DATA PREPARATION AND VISUALIZATION FOR THE SWAN REFRACTION MODEL

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF INFORMATION TECHNOLOGY

By
Nicholas Green
September 2003

Supervised by
Dr. James Gain

Data Preparation and Visualization for the SWAN Refraction Model

CONTENTS

Data Preparation and Visualization for the SWAN Refraction Model

1. INTRODUCTION

Scientific visualization can be described as a means of displaying information in a graphical context in order that the encapsulated data can be more readily understood. The means of achieving an appropriate visual format for the underlying data set can assist with identifying relationships and trends that may exist. The techniques used, allow the data set to be analysed from a visual perspective and provide more insight into the nature of the modelled phenomenon [12].

This research and development project seeks to provide a usable interactive graphical interface to an environment that otherwise involves primarily numerical data in a static, non-interactive format. Tools will be developed that will enable users to prepare numerical data required for the SWAN refraction model and to visualize the results in an interactive three-dimensional graphical context. SWAN (acronym for Simulating Waves Near shore) is a numerical wave model that is used to predict wave parameters according to a given set of conditions. The design of the 2-D and 3-D graphical interfaces and their impact on the system will be discussed.

Java will be used as the preferred development language. This language has the advantage of being portable between various platforms, and so, if the system is to be used in different environments, the program code will not have to be re-implemented for each specific purpose. The system will initially be developed in a Windows environment.

The 3-D code will be developed using OpenGL, with an API (Application Programming Interface) that maps the standard OpenGL libraries to Java. This will be needed since Java does not have built-in 3-D graphics capabilities. Two different 3-D APIs will be investigated to provide an appropriate framework in which to develop the graphical programming.

The background to the problem domain and the need for such a visualization tool will be outlined, followed by an introduction to the visualization techniques employed and a detailed design discussion of the 2-D user interface and 3-D graphical interface components.

1.1. Research

Data visualization using Java will be researched to determine the most appropriate framework within which to develop the program. Decisions that will influence this process include the following:

- The use of AWT or Swing components for the graphical user interface
- The use of Java3D or gl4Java for the OpenGL 3-D graphics

Investigations into scientific visualization techniques will be undertaken to determine the following:

- How to visualize layered components
- How to interactively adjust component transparency
- How to effectively render the 3-D model

The resulting output from the SWAN model provides a number of numerical models that need to be viewed together, so the process of layering graphical elements and the interactions between them will be examined. An appropriate data structure for the 3-D model will be considered as well as how to use the OpenGL geometric primitives and rendering options to good effect.

1.2. Development

The system development will focus on providing an interactive environment for the following two main areas:

1. Preparation of input data for the SWAN model.

The data preparation requires development of an intuitive interface mechanism to capture the required input. The various command and parameter selections will have to be written to a file in such a way that they can be understood by the SWAN model. Since there are numerous commands and data values that may need to be captured, a modular interface system will be investigated so that a core set of interfaces can be developed and then others added, following similar design patterns. This core set of interfaces will be used to evaluate the effectiveness of the design. Once a suitable input mechanism has been achieved, further interfaces can be added.

2. Visualization of the numerical results as generated by the SWAN model.

The output data visualization requires the numerical SWAN models to be interpreted in a graphical context and displayed in an appropriate format. More than one surface may need to be visualized at the same time in a kind of layered arrangement, so provision will have to be made for multiple graphical elements to be viewed simultaneously. The need to ascertain the relationship between the different layers means that they need to be viewed independently or in a combined format.

In order to visualize any relationships that may exist between different graphical components, a technique for enabling one of the layers to become translucent is required.

1.3. Scope of work

To accommodate the complete list of input commands that can be used within the SWAN program (see Appendix A) would require much repetitive development, so the focus will be aimed at designing a mechanism for creating modular panels that can be added to at a later stage. For the purposes of this dissertation, a core set of panels will be developed to highlight the design principles that can then be used as a basis for future work.

The 3-D graphical visualization work will focus on developing a computer model that has two surface layers. The transparency of the top layer can be altered so that both can be viewed together in an interactive manner. Controls will be supplied to enable the user to select from different rendering options, enable or disable surface layers and adjust component transparency. Model viewing transformations will be enabled through the use of various mouse and keyboard options.

1.4. Structure of dissertation

Background to the SWAN model is given, followed by a discussion of the need for a visualization tool. The data requirements are outlined to illustrate the structure of the input and output formats.

The process of visualizing computerised images is introduced, followed by a discussion on the use of OpenGL geometric primitives to render the 3-D model. In order to make use of native OpenGL libraries within a Java context, an application programming interface is needed. Two such APIs are discussed to determine the appropriate framework in which to develop the 3D code. Java has two options for displaying graphical user interface components: Abstract Window Toolkit (AWT) and Swing. These are discussed as well as their influence on the incorporated 3-D rendering container.

Design of the 2-D user interface is discussed according to various usability principles, and the layout of different components is presented. Different elements affecting the human-computer interaction of the system are outlined and how they influence the interface design.

Steps involved in the visualization process are presented and a discussion of how the 3-D graphical interface elements have been designed. The underlying data structure and design rationales are explained. Multiple surface layering, model rendering and object transparency issues are discussed as they apply to this project.

Different evaluation techniques are outlined followed by a more detailed look at the particular evaluation process that was followed in order to find any usability problems. The findings from this process are shown followed by a description of how some common usability problem areas were improved.

This dissertation concludes with a round-up of the major issues that were tackled and a description of related future work.

2. BACKGROUND

2.1. What is SWAN?

SWAN is a numerical wave model that has been developed at the Delft University of Technology in the Netherlands (http://fluidmechanics.tudelft.nl/). The program is used to obtain realistic estimates of wave parameters in coastal areas, lakes and estuaries from given wind conditions, bottom surface profiles and currents. The program is written in FORTRAN, and can be compiled for Windows and UNIX platforms. The input data and output results from the SWAN refraction program are in a numerical format.

2.2. Why the need for a data preparation and visualization tool?

The SWAN computational program expects input and output commands and related data to be supplied in a text file format. The instructions contained within this text file are used to determine the type of computation to perform as well as to supply details regarding the grid format and boundary conditions. These input parameters have to be entered manually into the text file which is a time-consuming process that can lead to complications if data is not entered in the correct format. The process of checking this text file is complicated by the many different command types and data formats.

A more user-friendly mechanism for formulating the SWAN command file is needed to:

- allow easier entry of data
- reduce the possibility of data format errors
- remove the necessity of entering default values that do not need to change
- provide visual assistance as to the type of data required
- generate the text file in the correct format

Once the computations have been completed by SWAN, and a result file generated, there needs to be a way of visualizing this in an appropriate format. Using a graphical environment in which to develop a 3-D model of the results will allow the user to view the wave and surface data in an interactive manner.

2.3. Overview of Input requirements and Output format

The input data is specified in a text file that can contain over 50 command codes and related parameters, and the process of entering this data into the file can be quite cumbersome. Not all of the input commands are required for each implementation, but the SWAN model expects the input file to be in a particular format.
The data includes both alphanumeric command codes and numeric information that will provide SWAN with the necessary instructions to generate the required results.

The general format of the input file is a list of commands each followed by related parameters.
The following is an extract from a SWAN command file template: (see Appendix A for complete listing)

```
$  PROJect  'name'  'nr'
$         'title1'
$         'title2'
$         'title3'
$
$  MODE    STATionary/NONSTationary     TWODimensional/ONEDimensional
$
$  COORDinates /  -> CARTesian       \
$                    \  SPHErical   CCM/QC  /
$
$  SET  [level] [nor] [depmin] [maxmes] [maxerr]     &
$     [grav] [rho] [inrhog] [hsrerr]            &
$     CARTesian/NAUTical  [pwtail] [froudmax]     &
$     [printf] [prtest]
$
$  CGRID  / REGular [xpc] [ypc] [alpc] [xlenc] [ylenc] [mxc] [myc]  \
$            \ CURVilinear [mxc] [myc] ( EXCeption [xexc] [yexc] )    /    &
$            / CIRcle                          \
$            \ SECtor [dir1] [dir2] [mdc] /      &
$             [flow] [fhig] [msc]
```

Figure 2.1 SWAN command file format

The SWAN commands shown in figure 2.1 are indicated by uppercase instructions and related parameters that follow in the adjacent square brackets. Some of these commands are required, and others are optional. Variables inside the square brackets represent numerical values. Within each group different commands or variables may be set according to specific requirements. The allowed options are determined by a control flow indicated by logical operators & (AND) and / (OR). Depending on which options are set, the corresponding variables need to be specified. As an example, the CGRID command parameters shown above can be interpreted as follows:

REGular [xpc] [ypc] [alpc] [xlenc] [ylenc] [mxc] [myc] **OR**
CURVilinear [mxc] [myc] (EXCeption [xexc] [yexc])
AND
CIRcle **OR** SECtor [dir1] [dir2] [mdc]
AND
[flow] [fhig] [msc]

In a numerical format, the CGRID command may appear as follows:

CGRID REG 525500.0 6107000.0 0.0 204000.0 137000.0 204 137 SEC 90 300 210 0.03 0.2 10
Each field is separated by white space.

The output file from SWAN (see Appendix B for sample) contains a list of *x* and *y* coordinate values and other attributes for *inter alia* Depth and Significant wave height. The output data is in a space delimited text format, so results can be extracted into a tabular structure. For the purpose of this dissertation, the following fields in the output file will be used to generate 3-D models of the ground profile and wave layers:

Parameter	Unit	Description
Xp	metres	X-coordinate
Yp	metres	Y-coordinate
Depth	metres	Distance to sub aqueous surface
Hsig	metres	Significant wave height

A graphical environment will be developed in which the resulting output files can be visualized. Two layers of detail will be modelled, one for the submarine surface (Depth) and the other representing the significant wave heights (Hsig) at each coordinate point. A transparency control will be provided to enable the user to see through the wave layer and view what lies on the underlying surface.

3. VISUALIZATION

Computerised images consist of of discrete pixels, whose number depends on the resolution. Displaying images involves the rendering of each of these pixels in a particular colour. The pixels are stored in a part of computer memory called the *frame buffer* [1]. The depth of this frame buffer determines the number of colours that can be represented, ranging from 1-bit for two colours to 24-bits or more for true colour.

This dissertation focuses on two areas of graphical representation, namely user interfaces (2-D) and graphical scientific visualization (3-D). Visualization of scientific information can be described as *the merging of data with the display of geometric objects through computer graphics* [1]. Representing objects that are derived from a coordinated list of points requires a process of sorting the data and assembling a model of geometric shapes based on vertex and vector information.

Model rendering involves data processing and transformations such that a list of points in a file can be read, interpreted and displayed on the computer screen as geometric shapes, each in a particular colour or texture. The forms of these shapes will be determined by the types of primitives to which the points or vertices are allocated. In OpenGL there are various geometric primitives:

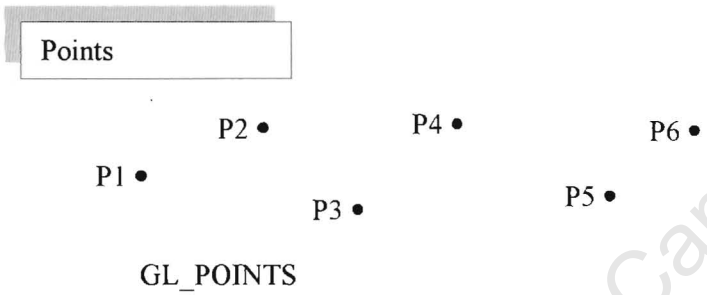


Figure 3.1 OpenGL points

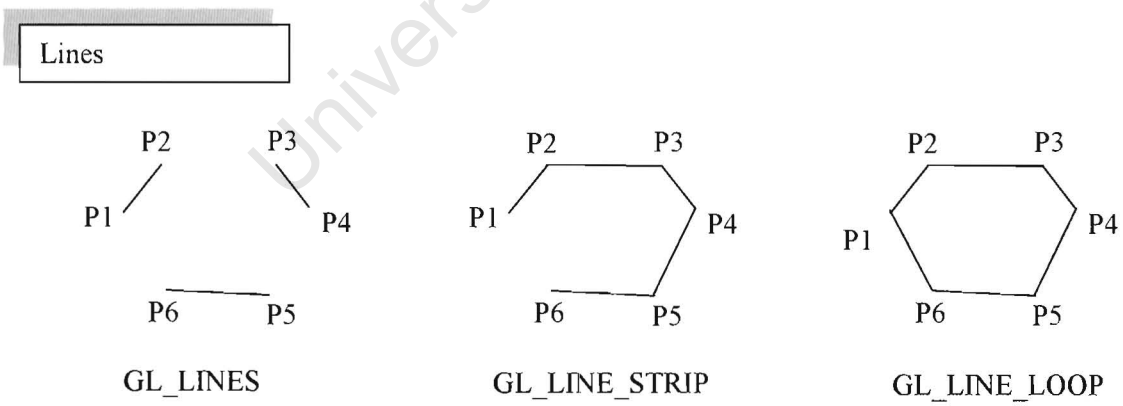


Figure 3.2 OpenGL line segments and polylines

The GL_LINES type is used to draw independent lines, whereas the GL_LINE_STRIP and GL_LINE_LOOP line types can be used where a continuous connection is required between a given set of points. With each of the line types shown in figure 3.2 the results are produced from the same set of points. In each case the required line type is set first and then the points are processed one at a time. The GL_LINES type requires two points to be specified for each line, whereas for the GL_LINE_STRIP and GL_LINE_LOOP types the first point specifies the starting point of the line segment and the second point specifies the segment's end point and the starting point of the next segment.

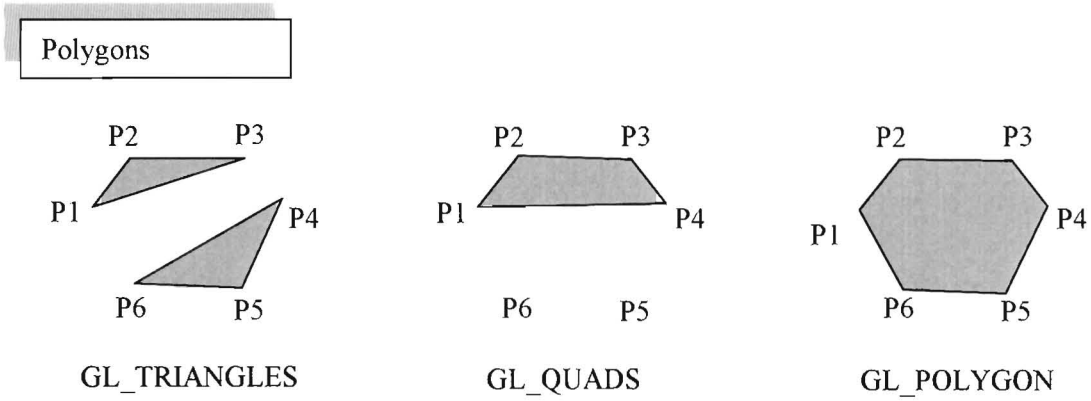


Figure 3.3 OpenGL polygon types

The polygon primitives shown in figure 3.3 above are used to draw independent shapes, and they can be drawn filled (as shown) or with edges only.
There is also an extended group of polygonal shapes that can be used to draw a series of triangles or quadrilaterals that share common vertices:

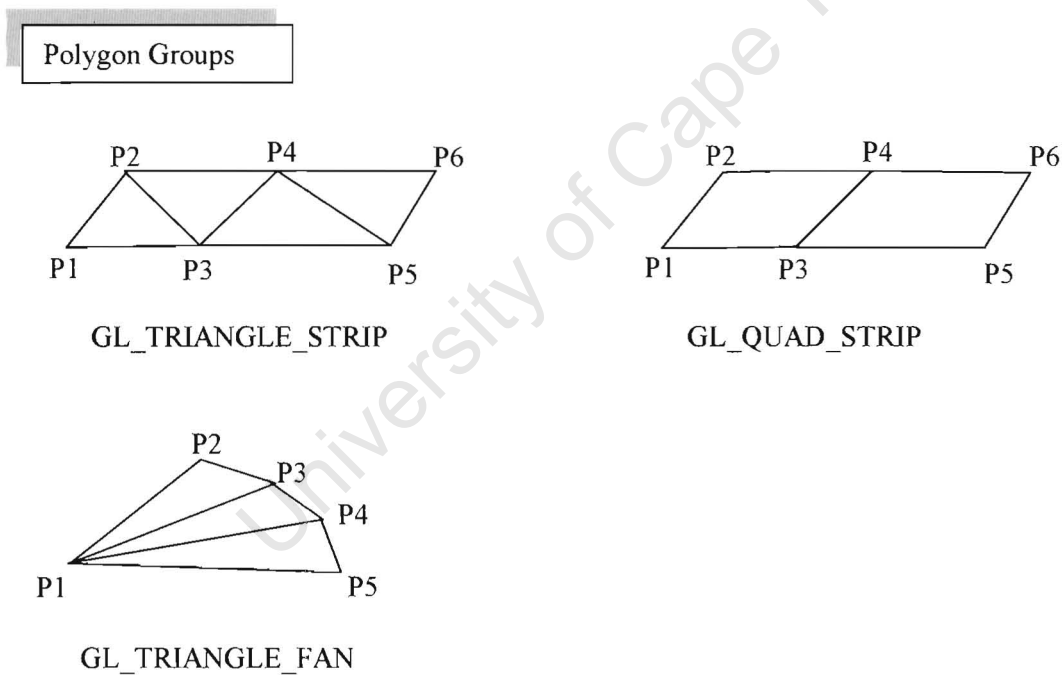


Figure 3.4 OpenGL triangle strip, quadrilateral strip and triangle fan

The 3-D model is constructed by drawing geometric primitives in 3-D space and rendering them with a particular colour or texture. The boundaries of shapes are determined by the (x,y,z) coordinates of vertices and the order in which they are presented to the processing engine. The final model may be seen from different viewpoints through the use of different projection transformations. Parallel orthogonal projections result in a flat view parallel to the projection plane, and perspective projections result in views where objects that are further from the viewer appear smaller.

In order to visualize a 3-D object on a computer screen the image needs to be transformed and projected from 3-D space to a 2-D surface. This process requires the manipulation of the underlying matrices used to represent the vertices in 3-D space. Transformations and rotations can be used to view the model from different perspectives, and the projected image may need to be clipped in order for it to fit into the visible view port.

The process of setting up the model and projecting it requires that certain OpenGL options be enabled. OpenGL is based on a procedural language, and rendering options need to be switched on (enabled) or switched off (disabled) programmatically. These include settings for enabling or disabling *inter alia* texture mapping, depth testing and blending. Texture mapping is the process of applying an image to the surface of a geometric primitive instead of specifying a colour. Depth testing is used when multiple objects are displayed, to determine which is closer to the viewer. The depth buffer keeps track of each pixel in the viewport to determine the distance between the viewpoint and the object occupying that pixel. If the specified depth test passes, the incoming object is displayed, thereby obscuring any hidden objects. Blending is used to combine the colours of current and incoming pixels in the frame buffer. The way that the model is rendered is also determined by the type of shading that is to be performed. This can be specified as being flat or smooth. Flat shading results in noticeable edges at boundary lines between geometric primitives that are rendered with different colours. As Edward Angel writes: "Flat shading will show differences in shading for the polygons in our mesh" [1]. This is due to the shading calculation using the normal associated with only one of the vertices for each of the primitives. To achieve a better result, the graphical model will be rendered using smooth shading, where the normals at each of the vertices are used. The boundary lines between primitives of differing colours will be blended to create a smoother finish. The type of shading to be used is called *Gouraud* shading [2]. The normal at each vertex is calculated by taking the average of the polygon normals that share that vertex. By using smooth shading, "OpenGL interpolates the colors across the faces of the polygons automatically" [1].

3.1. 3-D graphics API

The original OpenGL graphics API is written in C, and its procedures and functions are not directly accessible from Java. A middleware API is therefore necessary to make the OpenGL calls available to a Java program through a process of wrapping. This process involves making "calls through the Java Native Interface (JNI) to call the native OpenGL libraries" [3].

