OpenGL glLightfv function.



	X	Y	Z
☑ Lamp 1	0.5	0.5	1.0
☑ Lamp 2	-0.5	0.5	0.3
☑ Lamp 3	0.2	-0.5	0.7
☑ Lamp 4	2.0	2.0	1.0

Figure 26: User options for to enable lights and their positioning. All lights are enabled in this case.

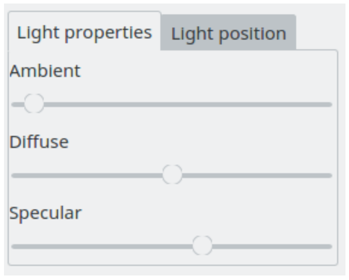


Figure 27: User options for lighting levels.

Once the lighting is set the paintGL function calls the drawFrame function. This function draws the outline of the flow domain by using the glBegin function with the GL_LINE_STRIP parameter. glVertex3f coordinate points are then used to outline the extremities of the flow domain according to the size set by the user. Finally the paintGL draws the velocity field according to the options as selected by the user.

Knowing which way the flow domain is orientated after rotating can be confusing. The orientation widget, on the bottom left of the user interface, was implemented to alleviate the confusion by using arrows to allow the user to see which way the flow domain is orientated. The orieGLWidget class manages the visualization in this pane, which is again inherited from the Qt OpenGLWidget class. Three coloured arrows are the only objects drawn in the pane using the paintGL function. A yellow arrow is drawn to represent the increasing y-axis, a red arrow for the x-axis and a green arrow for the z-axis. The arrows are drawn using the same glBegin function as in the GLWidget but with the GL_LINES parameter used instead. The same rotation values from the main 3D visualization pane GLWidget are used to rotate the arrows the same as in the main 3D visualization pane using the same glRotatef function.

The orieGLWidget class also uses the Qt QMouseEvent to reset the orientation back to the x-y face. Clicking the right mouse button resets the rotation factors in both the orieGLWidget and GLWidget. Clicking the left mouse button resets the zoom factor in the GLWidget putting the whole flow domain back into view within the 3D visualization pane.

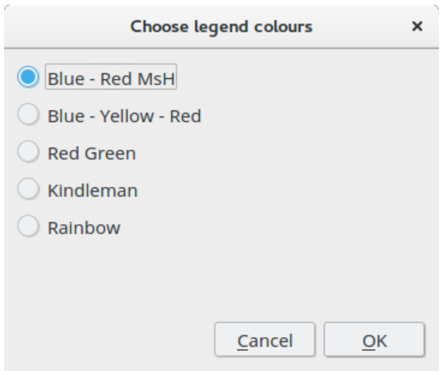
6.1.4 Colour Legend

Once data has been loaded a colour legend appears, as can be seen in figure 25. By default this colour legend is mapped to the speed of the vector field with a red to blue through white diverging colourmap. The coloured legend strip is managed by another class inherited form the Qt 'QOpenGLWidget' called legGLWidget. The paintGL function within the legGLWidget class simply draws a rectangle and applies a colourmap to it. A LoadAllTextures function is used to manage what colourmap is applied according to a parameter from the MainWindow class. This parameter is a number which when applied to an object of the colourmap class, returns an array of rgb values to be loaded as the colour texture. The colourmap class is a class containing arrays that make up the rgb values for the different colourmap options.

The legGLWidget class also uses the Qt 'QMouseEvent' to allow the user to modify the colourmap. When the user clicks the colour legend a new dialog opens to allow the user to chose a colourmap. This dialog is managed by the chooseMap class and can be seen in figure 28. The user has a choice of five colourmaps which can be selected and applied using the 'OK' button. This sends a value corresponding to the colourmap back to the MainWindow class where it is then used in the GLWidget and legGLWidget classes to colour vector field visualization the and the legend respectively.

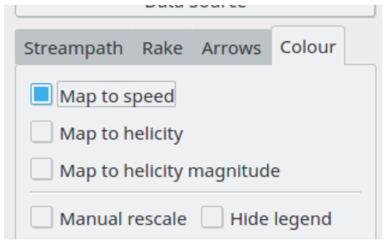
The defaut colourmap choice is a diverging colour from blue to red through white as presented by Moreland in [Mor09]. Other colourmap options are the Kindlmann colourmap presented in [KRC02], a red to green colourmap, a diverging blue to yellow to red colourmap and, despite the criticism from Moreland in [Mor09], a rainbow colourmap.

The user has the option to chose what parameter the colour is mapped to in the 'Colour' tab of the user options panel as can be seen in figure 29. These options are to map colour to the vector field speed, the vector field helicity or the vector field helicity magnitude. To calculate the scalar values at the extremities of the scale, a ordered class was created. The ordered class takes the x, y and z coordinates as an input along with a scalar value. The class uses the Qt QMultiMap container class to store the x, y and z coordianates as a QVector3D object, using the scalar value as the key. This allows the coordinates to be stored in ascending scalar value. By accessing the first and last values in the object, the extremities of the scale could be found.



[✓ Blue - Red MsH]
Blue - Yellow - Red
Red Green
Kindleman
Rainbow

Figure 28: Colourmap choice dialog.



[✓ Map to speed]
Map to helicity
Map to helicity magnitude
Manual rescale
Hide legend

Figure 29: User colour mapping options.

The ordered class has a set method which takes a set of coordinates as an input along with the scalar value to input into the class. A getFScal method can be used to return the highest scalar value in the object while a getLScal method returns the lowest values. While developing the software it was discovered that the RayleighBnard convection dataset has a few values that were far lower than the vast majority. This caused a skewing of the colourmap. To counter this a function was added into the ordered class which took the scalar values at the 99.9^{th} percentile and compared it to the extremities. If the value at the extremeties were found to be far beyond the scale to the 99.9^{th} percentile, the scalar value for the 99.9^{th} percentile position would be used. This function is accessed via the getUCS for the maximum value of the scale and via the getLCS for the minumum value of the colour scale.

A getScal function is also available within the class that return the scalar value for a given set of coordinates as an input. This is a slow accessors method however and was not implemented in the software. getFQVec and getLQVec functions return the first (lowest scalar value) and last (highest scalar value) coordinate positions. These functions take an integer input to return the value of the position of the integer, e.g. entering the integer value three into the getFQVec function will return the coordinates of the third lowest scalar value.

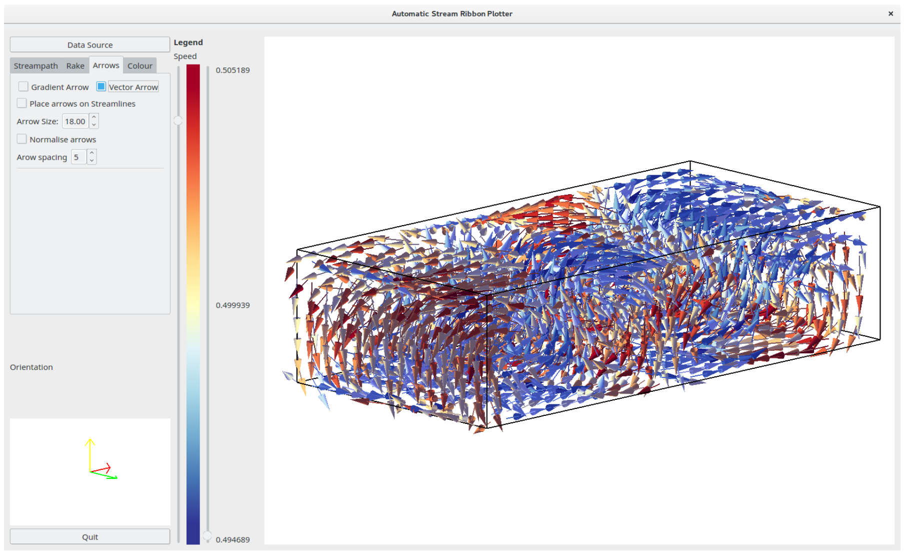
This was found to be a slow accessors method for larger inter inputs therefore a new function was added that copied the QVector3D values into a QList class, keeping the same order. The QList class has the capability of being directly accessed whereas the QMultiMap class does not.

When the option to map the colour is changed to helicity or helicity magnitude, the helicity must first be calculated. The helicity is calculated within the MainWindow class and is computed when required. Helicity is calculated according to equation 1, this requires vorticity to be calculated first. The x-axis vorticity is calculated by subtracting the mData vecField VZgrad from the WYgrad value. The y-axis vorticity by subtracting the WXgrad value from the UZgrad value and the z-axis vorticity by subtracting the UYgrad value from the VXgrad value. Once vorticity is calculated, the x-axis vorticity is multiplied by the x-axis velocity component, the y-axis vorticity by the y-axis velocity component and the z-axis vorticity by the z-axis velocity component. Adding all three values together give the helicity. Because helicity is directional, the helicity magnitude is also calculated by using the 'fabs' function from the Math.h library. The helicity values are stored in a scalarField class. The scalarField class is similar to the vecField class except that there is only one value to be stored at each vertex and gradient functions are absent. The helicity values are also passed into a ordered object to find the colour legend extremities. The helicity scalarField class is passed to the GLWidget class which then uses the appropriate scalar values to plot each vertex colour. The maximum and minimum values returned from the ordered class for the appropriate scalar value are then added to the legend, with a middle point calculated.

A hide legend option is available that allows the user to hide the colour legend, as well as manual rescale options. When the manual rescale option is selected, two vertical sliders appear either side of the legend as can be seen in figure 30. By moving these sliders it is possible to change the scale of the colourmap to allow greater colour definition.

6.1.5 User options

The user option panel has multiple options to allow the user to create visualizations. Within the 'Arrows' tab there are options to allow for the domain to be filled with arrow glyphs. Arrows can be used to show the velocity or the velocity gradient. An example of the arrows can be seen in figure 30. The size of the arrows displayed are by default proportional to the value that they are showing. The size of the arrows can however be increased by a multiplication factor used in the spin box labelled arrow size. A normalize arrows check box is also available to make all arrows the same size. Arrows are placed at vertices according to the arrow spacing spin box. A value of ten in the spin box signifies that arrows will be visualized at every tenth vertex.



[ ] Gradient Arrow [✓ Vector Arrow]
[ ] Place arrows on Streamlines
Arrow Size: 18.00
[ ] Normalise arrows
Arow spacing 5

Figure 30: Velocity arrows on the RayleighBnard flow at an spaced at an interval of five. Colour is mapped to speed using the Blue - Yellow - Red colourmap. Manual legend rescale sliders can be seen.

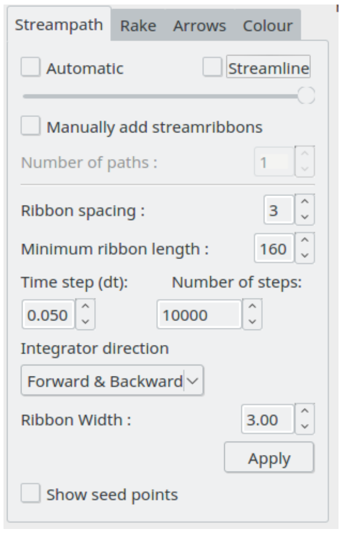
When the 'Vector Arrow' option is selected, a boolean value within the GLWidget class is switched to allow for the arrows to be drawn. The mData vecField dataset, generated when the data was loaded, is passed into the GLWidget so the vector information can be accessed and drawn. To generate the gradient arrows, the velocity gradient must first be calculated. The gradient is calculated within the MainWindow class and is computed when required, if only speed arrows are drawn the gradient is not calculated at all. Once calculated, the gradient vectors are stored until the program is terminated or a new dataset is loaded. The gradient vectors are stored in another vecField class which is then passed to the GLWidget for rendering. The vector gradient is calculated by comparing the speed at either side of a vertex in one axis of the flow domain, the difference between the values give the gradient component for the chosen axis. This is done for all three axis to give the gradient vector.

The GLWidget contains a function for rendering the arrows called drawArrows. This function is called with the location coordinates of the vertex the arrow is to be drawn on and the coordinates of the end of the arrow. The function then draws a the arrow line along with a cone object from the freeGLUT OpenGL libraries, orientating the cone in the right direction.

Another option on the 'Arrows' tab of the user interface allows for the arrows to be drawn in place of stream objects. This allows arrows to be drawn along the calculated stream lines generated in other tabs rather than filling the whole domain.

To render stream objects, an integrator must be applied to the vector at a seed point to find the line at a tangent to the flow. This software implements the fourth order Runge-Kutta method as seen in equation 3. The integrator options can be set under the 'Streampath' tab. Options available are the size of the time step, the integrator direction and the maximum number of steps. The integrator direction can be set to forward, backward or backward and forward. Figure 31 shows the Streampath tab along with the integrator options.

$$\begin{gathered} s_{i+1} = s_i + (a + 2 \bullet b + 2 \bullet c + d)/6 \\[1em] a = dt \bullet v(s_i) \\ b = dt \bullet v(s_i + a/2) \\ c = dt \bullet v(s_i + b/2) \\ d = dt \bullet v(s_i + c/2) \end{gathered} \tag{3}$$



Automatic ☐ Streamline ☐

Manually add streamribbons ☐

Number of paths : 1

Ribbon spacing : 3

Minimum ribbon length : 160

Time step (dt): 0.050

Number of steps: 10000

Integrator direction: Forward & Backward

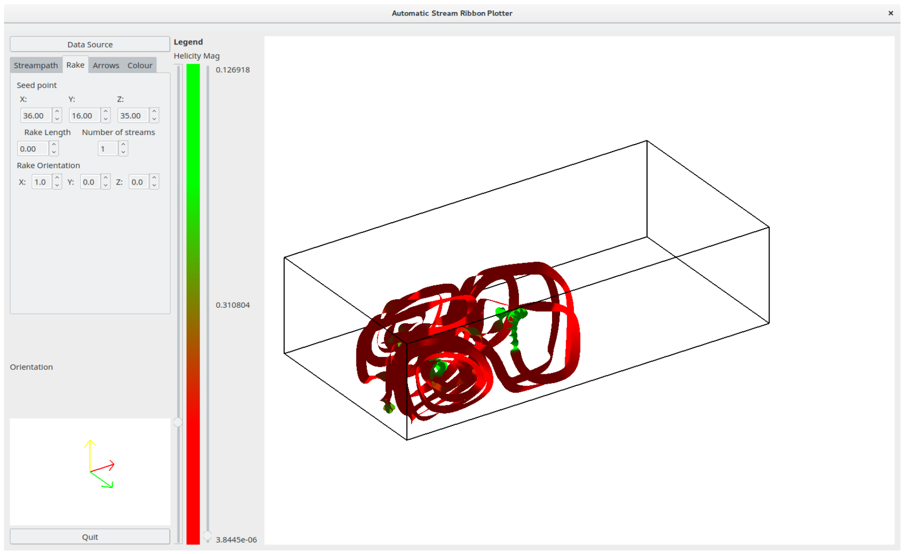
Ribbon Width : 3.00

Show seed points ☐

Figure 31: The Streampath tab of the user interface.

The tab labelled 'Rake'enables the user to manually set a seed point for a stream ribbon. Three spin boxes are available to be filled with the x, y and z coordinates for the ribbon seed point. Once coordinates have been entered, the stream ribbon calculated from that seed point shall be rendered according to the integrator settings and to the width stated in the 'Ribbon width' spin box, also on the Streampath tab. The stream ribbon is stored as a QList of QVector3D and is calculated using the matRibbon function within the MainWindow class. The matRibbon function takes the coordinates of the seed point as an argument and calculates the coordinates of the next step by using the forth order Runge-Kutta. If the coordinates of the next step exceed the flow domain the function exits. A normal is found to the vector between the seed point and the coordinates of the next step. The positions that are half the ribbon width either side of the seed point, along the normal line are stored in the QList. The next step is then performed but with the addition of a rotation of the normal about the streamline axis. The magnitude of the rotation is set according to the angular velocity. The stream ribbon continues until a termination condition has been reached. This ribbon is then passed into the GLWidget class where it can be rendered. A single manually seeded stream ribbon can be seen in figure 32.

As well as a singular stream ribbon, it is possible to draw a rake of stream ribbons. This is also done in the 'Rake' tab. The 'Rake' tab of the user interface can be seen in figure 32. The length of the rake is input in the 'Rake Length' spin box while the number of seeding points along the rake set in the 'Number of streams' spin box. The rake will be drawn from the seed point a distance set by the rake length spin box. The direction of the rake is set according to the normalised vector in the rake orientation x, y and z spin boxes.



Seed point

X: 36.00 Y: 16.00 Z: 35.00

Rake Length: 0.00 Number of streams: 1

Rake Orientation

X: 1.0 Y: 0.0 Z: 0.0

Figure 32: A single stream ribbon manually seeded in the RayleighBnard convection flow. The Red Green colour map is used and mapped to helicity magnitude and the scale manually adjusted. The Rake tab of the user interface can be seen with the seeding location.

The extra ribbons are constructed in the same manner as the single ribbon and again passed into the GLWidget class. Within the GLWidget class there is a QList container class that stores a QList class of QVector3D classes. This QList container class stores the array of ribbons to be rendered.

The automatic stream ribbon seeding algorithm is applied by selecting the 'Automatic' check box on the Streampath tab that can be seen in figure 31. The following section describes the algorithm used. The algorithm uses an additional class called clearPoint. The clearPoint class consists of two arrays of boolean values, with the length of each of the arrays equal to the number of vertices in the flow domain. One array is considered the permanent array while the other array is considered temporary. All values are initially set to true in both arrays. While a stream ribbon is being constructed, the boolean in the temporary array equivalent to the vertex that is being added to the ribbon object are set to false. The boolean in the temporary array is also set to false for all vertices that surround the vertex being added to the stream ribbon. The surrounding distance is set using a clearance value in the clearPoint class, which in turn is set to the value in the 'Ribbon Spacing' spin box. Therefore by increasing the value in the ribbon spacing spin box, more vertices will be set as false in the temporary array.

If a ribbon fails to meet the length specified in the minimum ribbon length spin box, the boolean values in the temporary array within the clearPoint class are all reset to true and the ribbon calculated discarded. If the ribbon calculated exceeds the minimum ribbon length, the values that are false in the temporary array are set to false in the permanent array and the ribbon is then added to the list to be rendered. When a new ribbon is being constructed, vertices are checked against the clearPoint class permanent array to check they are not too close to existing ribbons. If they vertex for the construction of a new ribbon has already been set in the clearPoint class, the construction of the new ribbon is then halted. If the 'Automatic' box is deselected the clearPoint class is reset.

A horizontal slider is enabled on the 'Streampath' tab when the 'Automatic' check box is checked. This slider controls the number of ribbons being rendered. Decreasing the slider removes the last constructed stream ribbon by stopping the GLWidget iterating through all ribbons in the QList.

A 'Manually add streamribbons' option is also enabled when the 'Automatic' check box is selected. When this option is selected, the 'Ribbon spacing' and 'Minimum ribbon length' options are disabled and the 'Number of paths' spin box is enabled. Deselecting the 'Manually add streamribbons' option reverses this. When enabled the automatic algorithm is overridden and a number of stream ribbons equal to the value in the 'Number of paths' spin box will be rendered. The ribbons will be seeded in the order of greatest helicity magnitude, disregarding the positioning of any other stream ribbon.

A 'Streamline' check box is also visible on the 'Streampath' tab. By selecting this option, all stream ribbons will be rendered as streamlines. Another check box on the 'Streampath' tab is the 'Show seed points' option. Selecting this renders a black sphere at the point where each stream ribbon was seeded. The sphere is generated in the GLWidget using the freeGLUT library function. Seeding points can be seen in figure 33.

6.2 Automatic Seeding Algorithm

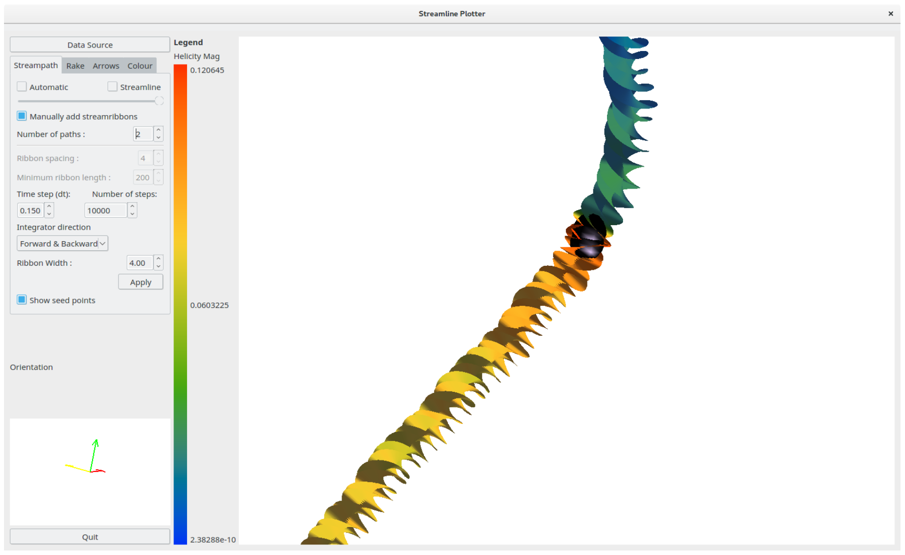
This subsection describes the seeding algorithm implemented for this project.

The advantage of stream ribbons over streamlines are that they have the ability to show helicity in a field. Therefore a stream ribbon placement should primarily take into account the helicity of the field. With this in mind the algorithm first calculates the helicity value for every vertex in the volume. These vertices are then ordered in ascending value according to the helicity magnitude. Helicity is directional, a anticlockwise motion is negative while a clockwise motion is a positive helicity value. Because the direction of helicity is unimportant, the helicity magnitude is used.

The program allows the user to seed the stream ribbon at the vertex showing the highest helicity value, and continue adding stream ribbons at the next highest values. This however showed to be inappropriate as often two vertices next to one-another would have a similar helicity magnitude and so the ribbons created from the seed points would wrap around each other as can be seen in figure 33. A this effect is more pronounced in simulations with more nodes.

To overcome this issue it is proposed to use a technique similar to the one used by Max et al. in [MCG94]. This only allows for seeding of streamlines, or in this case stream ribbons, a fixed distance from another streamline. Stream ribbons should also terminate when too close to another stream ribbon. The separation distance allowed between stream lines can be controlled by the user. Stream ribbon seeding order is dictated by the helicity magnitude as before, as long as the seeding point is the minimum distance away from previous stream ribbons. The ribbon is constructed forward and backward from each seed point. Once all vertices within the flow domain have been analysed as seed points, the algorithm is complete.

A tenancy for producing many short ribbons was seen while initially testing the algorithm. These ribbons offered minimal insight into the flow pattern. To prevent this a minimum stream ribbon length option is also proposed as a user option. This rejects seeding points where the resulting stream ribbon does not meet the set minimal length.



☐ Automatic ☐ Streamline

☑ Manually add streamribbons

Number of paths : 2

Ribbon spacing : 4

Minimum ribbon length : 200

Time step (dt): 0.150

Number of steps: 10000

Integrator direction: Forward & Backward

Ribbon Width : 4.00

☑ Show seed points

Figure 33: Two stream ribbons seeded next to one-another in a 128^3 node tornado simulation.

Ideally the stream ribbons should occupy all features within the vector field, even where the helicity magnitude is low, in order for the casual viewer to see a full representation of the flow. It is therefore important to indicate flow in all areas of the vector domain. This however can lead to a problem with occlusion in 3D fields, therefore a balance needs to be found. Occlusion is a particular problem with ribbons as they are essentially wide lines which take up more space, leaving less space to see into the centre of the volume.