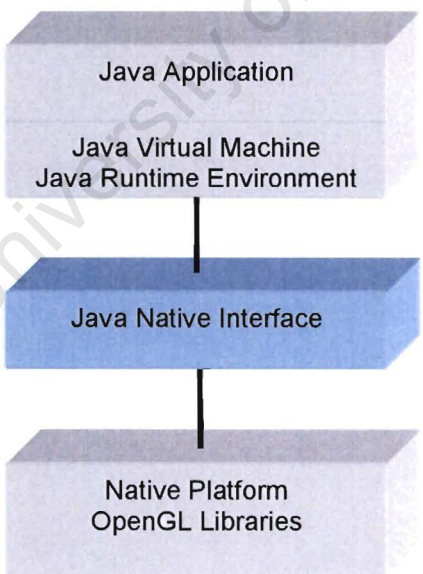


Figure 3.5 Java Native Interface enables Java applications to access native platform libraries

Since Java does not have built-in 3-D graphical capabilities, it is necessary to implement an add-on package that can deal with the complexities of 3-D rendering. Two such packages are investigated: Java3D from Sun Microsystems (http://java.sun.com/products/java-media/3D/) and gl4Java from Jausoft Software (http://www.jausoft.com).

3.1.1. Java3D

Java3D is a scene graph-based API that integrates with the standard Java environment through the use of an object oriented programming interface. Java3D programs "construct individual graphics elements as separate objects and connect them together in a tree-like structure called a *scene graph*. The application manipulates these objects using their pre-defined accessor, mutator, and node-linking methods" [4]. Accessor methods are used to get information from an object, mutator methods are used to change information that is stored in an object, and node-linking methods are used to relate the objects in the hierarchical structure. The scene graph is a collection of node objects that form a *virtual universe* [5]. The diagram below indicates a typical scene graph data structure.

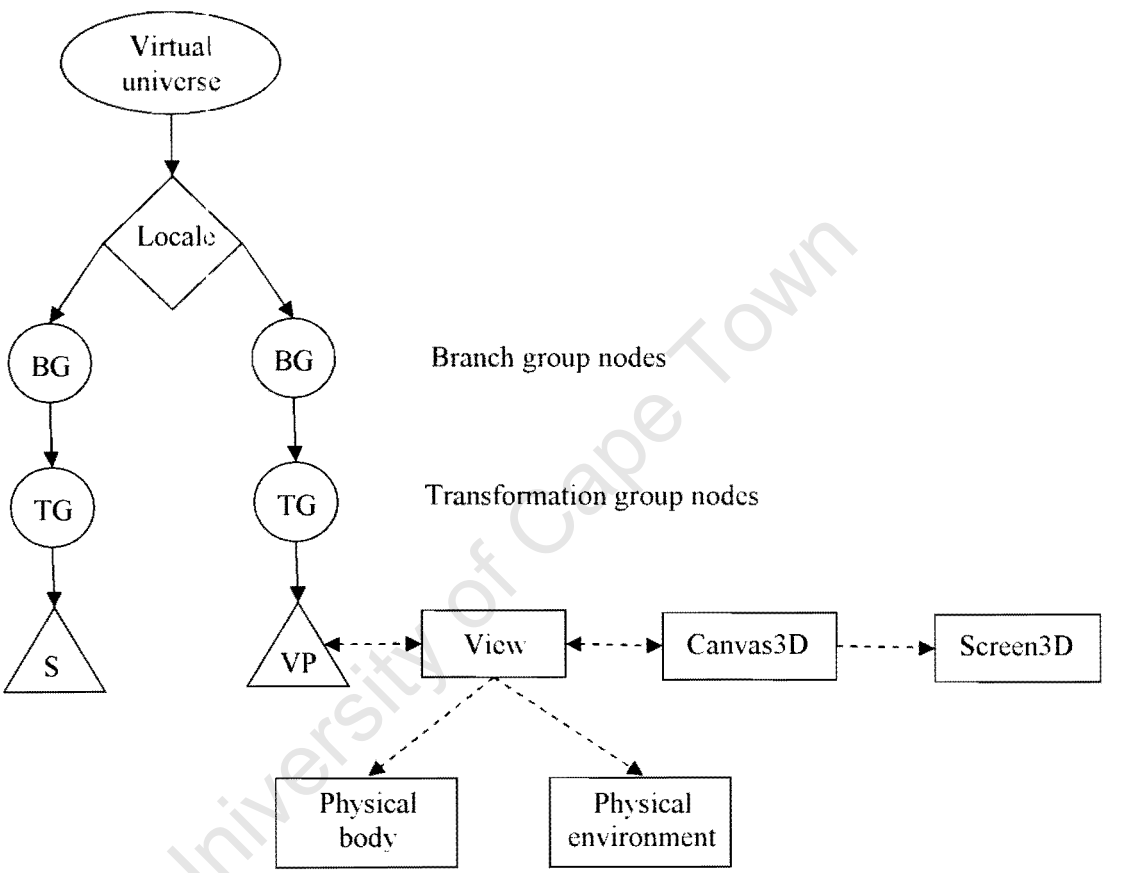


Figure 3.6 Typical scene graph data structure

The geometric shapes are indicated by the "S" triangle, and these objects are derived from the Shape3D class. This represents each shape that will be drawn to make up the 3-D model. The "VP" indicates the View Platform node which is responsible for the model viewing and orientation from the perspective of the viewer. The Virtual Universe may contain a number of Locale nodes, each responsible for their own set of graphical objects. If a Locale is active, then one of its Branch Groups needs to contain the View Platform.

An advantage of this environment is that the programming is object-oriented, and the API provides a high level abstraction of OpenGL or DirectX graphics programming. This allows the programmer to design a 3-D model based on objects and achieve the desired result by changing the behaviour of the objects concerned. Once objects have been defined, they can be re-used in true object-oriented fashion. Apart from objects that describe the 3-D shapes, there are also objects that are defined for other elements such as transformations, lights and materials.

For developers who want to program in 3-D using Java and not worry about the underlying graphics code, the use of Java3D has been described as "allowing them to concentrate on content creation and application logic, without choking on the details of rendering and arcane programming syntax" [3].

A disadvantage of abstracting the OpenGL graphics programming code is that the developer does not have total control over the model construction and rendering processes. This is a price that one has to pay for using pre-compiled application interface libraries.

3.1.2. gl4Java

gl4Java is an abbreviation of "OpenGL for Java" [6], and is an API that maps the native C OpenGL functions to Java. The methods have been named such that they are the same as standard OpenGL functions with the added prefixes "gl." for GL function calls and "glu." for GLU function calls. Using this naming convention enables one to apply C-based OpenGL code to a Java environment by prefixing the function calls appropriately. The method of coding 3-D graphics through the usage of gl4Java is therefore very similar to that of native OpenGL. The advantage of mapping the syntax closely to the original OpenGL is that one can predict its usage through reference to standard OpenGL literature and reference material.

gl4Java is not a scene graph-based API (as used in Java3D) and the developer is responsible for the entire model creation and rendering process. Using this type of API, the developer has more control and flexibility over the way that the graphic programming is implemented. Since gl4Java is not scene graph-based the environment is not entirely object-oriented. There are elements of abstraction contained within the API, but the majority of the OpenGL-related methods are based on a fundamentally procedural format. This is due to the fact that the native libraries are written in C, which is a procedural language. Object-oriented elements contained with the API include these drawing context components that interact with the Java environment:

- GLCanvas inherits from Java's AWT Canvas class
- GLAnimCanvas is a subclass of GLCanvas and, through implementation of the java.lang.Runnable interface, is used as a thread for animated graphics
- GLJPanel inherits from Java's Swing JPanel class
- GLAnimJPanel is a subclass of GLJPanel and, through implementation of the java.lang.Runnable interface, is used as a thread for animated graphics

There are other object based components for incorporating 3-D graphics in an applet, namely:

- SimpleGLApplet1 inherits from Java's AWT Applet class
- SimpleGLAnimApplet1 inherits from Java's AWT Applet class
- SimpleGLJApplet1 inherits from Java's Swing JApplet class

Composing a 3-D model using gl4Java requires that low-level geometric primitives are assembled to form the desired shape. OpenGL does not provide high-level commands for describing models of complex 3-D objects, so any complicated shapes need to be constructed programmatically from the basic points, lines and polygons. An appropriate data structure for describing the 3-D model needs to be designed, and the programmer is free to choose a suitable framework.

gl4Java includes texture loading objects which can load image files to be used as textures. One such class is PNGTextureLoader which can be used to load a portable network graphic file from disk.

Using gl4Java allows the developer the freedom to choose an appropriate structure for the object-oriented environment.

3.1.3. Selecting the appropriate API

In deciding whether to use Java3D or gl4Java as the 3-D graphics API, the following questions were asked:

- Is a scene graph necessary?
- How much control over the OpenGL library calls is required?
- Which option is able to integrate better with the user interface?

A scene graph-based API is useful for constructing models of objects that interact with each other. This is due to the hierarchical tree-like structure that is used to assemble the various object nodes. An example of where this can be applied is in the rendering of a robotic arm. Components that constitute the arm can be defined as individual objects with certain behaviour. The final model is assembled by tying these objects together.

Visualization of the SWAN output will involve the rendering of mesh-like geographic surfaces. The position of these layers relative to each other will not change, and each will be modelled as a separate surface. There is no need for a hierarchical object-oriented structure, since any viewing transformation changes will not affect the fundamental construction of the underlying model. The transparency properties of the surface will vary, but such changes will affect the layer as a whole.

Both Java3D and gl4Java act as wrappers around the native OpenGL libraries, but differ in their level of abstraction. Java3D provides classes and related methods that include pre-written libraries to deal with OpenGL integration, whereas those offered in gl4Java allow more control over the OpenGL calls. Having absolute control is not always necessary, but when the graphical interface design process is to be discussed, the steps involved can be more fully explained if they have been implemented directly.

In order to determine which API offers the most appropriate integration with the user interface, the different 3-D drawing canvas-type options were investigated. The following section discusses the choices made between the use of AWT or Swing components, and their impact on the rendering container that was eventually chosen.

3.2. AWT or Swing components?

The Java API contains two options for displaying user interface components: AWT (Abstract Window Toolkit) and Swing.

AWT components make use of *peers*, which are described as: "native GUI components that are manipulated by the AWT classes" [7]. Much of the component behaviour and rendering process is determined by the platform on which the code is run. This results in the *look and feel* of the interface components changing to suit the relevant platform. The program code is platform independent, but the way in which the components are painted is not. In fact the AWT classes act as wrappers around the peer components [7].

In an environment where the GUI display needs to conform to platform-dependent standards, then AWT components should be used to maintain a compatible design framework. With the flexibility that object oriented inheritance has to offer, standard AWT components are quite rigid. This is due to the fact that they delegate much of their functionality to the native GUI components. AWT components are described as heavyweight, which means that they associate with peers and are rendered in their own native contexts [7]. Lightweight components on the other hand are not associated directly with peers and are rendered in the context of their heavy weight containers [8].

Swing components are mostly lightweight, with the exception of top-level containers, and many new feature-rich components have been added to the GUI library. They also support a pluggable look and feel, which means that a consistent appearance can be maintained across platforms. This has the advantage of making the user interface look the same, regardless of the underlying hardware platform. Bruce Eckel describes this feature as follows: "the appearance of the UI can be dynamically changed to suit the expectations of users working under different platforms and operating systems" [9].

Some of the additional components that are available in the Swing framework are:

- JTable – for displaying rows and columns of data
- JTree – for displaying hierarchical data using branch and leaf nodes
- JTabbedPane – for providing convenient access to more than one panel
- JSplitPane – for displaying two components separated by a divider

Aside from the fact that these and other Swing components have useful functionality, they also have a fresh, modern appeal.

In designing the user interfaces for this system, it was found that the Swing GUI components were more versatile in behaviour than their AWT counterparts. Also the Swing library contains many more classes like the JTable and JTree which offer very useful additions to the standard library.

One area that was found to be problematic was when AWT and Swing components were both used in the same structure. This is mentioned by David Geary when he writes "To this day, mixing lightweight and heavyweight components in the same applet or application can be problematic, especially embedding heavyweight components inside lightweight containers"[8]. Where this irregularity became apparent was when the drawing canvas for the 3-D graphics object was added as one of the components inside a JSplitPane. The JSplitPane is a lightweight Swing container, and when a 3-D graphics canvas based on a heavyweight AWT component was used, the JSplitPane divider could not be moved in the direction of the AWT canvas. When a 3-D graphic canvas based on a Swing component was used, the divider could be moved without a problem.

The following 3-D drawing canvas types were tested:

- Canvas3D(Java3D) – based on an AWT component (problem moving divider)
- GLCanvas(gl4Java) – based on an AWT component (problem moving divider)
- GLAnimCanvas(gl4Java) – based on an AWT component (problem moving divider)
- GLJPanel (gl4Java) – based on a Swing component (no problem moving divider)
- GLAnimJPanel (gl4Java) – based on a Swing component (no problem moving divider)

Two were found to integrate properly with the user interface Swing components, and these are GLJPanel and GLAnimJPanel. The positions of these within the Java inheritance tree are as follows:

```
java.lang.Object
 |
 +-java.awt.Component
     |
     +-java.awt.Container
         |
         +-javax.swing.JComponent
             |
             +-javax.swing.JPanel
                 |
                 +-gl4java.swing.GLJPanel
                     |
                     +-gl4java.swing.GLAnimJPanel
```

Figure 3.7 Java inheritance tree for GLJPanel and GLAnimJPanel classes

The GLJPanel and GLAnimJPanel classes are contained within the gl4Java.swing package. The main difference between the two is that the GLAnimJPanel can be applied as a thread through its implementation of the *Runnable* interface. When the GLAnimJPanel class was used to contain the 3-D graphical interface, the performance of parts of the system was negatively affected. When the 3-D graphic was displayed and one of the drop-down menu options selected, the reaction time was very slow. The viewport itself was rendered fairly quickly (within a couple of seconds), but its integration with the user interface was found to be poor. Rendering the graphical model inside a thread causes the screen image to be redrawn repeatedly according to a particular number of frames per second. If the model is not changing, then constantly redrawing it on the screen is not necessary.

To test the system performance without implementing a thread, the GLJPanel class was used. This class requires that repainting the screen image be handled by the programmer. The viewport was only redrawn when a change occurred, and not repeatedly as in the case of the thread implementation. By only drawing the 3-D viewport as required, the performance of the rest of the system improved. Menu options could be selected without any noticeable degradation of response.

The GLJPanel class was found to be the most appropriate container in which to render the 3-D graphical interface. To reduce the chance of flicker in the rendered image, a system of *double buffering* is implemented where on and off screen images are used interchangeably. While an image is being displayed on the screen, an off screen image can be computed so that when it needs to be displayed, the buffers just need to be swapped. This process "allows the CPU to have uninterrupted access to one of the buffers while the video controller has uninterrupted access to the other" [2].

4. DESIGN OF THE 2-D USER INTERFACE

The 2-D interface was implemented using Swing components according to the following general usability principles as defined by Dix *et al* [10]:

Learnability – the ease with which users can learn how to use the system effectively.
Flexibility – providing users with different methods of interaction to suit their needs.
Robustness – providing support and feedback to assist users in achieving goals successfully.

Further design considerations outlined by Foley *et al* [2] have been applied:

- Be consistent
- Provide adequate feedback
- Minimize error possibilities
- Provide error recovery
- Accommodate multiple skill levels
- Minimize the need for memorization

The user interface will be described, and the functional components discussed according to the above mentioned usability and human-computer interaction principles.

4.1. GUI Component layout

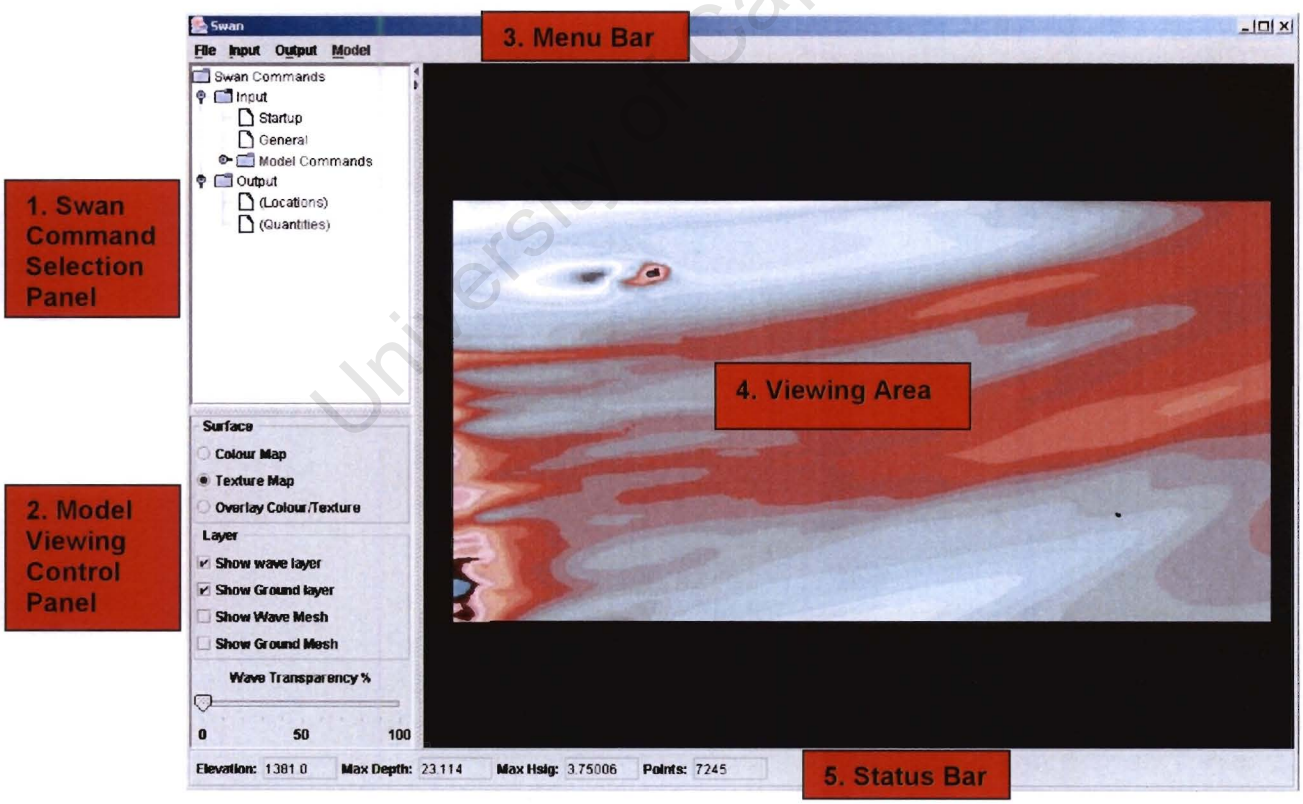


Figure 4.1 Graphical User Interface component layout

Graphical Interface components overview

1. Swan Command Selection Panel – This panel is used to select the relevant data entry group to be displayed in the Viewing Area. The input and output commands have been arranged in a tree-like structure to aid the navigability of the command categories.

2. Model Viewing Control Panel – This panel is used to change the model viewing parameters. Different selection components have been used according to their functional requirements.

3. Menu Bar – This is a standard drop-down menu bar from where program options can be selected.

4. Viewing Area – This is the main display area that will display the 3-D model, coordinate and detail table, or a 2-D graphical interface for data entry.

5. Status Bar – This panel displays information about the model that has been loaded from file.

4.2. Learnability

This usability principle concerns the design of a user interface that can be understood by users through the implementation of a supporting environment. Users need to feel comfortable with the look and feel of the interface and the way that the various controls operate in order to use the system effectively. There are certain usability principles that can aid the process of designing a learnable system without creating confusion. Dix *et al* have identified the following principles affecting learnability: [10]

- Predictability
- Synthesizability
- Familiarity
- Generalizability
- Consistency

These principles are geared to assist with the design of an environment in which the users feels comfortable. Those people that will initially use the system are expert users with experience in interacting with engineering software and graphical tools. They are exposed to other software products, and as such have developed an understanding of the way that common goals can be reached. Novice users might not have developed such a clear understanding of how to achieve the same goals, so the interface should be designed to assist them. Once users are familiar with the common environment, then learnability principles can be applied to support and strengthen the knowledge and understanding of what is already known.

The user interface has been designed in such a way that the different tasks to be undertaken are grouped according to their respective functionality. Two interface elements that have become quite common include the use of a drop-down menu bar at the top of the display, and a status bar at the bottom. The menu bar is used as a navigational aid to initiate a particular system function, and the status bar is used to provide feedback and display relevant information. The use of familiar elements such as these helps the user to understand the system and predict the steps that are necessary to accomplish a particular goal.

To reduce the chance of errors, certain menu options are disabled until their selection becomes appropriate. An example of this is that the user cannot view a 3-D model until the data file has been loaded from disk. The user is able to see all of the menu items, but those that have been *greyed-out* cannot be selected.
The effect of preventing errors from occurring is that the user is not confronted with unnecessary dialogue boxes to explain certain exceptional states.

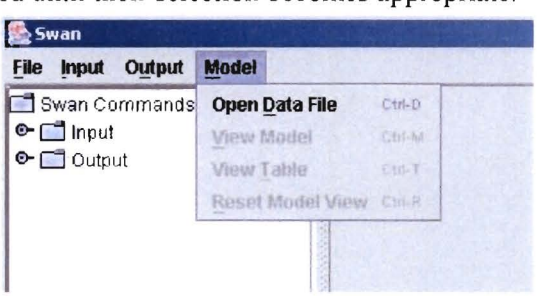
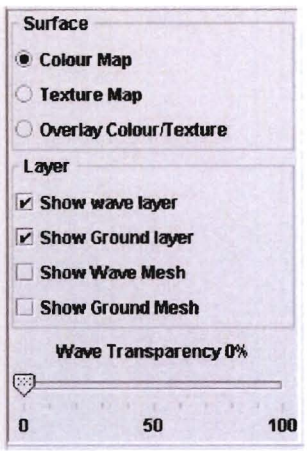


Figure 4.2 Menu showing disabled items

In a similar way, the model data information in the status bar is only displayed once the file has been loaded. The status bar elements are visible, but not activated until the details can be appropriately displayed.

Further examples of interface elements using familiarity and predictability are seen in the use of radio buttons and check boxes. These GUI components are used to set options using logical states.



Where only one option in a group may be selected, then radio buttons have been used. This is shown here by the "Surface" display options. The radio buttons have been surrounded by a titled border to further indicate their relationship to each other.

If multiple options in a list may be selected, then check boxes have been used. The state of each check box is clearly indicated by a tick when selected, or empty when deselected. These check box controls have also been surrounded by a titled border to show that they are related in some way.

The slider control indicated at the bottom may not be familiar to some users, but its use should be fairly predictable. As the slider is moved along the track, the heading "Wave Transparency 0%" is updated to indicate the current level of transparency on a scale of 0 to 100%.

Figure 4.3 Model viewing control options

The principle of synthesizability is the ability to assess the current state of a system based on past operations [10]. This means that the user should be able to determine the current state based on previous interactions with the system. The user needs to feel comfortable when a state change has occurred by having such a change acknowledged. State changes can be supported through the appropriate use of feedback to ensure that the user is aware that a change has occurred. The appropriate use of dialogues can help in this regard to inform the user of relevant changes.

The layout of interface components and their interaction with the system should be carried out in a consistent manner. Users will be able to learn a system more effectively if a consistent design approach has been applied. A system that does not behave in a consistent manner could result in the user becoming confused and frustrated. The interfaces that are used to capture the input data required by SWAN have been designed using a modular approach. Each of the data capture panels is grouped according to function. A series of tabbed panes have been developed so that within each group, the user is able to see all of the available panels.

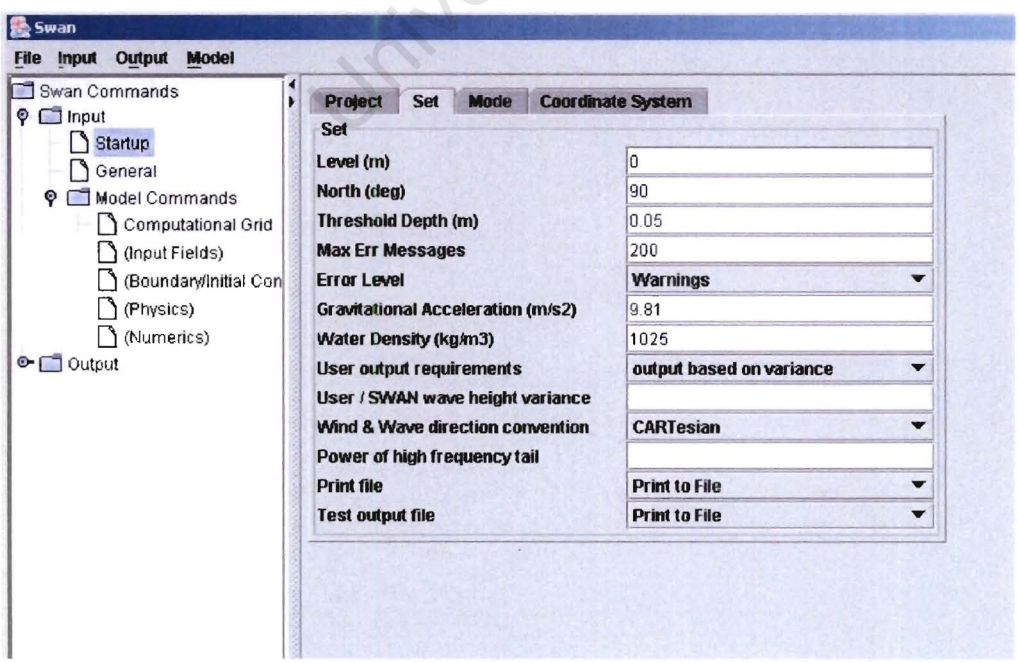


Figure 4.4 shows the *Startup* group with the *Set* panel selected. The other panels on the group; Project, Mode and Coordinate System can be selected by pressing the relevant tab heading. The user is guided as to the type of data required by a tool tip that displays when the mouse is positioned over a component.

Figure 4.4 Startup command group shown with *Set* panel in front

The principle of consistency can be extended to include more generally accepted computer interactions. These *generalized* interactions include those that promote visual familiarity and behavioural predictability. For example, the use of recognized shortcuts for common actions; <ctrl> N for "New", <ctrl> O for "Open" and <ctrl> S for "Save".
The behaviour of navigational aids like menus and trees should be applied consistently so that users, who are already familiar with their use in other environments, are able to use them without confusion.

The memorization required to operate this system effectively has been kept to a minimum. Menu and display options are presented in natural language, so the user does not have to try to memorize, for example, a set of numerical command codes. Perhaps the only area where some form of memorization is needed is in the navigation of the 3-D model. Here different mouse and keyboard options are used to perform the tasks of zooming, panning and rotating the model.

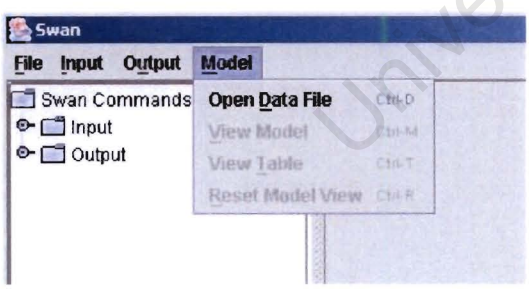
4.3. Flexibility

A flexible system is one that supports different forms of human-computer interaction in such a way that the user is able to use the system efficiently and effectively. To support users ranging from novice to expert, different navigational and interactive options have been incorporated. These include the use of accelerator keys, shortcuts and mouse controls.

The following factors influencing flexibility have been outlined by Dix *et al* [10]:

- Dialogue initiative
- Multi-threading
- Task migratability
- Substitutivity
- Customizability

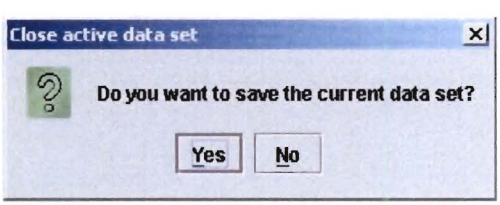
The dialogue between user and system is mostly *user pre-emptive*, which means that the user is able to decide what steps to take in performing a particular task. To allow the user the freedom to select options, those that would result in errors are disabled until they can be used appropriately. An example of this is shown in figure 4.5 below:



Here the "Model" menu has been activated, and only the "Open Data File" option can be selected. The other options in this menu are disabled until the data file has been successfully loaded from disk. This technique of error prevention has been applied in other parts of the system, and they will be discussed further in section 4.4. Figure 4.5 also shows the relevant shortcut keys associated with the menu options as well as the accelerator keys (indicated by the characters that are underlined).

Figure 4.5 Model menu showing different menu item states

In circumstances where the system needs a response from the user, then dialogue boxes have been used. For example, when the user selects 'create a new file', then the system prompts to save the existing open data set.



Here the user is required to make a selection before being able to continue with the current procedure. If the dialogue box is dismissed without selecting *Yes* or *No*, the system is returned to the state that it was in prior to the processing of the command that initiated the system dialogue.

Figure 4.6 Confirmation dialogue box

Multi-threading has been implemented in an *interleaved* format that "permits a temporal overlap between separate tasks but stipulates that at any given instant, the dialog is restricted to a single task" [10].
The user interface is divided into different functional areas (as described in section 4.1.) where the main viewing window will either display one of the data capture panel groups, the model coordinate table or the 3-D graphical model. Each of the objects responsible for these visual components is held in memory until required for display. The human-computer interaction will therefore be concentrated on one task at a time.

Flexibility is offered in the navigational and interactive structures of the system as follows:

- Data capture panel groups can be selected by using the drop-down menu or navigational tree structures. Both have been implemented using a top-down approach with the *input* and *output* nodes being the roots of their respective structures. Child nodes branch off to indicate sub-groups or categories.
- Accelerator keys have been assigned to menu items. These come into operation when the user presses the *alt* key on the keyboard followed by the key matching the relevant menu item. With these accelerator keys, the menu structure still has to be traversed one layer at a time.
- Shortcut keys have been provided for those tasks that may need to be performed a number of times. These are activated by pressing the *control* button on the keyboard and the key referring to the relevant task. Unlike the accelerator keys, shortcut keys activate the relevant option directly without having to first traverse any parent menu structure layers.
- The 3-D model can be viewed from different angles, panned and zoomed by using mouse and keyboard options. Those users who use a mouse with a scroll wheel can use it to zoom in and out of the model. For users who don't have this facility, then an alternative mouse and keyboard combination is available.

The different 3-D model viewing controls are as follows:

Zooming out (increasing the elevation)
Option 1: Whilst pressing the <alt> key, press the left mouse button and move the mouse pointer upwards.
Option 2: Using the scroll button on the mouse, rotate the button away from you. This will adjust the elevation level by 1 unit. To accelerate the zoom factor, press the <shift> key whilst turning the scroll button.

Zooming in (decreasing the elevation)
Option 1: Whilst pressing the <alt> key, press the left mouse button and move the mouse pointer downwards.
Option 2: Using the scroll button on the mouse, rotate the button towards you. This will adjust the elevation level by 1 unit. To accelerate the zoom factor, press the <shift> key whilst turning the scroll button.

Panning
Whilst pressing the <shift> key, press the left mouse button and move the mouse in the direction that you would like to move. To accelerate the pan, press the <shift> and <alt> keys together

Rotating the view
Press the left mouse button and move the mouse in the desired direction to rotate.

The main body of the user interface has panels that are contained within split pane objects. These are separated by a divider that can be moved by the user. The user can, for example, increase the viewing area of the 3-D model display by shifting the vertical divider to the desired offset. Alternatively if the user wants to increase the size of the control panels on the left of the display, then the divider can be dragged in the direction of the viewing window. The horizontal divider between the Swan command selection and model viewing control panels can also be moved up or down to the desired location. These are examples of user-initiated modifications.

A system-initiated interface modification occurs when a model data file is opened. The model viewing control panel is only displayed once the data file has been successfully loaded.

4.4. Robustness

The robustness of a system can be determined by the ability to successfully achieve a specified goal. Measures should be put in place to prevent or catch exceptional conditions in order to maintain a stable system. The following principles affecting robustness have been outlined by Dix *et al* [10]:

- Observability
- Recoverability
- Responsiveness
- Task conformance

When the system state changes due to user interaction or otherwise, then feedback of some sort should be provided. This is to assist the user in evaluating whether or not the task at hand is being performed correctly. Users will get frustrated if there is any doubt as to the success of a given task.

According to Foley *et al* [2]: "Feedback can be given at three possible levels, corresponding to the functional, sequencing, and hardware-binding (semantic, syntactic, and lexical) levels of user-interface design". Feedback at the hardware level concerns user interaction via the keyboard, mouse or other input device. When a text field has system focus and the user types a character, then it should be added to the text displayed in the text field. There are times when keystroke events are *consumed* to prevent the user from typing invalid characters. The validity of each key typed is checked and if an inappropriate key entry is encountered, the keystroke is prevented from having any affect on the system. This process of consuming events is used as a preventative measure against inappropriate numerical data capture. Two key listeners have been implemented; one responsible for allowing only integer types and the other for allowing floating point numbers. Any characters that would result in an incorrect numerical format are prevented. Hardware feedback responses to mouse movements are clearly observed by the cursor moving around the screen.

Sequencing feedback involves observing the actions required in performing a series of steps. This can be seen when browsing the menu structure for a specific option. The menu option at each step is highlighted to show the currently selected item. Other examples include a button that changes colour when it has been pressed, and the cursor visibly moving to the next text field in a sequence when the *tab* button is pressed.

Functional feedback displays information relating to changes in system state such that the user can tell whether something has happened. This type of feedback has been implemented in a number of instances:

- When the wave transparency slider is moved, the current percentage transparency is displayed as a number from 0 to 100%.
- When the zoom factor for the 3-D model is changed, the current elevation is displayed in the status bar.
- Dialogue boxes are displayed to present information of the following types:
    - Confirmation message dialogue
    - Input dialogue
    - Message dialogue of one of the following types
        - Error message
        - Informational message
        - Warning message
        - Question message
        - Plain message

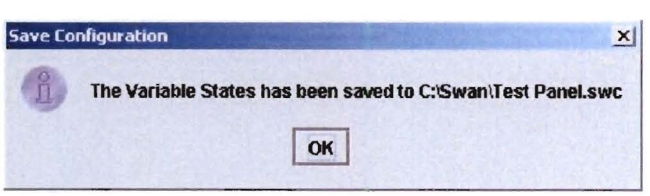
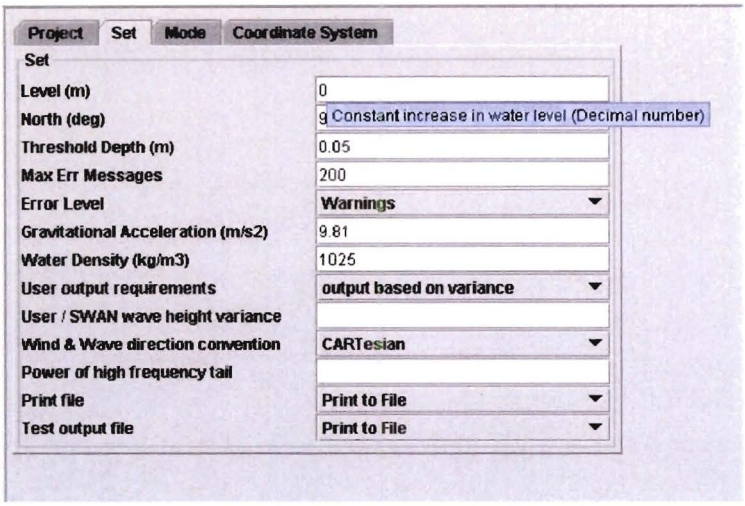


Figure 4.7 shows a dialogue box that is displayed once a data set has been successfully saved to disk. The path and name of the saved file are displayed to inform the user of its location.

Figure 4.7 Informational dialogue box

Another type of feedback that has been deployed includes the use of tool tips. This is especially useful when used with the data capture panels. A lot of the required fields are domain specific, and therefore a brief description of the function and required data type of each can assist the user in entering the correct information.



In figure 4.8 the mouse is positioned over the text field next to the label "Level (m)". The tool tip that pops up provides more detail as to the type of information that is required as well as to the relevant numerical format. The use of tool tips in this regard can provide a lot of help to the user in a *passive* way. The user does not have to go hunting through help dialogues for an explanation of what is required for a particular data item. As the mouse is moved over the item, the description is displayed just beneath the current pointer location. This is an advantage of using tool tips since the focus is concentrated around the current mouse pointer position.

Figure 4.8 Tooltip usage

Through the use of tool tips the user does not have to refocus on another part of the screen in order to get the relevant help or information.

The system has been designed to eliminate the possibility of errors, where possible. Where an error does occur, the user is presented with a dialogue box that explains what has happened, and suggests a possible remedy. The user is then able to take corrective measures to recover from an exceptional condition.

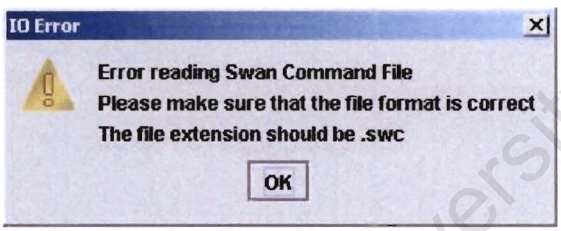
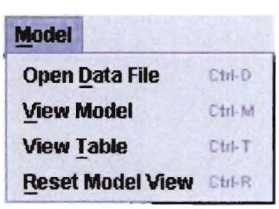


Figure 4.9 shows an example of a dialogue box that is displayed when the user tries to open a file of the incorrect type. Plain language has been used for error messages so that the user is adequately informed of what has gone wrong. If error codes were used, the user may still have needed to peruse further documentation to ascertain the reason for an error.

Figure 4.9 Error message dialogue box

Once data has been typed into a text field, it remains editable so that values can be amended after they have been entered. If the user finds that too many incorrect values have been entered into the numerical fields, then a new data set can be initialized by selecting to start a new file. The user is therefore able to recover to the initial state and those text fields that have default values will be reinstated.

Further feedback is seen where, before the current data set is closed, the user will be prompted to save the values. If the values are to be saved for future use, then the states of the objects representing the panels are serialized to disk in a binary format. This means that the current object states including text field values and combo box settings will be saved. The objects themselves can be read back at a future time through a process of de-serialization when the encapsulated states of the objects can be reconstructed.



Recoverability can also be seen within the context of the 3-D model viewport where the initial view can be returned to at any time. If the user finds that the current view of the model is not appropriate and would like to reset the view, then this can be achieved by either selecting "Reset Model View" from the View menu, or using the shortcut key <ctrl> R, as shown in figure 4.10.

Figure 4.10 Model menu enables recovery of the model view

Feedback is presented to the user as soon as possible after the need for such communication, and so the perceived responsiveness of the system should be instantaneous. Certain system responses are in the form of dialogue boxes that are displayed after the user requests to perform a certain action. These are mostly in the form of confirmation dialogues, like when the user wants to open a new data set or exit the program.

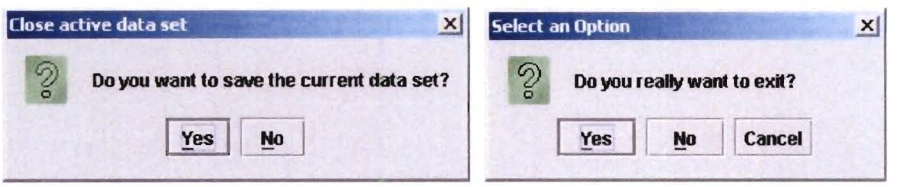


Figure 4.11 Examples of confirmation dialogues

Another aspect of system responsiveness that does not require any feedback from the user is evident where information is updated in real-time. Two such instances of this type of feedback are seen in the display of the current model elevation in the status bar and transparency level indicated above the transparency slider. Each time the user zooms in or out of the model viewport, the current elevation is updated as the change occurs, and when the wave transparency percentage is adjusted by moving the slider, the numerical representation is updated automatically. These updated display components are indicated in figure 4.12.