To counter the occlusion problem, a filtering option is proposed that removes the last seeded ribbon, i.e. the ribbon with the least helicity magnitude. Further filtering removes the ribbon with the next smallest helicity magnitude and so on until all ribbons are removed.

Another consideration with producing stream ribbons is their width. As previously mentioned, if ribbons are too wide they will cause greater occlusion but if too small the twisting will be difficult to see. A guideline ribbon width is suggested that is proportional to the size of the flow domain. This guideline applies to rectangular flow domains. The distance from the centre of the domain to a corner is calculated, this is then divided by sixty. The ceiling function is then used to give an integer value used as the ribbon width.

Given a suggested ribbon width, a suggested stream ribbon separation distance is also suggested as the same distance as the ribbon width. The minimum ribbon length is also suggested as eight times the ribbon width. These values are only initial suggestions however and the user should be able to adjust these values to their own requirements.

7 Testing and Evaluation

This section describes the testing of the software and shows the results of the testing.

Unit testing and integration testing were continuously performed while development of the software was ongoing. New features implemented would be tested alongside other features with the three separate datasets. Any bugs found were addressed. Testing of invalid inputs that would cause the application to crash was also performed. Three datasets were used to test both the program and the algorithm.

The first dataset is the RayleighBnard convection flow. The dataset provided has a fixed resolution of 128 x 32 x 64 and is provided as a file with a binary .raw format.

The second dataset is a tornado simulation and is an analytical data set produced from a procedural function on a regular Cartesian grid. The size of the tornado dataset can be adjusted by manually specifying the dimensions of the dataset. Testing will primarily use a 128^3 node model.

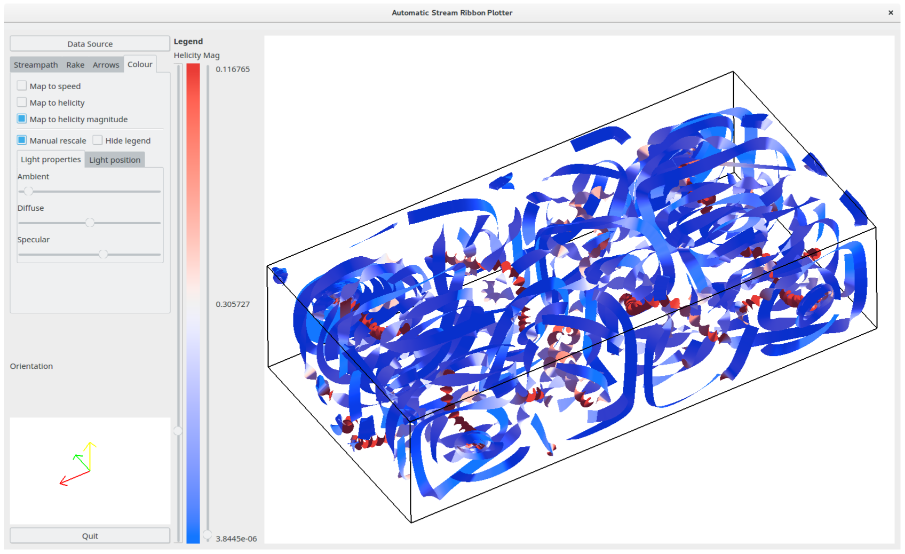
The final dataset is a ArnoldBeltramiChildress (ABC) flow. It is another flow field that is produced from a procedural function as seen in equation 2. A, B and C are constants that can be adjusted to create different flows. For this project the parameters be used are: A = 0.1, B = 0.5 and C = 0.5. The size of the flow field can also be adjusted as for the tornado dataset, testing will primarily use a 128^3 node model.

7.1 Results

Figures 34, 35 and 36 show visualizations of the three datasets created by the software, using the automatic stream ribbon seeding algorithm. A demonstration video can be seen at:

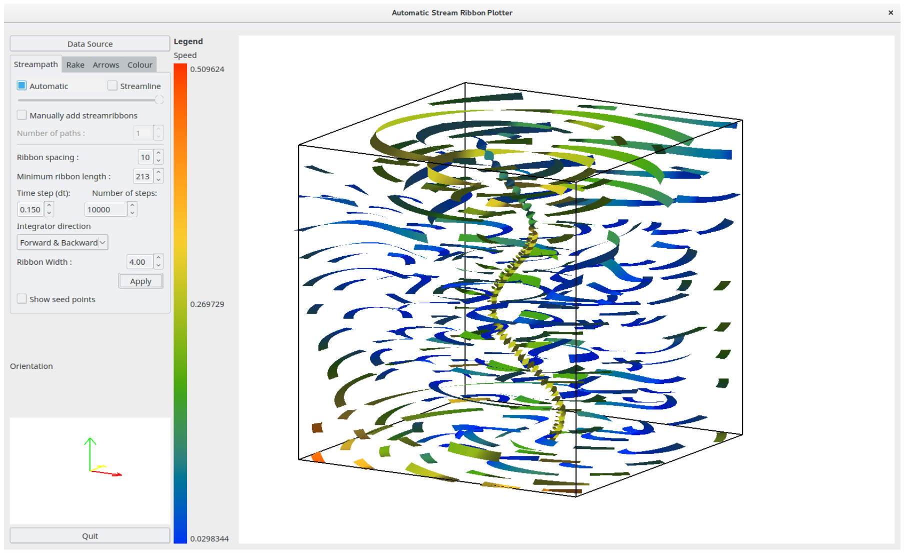
`http://cs-ugrad.swan.ac.uk/849119/`

Figure 37 shows an example of the rake implementation along with the seeding locations and the streamline implementation. The Kindlmann colourmap can also be seen in the figure which is mapped to helicity. An example of arrows placed on streamlines can be seen in figure 39.



Map to speed
Map to helicity
[✓ Map to helicity magnitude]
[✓ Manual rescale] Hide legend

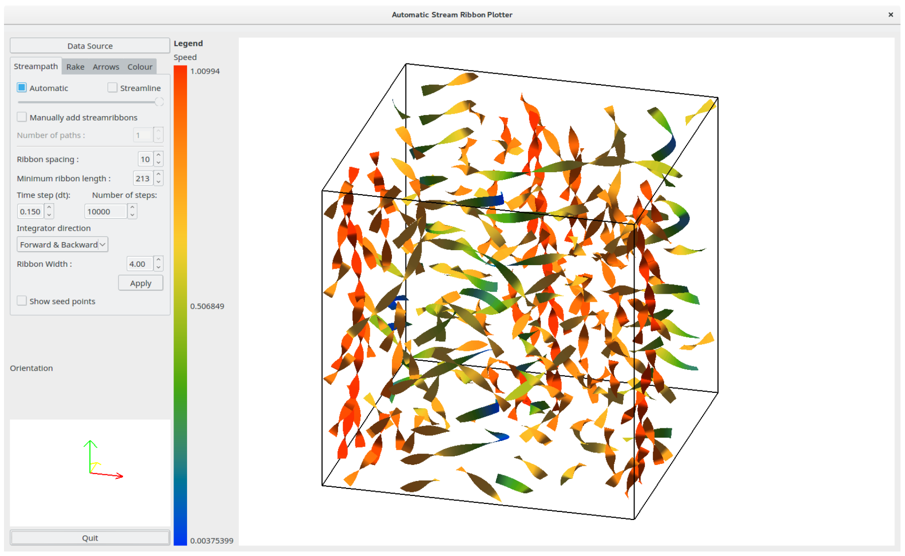
Figure 34: Automatic stream ribbon seeding applied to a RayleighBnard convection flow. The default colourmap is used and colour is mapped to helicity magnitude. The legend has been manually rescaled.



[✓ Automatic] Streamline
Manually add streamribbons
Number of paths : 1
Ribbon spacing : 10
Minimum ribbon length : 213
Time step (dt): 0.150
Number of steps: 10000
Integrator direction: Forward & Backward
Ribbon Width : 4.00
Show seed points

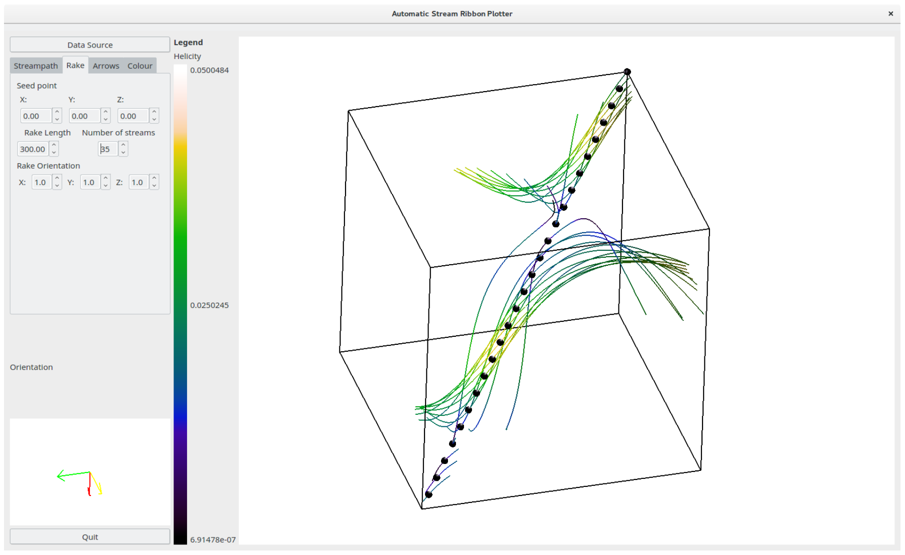
Figure 35: Automatic stream ribbon seeding applied to a 128^3 node tornado simulation. Colour is mapped to speed using a rainbow colourmap.

The available colourmaps, blue to red through white diverging, blue to red through yellow diverging, red - green, Kindlmann and rainbow can be seen in



[✓ Automatic] Streamline
Manually add streamribbons
Number of paths : 1
Ribbon spacing : 10
Minimum ribbon length : 213
Time step (dt): 0.150
Number of steps: 10000
Integrator direction: Forward & Backward
Ribbon Width : 4.00
Show seed points

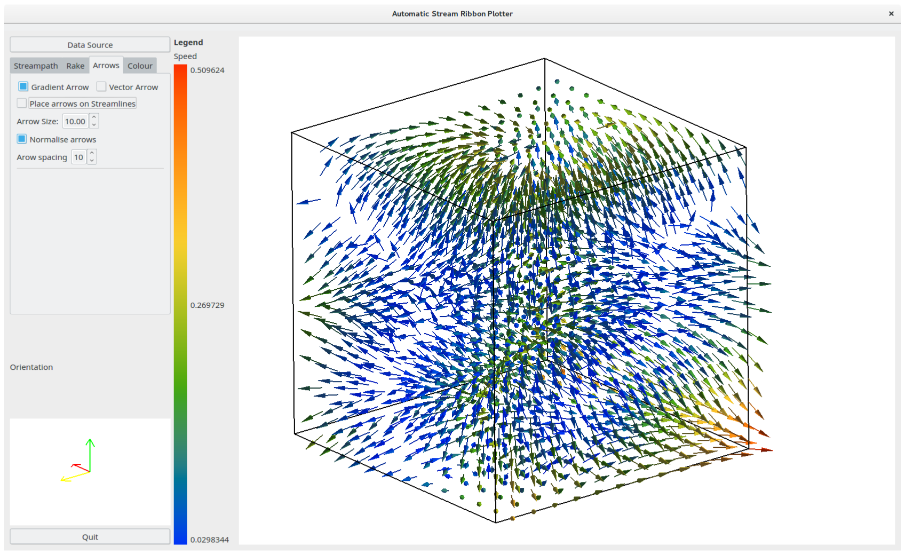
Figure 36: Automatic stream ribbon seeding applied to a 128^3 node ABC simulation. Colour is mapped to speed using a rainbow colourmap.



Seed point
X: 0.00 Y: 0.00 Z: 0.00
Rake Length: 300.00
Number of streams: 35
Rake Orientation
X: 1.0 Y: 1.0 Z: 1.0

Figure 37: A streamline diagonal rake through a 128^3 node ABC simulation. Seeding locations are shown as black spheres. Colour is mapped to helicity using the Kindlmann colourmap.

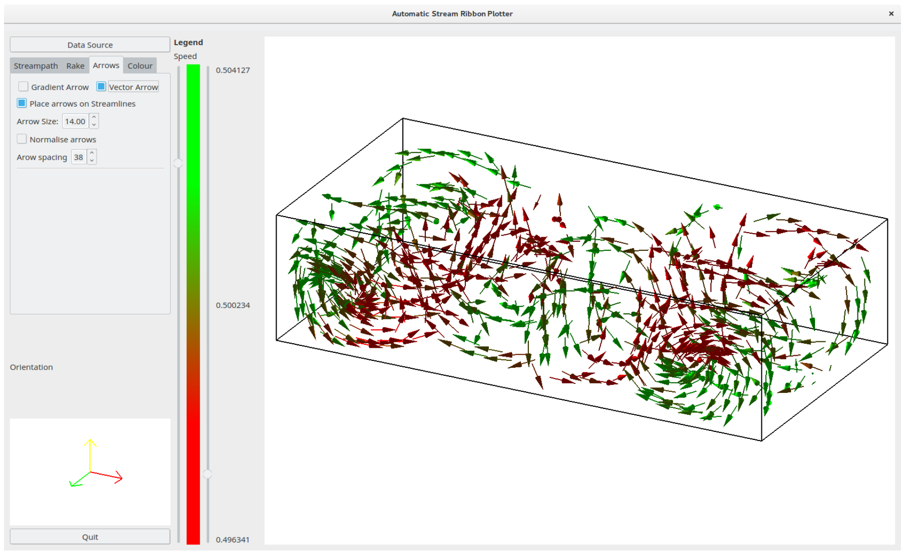
figures 34, 30 39, 37 and 38 respectively.



☑ Gradient Arrow ☐ Vector Arrow
☐ Place arrows on Streamlines
Arrow Size: 10.00
☑ Normalise arrows
Arrow spacing 10

Speed
0.509624
0.269729
0.0298344

Figure 38: Speed gradient arrows applied to a 128^3 node tornado simulation. Arrows are normalised and are spaced at every 10^{th} node. Colour is mapped to speed using the rainbow colourmap.



☐ Gradient Arrow ☑ Vector Arrow
☑ Place arrows on Streamlines
Arrow Size: 14.00
☐ Normalise arrows
Arrow spacing 38

Speed
0.504127
0.500234
0.496341

Figure 39: Velocity arrows placed on streamlines in a RayleighBnard convection flow. Streamline seeding locations were generated using the automated algorithm and filtered. Colour is mapped to speed using the Green-Red colourmap.

To test that the file read is performing correctly and that vector fields are correct, the RayleighBnard convection flow was imported into both the software developed for this project and ParaView, a commercial application [Kit00] to compare results. The results, which can be seen in figure 40, show a very similar likeness verifying the software.

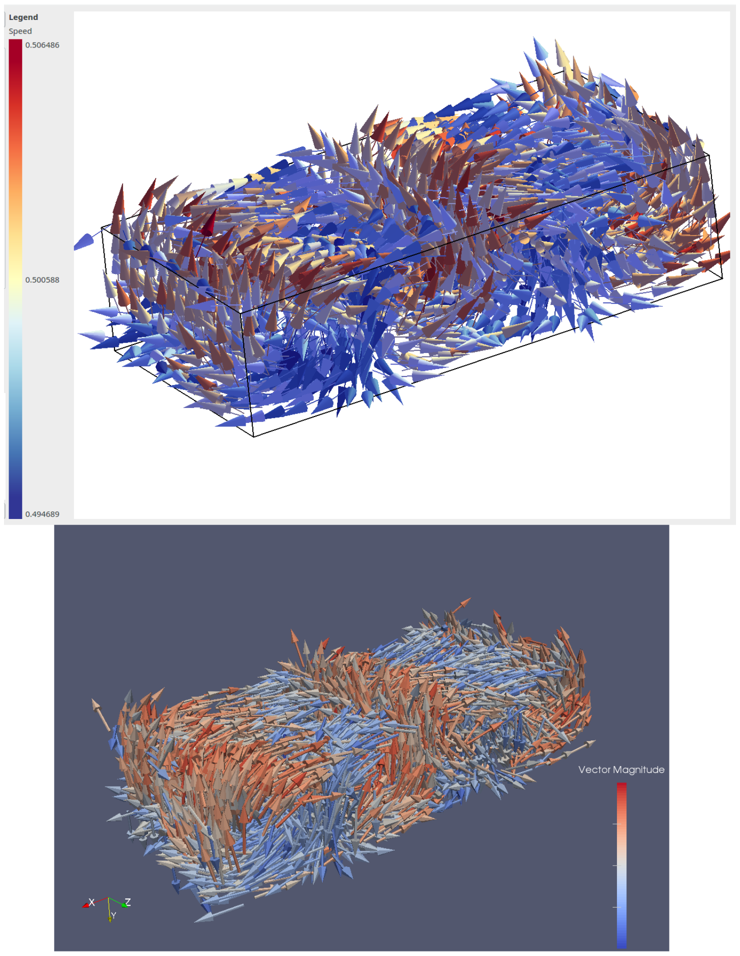


Figure 40: A top and bottom comparison of a RayleighBnard convection flow rendered in the software developed for this project (top) and ParaView (bottom).

A similar test was carried out on the tornado dataset. The function was loaded into ParaView and a seed point added to create a stream ribbon. The same seed point was entered into the software created for this project to compare the results. Figure 41 shows a top and bottom comparison. The results are broadly similar validating the stream ribbon calculation method, the stream line integration method and the tornado dataset generation.

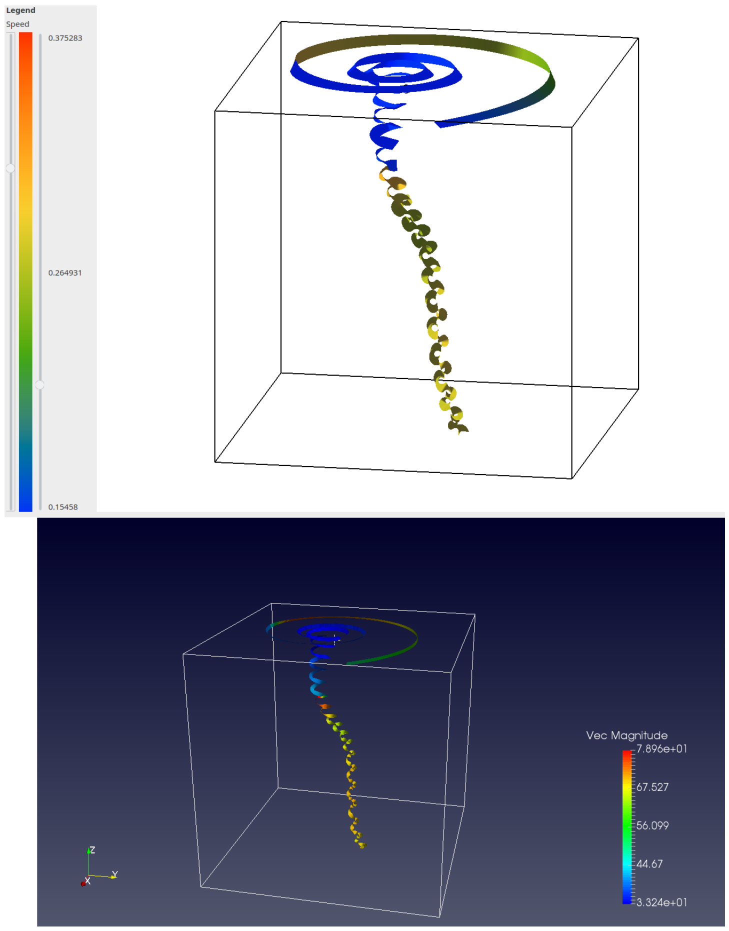


Figure 41: A top and bottom comparison of a single stream ribbon seeded in the 128^3 node tornado simulation. The top image was rendered in the software developed for this project and the bottom rendered in ParaView.

7.2 Performance

A noticeable performance issue is seen with the automatic seeding algorithm on larger datasets. This is exasperated by increasing the minimum length allowed for a ribbon. This is because there are more rejected stream ribbons meaning that there are more seed candidates to investigate. Running the algorithm on a 128^3 node dataset would take over ten minutes on the development PC. Using a smaller dataset rectifies this due to the cubic nature of the flow domain.

Improvements could also be made by using multithread optimizations.

The software shows signs of limitation when rendering many vertices such as vector arrows for every vertex in a 128^3 node dataset. A slow juddering is experienced instead of a smooth user experience when trying to interact with the visualization. This highlights some improvements that could be made to the way the software renders the 3D OpenGL pane. Alternatively it could be rectified with the use of a more powerful dedicated graphics card. The same affect is seen when rendering many vertices within the established flow visualization tool ParaView.

A limitation with the filtering of the automated seeded stream ribbon has also been identified. When the most helical parts of a vector field are on the outside of the domain, the stream ribbons here would be filtered last. This would obscure the flow features in the centre of the flow domain, which would be the first to be filtered.

Outside of the performance issues outlined, the software and the algorithm preform effectively.

8 Conclusions

The first aim of this project was to create a novel stream ribbon seeding algorithm. Stream ribbons are ribbon representations that run at a tangent to the velocity at any point in a vector field. Stream ribbons are most effective demonstrating the helicity of a vector field by their twisting. This aim was met with an algorithm created that prioritises seeding points in order of greatest helicity magnitude. Stream ribbons must be a minimum distance from other ribbons and must meet a minimum length criteria set as a user option. Visualizations of the algorithm results can be seen in this document, however validation from a domain expert is absent.

The second aim was to create a software application capable of visualizing the algorithm which was also met. A C++ based program was created using the Qt framework and OpenGL API. Visualizations produced by the software can also be seen in this document.

A comparison with visualizations created using ParaView [Kit00] show similar results validating the software. Although ParaView has many additional features, an automatic seeding option is unavailable.

Overall the project can be deemed to be a success, however there are some issues with the performance of the algorithm on larger datasets, especially when a longer minimum stream ribbon length is stipulated. A limitation with the filtering has also been identified when the most helical parts of a vector field are on the outside of the domain.

The software also shows signs of limitation when rendering many vertices, a slow juddering is experienced instead of a smooth user experience. The software would also benefit from extensive user testing to identify any bugs.

A key objective of this project was for the author to learn how to use the technologies used for creating computerised data visualization. By performing this project the author has had experience with C++, Qt and OpenGL all of which they were previously unfamiliar with. Although there is more to be learnt about these technologies, this project has created a good introduction.

A video demonstration of the software is available along with other resources produced for the project at the following web address:

`http://cs-ugrad.swan.ac.uk/849119/`

9 Further Work

The first priority to continue this project would be to seek the view of a domain expert in order to validate the algorithm produced. With feedback the algorithm can be tuned to display visualizations that would be of benefit to domain experts. The testing of the algorithm on simulations that are less theoretical and more practical, such as CFD data as seen in figure2, could also benefit. In order to do this, the load data functions will need to be extended to enable a greater variety of file types to be read.

The software could also be extended to enable handling of data on non Cartesian grids, such as polar coordinates or tetrahedron grids. Tetrahedron grids are often used in CFD simulations therefore to implement this option would be beneficial. The software could also be expanded to allow for mapping of different scalar values. CFD simulations generate many scalar values for a flow simulation such as temperature and pressure, the ability to map colour to these scalar values would also be beneficial. Using the scalar values as seeding points using a similar algorithm to the one outlined in this project could also yield interesting results.