Figure 4.12 Real-time feedback

One performance issue regarding responsiveness of the drop-down menus was encountered. The 2-D user interface includes a component that is used to house the 3-D model viewport. When different graphical contexts were tested, it was found that those running as threads had a negative impact on the responsiveness of the system and the reaction time of the drop-down menus. When non-threaded graphical contexts were used, then the drop-down menus were no longer sluggish, and their responsiveness was improved dramatically.

The issues surrounding task conformance are user-centric and any concerns in this regard will be observed more fully during the system evaluation process, but the following aspects of user expectations have been catered for:

- providing a user friendly means of capturing SWAN command input parameters
- providing a means to generate the SWAN command text file in the correct format
- viewing the SWAN output file in a graphical context
- allowing free movement to view the graphical model from different perspectives
- providing a transparency tool to view multiple layers simultaneously
- providing a rendered model that describes the underlying data set with sufficient detail

5. DESIGN OF THE 3-D GRAPHICAL INTERFACE

The process of visualizing numerical data in a graphical context consists of the following three basic stages: [13]

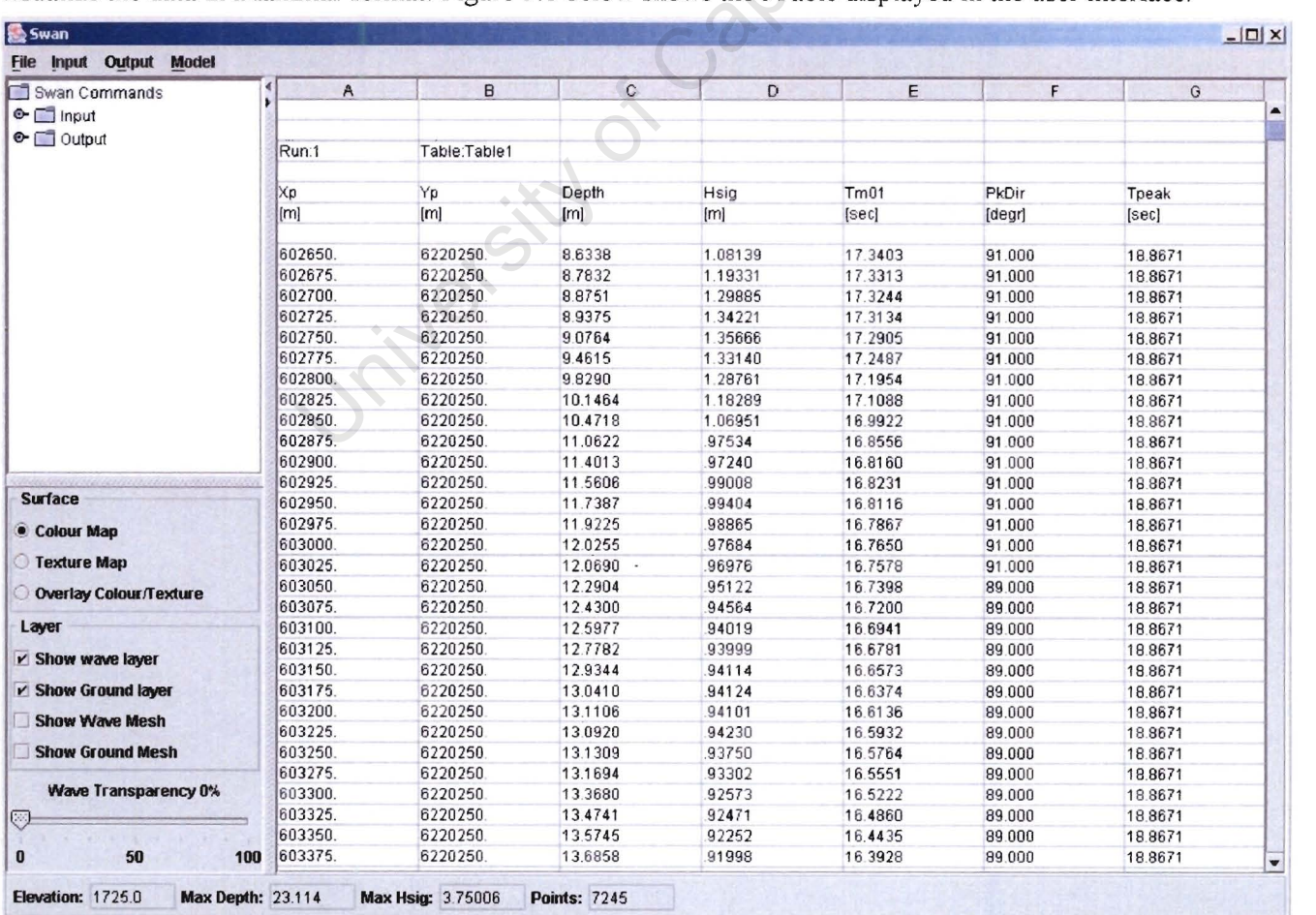
Data structure - the construction of a data set that represents the physical forms to be modelled
Model representation - the selection of an internal means of representing the model
Model rendering - the rendering of the model as an image on a graphical display

The graphical programming has been implemented using OpenGL with gl4Java acting as a Java wrapper for the native OpenGL libraries. The data structure used to formulate the 3-D model will be discussed, followed by rendering solutions and issues surrounding the implementation of transparency.

5.1. Data structure

The file used to generate the 3-D model consists of a list of coordinates for points and related attributes. This information needs to be read from disk and converted to an appropriate internal format to be used in the model generation process. The numerical data stored in the file is in a tabular format with each field separated by white space.

The JTable class is one of the Java Swing components which has been used to store the values contained within the output file. The JTable serves a dual purpose; firstly as a storage container for the data fields, and secondly as a means of displaying the data in a *spreadsheet* format. Using this type of display structure enables the users to visualize the data in a familiar format. Figure 3.1 below shows the JTable displayed in the user interface.



	A	B	C	D	E	F	G
	Run:1	Table:Table1					
	Xp [m]	Yp [m]	Depth [m]	Hsig [m]	Tm01 [sec]	PkDir [degr]	Tpeak [sec]
	602650.	6220250.	8.6338	1.08139	17.3403	91.000	18.8671
	602675.	6220250.	8.7832	1.19331	17.3313	91.000	18.8671
	602700.	6220250.	8.8751	1.29885	17.3244	91.000	18.8671
	602725.	6220250.	8.9375	1.34221	17.3134	91.000	18.8671
	602750.	6220250.	9.0764	1.35666	17.2905	91.000	18.8671
	602775.	6220250.	9.4615	1.33140	17.2487	91.000	18.8671
	602800.	6220250.	9.8290	1.28761	17.1954	91.000	18.8671
	602825.	6220250.	10.1464	1.18289	17.1088	91.000	18.8671
	602850.	6220250.	10.4718	1.06951	16.9922	91.000	18.8671
	602875.	6220250.	11.0622	.97534	16.8556	91.000	18.8671
	602900.	6220250.	11.4013	.97240	16.8160	91.000	18.8671
	602925.	6220250.	11.5606	.99008	16.8231	91.000	18.8671
	602950.	6220250.	11.7387	.99404	16.8116	91.000	18.8671
	602975.	6220250.	11.9225	.98865	16.7867	91.000	18.8671
	603000.	6220250.	12.0255	.97684	16.7650	91.000	18.8671
	603025.	6220250.	12.0690	.96976	16.7578	91.000	18.8671
	603050.	6220250.	12.2904	.95122	16.7398	89.000	18.8671
	603075.	6220250.	12.4300	.94564	16.7200	89.000	18.8671
	603100.	6220250.	12.5977	.94019	16.6941	89.000	18.8671
	603125.	6220250.	12.7782	.93999	16.6781	89.000	18.8671
	603150.	6220250.	12.9344	.94114	16.6573	89.000	18.8671
	603175.	6220250.	13.0410	.94124	16.6374	89.000	18.8671
	603200.	6220250.	13.1106	.94101	16.6136	89.000	18.8671
	603225.	6220250.	13.0920	.94230	16.5932	89.000	18.8671
	603250.	6220250.	13.1309	.93750	16.5764	89.000	18.8671
	603275.	6220250.	13.1694	.93302	16.5551	89.000	18.8671
	603300.	6220250.	13.3680	.92573	16.5222	89.000	18.8671
	603325.	6220250.	13.4741	.92471	16.4860	89.000	18.8671
	603350.	6220250.	13.5745	.92252	16.4435	89.000	18.8671
	603375.	6220250.	13.6858	.91998	16.3928	89.000	18.8671

Figure 5.1 JTable used to display coordinate detail list

The JTable is placed within a JScrollPane so that any row in the table can be viewed by adjusting the vertical scroll bar to the desired position.

Before the input data file is accepted into the program, it is checked as follows:

1. Each line of the file is read into a StringTokenizer using the white space character as delimiter. This object type takes a string and breaks it up into individual tokens by extracting the characters between the specified delimiter character(s). The fields in each line can then be extracted and individually referenced (the white space separating the fields will have been removed).
2. The column names "Xp", "Yp", "Depth" and "Hsig" are searched for, and if found, their column numbers are recorded. If the column names are not found, then the file is rejected and assumed to be of the wrong format. As long as the column headers are found, their position in the file does not matter since the determined column number references will be used when reading from the table.
3. The starting row of the numerical data is determined by finding the first row that contains only numerical values. This is achieved by trying to parse the strings as floating point numbers – if an exception is thrown, the next row is checked. Once each value in a row can be converted to a floating point number type then this is assumed to be the starting row of the numerical data.
4. Each value from the data file is read and stored in an array of strings that will constitute the data structure of the JTable. This is a double array that will contain the same number of columns and rows that are present in the data file.

Once the table has been created, the process of assembling the 3-D model data structure continues.
Each row in the table contains x, y and z values that will be extracted into a vertex list to be used when creating the OpenGL geometric primitives. Two levels of information are to be modelled; one based on the depth of the sea bed, and the other based on the significant wave height (Hsig) at each coordinate point.
To generate vertex lists for each of these surfaces, the x, y & z values need to be extracted from the data table as follows:

Ground profile: *Xp, Yp* and *Depth*
Wave surface: *Xp, Yp* and *Hsig*

A class called *Vertex3f* has been defined from which to create objects that represent the 3-D point information. Each vertex object is created by passing three float values representing the x, y and z coordinates of that point. Two arrays of Vertex3f objects are created; one for the ground profile points and the other for the wave surface points. The vertices representing the ground profile each have their *z* values negated to ensure that the surface to be generated will be drawn below sea level (zero). The wave layer vertices have positive *z* values to indicate their significant wave heights above zero.

The vertex lists encode a linear representation of a regular point grid. The distances between in the grid points in the x and y planes are constant. The values for the number of columns and rows in the grid as well as the constant x and y increments are determined by first establishing the number of columns in the model. Starting with the first vertex, the x value is read and compared to each subsequent x value. As soon as the same value is encountered then the next row in the model has been reached. The number of columns is then equal to the number of points with a changing x value. The incremental amount is equal the constant change in distance between the points along the x-axis. The number of rows is determined by comparing the y values, and when a *different* value is encountered, the next column has been reached. The number of rows is equal to the number of sets of vertices that have a constant y value and the incremental change is the distance between the points along the y-axis. If the vertices are viewed along the z-axis, then the points will form a regular grid.

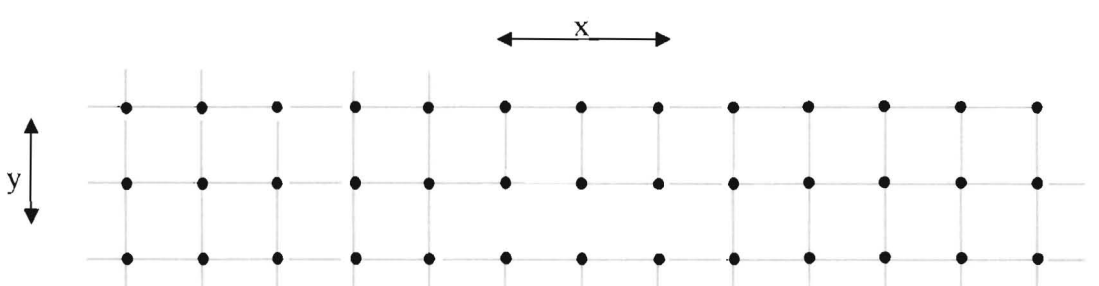


Figure 5.2 Regular grid format

5.2. Model description

To display a surface that covers each of the vertices, filled OpenGL geometric polygon types will have to be used. Since the grid can be seen as a series of rectangular shapes, the surface could be constructed using quadrilateral or triangular primitives. Using triangular shapes will require twice as many primitives to be rendered, but offers the advantage of providing more definition and clarity in the final rendering. This is due to the fact that using quadrilaterals could result in non-planar surface segments if the vertices do not lie within the same plane. On the other hand, triangular primitives bound by three vertices will always be planar, since a flat surface can be derived from any three points.

If individual triangles are used to render the surface then each requires three vertices to be passed. A more effective structure is to use triangle strips which share common vertices. In rendering a series of triangles, the first will require all three vertices to be passed, but subsequent triangles will only require one additional vertex each. This is due to the fact that adjacent triangles share two common vertices. Figure 5.3 shows numbered vertices followed by the resulting triangles.

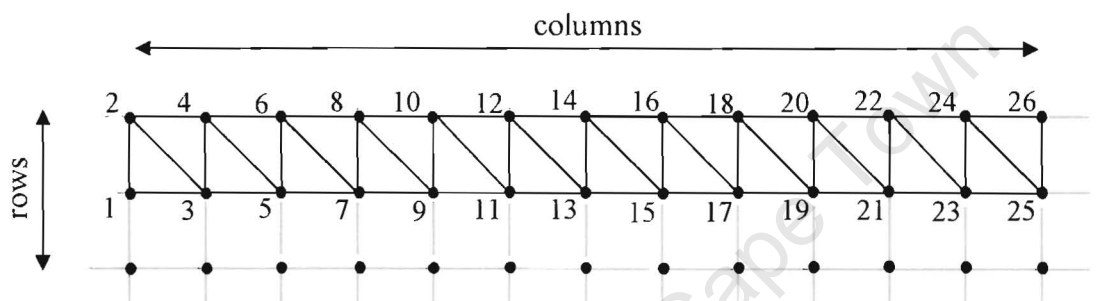


Figure 5.3 Triangle strip vertex arrangements

Resulting triangles from the above triangle strip:
1-2-3, 2-4-3, 3-4-5, 4-6-5, ... to ... 22-24-23, 23-24-25 and 24-26-25
To create sufficient triangles to render the entire surface will require (rows -1) triangle strips.

In order to achieve a smooth *Gouraud* shading of the final 3-D model, the normals at the vertices need to be calculated. These are determined by first calculating the normals for each triangle in the mesh and then taking the average around each vertex. Care must be taken to ensure that the front facing normals are calculated for each triangle. This is achieved by using counter-clockwise winding of the vertices used to calculate the normals.

A *Vector3f* class has been defined for dealing with functions like adding vectors, calculating the cross product of two vectors and determining the magnitude of a vector. The steps involved in calculating the front face triangle normals involves taking the cross-product of two vectors in a counter-clockwise winding, so if the normal of Vector 1 and Vector 2 below is to be determined:

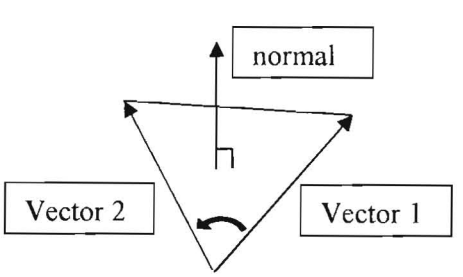


Figure 5.4 Front-face triangle normal

Assuming Vector 1 = (a, b, c) and Vector 2 = (x, y, z), then the normal is determined by taking their cross-product:

normal = Vector 1 x Vector 2
normal = (bz – cy, cx – az, ay – bx)

and the unit normal vector is established by dividing the vector components by the magnitude $\sqrt{(x^2 + y^2 + z^2)}$

Once the normal for each triangle has been calculated, then the normal at each vertex can be determined by taking the average normal of the triangles that share the vertex, taking note of the number of triangles adjacent to each vertex:

Top right and bottom left grid points: one triangle
Top left and bottom right grid points: two adjacent triangles
Other grid points along the edges: three adjacent triangles
Points inside the grid: six adjacent triangles

Figure 5.5 shows the number of triangles that surround each vertex. The different triangle arrangements have been shaded to illustrate the relative numbers as described above.

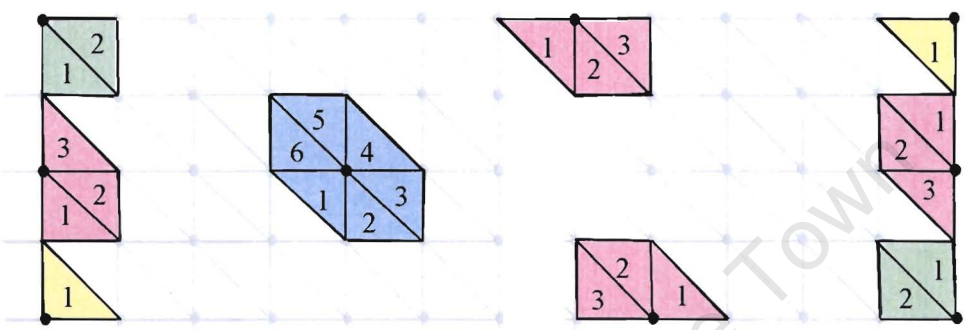


Figure 5.5 Number of triangles per vertex

Once the vertex normals have all been calculated, the data structure for model representation is complete, and consists of the following object arrays for each surface:

- Vertex list as an array of *Vertex3f* objects
- Triangle normal list as an array of *Vector3f* objects
- Vertex normal list as an array of *Vector3f* objects

When the surfaces are assembled using this data structure, the vertices and related normals are processed using triangle strips for rendered surfaces, and line strips for wire-frame layers. Triangle strips, as shown in figure 5.3, are processed one row at a time until the entire surface has been constructed. The vertices are handled in the order indicated in figure 5.3 in a looping structure such that each iteration processes two vertex points.

The construction of each wire-frame surface is achieved by using two sets of line strips. One is used to draw lines representing each grid row, and the other to stitch the rows together in a triangular formation. These are illustrated in figures 5.6 to 5.8. The vertices are processed one at a time and the result is a continuous line joining each of the vertex points.

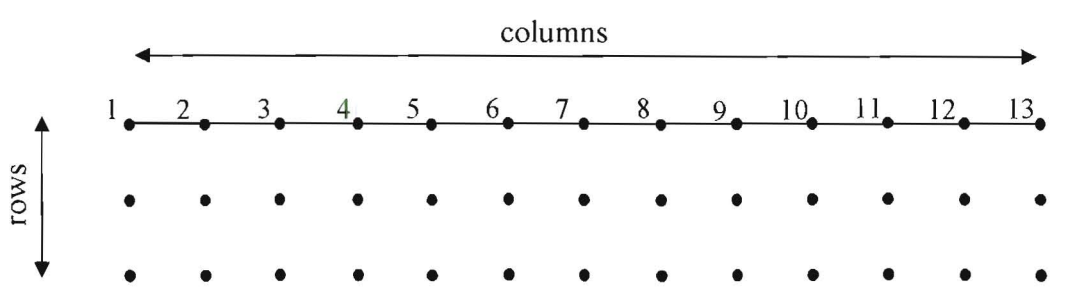


Figure 5.6 Line strip drawn for each row

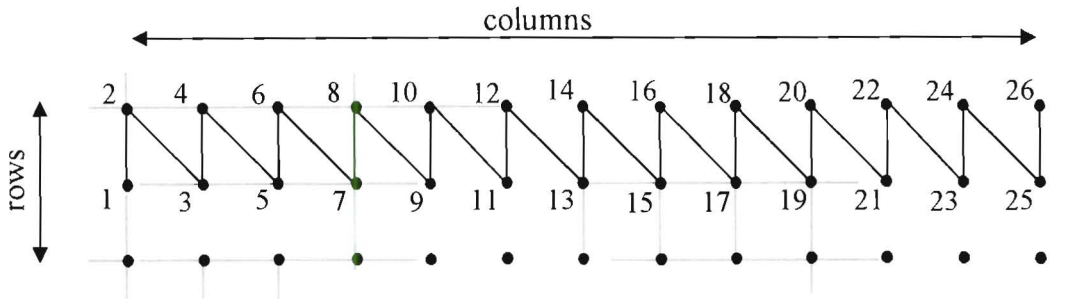


Figure 5.7 Line strip used to join two rows

When the two line strips shown in figures 5.6 and 5.7 are superimposed, a triangular wire-frame surface similar to that used to construct the triangle strips emerges.