Extending the software to accommodate time dependant data is another option to extend the project. This would allow for analysis of streak lines as well as streamlines. Animations would then most likely be required to display the data.

Further improvements could be made to the software such as using multi-threading capabilities to speed up calculations of field attributes and to reduce the time taken to run the automatic stream ribbon seeding algorithm on larger flow domains. A HCI appraisal of the software user interface is another example of further work that could be carried out.

The algorithm used for the automatic seeding of the stream ribbons could also be expanded. Stream ribbon similarity could be mapped in order to create an index, similar to the streamline similarity concept proposed by Chen et al. in [CCK07]. The aim would be to represent the whole flow domain with the minimum possible number of stream ribbons to avoid occlusion problems.

10 Acknowledgements

I would like thank my supervisor Robert S. Laramee for the guidance and support through the learning process of this masters thesis.

References

[Arn11] Arnuphaptrairong T. Top ten lists of software project risks: Evidence from the literature survey. InProceedings of the International MultiConference of Engineers and Computer Scientists 2011 Mar (Vol. 1, pp. 1-6).

[BS08] Blanchette J, Summerfield M. C++ Gui Programming with Qt 4, Second Edition. Prentice Hall Press. 2008

[BT07] Borland D, Taylor II RM. Rainbow color map (still) considered harmful. IEEE computer graphics and applications. 2007 Mar 1;27(2):14-7.

[CCK07] Chen Y, Cohen J, Krolik J. Similarity-Guided Streamline Placement with Error Evaluation. IEEE Transactions on Visualization and Computer Graphics. Volume 13, No. 6. 2007 0-201-14192-2.

[DMG+04] Doleisch H, Mayer M, Gasser M, Wanker R, Hauser H. Case Study: Visual Analysis of Complex, Time-Dependent Simulation Results of a Diesel Exhaust System. In: Data Visualization, Proceedings of the 6th Joint IEEE TCVGEUROGRAPHICS Symposium on Visualization (VisSym 2004). 2004, p. 916.

[DMH04] Doleisch H, Muigg P, Hauser H. Interactive visual analysis of hurricane isabel with SimVis. In IEEE Visualization 2004.

[Dol07] Doleisch H. Simvis: Interactive Visual Analysis of Large and TimeDependent 3D Simulation Data. In: WSC 07: Proceedings of the 39th Conference on Winter Simulation. Piscataway, NJ, USA: IEEE Press. ISBN 1-4244-1306-0; 2007, p. 71220.

[EMcLL+11] Edmunds M, McLoughlin T, Laramee RS, Chen G, Zhang E, Max N. Automatic Stream Surface Seeding. EUROGRAPHICS 2011

[FG98] Fuhrmann A, Grller E. Real-time techniques for 3D flow visualization. InProceedings of the conference on Visualization'98 1998 Oct 18 (pp. 305-312). IEEE Computer Society Press.

[JL97] Jobard B, Lefer W. Creating evenly-spaced streamlines of arbitrary density. InVisualization in Scientific Computing97 1997 (pp. 43-55). Springer Vienna.

[Kit00] Kitware Inc. ParaView. Kitware Inc. http://www.paraview.org/ [September 2016].

[KRC02] Kindlmann G, Reinhard E, Creem S. Face-based luminance matching for perceptual colormap generation. InProceedings of the conference on Visualization'02 2002 Oct 27 (pp. 299-306). IEEE Computer Society.

[Lar10] Laramee RS. Bobs Project Guidelines: Writing a Dissertation for a BSC. in Computer Science. Innovation in Teaching and Learning in Information and Computer Sciences, 10(1):4354, February 2011. (A publication of the UK Higher Education Academy (HEA), available online).

[Lar10a] Laramee RS. Bobs Concise Coding Conventions (C3). Advances in Computer Science and Engineering (ACSE). 2010 Feb;4(1):23-6.

[LJH03] Laramee RS, Jobard B, Hauser H. Image Space Based Visualization of Unsteady Flow on Surfaces. In: Proceedings IEEE Visualization 03. IEEE Computer Society; 2003, p. 1318.

[Mat84] MathWorks. MATLAB. MathWorks Inc. http://uk.mathworks.com/products/matlab/ [September 2016].

[McLL+09] McLoughlin T, Laramee RS, Peikert R, Post FH, Chen M. Over Two Decades of Integration Based, Geometric Flow Visualization. InComputer Graphics Forum 2010 Sep 1 (Vol. 29, No. 6, pp. 1807-1829). Blackwell Publishing Ltd.

[MCG94] Max N, Crawfis R, Grant C. Visualizing 3D velocity fields near contour surfaces. In Proceedings of the conference on Visualization'94 1994 Oct 17 (pp. 248-255). IEEE Computer Society Press.

[Mor09] Moreland K. Diverging color maps for scientific visualization expanded. Advances in Visual Computing. 2009 Jul 1;5876.

[NCE91] Nord H, Chambe-Eng E. Qt. The Qt Company. https://www.qt.io [September 2016]

[Para07] Default Color Map - ParaQ Wiki. June 2007. http://www.paraview.org/ParaView/index.php/Default_Color_Map [September 2016]

[PGL12] Peng Z, Geng Z, Laramee RS. Design and Implementation of a System for Interactive High-Dimensional Vector Flow Visualization. Computer Modeling: New Research 2012.

[PGN+14] Peng Z, Geng Z, Nicholas M, Laramee RS, Croft N, Malki R, Masters I, Hansen C. Visualization of Flow Past a Marine Turbine: The Information-Assisted Search for Sustainable Energy. Computing and Visualization in Science, Vol 17, No 6, December 2014, pages 89-103.

[Ree16] Rees DG, Automatic Stream Ribbon Seeding. CSCM10 - Computer Science Project Research Methods. Swansea University department of computer science. May 2016.

[Roth00] Roth, Martin. Automatic extraction of vortex core lines and other line type features for scientific visualization. Hartung-Gorre, 2000.

[Sma94] Smart J. wxWidgets: cross platform GUI library. https://www.wxwidgets.org/ [September 2016]

[Spi98] Spitzak B. Fast Light Toolkit (FLTK). http://www.fltk.org/ [September 2016]

[Sto04] Storer J. JUCE (Jules' Utility Class Extensions). ROLI Ltd. https://www.juce.com/ [September 2016]

[Str15] Bjarne Stroustrup. The C++ Programming Language Fourth Edition. Addison-Wesley. 2015

[SWH15] Sellers G, Wright RS, Haemel N. OpenGL SuperBible Seventh Edition. Addison-Wesley. 2015

[Tec81] Tecplot Inc. Tecplot 360 http://www.tecplot.com [September 2016].

[Tel08] Telea AC. Data Visualization Principles and Practice. A K Peters Ltd. 2008

[Tsi] Tsinober A. Fundamental and conceptual aspects of turbulent flows - Helicity. Slides from Imperial College London. Available online: http://www3.imperial.ac.uk/portal/pls/portallive/docs/1/10733697.PDF [May 2016]

[TW03] Telea A, Wijk JJ. 3D IBFV: Hardware-accelerated 3D flow visualization. InProceedings of the 14th IEEE Visualization 2003 (VIS'03) 2003 Oct 22 (p. 31). IEEE Computer Society.

[YKP05] Ye X, Kao D, Pang A. Strategy for seeding 3D streamlines. InVisualization, 2005. VIS 05. IEEE 2005 Oct 23 (pp. 471-478). IEEE.

11 Apendix

11.1 Minutes of Meetings

Minutes of Meeting: Bob, Dylan

Date: 08 Feb '16
Start time: 12:00
End time: 12:10

Date and time of next meeting: TBA

Topics discussed:
– Ian Masters
– First document submission 10 Mar '16 (Confirmed)
– Bob teaches software engineering II for undergraduates Tuesdays, Fridays 10:00 AM

Progress since last meeting:
– Masters project chosen

TODO (for next meeting):
– Dylan: Read Bob's Minutes of Meeting Protocol
– Dylan: Place minutes of meeting on web server
– Dylan: Read "Visualisation of flow past a marine turbine" paper
– Bob: Get in touch with Ian Masters for marine turbine data
– Dylan: Read Bob's project guidelines

Minutes of Meeting: Bob, Donal, Dylan

Date: 25 Feb '16
Start time: 11:00
End time: 11:50

Date and time of next meeting: Donal: 29 Feb '16 11:00 with Nai, 2 March '16 11:00

Topics discussed:
– Matt Edmunds
– First document submission 10 Mar '16

Progress since last meeting:
– Minutes of Meeting Protocol
– Dylan read "Visualisation of flow past a marine turbine" paper
– Both read Bob's project guidelines
– Bob wrote to Ian for turbine data

TODO (for next meeting):
– Donal Put minutes of meeting online from first meeting
– Dylan get copy of Matt's software
– Dylan read preface and chapter one of Qt book
– Dylan read chapter on vector visualization
– Donal look at data visualization book
– Donal read "Visualization of Biomolecular Structures: State of the Art"
– Both start writing related work section of thesis - 1/2 - 1 pages per assignment

Minutes of Meeting: Bob, Donal, Dylan, Dimitrios

Date: 03 Mar 2016
Start time: 14:30
End time: 16:00

Date and time of next meeting: Donal + Dimitrios: 07 Mar 2016 11:00 with Nai, 09 March 2016 11:00

Topics discussed:
– Tecplot
– 10 March Initial Document Deadline
– Rich's phone number - 07852377314

Progress since last meeting:
– Donal put minutes online
– Dylan read chapter on vector field vis. and chapter 1 of QT
– Donal read more survey papers

TODO (for next meeting):
– Rees - Matt's software is avilable for download. Add "software" to Bob's URL
– Rees - read "Design and Implemnetation of a System for Interactive, High-Dimensional Vector
Field Visualization" + Sim vis paper
– All: review Bob's project guidelines
– All : look at Markus' marking scheme
– All: Attempt to describe data
– Donal: Strategy for a survey 1) Describe the topics covered in survey, 1-2 paragraphs, describe how the literature is classified or categorised 1-2 paragraphs
– Donal: Ask for a copy of Nai's work-in-progress survey paper. Choose a paper in there for related work
– Dimitrios: Read Bob's project guidelines
– Dimitrios: Read Bob's minutes of meeting protocol

Minutes of Meeting: Bob, Dylan, Donal, Dimitrisl

Date: 03 Mar 2016
Start time: 11:00
End time: 11:45

Date and time of next meeting: 14 March 11:00, 16 March 11:00

Topics Discussed:
–10 March 17 Taught MSc
–17 March 16 Advanced MSc
–Tecplot
–Format, Gro Files

Pregress since last meeting:
–Donal started initial document

–Dylan has copy of Maths software
–Donal got a copy of Nai's survey

Todo
–Submit initial document: Intro,
Advenced topics: Data characteristics, Features-Specifications
–Donal keep asking Nai foa a copy of his software
–Dimitris Read "Visualizations of sensor data from animal movement"
–Dimitris add "data" in lower-case to Bob's url
–Dimitris think about technology choices
–Donal place literatures in chronological order by default

Minutes of Meeting: Bob, Dylan, Donal, Dimitris

---

date: 16 Mar 2016
start time: 11:00
end time: 12:15

date and time of next meeting: Donal 21 March 11:00, TBA

Topics Discussed:
–Advanced MSc deadline tomorrow - Initial document
–Thursday 21st April '16 CSCM10 presentations
–Bob away 04-07 April '16
–Bob away 11-17 April '16

Progress since last meeting:
–Donal, Dylan Initial documents submitted
–Bob gives MDS data to Donal
–Bob gives marine turbine data to Dylan

Todo
–Dimitris Submit initial document
–Dimitris Read "Smooth Graphs for Visual Exploration of Higher-Order State Transitions"
–Dimitris Google "Visualization of time-sensitive data, free tools"
–Dimitris Look at links on Bob's data visualization web page
–Dimitris Send Bob copy of initial document requirements
–Donal Try visualizing MDS data with existing tools
–Donal Add "/data" to Bob's URL to access data
–Dylan Write to Matt Edmunds and Ian Fairley about data
–Dylan Review different versions of Tecplot
–Dylan Read Bloodhound paper

Minutes of Meeting: Bob, Dylan, Donal

---

date: 24 Mar 2016
start time: 15:00
end time: 16:00

date and time of next meeting: 7 April 2016, 21 April otherwise, subject to change

Topics Discussed:
–QT and C++
–QT Creator
–Swansea used PC for sale
–Hexahedra
–Bob's visa to China

Progress since last meeting:
–Dimitrios submitted initial document
–Donal got data
–Donla visualized MDS data set with VMD
–Donal has code to read data
–Donal has Nai's survey paper
–Dylan has marine turbine data
–Dylan looked at Bloodhound paper

Todo
–Donal: show screenshots of MDS data with MVD- upload to web server
–Both have a look at QT Creator tutorials
–Donal: see if you can write a program to read the MDS data and print it to console
–Dylan: think about getting a new used PC
–Dylan: try viewing data in paraview

Minutes of Meeting: Bob, Dylan, Donal

---

date: 28 Apr 2016
start time: 15:00
end time: 15:50

date and time of next meeting: 19 May 2016 15:00 (9 May meeting with Nai for Donal)

Topics Discussed:
–Project fair 12 May 2016, all day
–Bob away next Thursday 5 May

–11 May 2016 interim/final document
–Real problem, real challenges, digital solutions, digital challenges

Progress since last meeting:
–Gave presentations

Todo
–Dylan decide between turbine vis vs streamribbon seeding
–Dylan review Tony's survey paper with a focus on seeding
–Dylan look at Matt Edward's papers
–Cite images
–Donal: ask Nai and Mattieu for more data sets
–Donal: Join us for 11:00 Monday meetings with Nai
–Donal: Read data in Qt
–Donal: Ask Nai for updated survey

Minutes of Meeting: Bob, Dylan, Donal

---

date: 02 Jun 2016
start time: 15:00
end time: 15:50

date and time of next meeting: 16 Jun 2016 15:00 (13 Jun meeting with Nai for Donal)

Topics Discussed:
–Dylan returned computer
–Bob away at Eurovis 2016 6-10 June
–Flow data: 5 jets, ABC flow, Bernard, Marine turbine, square cylinder, tornado
–Real problem, real challenges, digital solutions, digital challenges

Progress since last meeeing:
–Both submitted final documents
–Dylan new PC parts
–Donal has data

Todo
–Both read data and print to console
–Bob's coding conventions - follow them
–Both get a C++ reference book

Minutes of Meeting: Bob, Dylan, Donal, Dimitris

---

date: 16 Jun 2016
start time: 15:00
end time: 15:40

date and time of next meeting: 23 Jun 2016 15:00 (20 Jun meeting with Nai for Donal)

Topics Discussed:
–Bob away 25 June - 2 July - CGI 2016 conference
–QWT
–QCustomPlot
–23 YouTube video
–idvbook.com
–Ptr, Ref, (Copy), Naming conventions

Progress since last meeting:
–Dylan Sally, Betty, ABC data format
–Donal MDS data reader
–Dimitris acquired data

Todo
–Donal XY canvas, one point, N points
–Both read QT chapter on 2D graphics
–Dylan XY canvas, one line segment per vector, N vectors
–Dimitris read in data and print to console
–Dimitris send PDFs of books to Bob (all)
–All have a look at Bob's sample code

Minutes of Meeting: Bob, Dylan, Donal

---

date: 23 Jun 2016
start time: 15:00
end time: 15:50

date and time of next meeting: TBA, 14 July 2016 - tentative

Topics Discussed:
–PhD Studentships
–EU
–Bob at conference next week
–Swansea computer science society

Progress since last meeting:
–Donal - Rendered circles, panning, zooming, colour
–Dylan - Edges on a plane

Todo
–Donal - 1 point per atom trajectory
–Donal - paths (a set of edges)
–Dylan - Arrow glyphs - 2D and 3D
–Dylan - Encode vector magnitude with colour
–Read chapter on 3D graphics
–Donal proof-read Nai's paper - deadline on Monday - CGVC 2016

Minutes of Meeting: Bob, Dylan, Donal
———————————

date: 07 Jul 2016
start time: 15:00
end time: 15:45

date and time of next meeting: 14 July 2016

Topics Discussed:
–KESS Studentships
–Brexit
–Hadling periodric boundaries
–Nai away 15 July - 20 August

Progress since last meeting:
–Donal - 1 point per atom trajectory and paths based using QPainter
–Dylan - Coloured arrows based on a QT colour map (2D)

Todo
–Donal - 1 point per atom trajectory using OpenGL
–Dylan - paths (a set of edges)3D coloured arrows

Minutes of Meeting: Bob, Dylan, Donal
———————————

date: 14 Jul 2016
start time: 15:00
end time: 15:45

date and time of next meeting: 21 July 2016

Topics Discussed:
–Distant supervision
–Nai away until 20 August 2016
–PhD requirements
–Integrating Qt and OpenGL

–www.trentreed.net/topics/cc
–User options between swithing back and forth between Qt and OpenGL

Progress since last meeting:
–Donal - Drawing a triangle in OpenGL
–Dylan - Drawing a cone in OpenGl

Todo
–Donal - 1 point per atom trajectory using OpenGL
–Dylan - paths (a set of edges)3D coloured arrows

Minutes of Meeting: Bob, Dylan, Donal

---

date: 21 Jul 2016
start time: 15:00
end time: 16:00

date and time of next meeting: 28 July 2016

Topics Discussed:
–Bob away until 3 -11 August 2016
–Bob's Skype name is rlaramee

Progress since last meeting:
–Donal - Rendering of atom positions and paths with OpenGL
–Donal - Atoms as paths
–Dylan - One arrow glyph

Todo
–Dylan - Test out Skype
–Donal - User Option: Atom positions as circles
–Both - Read chapter 4 of the OpenGL superbible : Math for 3D graphics
–Donal - Store the data in memory
–Both - Read chapter 15: Debugging and stability of OpenGL superbible
–Dylan - Two arrow glyphs
–Dylan - Draw a grid of points
–Donal - Ask Matthieu what the term 'residue' means in this case
Also ask him how he would cluster trajectories

Minutes of Meeting: Bob, Dylan, Donal

---

date: 28 Jul 2016
start time: 15:00
end time: 16:00

date and time of next meeting: 11 Aug 2016, 15:00

Topics Discussed:
–Bob away until 3 -11 August 2016

Progress since last meeting:
–Donal - Exchanged emails with Matthieu
–Donal - Getting data loaded into memory
–Dylan - Zooming of viewport - rendering all atoms -
initial work on colour legend - collection of arrow glyphs
–Both - Read chapter 4 of OpenGL Superbible

Todo
–Donal - User Option: Atom positions as circles, reading of data into memory
–Dylan - Streamlines
–Donal - Colour mapping
–Both - Read chapters 1 and 2 of OpenGL Superbible
–Dylan - Read chapter 6 of Data Visualization Principles and Practice

Minutes of Meeting: Bob, Dylan, Donal

---

date: 12 Aug 2016
start time: 15:00
end time: 16:00

date and time of next meeting: 18 Aug 2016, 15:00

Topics Discussed:
–Qvectors
–2 samples/dimension
–Vector field can be normalised for streamlines
–Over Two Decades of Integration-Based, Geometric Flow Visualization
–Sci-hub

Progress since last meeting:
–Donal - Demo video
–Donal - Colour mapping
–Both - Read OpenGL Superbible chapters 1 and 2
–Dylan - Streamlines, Euler, Runge-Kutta2 and 4
–Dylan - User option dt

Todo
–Donal - Ask for a copy of Nai's video
–Donal - Try out Nai's dataset

–Donal - Add zooming and panning
–Donal - Try to find how much memory is available in vertex buffer
–Donal - Ask Nai for a copy of his code and Doxygen
–Donal - Colour Legend
–Donal - Ask Matthieu if he thinks your video is correct (as a link)
–Both - Read the Qt chapter on container clases
–Dylan - Introduce a vector class with a normalize function
–Dylan - Streamribbons

Minutes of Meeting: Bob, Dylan, Donal

---

date: 18 Aug 2016
start time: 15:00
end time: 16:00

date and time of next meeting: 25 Aug 2016, 15:00

Topics Discussed:
–Clearing
–Jet lag
–2GB of memory on graphics card
–VTK
–QElapsedTimer
–Bob away 31 Aug - 7 Sept

Progress since last meeting:
–Donal - Lipid visualization with Nai's data
–Bob sent email to Nai
–Donal - Colour legend
–Donal - Read chapter on container classes
–Dylan - Streamribbons

Todo
–Donal - Zooming and panning (try w/o Nai)
–Donal - Set up new laptop
–Dylan - Read Qt chapter on container classes
–Dylan - Try to finish streamribbons

Minutes of Meeting: Bob, Dylan, Donal

---

date: 25 Aug 2016
start time: 15:00
end time: 16:00

date and time of next meeting: 08 Sep 2016, 15:00

Topics Discussed:
–Texture advection on streamsurfaces
–VMD mailing list
–Bob away 31 Aug - 7 Sept

Progress since last meeting:
–Donal - Rotation, zooming and panning
–Dylan - First implementation of streamribbons

Todo
–Both - Watch advanced visualization of electroencephalography (EEC) data
–Dylan - Add lighting options
–Dylan - User option - maximum length
–Dylan - Add a seeding rake
–Dylan - Have a look at "Automatic extraction of vortex core lines and other
–Dylan - User option: Vector velocity at each point
–Dylan - User option: Gradient vector at each point
–Donal - Zooming and panning with buffers
–Donal - Handle boundary issues
–Both - Propose thesis outline (show to Bob at next meeting)

Minutes of Meeting: Bob, Dylan, Donal

---

date: 08 Sep 2016
start time: 15:00
end time: 16:30

date and time of next meeting: 15 Sep 2016, 15:00

Topics Discussed:
–Trip to Portugal
–Visible lunch on Monday at 13:00 - 4th floor kitchen
–Fire alarm
–Add "software" to Bob's URL (lowercase)
–Illuminated lines
–ABC data
–Streamlines similarity video

Progress since last meeting:
–Donal - Buffered rendering - quick, panning, zooming, rotation, more colour mapping
–Donal - Handled boundaries
–Dylan - Ribbons with shading, forward and backward integration

–Dylan - Vector and gradient glyphs, colour map clamping
–Dylan - Seeding based on helicity magnitude

Todo
–Donal - Experiment with shading
–Donal - Spheres user option
–Donal - Trajectories as tubes (ask Nai how he's rendering spheres)
–Both - Cite and enhance introductory/final document
–Dylan - User options for lighting and ambient light
–Dylan - White background
–Dylan - Focus on aesthetics - try out Alex Telea
–Dylan - Try out illuminated lines?
–Both look at colour map from chapter 5 of Telea's book

Minutes of Meeting: Bob, Dylan, Donal

---

date: 15 Sep 2016
start time: 15:00
end time: 16:05

date and time of next meeting: 22 Sep 2016, 15:00

Topics Discussed:
–CoS DTC
–Animate streamline evolution: Current cell, current vectors at cell vertices, current length

Progress since last meeting:
–Donal - Next video
–Donal - Point sprites, interactive light source and ambient light
–Donal - Animation frame rates
–Dylan - Normals and ambient lighting, interactive light sources
–Dylan - Version of seeding algorithm

Todo
–Dylan - Backup copy of offer letter
–Donal - Test out/demo both lipid and protein data
–Both - Thesis writing
–Donal - Colour map selection, illuminated lines?
–Dylan - Double check integrator for smoothness
–Dylan - User option - render seed points as spheres

Minutes of Meeting: Bob, Dylan, Donal

---

date: 22 Sep 2016
start time: 15:00
end time: 16:00

date and time of next meeting: 29 Sep 2016, 15:00 (optional)

Topics Discussed:
–Distractions
–Design chapter
–Implementation: list of features + images to support + link to video demo
–Doxygen as appendix, but code only in digital form (pdf on web page)
–Open Broadcaster
–The Lorenz attractor

Progress since last meeting:
–Donal - Filtering by path length, curvature, velocity, 30+ colour maps
–Donal - Next video
–Dylan - Next video, smoother streamlines
–Both - Dissertation writing

Todo
–Dylan - Try adding/modifying light sources
–Dylan - 50% Sally/50% Betty - maybe a third data set
–Both - Write thesis and submit

11.2 Class Documentation

The code documentation created using Doxygen can be found in this subsection.

chooseMap Class Reference

Signals

void **mapNo** (int choice)
Signal to emit mapNo.