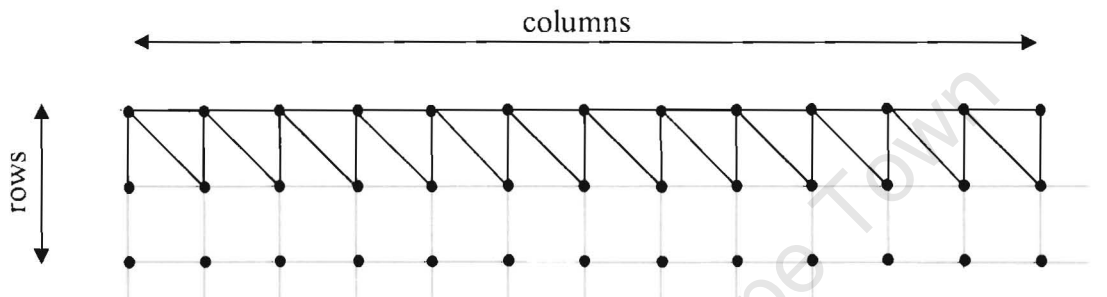


Figure 5.8 Wire-frame constructed from two line strips

The wire-frame layer is drawn one row at a time until the entire surface has been modelled.

5.3. Multiple surface layering

The surfaces to be rendered represent both static and dynamic objects. Static layers are those surfaces that do not have adjustable attributes; i.e.

- Ground surface profile
- Ground surface mesh (wire-frame)
- Wave surface mesh (wire-frame)

The term *mesh* as applied to the surface layers is used to indicate a wire-frame outline.

Once these layers have been constructed they can be turned on or off, with no intermediate state. The only property that can change is the appearance depending on whether the model is being rendered with a colour or texture map.

The wave surface is a dynamic layer which has an adjustable element of transparency. This layer can be turned on or off, but can also assume intermediate states depending on the current level of transparency that is set by the user.

Whenever an element of the rendered model changes through user interaction, the displayed image has to be repainted to the screen. Depending on the size and complexity of the model, this process can demand a fair amount of computational processing. So the idea of reducing the required processing was investigated, and it was found that OpenGL can render images in *immediate mode* and *retained mode* [1].

When rendering the displayed image using immediate mode, the vertices and related attributes are processed to draw the specified geometric shapes on the display. Each time the display needs to be refreshed the geometric shapes have to be re-processed and redrawn. This type of processing is used for the wave surface layer that has adjustable transparency. But what about the other displayed layers that do not have adjustable properties? The ground surface layer as well as ground and wave mesh layers are rendered in retained mode through the use of *display lists*.

Using display lists allows the relevant objects to be defined once, and stored in an area of memory for re-use at a later stage. Whenever the object is required for rendering it is called from memory and displayed. By not having to re-process the vertices of those layers that do not have changing properties, the load on the graphics processor can be reduced. These compiled lists are then available to be called when required. Whenever the displayed image needs to be repainted, a check is made to determine which layers are to be shown and the relevant display lists are called.

The process of layering multiple surfaces requires that they are ordered according to depth. This is to ensure that when the user is viewing the model, the ground surface is hidden beneath the wave layer, unless the wave transparency is set to a value higher than zero. As the wave transparency is adjusted, the underlying ground surface becomes more prominent.

The ground and wave mesh layers are generated slightly proud of their underlying feature surfaces to prevent them from merging, or the wire-frame outlines getting lost inside the surfaces they are highlighting. This phenomenon is known as *stitching* or *z-fighting*. Since the rendered feature surface and wire-frame details are based on the same vertices, they share similar depth z-values. When the image is displayed in the viewport, there can be times when the rendered surface is draw in front of the wire-frame, and other times when the wire-frame is drawn in front. To ensure that the wire-frame is always shown in front of the underlying surface, the depth value at each vertex point is elevated slightly. The net result is that the undesirable stitching effect is removed.

Figures 5.9 and 5.10 illustrate the effect of stitching.

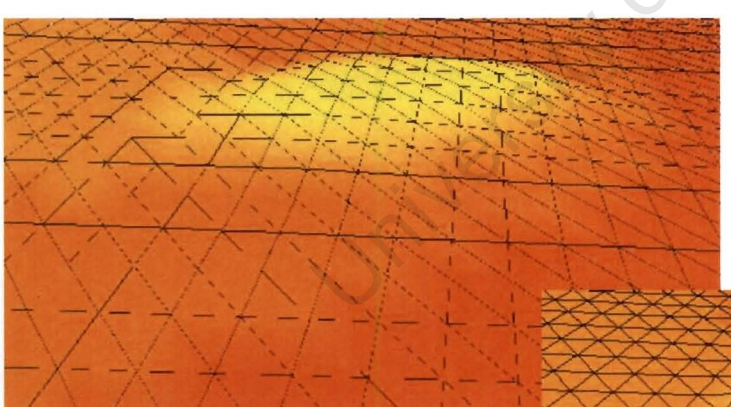


Figure 5.9 View of the wire-frame mesh applied to the ground surface showing the effect of stitching

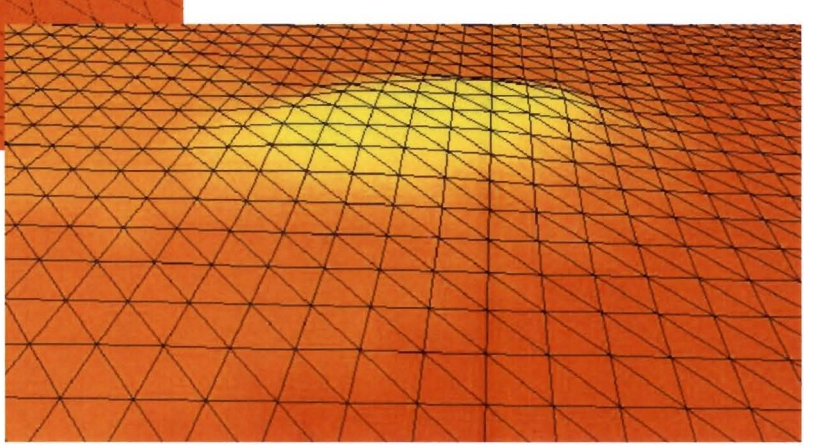


Figure 5.10 View of the wire-frame mesh applied to the ground surface. Stitching has been removed.

5.4. Model Rendering

The 3-D model needs to be rendered so that the user is able to identify the different properties of the ground and wave surfaces. These properties include differences in depth and contour patterns. To aid this visualization two rendering options have been used; one based on a colour map, and the other based on a texture map.

Using colour to illustrate varying depth values or patterns is referred to by a technique known as *pseudo colouring* , and is described as "the technique of representing continuously varying map values using a sequence of colors" [14]. Model regions are rendered in varying colours to assist with the visualization of the following:

- perception of ground and wave features
- recognition of high and low points in the model

Using a colour map, the 3-D graphic can be rendered with different colours or shades depending on the different ground depths or wave heights that are present in each region of the model. Each model file that is rendered may have differing depth and height ranges, so the colour regions have been set up on a relative scale of one to ten and are not dependent on hard-coded values. Each layer has a light colour for indicating minimum levels and a dark colour for maximum levels.

The different light and dark colours for the wave and ground surface layers are as follows:

Wave surface

	Light Colour This light blue shade is used to indicate those wave regions that have the minimum wave heights on a scale of one to ten.
	Dark Colour This dark blue shade is used to indicate those wave regions that have the maximum wave heights on a scale of one to ten.

Figure 5.11 Light and dark colour values for wave surface

Ground profile surface

	Light Colour This yellow colour is used to indicate those ground surface regions that have the minimum depths on a scale of one to ten.
	Dark Colour This red colour is used to indicate those ground surface regions that have the maximum depths on a scale of one to ten.

Figure 5.12 Light and dark colour values for ground profile surface

In order to achieve a spectrum of colours that are perceived to be ordered, a visualization technique outlined by Ware has been applied. He describes the technique as follows: "a perceptually ordered sequence will result from a series of colors that monotonically increases or decreases with respect to one or more of the color opponent channels" [14].

The colours are described by specifying values for the red, green and blue channels that make up the each RGB colour. Each colour is set by keeping two of the channels constant, and adjusting the third on a sliding scale. Following are the colour palettes that indicate the different colours used in the model rendering process: (the percentage saturation of each colour channel is indicated)

R%	G%	B%	Wave Height
0	100	100	
0	90	100	
0	80	100	
0	70	100	
0	60	100	
0	50	100	
0	40	100	
0	30	100	
0	20	100	
0	0	100	

Shallow

Deep

Ground Depth	R%	G%	B%
	100	100	0
	100	90	0
	100	80	0
	100	70	0
	100	60	0
	100	50	0
	100	40	0
	100	30	0
	100	20	0
	100	0	0

Figure 5.13 Colour map showing gradient levels

The resulting height or depth gradients are represented by colour bands of the relevant colour. The minimum value in each case is assumed to be zero, and the maximum is determined by recording the highest value present in the model file. This means that the visual representations of the different wave height or ground depths are constrained to a range that is proportional to the maximum value present in the model.

By constraining depth or height ranges to a particular colour, noticeable borders will be displayed at the transitions from one colour region to another. These visible boundaries will help the user to identify high and low areas in the model, as well as any ground features or wave formations that get outlined in the process.

Wave colours vary from light blue RGB (0, 255, 255) to dark blue RGB (0, 0, 255). The value for the green channel is reduced on a sliding scale. The changes in shades of blue from light to dark have been used to approximate the real world appearance of water where shallow areas appear lighter and deeper areas appear darker blue, as shown below in figure 5.14.

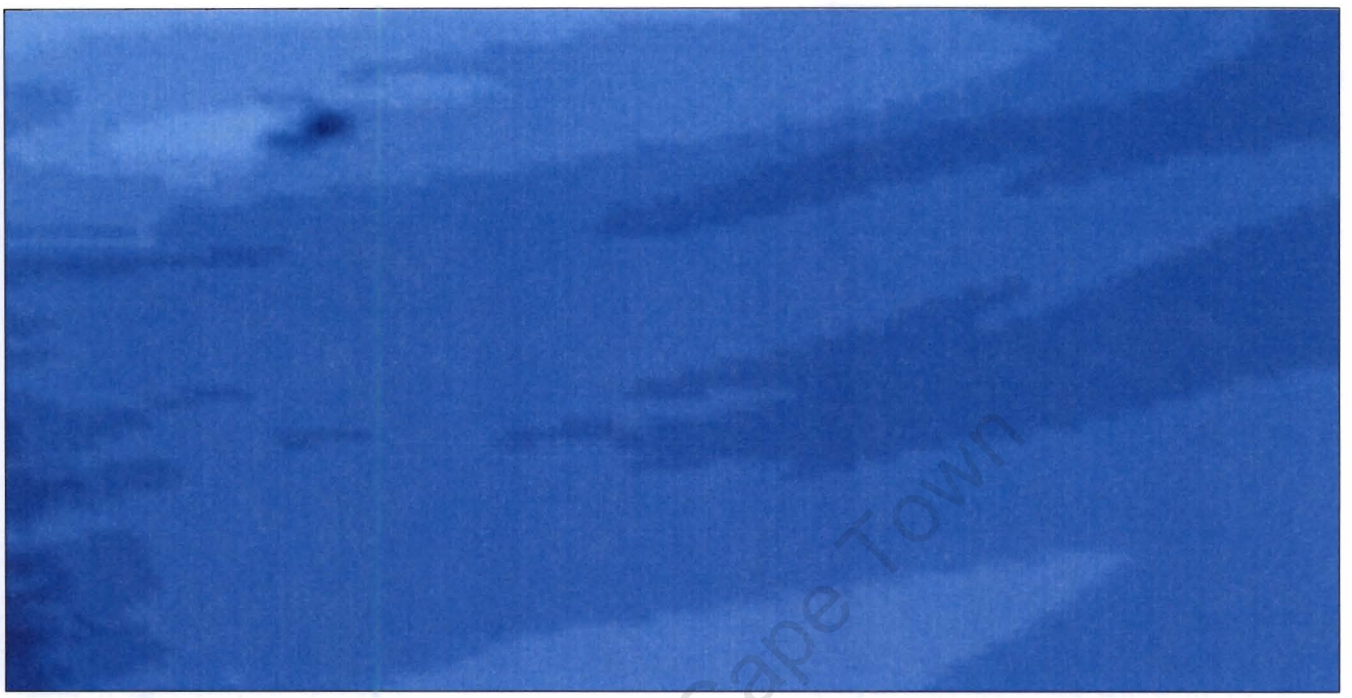


Figure 5.14 Wave surface rendered with colour map showing shallow and deep water areas

Ground colours vary from yellow RGB (255, 255, 0) to red RGB (255, 0, 0). The value for the green channel is reduced on a sliding scale. Figure 5.15 shows the ground profile rendered with a colour map.

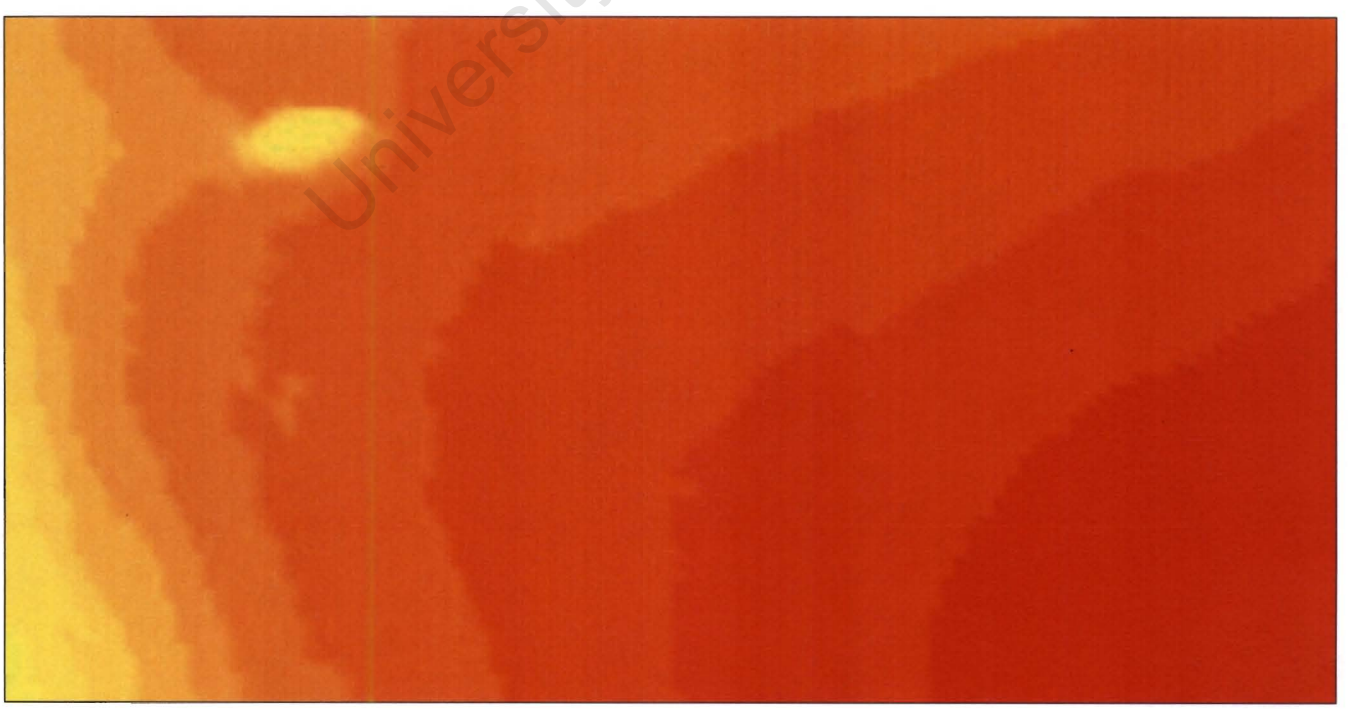


Figure 5.15 Ground profile rendered with colour map showing low and high regions

The boundaries between the different colour ranges result in visible edges, indicating the steps in depth or height gradation.

Rendering the model using a texture map yields different results that highlight the ground profile and wave height patterns more vividly. The texture used is based on an image such that a more uneven colour spread results in distinct boundaries between the regions. This aids the visualization from the perspective of identifying more detailed patterns and formations.