Public Member Functions

chooseMap (QWidget *parent=0)

~chooseMap ()

Constructor & Destructor Documentation

chooseMap::~chooseMap ()

destructor

Member Function Documentation

void chooseMap::mapNo (int *choice*)[signal]

Signal to emit mapNo.

Parameters:

choice	is the map number

clearPoint Class Reference

Public Member Functions

clearPoint ()
constructor
clearPoint (int x, int y, int z)
constructor setting parameters
void **setSize** (int x, int y, int z)
set Size of dataset
void **setClearance** (int clearance)
set Clearance value
void **setPoint** (int x, int y, int z)\\
setPoint as occupied
void **setPoint** (float x, float y, float z)
setPoint as occupied if location is float
void **clearTemp** ()
resets temporary positions without storing in permanent
bool **checkSetPoint** (float x, float y, float z)
checkSetPoint sets a point as occupied and returns if available
void **newRibbon** ()
indicates a new ribbon is started, converts temporary points to permanent and resets temporary points
bool **isClear** (int x, int y, int z)
Checks if location is clear.
bool **isClear** (float x, float y, float z)
Checks if location is clear.
void **reset** ()
resets all data to clear
int **size** ()
returns length of object

Constructor & Destructor Documentation

clearPoint::clearPoint (int *x*, int *y*, int *z*)

constructor setting parameters

Parameters:

x,*y*,*z*	sizes

Member Function Documentation

bool clearPoint::checkSetPoint (float *x*, float *y*, float *z*)

checkSetPoint sets a point as occupied and returns if available

Parameters:

x,*y*,*z*	coordinates

Returns:

if position is available or not

bool clearPoint::isClear (int *x*, int *y*, int *z*)

Checks if location is clear.

Parameters:

x,*y*,*z*	coordinates

Returns:

if location is clear

bool clearPoint::isClear (float *x*, float *y*, float *z*)

Checks if location is clear.

Parameters:

x,*y*,*z*	coordinates

Returns:

if location is clear

void clearPoint::setClearance (int *clearance*)

set Clearance value

Parameters:

c*learance*	value

void clearPoint::setPoint (int x, int y, int z)

setPoint as occupied

Parameters:

x,y,z	coordinates

void clearPoint::setPoint (float x, float y, float z)

setPoint as occupied if location is float

Parameters:

x,y,z	coordinates

void clearPoint::setSize (int x, int y, int z)

set Size of dataset

Parameters:

x,y,z	sizes

int clearPoint::size ()

returns length of object

Returns:

length of object

ColourMap Class Reference

Public Member Functions

ColourMap ()
Constructor.
ColourMap (int tex)
Constructor setting map.
float(* **getMap** ())[3]
Constructor setting map.
int **getSize** ()
void **setMap** (int tex)
sets Map

Constructor & Destructor Documentation

ColourMap::ColourMap (int *tex*)

Constructor setting map.

Parameters:

m*ap*	number

Member Function Documentation

float(* ColourMap::getMap ())[3]

Constructor setting map.

Returns:

Map rgb array

int ColourMap::getSize ()

Returns:

size of map

void ColourMap::setMap (int *tex*)

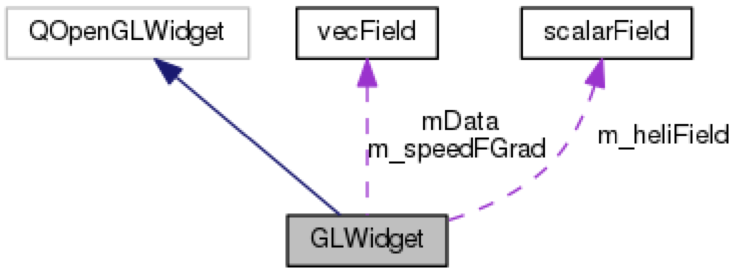
sets Map

Parameters:

m*ap*	number

GLWidget Class Reference

Collaboration diagram for GLWidget:



Public Slots

void **dSw** (bool sw1)
Slot indicating that data is present to allow drawing.
void **resetRot** (float xORot, float yORot)
Slot for rotatation reset from orientation widget.
void **resetZoom** (float zoomFactor)
Slot to reset zoom.

Signals

void **rotate** (float xRot, float yRot, float zRot)
rotatation values to be emitted to orientation widget

Public Member Functions

GLWidget (QWidget *parent=0)
Constructor.
void **initializeGL** ()
initialize OpenGL sets up backround
void **paintGL** ()

Draws scene, sets positions and colours and draws according to variable switches.

void **resizeGL** (int w, int h)

Behaviour on resizing GL pane.

Public Attributes

float **xSize**

Sizes of data field.

float **ySize**

float **zSize**

QVector3D **m_seed**

seed position

bool **m_dSw** = false

Switch for allowing draw.

bool **m_line** = false

Switch for drawing stream lines.

bool **m_VArrows** = false

Switch for drawing vector arrows.

bool **m_GArrows** = false

Switch for drawing gradient arrows.

bool **m_streArrow** = false

Switch for drawing arrows on stream line.

bool **m_normArrow** = false

Switch for drawing normalized arrows.

float **m_aScale** = 1

Switch for setting arrow scale.

vecField * **mData**

Contains vector field vectors for drawing arrows and colour mapping.

vecField * **m_speedFGrad**

Contains gradient field vectors for drawing arrows.

int **xSpace** = 10

Sets spacing of arrow glyphs on x axis.

int **ySpace** = 10

Sets spacing of arrow glyphs on y axis.

int **zSpace** = 10

Sets spacing of arrow glyphs on z axis.

float **minScal**

Sets maximum and minimum values of colour scale.

float **maxScal**

scalarField **m_heliField**

values of helical field

int **m_colourHeli** = 0

Sets what colour is mapped to.

QVector< QList< QVector3D > > **m_ribbon**

m_ribbon

QVector< QVector3D > **mRibSeed**

Ribbon seed locaions.

GLdouble **mSphereRad** = 0.25f

Size of seed position sphere.

bool **mDrawSeed** = false

Switch for drawing seed positions.

float **mL0X** = 0.5f

Light 0 x position.

float **mL0Y** = 0.5f

Light 0 y position.

float **mL0Z** = 0.1f

Light 0 z position.

float **mL1X** = -0.5f

Light 1 x position.

float **mL1Y** = 0.5f

Light 1 y position.

float **mL1Z** = 0.3f

Light 1 z position.

float **mL2X** = 0.2f

Light 2 x position.

float **mL2Y** = -0.5f

Light 2 y position.

float **mL2Z** = 0.7f

Light 2 z position.

float **mL3X** = 0.2f

Light 3 x position.

float **mL3Y** = -0.5f

Light 3 y position.

float **mL3Z** = 0.7f

Light 3 z position.

GLfloat **mlight0_position** [3]

Light 0 position array.

GLfloat **mlight1_position** [3]

Light 1 position array.

GLfloat **mlight2_position** [3]

Light 2 position array.

GLfloat **mlight3_position** [3]

Light 3 position array.

float **mAmbient** = 0.05

Indicates ambient light value.

float **mDiffuse** = 0.5

Indicates diffuse light value.

float **mSpecular** = 0.6

Indicates specular light value.

GLfloat **mlight_diffuse** [4]

Indicates diffuse light conditions.

GLfloat **mlight_specular** [4]

Indicates specular light conditions.

GLfloat **mlight_ambient** [4]

Indicates ambient light conditions.

bool **mL0** = true

Indicates the use of lamp 0.

bool **mL1** = true

Indicates the use of lamp 1.

bool **mL2** = true

Indicates the use of lamp 2.

bool **mL3** = true

Indicates the use of lamp 3.

int **mTextureNo**

Sets colour map.

int **mPCRibbons** = 100

Sets number of ribbons to draw from filter.

int **mNoOfRibbs** = 100

Sets maximum number of ribbons to draw.

Protected Member Functions

void **mousePressEvent** (QMouseEvent *event)

Tracks mouse inputs on widget, left button for panning right button for rotating.

void **wheelEvent** (QWheelEvent *event)

Tracks mouse wheel inputs on widget for zoom.

void **mouseMoveEvent** (QMouseEvent *event)

Tracks mouse movements.

legGLWidget Class Reference

Signals

void **mapNum** (int)
Sends map number returned from choose dialog.

Public Member Functions

legGLWidget (QWidget *parent=0)
Constructor.
void **initializeGL** ()
Behaviour on initializing GL pane.
void **paintGL** ()
Behaviour on initializing GL pane.
void **resizeGL** (int w, int h)
Colours widget according to chosen map/texture.

Public Attributes

int **mTextureNo**
Textur map number.

load Class Reference

Signals

void **draw** (bool sw)
Signal to switch when data is available to draw.

Public Member Functions

load (**vecField** *pData, QWidget *parent=0)
Constructor, takes in array to fill with loaded data.
~load ()
Deconstructor.

MainWindow Class Reference

Public Member Functions

MainWindow (QWidget *parent=0)
Constructor.
~**MainWindow** ()
Destructor.

ordered Class Reference

Public Member Functions

ordered ()
Constructor.

void **set** (int x, int y, int z, float val)
sets new value at coordinates creates QVector3D of coordinates and insetrts into map with val as key

int **length** ()
length of

float **getFScal** (int pos)
getter for pos-th highest svalar value

float **getLScal** (int pos)
getter for pos-th lowest scalar value

float **getUCS** ()
Getter for the upper value for legend scale value would be highest scalar value unless it is significantly higher that the value at the 99.9% position or will return 99.9% position to filter stray high values.

float **getLCS** ()
Getter for the lower value for legend scale value would be lowest scalar value unless it is significantly lower that the value at the 99.9% position or will return 99.9% position to filter stray low values.

void **clear** ()
clear all values for new data set

float **getScal** (int x, int y, int z)
Getter for scalar value at coordinates.

QVector3D **getFQVec** (int pos)
getter for pos-th coordinates according to highest scalar value

QVector3D **getLQVec** (int pos)
getter for pos-th lowest coordinates according to scalar value

void **convert** (QVector< QVector3D > *orderFast)
Coppys map into array in same order for quicker indexed access.

Member Function Documentation

void ordered::convert (QVector< QVector3D > * *orderFast*)

Coppys map into array in same order for quicker indexed access.

Parameters:

orderFast	array passed in to fill

QVector3D ordered::getFQVec (int *pos*)

getter for pos-th coordinates according to highest scalar value

Parameters:

pos	from end of map

Returns:

Coordinates of scalar value at posth position from end

float ordered::getFScal (int *pos*)

getter for pos-th highest svalar value

Parameters:

pos	from start of map

Returns:

Scalar value at posth position

float ordered::getLCS ()

Getter for the lower value for legend scale value would be lowest scalar value unless it is significantly lower that the value at the 99.9% position or will return 99.9% position to filter stray low values.