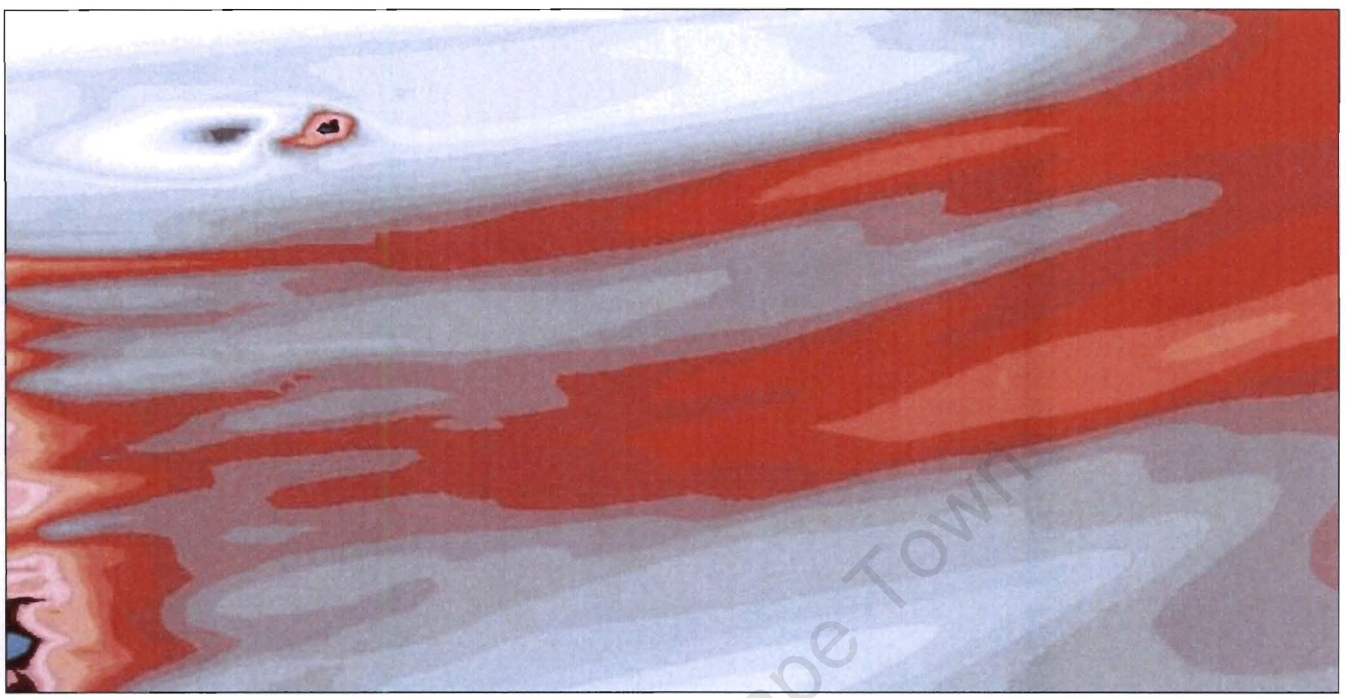


Figure 5.16 Wave Layer shown with Texture Map surface

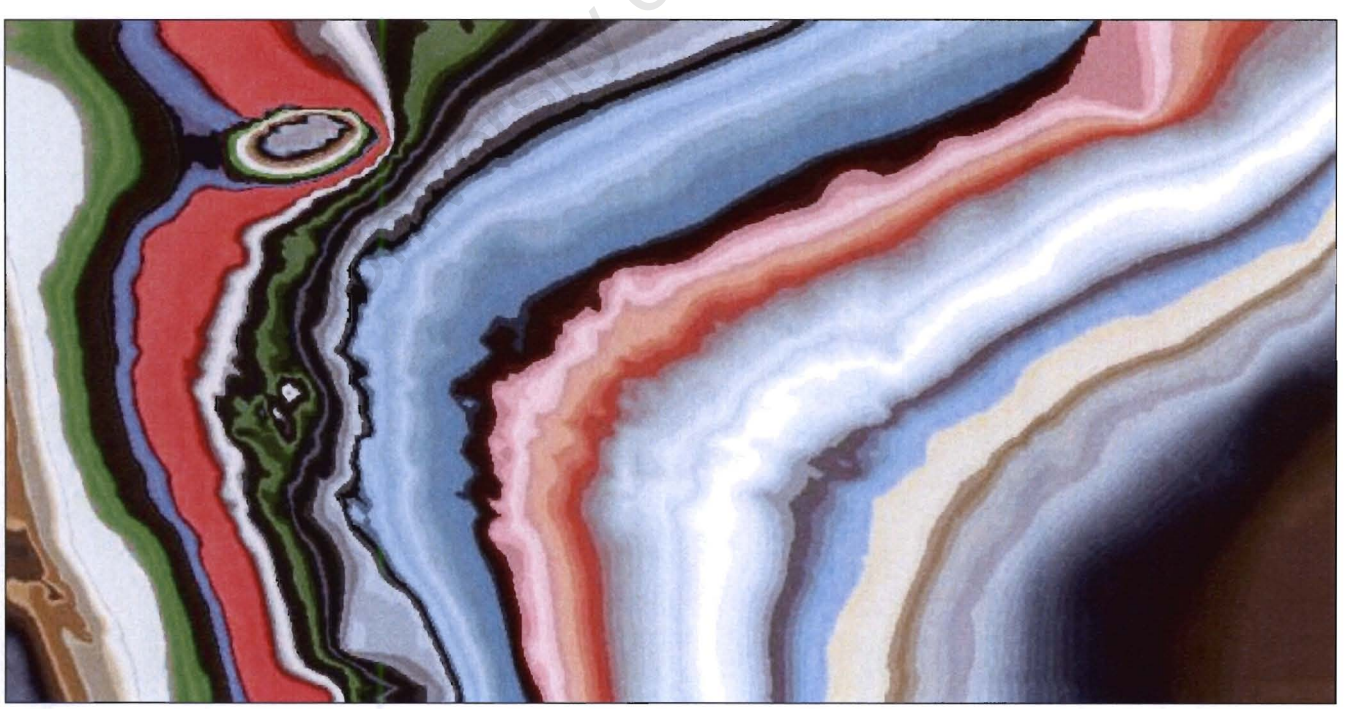


Figure 5.17 Ground profile shown with Texture Map surface

The texture map rendering of the model can be used to view any relationships that may exist between the surfaces. The distinct boundary lines aid the visualization process especially when the wave transparency is adjusted.

5.5. Transparency

The idea of using semi-transparency to assist in identifying object relationships in a 3-D model is outlined by Zhai *et al* [11] as he writes "One of the key challenges in 3-D interface design is to effectively reveal spatial relationships among objects within a 3-D space, particularly in the depth dimension". In his discussion on the properties of semi-transparency, Zhai covers two aspects that are relevant to the design of this system:

- semi-transparent surfaces do not block the view of underlying objects and allow an awareness of the background information to be maintained
- relational and discrete depth information about the position of a semi-transparent surface, relative to others, can be observed [11].

Transparency, as applied in this project, is used to assist users with identifying relationships that may exist between the wave surface and underlying ground profile.

To enable the rendering of transparent objects using OpenGL, the technique of *alpha-blending* [1] has been used. An additional alpha (α) channel is added to the standard RGB colour model, resulting in colours that are defined as RGBα. The RGB channels still refer to the red, green and blue aspects of the colour, and the α channel is used to determine relative transparency. To use the RGBα colour mode OpenGL blending needs to be enabled and a blending function specified. Blending is required since the colour to be displayed on the screen will be a combination of colours from different objects.

The range of transparency used in OpenGL ranges from opaque (α = 1) to fully translucent (α = 0). To make the value selection more meaningful to the user, a slider tool has been provided where a percentage value from 0 to 100 can be selected. The chosen percentage value is then interpolated to fall within the range of 0 to 1.

Opaque objects block all light from passing through so that any objects rendered behind it are obscured from view. Translucent objects with α = 0 allow all light to pass through such that they appear invisible. An α value between 0 and 1 results in a corresponding amount of light passing through the object, thus giving the effect of relative transparency. The OpenGL blend function is set up by specifying source and destination blend factors.

The blend factors have been specified as follows:

- source factor = source α. This refers to the alpha value of the display pixel colour to be drawn.
- destination factor = (1- source α). This refers to the corresponding currently stored alpha value for the display pixel.

These factors have been chosen to ensure that "both transparent and opaque polygons are handled correctly and that neither colors nor opacities can saturate" [1].

The resulting colour and transparency values are found by combining the source and destination colour channels with the source and destination factors specified above.
If the source colour is specified as $(R_s, G_s, B_s, \alpha_s)$ and the destination as $(R_d, G_d, B_d, \alpha_d)$ then the resulting colour is calculated as follows:

$$(R, G, B, \alpha) = (R_s\alpha_s + R_d(1-\alpha_s), G_s\alpha_s + G_d(1-\alpha_s), B_s\alpha_s + B_d(1-\alpha_s), \alpha_s\alpha_s + \alpha_d(1-\alpha_s))$$

Figure 5.18 Calculating composite colour and α channel values

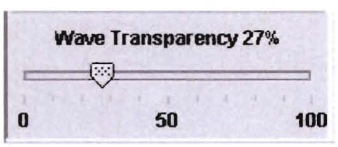
The resulting colour and α values still need to fall within the range 0 to 1, so if a combined colour value would exceed these bounds, it is restricted to 0 or 1.

Those surfaces that are not transparent have been hard-coded with an α value of 1, and the surface representing the wave layer with a variable α value that changes according to a user-selectable value.

Using alpha-blending as a means to view multiple opaque and semi-transparent objects has shortcomings. OpenGL maintains a z-buffer that is used to store information about the depth attribute of each pixel. The problem is that the z-buffer does not record sufficient information to determine the relative ordering of objects when multiple transparent and opaque objects are combined [2]. This means that the rendered image may produce different results depending on the order in which the objects are drawn. To work around this problem, the opaque objects can be rendered first followed by the semi-transparent ones. The translucent objects will also have to be ordered to achieve a consistent rendering. This does not strictly apply to the system being developed, since only a single semi-transparent layer is being modelled and this is always positioned as the top surface. The different surface layers are sorted from bottom to top (back to front) to ensure that the alpha-blending results produce the desired effect.

The surface to which adjustable transparency can be applied is the wave layer which is always rendered above the underlying ground surface. This means that once the opaque ground surface has been drawn, the semi-transparent wave layer is rendered using alpha-blending to combine the colours of the ground and wave surfaces proportional to the alpha value at each point.

The user is able to dynamically adjust the level of transparency that is applied to the wave layer by moving a slider to indicate a particular percentage.



The slider tool shown in figure 5.19 displays the current transparency percentage to adequately inform the user of the current level of wave transparency.

Figure 5.19 Transparency slider tool

Figure 5.20 shows example of viewing the wave and ground layers using texture mapping and the wave transparency set to 25%

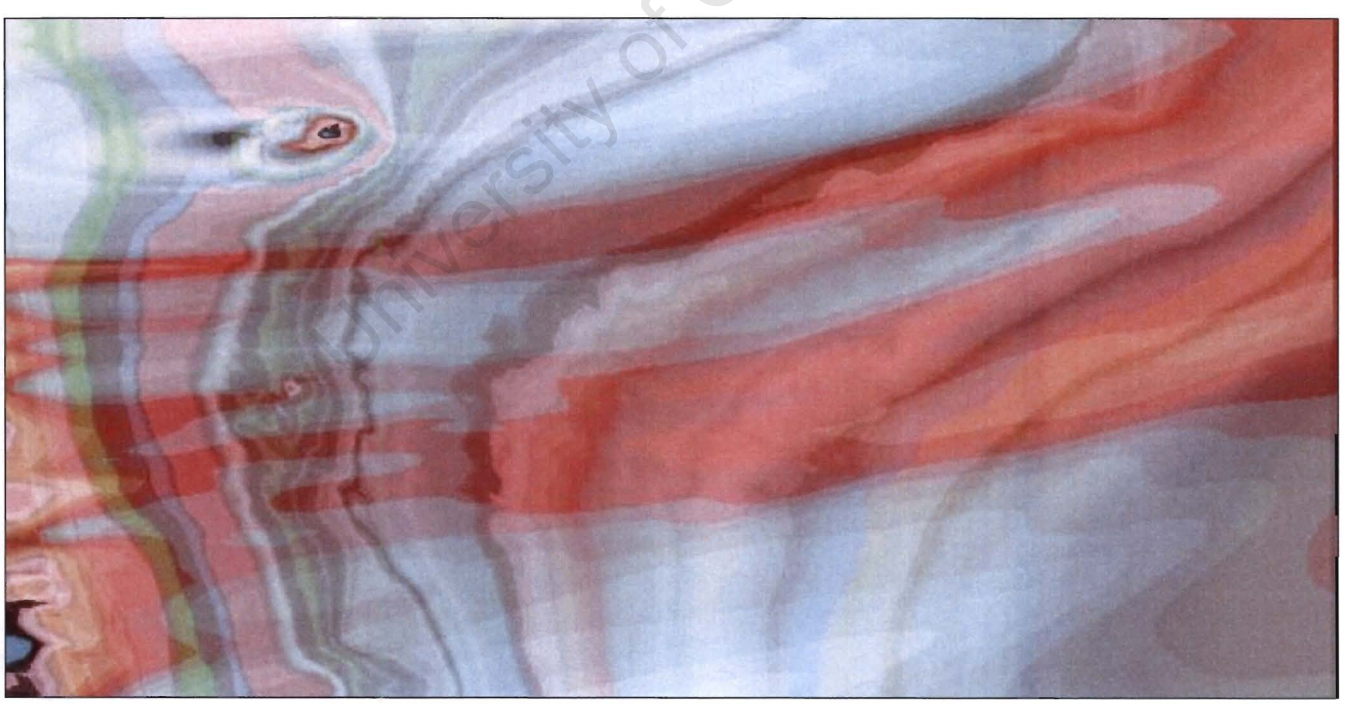


Figure 5.20 Wave and ground surfaces viewed together with 25% wave transparency applied

6. EVALUATION

The evaluation of the system involves a process of usability testing to ensure that the users are able to use the software effectively. Any issues relating to problems that users have with the system should be documented in order to implement any improvements that are deemed necessary.

6.1. Usability testing

User interfaces can be evaluated in one of four basic ways, as outlined by Nielsen: [15]

- **Automatic evaluation** where the usability is computed by a program according to some user interface specifications.
- **Empirical evaluation** where the user interface is tested by real users to find any problems that may exist.
- **Formal evaluation** where a strict set of formulae is used to calculate interface usability.
- **Informal evaluation** where no set guidelines are applied, but evaluators adjudicate an interface according to their own experience.

The use of automatic and formal evaluation methods could require a substantial investment in time and money to set up the required scientific guideline framework and suitable environment. Empirical and informal methods would require less resources since users can test the interface in their own existing working environments. Involving users in the testing phase has the benefit of making them feel that they have been part of the development process. Their input is important and any comments made should be taken into consideration in order to improve the usability of the system.

The user interface to be tested has been developed for a specific domain and therefore some prior knowledge of what the system is about will assist with the testing process. The best people to test the system are those who are familiar with the problem domain, and are also familiar with basic human-computer interaction needs. Expert users can be described as those who are familiar with the problem domain.

To decide which evaluation procedure to follow the following list highlights those that involve testing by expert users: [16]

- **Heuristic evaluation**. Expert reviewers make comments on the usability of an interface according to a list of heuristics to determine problem areas. This fairly informal inspection method involves a few evaluators and is inexpensive and relatively easy to implement.
- **Guidelines review**. The interface is checked for compliance with organizational guideline documents. This inspection method can be very time consuming if the guideline documents are lengthy.
- **Consistency inspection**. Experts check a series of interfaces to ensure that a consistent design approach has been implemented.
- **Cognitive walkthrough**. Experts simulate users walking through a system to achieve certain goals. Typical use cases and tasks are carried out by the evaluators.
- **Formal usability inspection**. The interface is discussed in a meeting by a team of experts and chaired by a moderator. The merits and weaknesses of the interface are presented and discussed.

In a comparison of four different techniques: heuristic evaluation, software guidelines, cognitive walkthroughs and usability testing, Jeffries *et al* [17] found heuristic evaluation to be the most effective. In their comparative testing, the heuristic evaluation process identified more problems than the other methods and at the lowest cost.

Heuristic evaluation is referred to as a *discount* usability engineering method because of the fact that it is cheap and easy to implement. In a discussion on heuristic evaluation, Nielsen and Molich [18] recommend that between three and five evaluators are used. The average number of usability problems found grows rapidly with up to five evaluators, and then starts to diminish. Using a minimum of three evaluators should provide sufficient insight into any usability problems that exist.

6.2. Heuristic evaluation process

The usability testing of the Swan interface program was carried by three expert users through a process of heuristic evaluation. The evaluators are domain experts with experience in interacting with other 3-D graphical software and will therefore have some prior knowledge of particular usability issues that may be encountered.

The evaluators were each presented with an evaluation pack (see Appendix C) which contains the list of heuristics and a user guide explaining the program operation. They were also given a piece of paper on which to write their comments relating to the usability of the system. The users were asked to critique the system not only according to the guidelines presented but also based on any other criteria that they saw fit.

Heuristics are guidelines that relate to recognized usability principles. Ten such heuristics were used as a framework for usability testing. These were originally developed by Molich and Nielsen [19] and later refined to the following: [10]

1. Visibility of system status.
The system should provide sufficient feedback so that users are aware of what is happening.

2. Match between system and the real world.
Information should be displayed in a logical format that is meaningful to the users.

3. User control and freedom.
Users are in control of the system, and need to be able to recover from unwanted states without too much difficulty.

4. Consistency and standards.
Once users have come across particular words, situations or actions, they should feel comfortable that similar standards have been applied across the system

5. Error prevention.
Measures should be put in place to prevent avoidable errors from happening.

6. Recognition rather than recall.
The user should not need to memorise system actions, but clearly presented options should be made available. Assistance in the form of clearly presented instructions should be made available when necessary.

7. Flexibility and efficiency of use.
Accelerators and shortcut keys should be made available and provision should be made for different methods of navigation and option selections. This can improve the interaction for more experienced users.

8. Aesthetic and minimalist design.
Only relevant information should be presented. Additional detail should be kept to a minimum, so as not to detract from the essential data.

9. Help users recognize, diagnose, and recover from errors.
Error messages should be displayed using plain language, and explanations should be offered as to the reason for the problem occurring. Possible solutions should be offered to assist the user to recover from an exceptional state.

10. Help and documentation.
Documentation that is provided should be helpful and easy to understand by the users. Information should be focused and not too verbose.

6.3. Findings

The result of the evaluation process is a list of written comments received from each of the users. The comments are documented as recorded by the evaluators.
Copies of the original comment sheets can be found in Appendix 4.

User 1

Guideline Number	Comment
1.	When selecting "save" or "Write SWAN Command File", it would make the selection clearer if the required file type (extension) was displayed in the file name box.
3.	Rotating, panning & zooming is a bit difficult to control. An "undo" command would be useful to return to the previous view.
6.	It is difficult to remember which keys should be pressed for panning, rotating, zooming, etc. A set of buttons on the toolbar would help.

User 2

1. Visibility of system status: Good
2. Match between system and real world: Good
3. User control and freedom: Good
4. Consistency and standards: Good
5. Error prevention: Good
6. Recognition rather than recall: Good recognition
7. Flexibility and efficiency of use: Good
8. Aesthetic and minimalist design: Good
9. Help users recognize, diagnose and recover from errors: Acceptable, Good, but difficult to generate an error
10. Help and documentation: Help and documentation needs to be added.

User 3

1. When opening a data file – should only list 'file types' that can be opened in the program.
2. View Table: The header lines Xp, Yp, Depth etc should remain visible when scrolling down the table.
3. Commands are not visible as "buttons" on the screen, eg zoom/pan etc..
4. When hitting X the program closes without prompting to save.
5. Option: When pressing the key to "pan a view" the icon could change to a hand symbol?
6. Option: When moving the mouse over the model – the elevation / Depth could reflect current position? – Probably not possible at this stage.
7. Left and right buttons have same function in panning.

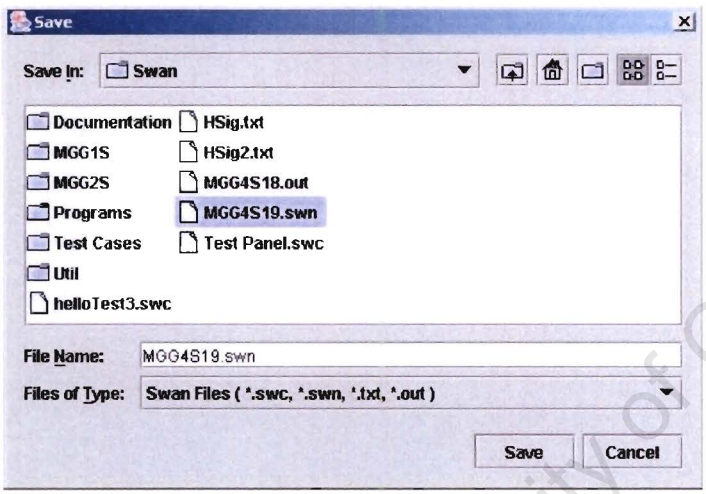
On the whole the users were quite happy with the screen layout and program functionality. The wave transparency tool was found to be effective, and none of the users had difficulty in selecting between the various rendering options and layer visibility settings. Positive feedback was received on the functionality of the program, and the 3-D graphical display of the rendered model.

When compiling the results of the evaluation process, the following areas of concern were noted by more than one of the users:

- The file types need to be filtered so that when opening or saving a file only those types relevant to the program are displayed in the file selection dialogue boxes.
- Buttons are needed to select between the different 3-D navigational options; zoom, pan and rotate.

These two problematic areas have been addressed as follows:

1. Display only relevant file types



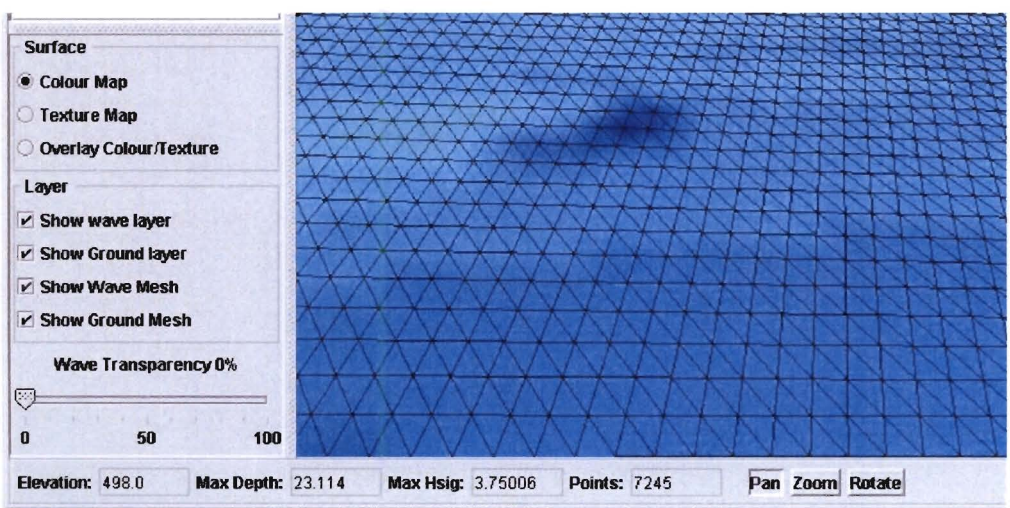
A file filter has been implemented to show only the relevant file types in the *open* and *save* file dialogue boxes.

Figure 6.1 shows the improved *save* dialogue where only the file types *.swc, *.swn, *.txt and *.out are displayed (all other file types are hidden from view).

Figure 6.1 File selection dialogue with file filter applied

2. Display buttons to select between pan, zoom and rotate options

Three buttons have been added to the status bar, representing the pan, zoom and rotate functions for model navigation. They have been implemented so that they are mutually exclusive in that only one can be pressed at a time. Figure 6.2 shows the buttons positioned on the status bar:



The buttons have been rendered with two visible states; a raised bevel border for *off* and a lowered bevel border for *on*. This arrangement helps the user to check which option has been selected.

Figure 6.2 Status bar showing pan, zoom and rotate buttons

7. CONCLUSION

The task of developing an interactive user-centric graphical interface based on an existing scientific model was found to contain a number of elements that could apply equally to other environments. These include

- providing a means of collecting large amounts of data in an effective manner
- reducing or eliminating the need for the user to check input data formats
- using error prevention rather than error correction
- providing usable interface controls
- integrating OpenGL graphics effectively in a Java environment
- rendering the 3-D graphical model according to different visualization needs
- layering different surfaces in a 3-D model with each able to be turned on or off
- visualizing relationships between 3-D surfaces through the use of transparency

The SWAN refraction model upon which this dissertation is based has particular input data format requirements. The means of designing an intuitive data capture mechanism for gathering the required input was examined, and a system of modular control panels was developed. The quantity of data to be captured can be quite substantial, so a means of preventing errors was implemented, as follows:

- default values are set, where applicable, to remove the necessity of the user to enter these. If necessary, these values can be changed to suit user requirements.
- key listeners were assigned on numerical fields to check for integer or floating point formats. If the user tries to type a character that would invalidate the number format, then the key stroke is *consumed*.
- where the user needs to enter one value from a set or group, then drop-down combo boxes were used to allow only a single selection. Here the user is not typing in a value, but selecting from a given list of options, thereby reducing the chance of error.
- the states of input fields are monitored to ensure that only appropriate ones are enabled. This implies that data fields are disabled to prevent user selection when their use would result in inappropriate data capture.
- consistently designed input panels were applied to enable the user to predict the usability of the system through familiarity.

The principle of disabling components until such a time as their selection is appropriate was also applied to the menu structure. Certain menu options are only made available once prerequisite conditions have been met. For example, the 3-D graphical model or coordinate table views can only be selected once the underlying data file has been loaded. Preventing errors rather than providing a means of correcting them has the added advantage of not presenting the user with unnecessary error notifications.

Common user interface components which may be regarded as *general* in their usage were applied to promote effective learnability through familiarity and predictability. These components include drop-down menus, combo-boxes, radio buttons and the usage of a tree-like structure for navigation of the SWAN input command group selection. The use of a slider for transparency selection was used, with adequate labelling and feedback to assist the user with its usage. During the evaluation of the system, the users did not find any difficulty in using the various selection options and data capture mechanisms.

Two different application programming interfaces for including OpenGL graphics in a Java environment were examined, and it was found that *gl4Java* allowed for a smoother integration with the user interface. This was partly due to the fact that this API allows for more direct access to the native OpenGL libraries, and also its provision of drawing components that inherit Java swing functionality. Swing components were chosen in preference to AWT interface components for their added functionality and appeal. These components also have the advantage of a platform-independent look and feel. This implies that in a mixed environment the program will look the same, regardless of the underlying platform.

Visualization techniques using OpenGL were researched, and an appropriate data structure using triangle strips and line strips was developed. Utility classes for *Vertex3f* and *Vector3f* objects were implemented to deal with the preparation of the data structure arrays. The coordinate list containing vertex and associated detail information was stored in a *JTable* object which was used for displaying the tabular information on the user interface, and also as a means of extracting the individual data items to be used in the vertex lists.

OpenGL triangle strips were used for assembling the 3-D rendered surfaces and line strips for the superimposed wire-frame mesh layers, the latter offset slightly to improve definition. Each of the generated surfaces could be viewed independently or jointly through the provision of layer selection controls. These controls were found to be effective and useful by allowing the user the flexibility to choose which surface(s) to display.

Visualization of the 3-D model was enhanced by the provision of a transparency modulator. Object transparency was implemented by blending source and destination pixel RGB colours with an added alpha channel accounting for the transparency factor. A slider control was implemented whereby the user could select the percentage transparency of the top wave surface. The relationships between the wave layer and underlying ground profile could then be examined in an interactive manner.

The model could either be rendered using a colour map to highlight depth levels or texture map to highlight variance patterns. Each of these rendering options offered a different perspective when viewed using the transparency tool. During the evaluation process, the users were able to use the transparency slider without difficulty, and comments were verbalized regarding effectiveness and helpfulness of the transparency element as an aid to visualization.

Navigation of the 3-D model was provided through the use of various mouse and keyboard controls to allow for panning, zooming and rotation. Two of the evaluators requested that buttons be added to the interface to reduce the need to memorize the different control options. Three buttons were added to the status bar, one each for panning, zooming and rotating the model and these reduced the memorization requirement by providing visual feedback as to the currently selected mode.

7.1. Future work

Common areas of concern with regard to usability issues were implemented following the evaluation process, and others that were noted can be added at a later stage. These include:

- the addition of a toolbar with icons for option selections
- the provision for undoing actions, possibly allowing for a number of steps to be undone
- the addition of a *help* menu to provide for general usage instructions and related information
- improving the feedback with regard to current navigational location within the model through the addition of x and y coordinates alongside the displayed elevation.

Additional data capture screens will be added, following similar design principles to those discussed, and further improvements to the system will include the visualization of additional details as computed by the SWAN refraction model for *inter alia* wave period and direction.

Visualization of further surface layers will add additional complexity to the model through the quantity and type of information that may be required to be viewed at any time. Research will have to be conducted into finding appropriate means of visualizing more complex models whilst maintaining acceptable levels of clarity and usability.

8. REFERENCES

1. E. Angel. *Interactive Computer Graphics A Top-Down Approach Using OpenGL*. Addison-Wesley, 3rd Edition, 2003.
2. J.D. Foley, A. van Dam, S.K. Feiner and J.F. Hughes. *Computer Graphics: Principles and Practice in C*. Addison-Wesley, 2nd Edition, 1995.
3. D. Selman. *JAVA 3D Programming*. Manning Publications, 2002.
4. H. Sowizral, K. Rushforth and M. Deering. *The Java 3D API Specification*. Addison Wesley, 1998.
5. G. Rowe. *Computer Graphics with Java*. Palgrave, 2001.
6. S. Goethel. *OpenGL for Java (formerly gl4Java) Implementation of a Native OpenGl Interface to Java, X-Window and Windows (95 NT)*. Jausoft, 2001.
7. D. Geary. *Graphic Java Mastering the JFC. Volume I: AWT*. Sun Microsystems Press, 3rd Edition, 1999.
8. D. Geary. *Graphic Java Mastering the JFC. Volume II: SWING*. Sun Microsystems Press, 3rd Edition, 1999.
9. B. Eckel. *Thinking in Java*. Prentice Hall, 2nd Edition, 2000.
10. A. Dix, J.Finlay, G.Abowd and R.Beale. *Human-Computer Interaction*. Prentice Hall, 2nd Edition, 1997.
11. S. Zhai, W.Buxton and P. Milgram. *The Partial-Occlusion Effect: Utilizing Semitransparency in 3D Human-Computer Interaction*. ACM Transactions on Computer-Human Interaction, Vol. 3, No. 3, September 1996.
12. R.A. Earnshaw and N. Wiseman. *An Introductory Guide to Scientific Visualization*. Springer-Verlag, 1992.
13. K. Brodie, L.A. Carpenter, R.A. Earnshaw, J.R. Gallop, R.J. Hubbold, A.M. Mumford, C.D. Osland and P. Quarendon. *Scientific Visualization Techniques and Applications*. Springer-Verlag, 1992.
14. C. Ware. *Information Visualization Perception for Design*. Morgan Kaufmann, 2000.
15. J. Nielsen. *Usability Inspection Methods*. ACM Proceedings of CHI 1994, 413-414.
16. B. Shneiderman. *Designing the User Interface. Strategies for Effective Human-Computer Interaction*. Addison Wesley, 3rd Edition, 1998.
17. R. Jeffries, J.R. Miller, C. Wharton and K.M. Uyeda. *User Interface Evaluation in the Real World: A Comparison of Four Techniques*. ACM proceedings of CHI 1991, 119-124.
18. J. Nielsen and R. Molich. *Heuristic Evaluation of User Interface*. ACM Proceedings of CHI 1990, 249-256.
19. R. Molich and J. Nielsen. *Improving a Human-Computer Dialogue*. Communications of the ACM Vol. 33, No. 3, March 1990.

APPENDIX A –SWAN INPUT FILE TEMPLATE

This listing of the SWAN input file template includes all of the available options, although perhaps only a few may be required for any particular implementation.

```
$  PROJect  'name'  'nr'
$         'title1'
$         'title2'
$         'title3'
$
$  MODE    STATionary/NONSTationary     TWODimensional/ONEDimensional
$
$  COORDinates /  -> CARTesian       \
$           \  SPHErical   CCM/QC  /
$
$  SET  [level] [nor] [depmin] [maxmes] [maxerr]     &
$     [grav] [rho] [inrhog] [hsrerr]            &
$     CARTesian/NAUTical  [pwtail] [froudmax]     &
$     [printf] [prtest]
$
$  CGRID  / REGular [xpc] [ypc] [alpc] [xlenc] [ylenc] [mxc] [myc]  \
$        \ CURVilinear [mxc] [myc] ( EXCeption [xexc] [yexc] )     /    &
$        / CIRcle                \
$        \ SECtor [dir1] [dir2] [mdc] /                          &
$          [flow] [fhig] [msc]
$
$  TEST [itest] [itrace]                        &
$        POINTS  IJ < [i] [j] > / XY < [x] [y] >     &
$        (PAR 'fname')  (S1D 'fname')  (S2D 'fname')
$
$  INPgrid   BOTtom/WLEVel/CURrent/VX/VY/FRiction/WInd/WX/WY       &
$    / REGular [xpinp] [ypinp] [alpinp] [mxinp] [myinp] [dxinp] [dyinp] \
$    \ CURVilinear [stagrx] [stagry] [mxinp] [myinp]                  / &
$    (EXCeption [excval] )                                  &
$    (NONSTATionary [tbeginp] [deltinp] Sec/MIn/HR/DAy [tendinp])
$
$  READ   BOTtom/WLevel/CURrent/FRiction/WInd/COORdinates                &
$     [fac] 'fname1'/SERIES 'fname2' [idla] [nhedf] ([nhedt]) (nhedvec]) &
$      / FREe              \
$     <  FORmat  'form'/[idfm]  >
$      \ UNFormatted         /
$
$  WIND  [vel] [dir]
$
$  BOUNdpar1 SHAPespec  JON [gamma] / PM / GAUSS [sigfr] / BIN      &
$     PEAK/MEAN    DSPR POWER/DEGRees
$
$  BOUNdpar2  / SIDE  N/NW/W/SW/S/SE/E/NE   CCW/CLOCKWise \
$           \ SEGMent  XY < [x] [y] > / IJ < [i] [j] >  /       &
$     / CONSTANT PAR [hs] [per] [dir] [dd]          \
$     \ VARIABLE PAR < [len] [hs] [per] [dir] [dd] >  /
$
$  BOUNdspec  / SIDE  N/NW/W/SW/S/SE/E/NE   CCW/CLOCKWise \
$           \ SEGMent  XY < [x] [y] > / IJ < [i] [j] >  /       &
$     / CONSTANT FILE 'fname' [seq]          \
$     \ VARIABLE FILE < [len] 'fname' [seq] >  /
```

```
$
$  BOUNdnest1  NEst 'fname'  CLOSed/OPEN
$
$  BOUNdnest2  WAMNest 'fname' / UNFormatted CRAY/WKstat \
$                    \ FREe              /     &
$      [xgc] [ygc]
$
$  BOUNdnest3  WW3  'fname'  CLOSed/OPEN  ([xgc] [ygc])
$
$         / DEFault
$  INITial  <  ZERO
$         \  PAR [hs] [per] [dir] [dd]
$          \ HOTStart 'fname'
$
$  GEN1  [cf10] [cf20] [cf30] [cf40] [cdmlpm] [cdrag] [umin] [cfpm]
$
$  GEN2  [cf10] [cf20] [cf30] [cf40] [cf50] [cf60]    &
$      [edmlpm] [cdrag] [umin] [cfpm]
$
$  GEN3  / KOMen [cds2] [stpm]    \
$      \ JANSsen  [cds1] [delta] /                    &
$      (QUADrupl [iquad] [lambda] [cnl4] [csh1] [csh2] [csh3])  &
$      (AGROW [a])
$
$  BREaking  CONstant [alpha] [gamma]
$
$         / JONswap [cfjon]
$  FRICtion  <  COLLins [cfw]
$         \ MADsen [kn]
$
$  TRIad   [trfac] [cutfr]
$
$  OBSTacle   (TRANSm [trcoef] / DAM [hgt] [alpha] [beta])          &
$         (REFL  [reflc])   LINE  < [xp] [yp] >
$
$  SETUP  [supcor]
$
$  OFF   BREaking / WCAPping / REFrac / FSHift / QUADrupl / WINDGrowth
$
$  PROP  ??  T / SORDup \  NONST  / STELling [waveage] Sec/MIn/HR/DAy \
$          \ BSBT  /       \ BSBT                    /
$
$  NUMeric   ( ACCUR [drel] [dhabs] [dtabs] [npnts]            &
$           / -> STATionary [mxitst]  \
$           \ NONSTationary [mxitns]  /     [limiter] )     &
$         (DIRimpl [cdd] [cdlim]                    )     &
$         (SIGIMpl [css] [prec] [eps1] [eps2] [outp] [niter])  &
$         (SETUP rpec] [eps2] [outp] [niter])
$
$  FRAme   'sname'  [xpfr] [ypfr] [alpfr] [xlenfr] [ylenfr]      &
$        [mxfr] [myfr] [scale]
$
$  GROUP   'sname'  SUBGrid [ix1] [ix2] [iy1] [iy2]
$
$  NGRid   'sname'  [xpn] [ypn] [alpn] [xlenn] [ylenn] [mxn] [myn]
$
$  CURve   'sname'  [xp1] [yp1]   < [int]  [xp]  [yp] >
```

```
$
$ POINts 'sname'   < [xp]  [yp] > / FILE 'fname'
$
$ RAY    'rname' [xp1] [yp1] [xq1] [yq1]      &
$      < [int]  [xp]  [yp]  [xq]  [vq]  >
$
$ ISOline 'sname'  'rname'  DEPth/BOTtom  [dep]
$
$ LINe    CONTinuous / DAShed [pat] [len]        &
$       (COLor [ipen])  < [xp]  [yp] >
$
$ SITEs   < 'pname'  [xp]  [yp]  [size]  REGion / TOWN >
$
$ QUANTity  DSPR/HSign/DIR/PDIR/TDIR/TM01/RTM01/RTP/TM02/FSPR/PER/RPER/
$         DEPth/VEL/FRCoeff/WIND/DISSip/QB/TRAnsp/FORce/UBOT/URMS/WLEN/
$         STEEpness/HSWEll/DHSign/DRTM01/LEAK/XP/YP/DIST          &
$      'short' 'long' [lexp] [hexp] [excv]                  &
$      ([power])  ([ref])  ([fswell])                       &
$      (PROBLEMcoord/FRAME)
$
$ BLOck   'sname'  HEADer / NOHEADer  'fname' (LAYout [idla])        &
$     < DSPR/HSign/DIR/PDIR/TDIR/TM01/RTM01/RTP/TM02/FSPR/PER/RPER/
$      DEPth/VEL/FRCoeff/WIND/DISSip/QB/TRAnsp/FORce/UBOT/URMS/WLEN/
$      STEEpness/HSWEll/DHSign/DRTM01/LEAK              [unit] >   &
$      (OUTput [tbegblk] [deltblk] Sec/MIn/HR/DAy)
$
$ TABle   'sname'  HEADer / NOHEADer / INDexed 'fname'               &
$     < DSPR/HSign/DIR/PDIR/TDIR/TM01/RTM01/RTP/TM02/FSPR/PER/RPER/
$      DEPth/VEL/FRCoeff/WIND/DISSip/QB/TRAnsp/FORce/UBOT/URMS/WLEN/
$      STEEpness/HSWEll/DHSign/DRTM01/LEAK/XP/YP/DIST       >       &
$      (OUTput [tbegtbl] [delttbl] Sec/MIn/HR/DAy)
$
$ SPECout 'sname'  SPEC1D/SPEC2D  ABs/REl  'fname'                  &
$      (OUTput [tbegspc] [deltspc] Sec/MIn/HR/DAy)
$
$ NESTout 'sname'  'fname'                                  &
$      (OUTput [tbegnst] [deltnst] Sec/MIn/HR/DAy)
$
$ PLOtgeogr  'sname'  (FIle 'fname')  'title'  (COORD [marg])        &
$      ISO  DSPR/HSign/TM01/RTM01/RTP/TM02/FSPR/PER/RPER/DEPTH/
$         FRCoeff/DISSip/QB/UBOT/URMS/WLEN/STEEpness/HSWEll/
$         DHSign/DRTM01/LEAK              [step]  [min]  [max]   &
$      VEC  VEL/TRAnsp/FORce/WIND/DIR/PDIR/TDIR   [scale] [dist]      &
$      (CMESH) [ipen])                                      &
$      (SITes [ipen])  (LINes [ipen])  (LOCation ('sname') [ipen])     &
$      (OUTput [tbegnst] [deltnst] Sec/MIn/HR/DAy)
$
$ PLOtstar  'sname'  (FIle 'fname')  'title'  (COORD [marg])        &
$      ISO  DSPR/HSign/TM01/RTM01/RTP/TM02/FSPR/PER/RPER/DEPTH/
$         FRCoeff/DISSip/QB/UBOT/URMS/WLEN/STEEpness/HSWEll/
$         DHSign/DRTM01/LEAK              [step]  [min]  [max]   &
$      STAR   [scale] [dist] [bundle]                       &
$      (CMESH) [ipen])                                      &
$      (SITes [ipen])  (LINes [ipen])  (LOCation ('sname') [ipen])     &
$      (OUTput [tbegnst] [deltnst] Sec/MIn/HR/DAy)
$
$ PLOtpp  'sname'  FIle 'fname' 'title'                      &
```

```
$       PROBlems FROUDe / PCONV   |symsiz|                    &
$       (SITes |ipen|)  (LINes |ipen|)  (LOCation ('sname') |ipen|)    &
$       (OUTput |tbegnst| |deltnst| Sec/MIn/HR/DAy)
$
$  PLOtspec 'sname'  (FIle  'fname')   'title'                 &
$       SPECtrum  NORMalized / NONORM   ( < |ch| > )              &
$       FReq  NORm / |fmax|  |fmid|     ABS/REL                 &
$       (OUTput |tbegnst| |deltnst| Sec/MIn/HR/DAy)
$
$  COMPUTE
$  COMPUTE  STATionary |time|
$  COMPUTE  NONSTat |tbegc| |deltc| Sec/MIn/HR/DAy |tendc|)
$
$  HOTFile  'fname'
$
$  POOL
$
$  STOP
```