Returns:

Lower scale value

QVector3D ordered::getLQVec (int *pos*)

getter for pos-th lowest coordinates according to scalar value

Parameters:

pos	from end of map

Returns:

Coordinates of scalar value at posth position from end

float ordered::getLScal (int *pos*)

getter for pos-th lowest scalar value

Parameters:

p*os*	from end of map

Returns:

Scalar value at posth position from end

float ordered::getScal (int *x*, int *y*, int *z*)

Getter for scalar value at coordinates.

Parameters:

x,*y*,*z*	coordinates

Returns:

scalar value at coordinates

float ordered::getUCS ()

Getter for the upper value for legend scale value would be highest scalar value unless it is significantly higher that the value at the 99.9% position or will return 99.9% position to filter stray high values.

Returns:

Upper scale value

int ordered::length ()

length of

Returns:

length of scalar field

void ordered::set (int *x*, int *y*, int *z*, float *val*)

sets new value at coordinates creates QVector3D of coordinates and insetrts into map with val as key

Parameters:

x	y z coordinates
v*al*	

OrieGLWidget Class Reference

Public Slots

void **rot** (float xORot, float yORot, float zORot)
Slot to recieve the rotation about various axes.

Signals

void **resetRot** (float m_xORot, float m_yORot)
Signal to reset orientation to 'home'.
void **resetZoom** (float zoomFactor)
Signal to reset zoom to 'home'.

Public Member Functions

OrieGLWidget (QWidget *parent=0)
Constructor.
void **initializeGL** ()
Behaviour on initializing GL pane.
void **paintGL** ()
Draws and rotates arrows.
void **resizeGL** (int w, int h)
Behaviour on resizing GL pane.

Member Function Documentation

void OrieGLWidget::rot (float *xORot*, float *yORot*, float *zORot*)[slot]

Slot to recieve the rotation about various axes.

Parameters:

r*otation*	arround x, y, z axis

scalarField Class Reference

Public Member Functions

scalarField ()
Constructor.
scalarField (int x, int y, int z)
Constructor setting size of field.
void **setSize** (int x, int y, int z)
set Size of field
void **setValue** (int x_, int y_, int z_, float val)
set scalar value at coordinates
float **getValue** (int x_, int y_, int z_)
get scalar value at coordinates
float **getValue** (float x_, float y_, float z_)
get scalar value at coordinates
int **getXSize** ()
Getter that returns the x size of field.
int **getYSize** ()
Getter that returns the y size of field.
int **getZSize** ()
Getter that returns the z size of field.

Member Function Documentation

float scalarField::getValue (int *x_*, int *y_*, int *z_*)

get scalar value at coordinates

Parameters:

x,*y*,*z*	coordinates

Returns:

Scalar value

float scalarField::getValue (float *x_*, float *y_*, float *z_*)

get scalar value at coordinates

Parameters:

x,y,z	float coordinates

Returns:

Scalar value

int scalarField::getXSize ()

Getter that returns the x size of field.

Returns:

the x size of field

int scalarField::getYSize ()

Getter that returns the y size of field.

Returns:

the y size of field

int scalarField::getZSize ()

Getter that returns the z size of field.

Returns:

the z size of field

void scalarField::setSize (int *x*, int *y*, int *z*)

set Size of field

Parameters:

x,y,z	size

void scalarField::setValue (int *x_*, int *y_*, int *z_*, float *val*)

set scalar value at coordinates

Parameters:

x,y,z	coordinates

vecA Struct Reference

The **vecA** struct a structure to return from rung kutta calculations.
#include <mainwindow.h>

Public Attributes

float **x**

float textbfy

float **z**

float **ang**

bool **ok**

Detailed Description

The **vecA** struct a structure to return from rung kutta calculations.

vecField Class Reference

Public Member Functions

vecField ()
Constructor.
vecField (int x, int y, int z)
Constructor setting size of field.
float * **getVec** (int x_, int y_, int z_)
Getter returning vectors at coordinates.
float **getXVec** (int x_, int y_, int z_)
Getter returning x vector at coordinates.
float **getYVec** (int x_, int y_, int z_)
Getter returning y vector at coordinates.
float **getZVec** (int x_, int y_, int z_)
Getter returningz vector at coordinates.
float **getXVec** (float x_, float y_, float z_)
Getter returning x vector at coordinates.
float **getYVec** (float x_, float y_, float z_)
Getter returning y vector at coordinates.
float **getZVec** (float x_, float y_, float z_)
Getter returning z vector at coordinates.
float * **getNVec** (int x_, int y_, int z_)
Getter returning normalized vectors at coordinates.
float * **getNVec** (float x_, float y_, float z_)
Getter returning normalized vectors at coordinates.
int **getXSize** ()
Getter that returns the x size of field.
int **getYSize** ()
Getter that returns the y size of field.
int **getZSize** ()
Getter that returns the z size of field.
float **getSpeed** (int x_, int y_, int z_)
Getter that returns the speed at coordinates.
void **setSize** (int x, int y, int z)
set Size of field
void **setX** (int x_, int y_, int z_, float xVal)
set X vector in field
void **setY** (int x_, int y_, int z_, float yVal)
set Y vector in field
void **setZ** (int x_, int y_, int z_, float zVal)

set Z vector in field

~vecField ()
destructor

float **UYgrad** (int x_, int y_, int z_)
Calculates gradient of x velocity in y direction.

float **UZgrad** (int x_, int y_, int z_)
Calculates gradient of x velocity in z direction.

float **VXgrad** (int x_, int y_, int z_)
Calculates gradient of y velocity in x direction.

float **VZgrad** (int x_, int y_, int z_)
Calculates gradient of y velocity in z direction.

float **WXgrad** (int x_, int y_, int z_)
Calculates gradient of z velocity in x direction.

float **WYgrad** (int x_, int y_, int z_)
Calculates gradient of z velocity in y direction.

float **angV** (int x_, int y_, int z_)
Calculates angular velocity.

float **UYgrad** (float x_, float y_, float z_)
Calculates gradient of x velocity in y direction.

float **UZgrad** (float x_, float y_, float z_)
Calculates gradient of x velocity in z direction.

float **VXgrad** (float x_, float y_, float z_)
Calculates gradient of y velocity in x direction.

float **VZgrad** (float x_, float y_, float z_)
Calculates gradient of y velocity in z direction.

float **WXgrad** (float x_, float y_, float z_)
Calculates gradient of z velocity in x direction.

float **WYgrad** (float x_, float y_, float z_)
Calculates gradient of z velocity in y direction.

float **angV** (float x_, float y_, float z_)
Calculates angular velocity.

Member Function Documentation

float vecField::angV (int *x_*, int *y_*, int *z_*)

Calculates angular velocity.

Parameters:

x,*y*,*z*	coordinates

Returns:

angular velocity

float vecField::angV (float *x_*, float *y_*, float *z_*)

Calculates angular velocity.

Parameters:

x,*y*,*z*	coordinates

Returns:

angular velocity

float * vecField::getNVec (int *x_*, int *y_*, int *z_*)

Getter returning normalized vectors at coordinates.

Parameters:

x,*y*,*z*	coordinates

Returns:

Normalized vectors

float * vecField::getNVec (float *x_*, float *y_*, float *z_*)

Getter returning normalized vectors at coordinates.

Parameters:

x,*y*,*z*	float coordinates

Returns:

Normalized vectors

float vecField::getSpeed (int *x_*, int *y_*, int *z_*)

Getter that returns the speed at coordinates.

Parameters:

x,*y*,*z*	coordinates

Returns:

Speed

float * vecField::getVec (int *x_*, int *y_*, int *z_*)

Getter returning vectors at coordinates.

Parameters:

x,*y*,*z*	coordinates

Returns:

Vector

int vecField::getXSize ()

Getter that returns the x size of field.

Returns:

the x size of field

float vecField::getXVec (int *x_*, int *y_*, int *z_*)

Getter returning x vector at coordinates.

Parameters:

x,*y*,*z*	coordinates

Returns:

X Vector

float vecField::getXVec (float *x_*, float *y_*, float *z_*)

Getter returning x vector at coordinates.

Parameters:

x,*y*,*z*	float coordinates

Returns:

X Vector

int vecField::getYSize ()

Getter that returns the y size of field.

Returns:

the y size of field

float vecField::getYVec (int *x_*, int *y_*, int *z_*)

Getter returning y vector at coordinates.

Parameters:

x,*y*,*z*	coordinates

Returns:

Y Vector

float vecField::getYVec (float *x_*, float *y_*, float *z_*)

Getter returning y vector at coordinates.

Parameters:

x,*y*,*z*	float coordinates

Returns:

Y Vector

int vecField::getZSize ()

Getter that returns the z size of field.

Returns:

the z size of field

float vecField::getZVec (int *x_*, int *y_*, int *z_*)

Getter returning z vector at coordinates.

Parameters:

x,*y*,*z*	coordinates

Returns:

Z Vector

float vecField::getZVec (float *x_*, float *y_*, float *z_*)

Getter returning z vector at coordinates.

Parameters:

x,*y*,*z*	float coordinates

Returns:

Z Vector

void vecField::setSize (int *x*, int *y*, int *z*)

set Size of field

Parameters:

x,y,z	size

void vecField::setX (int *x_*, int *y_*, int *z_*, float *xVal*)

set X vector in field

Parameters:

x,y,z	coordinates
x	velocity value

void vecField::setY (int *x_*, int *y_*, int *z_*, float *yVal*)

set Y vector in field

Parameters:

x,y,z	coordinates
y	velocity value

void vecField::setZ (int *x_*, int *y_*, int *z_*, float *zVal*)

set Z vector in field

Parameters:

x,y,z	coordinates
z	velocity value

float vecField::UYgrad (int *x_*, int *y_*, int *z_*)

Calculates gradient of x velocity in y direction.

Parameters:

x,y,z	coordinates

Returns:

gradient

float vecField::UYgrad (float *x_*, float *y_*, float *z_*)

Calculates gradient of x velocity in y direction.

Parameters:

x,*y*,*z*	float coordinates

Returns:

gradient

float vecField::UZgrad (int *x_*, int *y_*, int *z_*)

Calculates gradient of x velocity in z direction.

Parameters:

x,*y*,*z*	coordinates

Returns:

gradient

float vecField::UZgrad (float *x_*, float *y_*, float *z_*)

Calculates gradient of x velocity in z direction.

Parameters:

x,*y*,*z*	float coordinates

Returns:

gradient

float vecField::VXgrad (int *x_*, int *y_*, int *z_*)

Calculates gradient of y velocity in x direction.

Parameters:

x,*y*,*z*	coordinates

Returns:

gradient

float vecField::VXgrad (float *x_*, float *y_*, float *z_*)

Calculates gradient of y velocity in x direction.

Parameters:

x,*y*,*z*	float coordinates

Returns:

gradient

float vecField::VZgrad (int *x_*, int *y_*, int *z_*)

Calculates gradient of y velocity in z direction.

Parameters:

x,*y*,*z*	coordinates

Returns:

gradient

float vecField::VZgrad (float *x_*, float *y_*, float *z_*)

Calculates gradient of y velocity in z direction.

Parameters:

x,*y*,*z*	float coordinates

Returns:

gradient

float vecField::WXgrad (int *x_*, int *y_*, int *z_*)

Calculates gradient of z velocity in x direction.

Parameters:

x,*y*,*z*	coordinates

Returns:

gradient

float vecField::WXgrad (float *x_*, float *y_*, float *z_*)

Calculates gradient of z velocity in x direction.

Parameters:

x,*y*,*z*	float coordinates

Returns:

gradient

float vecField::WYgrad (int *x_*, int *y_*, int *z_*)

Calculates gradient of z velocity in y direction.

Parameters:

x,*y*,*z*	coordinates

Returns:

gradient

float vecField::WYgrad (float *x_*, float *y_*, float *z_*)

Calculates gradient of z velocity in y direction.

Parameters:

x,*y*,*z*	float coordinates

Returns:

gradient