APPENDIX B – SAMPLE SWAN OUTPUT FILE

Run:1 Table:Table1

Xp [m]	Yp [m]	Depth [m]	Hsig [m]	Tm01 [sec]	PkDir [degr]	Tpeak [sec]
602650.	6220250.	8.6338	1.08139	17.3403	91.000	18.8671
602675.	6220250.	8.7832	1.19331	17.3313	91.000	18.8671
602700.	6220250.	8.8751	1.29885	17.3244	91.000	18.8671
602725.	6220250.	8.9375	1.34221	17.3134	91.000	18.8671
602750.	6220250.	9.0764	1.35666	17.2905	91.000	18.8671
602775.	6220250.	9.4615	1.33140	17.2487	91.000	18.8671
602800.	6220250.	9.8290	1.28761	17.1954	91.000	18.8671
602825.	6220250.	10.1464	1.18289	17.1088	91.000	18.8671
602850.	6220250.	10.4718	1.06951	16.9922	91.000	18.8671
602875.	6220250.	11.0622	.97534	16.8556	91.000	18.8671
602900.	6220250.	11.4013	.97240	16.8160	91.000	18.8671
602925.	6220250.	11.5606	.99008	16.8231	91.000	18.8671
602950.	6220250.	11.7387	.99404	16.8116	91.000	18.8671
602975.	6220250.	11.9225	.98865	16.7867	91.000	18.8671
603000.	6220250.	12.0255	.97684	16.7650	91.000	18.8671
603025.	6220250.	12.0690	.96976	16.7578	91.000	18.8671
603050.	6220250.	12.2904	.95122	16.7398	89.000	18.8671
603075.	6220250.	12.4300	.94564	16.7200	89.000	18.8671
603100.	6220250.	12.5977	.94019	16.6941	89.000	18.8671
603125.	6220250.	12.7782	.93999	16.6781	89.000	18.8671
603150.	6220250.	12.9344	.94114	16.6573	89.000	18.8671
603175.	6220250.	13.0410	.94124	16.6374	89.000	18.8671
603200.	6220250.	13.1106	.94101	16.6136	89.000	18.8671
603225.	6220250.	13.0920	.94230	16.5932	89.000	18.8671
603250.	6220250.	13.1309	.93750	16.5764	89.000	18.8671
603275.	6220250.	13.1694	.93302	16.5551	89.000	18.8671
603300.	6220250.	13.3680	.92573	16.5222	89.000	18.8671
603325.	6220250.	13.4741	.92471	16.4860	89.000	18.8671
603350.	6220250.	13.5745	.92252	16.4435	89.000	18.8671
603375.	6220250.	13.6858	.91998	16.3928	89.000	18.8671
603400.	6220250.	13.8032	.91749	16.3410	89.000	18.8671
603425.	6220250.	13.8215	.92067	16.2988	89.000	18.8671
603450.	6220250.	13.9925	.92186	16.2518	89.000	18.8671
603475.	6220250.	13.9946	.92902	16.2257	89.000	18.8671
603500.	6220250.	13.9932	.93433	16.1949	89.000	18.8671
603525.	6220250.	13.9915	.94540	16.1863	89.000	18.8671
603550.	6220250.	13.9914	.95703	16.1781	89.000	18.8671
603575.	6220250.	13.9885	.97101	16.1787	89.000	18.8671
603600.	6220250.	14.0396	.97986	16.1726	89.000	18.8671
603625.	6220250.	14.1254	.99523	16.1916	89.000	18.8671

..... numerical data continues .....

APPENDIX C – EVALUATION PACK

HEURISTIC EVALUATION OF SWAN VISUALIZATION PROGRAM

The goal of this evaluation process is to discover any usability problems that may exist within the design of the Swan user interface. Please use the following list of heuristics as a guideline, and make comments where you feel the usability of the system can be improved (not limited only to those points listed below).

The list of heuristics were originally developed by Molich and Nielsen [A] and later refined to the following: [B]

1. Visibility of system status.
The system should provide sufficient feedback so that users are aware of what is happening.

2. Match between system and the real world.
Information should be displayed in a logical format that is meaningful to the users.

3. User control and freedom.
Users are in control of the system, and need to be able to recover from unwanted states without too much difficulty.

4. Consistency and standards.
Once users have come across particular words, situations or actions, they should feel comfortable that similar standards have been applied across the system

5. Error prevention.
Measures should be put in place to prevent avoidable errors from happening.

6. Recognition rather than recall.
The user should not need to memorise system actions, but clearly presented options should be made available. Assistance in the form of clearly presented instructions should be made available when necessary.

7. Flexibility and efficiency of use.
Accelerators and shortcut keys should be made available and provision should be made for different methods of navigation and option selections. This can improve the interaction for more experienced users.

8. Aesthetic and minimalist design.
Only relevant information should be presented. Additional detail should be kept to a minimum, so as not to detract from the essential data.

9. Help users recognize, diagnose, and recover from errors.
Error messages should be displayed using plain language, and explanations should be offered as to the reason for the problem occurring. Possible solutions should be offered to assist the user to recover from an exceptional state.

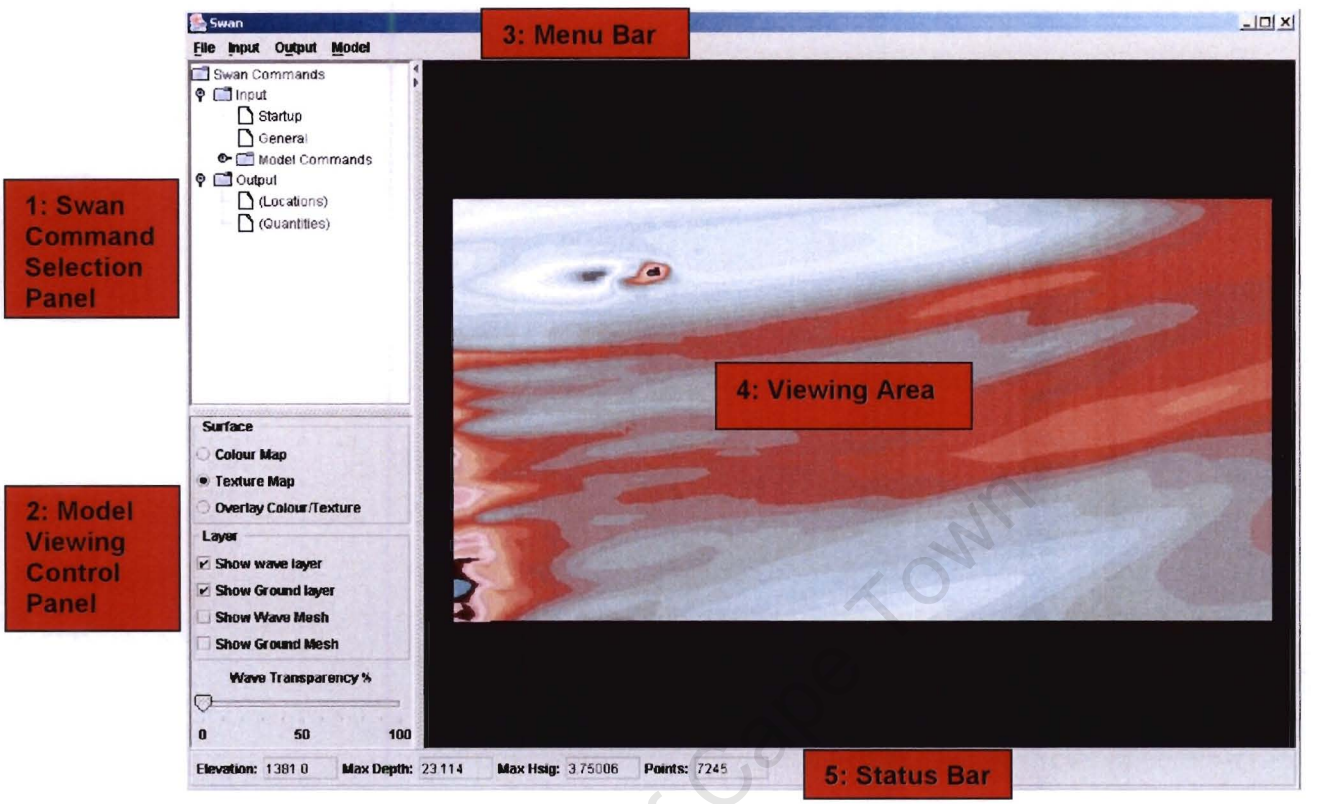
10. Help and documentation.
Documentation that is provided should be helpful and easy to understand by the users. Information should be focused and not too verbose.

References:

[A] R. Molich and J. Nielsen. *Improving a Human-Computer Dialogue*. Communications of the ACM Vol. 33, No. 3, March 1990.
[B] A. Dix, J.Finlay, G.Abowd and R.Beale. *Human-Computer Interaction*. Prentice Hall, 2nd Edition, 1997.

USER GUIDE

Graphical User Interface Description



Graphical Interface components description

1. Swan Command Selection Panel – This panel is used to select the relevant data entry group to be displayed in the Viewing Area. The input and output commands have been arranged in a tree-like structure to aid the navigability of the command categories.

2. Model Viewing Control Panel – This panel is used to change the model viewing parameters. Different selection components have been used according to their functional requirements.

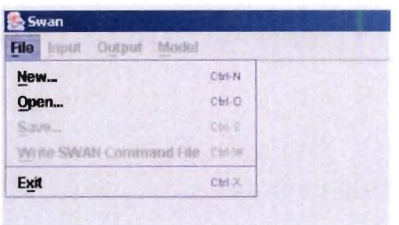
3. Menu Bar – This is a standard drop-down menu bar from where program options can be selected.

4. Viewing Area – This is the main display area that will display the 3-D model, coordinate and detail table, or a 2-D graphical interface for data entry.

5. Status Bar – This panel displays information about the model that has been loaded from file.

PROGRAM OPERATION

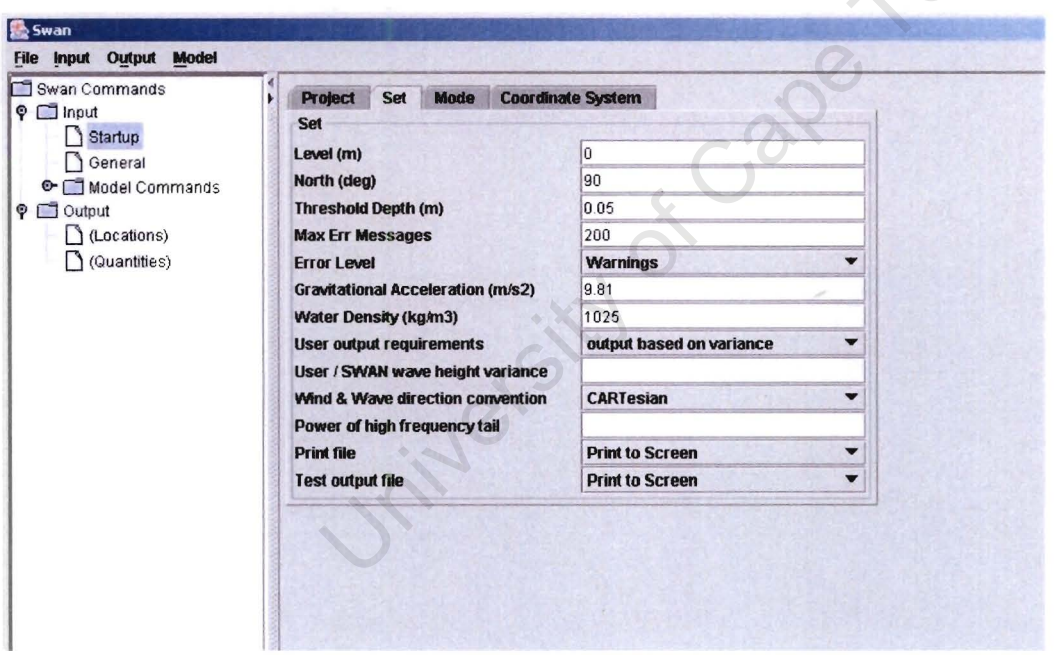
1. Starting the program



When the program is first run, the user needs to select to start a new workspace, or open a set of Swan command panels that have previously been saved to disk. The other menus and menu items will not be activated until the "new" or "open" procedure has been completed.
Shortcut keys: New <ctrl> N
Open <ctrl> O

2. Entering Swan Commands

The relevant Swan command group can be selected either from the Swan Command navigational tree on the left, or from the input and output drop-down menu options at the top of the display.
Once the desired command group has been selected, the command sets will be displayed in the main viewing panel in the centre of the display.



The command panels are arranged according to their category, and these are displayed as tabbed panes that can be selected by clicking the relevant tab heading with the mouse. Values are entered by either typing values in the spaces provided or by making selections from the combo box options. Default values have been set, where applicable.

Text fields that require numerical input have specific *key listeners* assigned to make sure that only valid characters can be entered. These key listeners check each character as it is typed, and only allow integer or floating point numbers to be entered, as appropriate. In some cases numbers are also checked to make sure that they fall within a required range.

Those Swan command groups that have not yet been implemented are indicated in the menu bar as greyed-out items and in the command tree as items between parentheses.

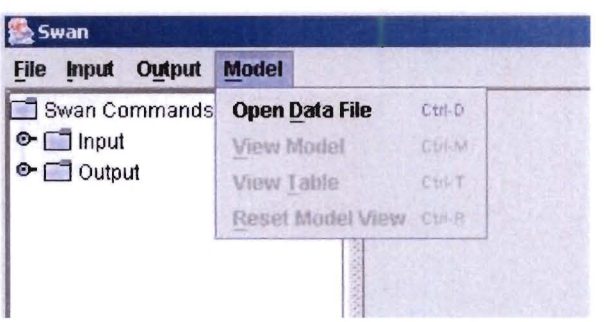
3. Saving Swan Commands

Swan commands can be saved in two formats; as a text file to be used for Swan computations, and as a binary file to capture the state of each of the Swan command panels for later retrieval.

To write the Swan commands to a text file, select "Write SWAN Command file" from the File menu or use the shortcut <ctrl> W. A file save dialogue box will be displayed where the name of the file to be saved can be entered.

To save the state of the Swan command panels for later retrieval, select "Save" from the File menu or use the shortcut <ctrl> S. A file save dialogue box will be displayed where the name of the file to be saved can be entered.

4. Opening a Model File

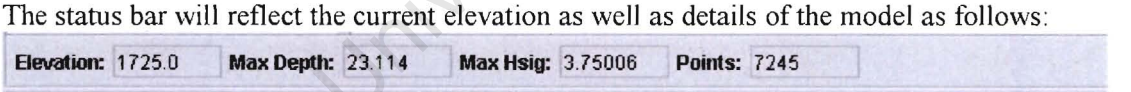


The model file to be opened is an output text file that has been generated by the SWAN computational program.
To open the model data file, select "Open Data File" from the "Model" menu, or use the shortcut <ctrl> D. A file chooser dialogue box will be displayed from which the relevant file can be browsed and selected.
Once the file has been successfully opened, the model viewing menu options will be activated, and the 3-D graphical model will be displayed in the main viewing panel in the centre of the display.

5. Viewing the model

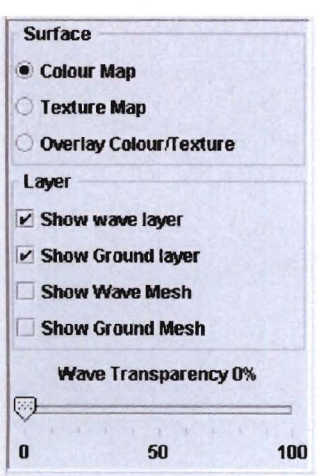
Once the model file has been loaded into memory, the surfaces will be generated and displayed on the screen. The initial view shows the model from the top (Z direction) from an elevation that can accommodate the whole model in the viewport. This view can be returned to at any time by selecting "Reset Model View" from the "Model" menu, or using the shortcut <ctrl> R.

The status bar will reflect the current elevation as well as details of the model as follows:



Elevation – Current distance from the model in the Z-direction
Max Depth– Maximum sub aqueous ground depth
Max HSig – Maximum significant wave height
Points – Number of coordinate points in the model

The model control panel will now be visible on the bottom left hand side of the display, as follows:



Surface appearance options
Toggle between options.

Layer control options
Select any of the desired layers to be shown.

Wave transparency Control
Slide the control to set the Wave transparency percentage.

The different surface rendering options have the following characteristics:

Ground layer colour Map: High points are shaded with a yellow colour, and low points are indicated with red. The darker shades represent deeper areas. There are ten equally graded transitional levels from high (yellow) to low (red).

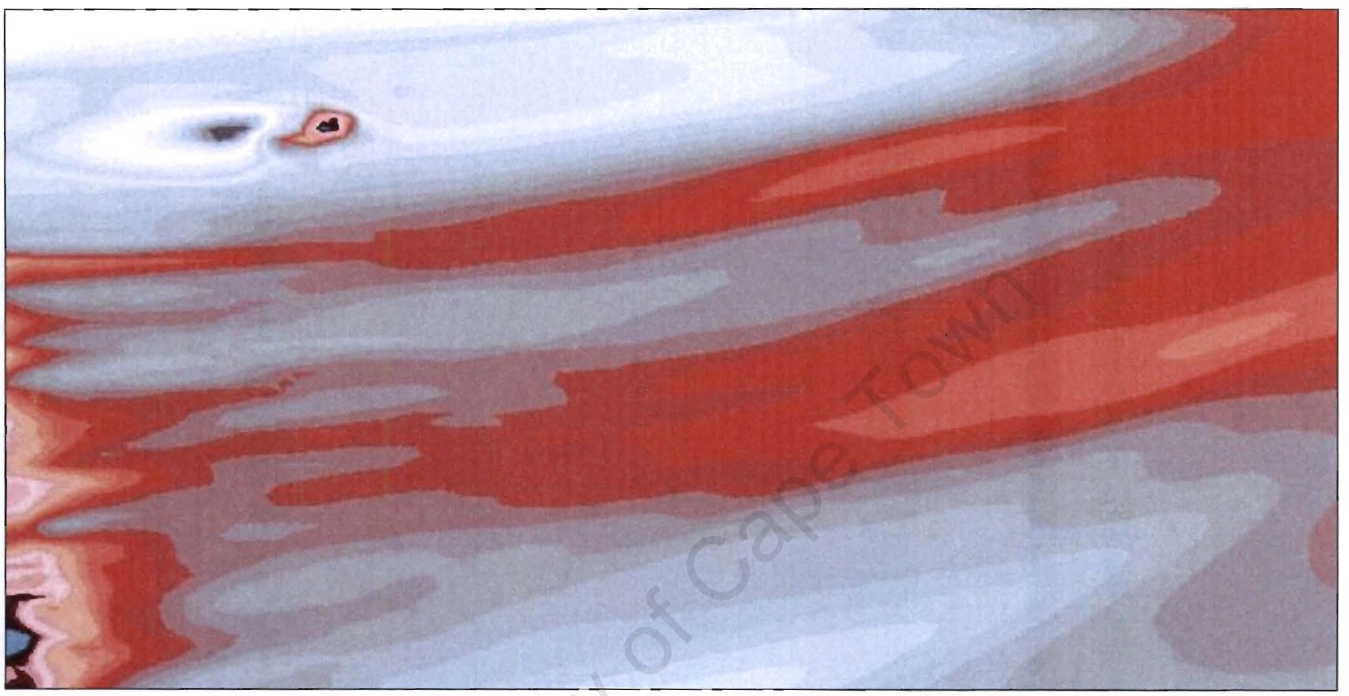
Wave layer colour Map: The colours vary from light blue to dark blue depending on the significant wave height at each point. There are ten equally graded transitional levels from shallow (light blue) to deep (dark blue).

The colour map option is used to identify high and low areas in the model.

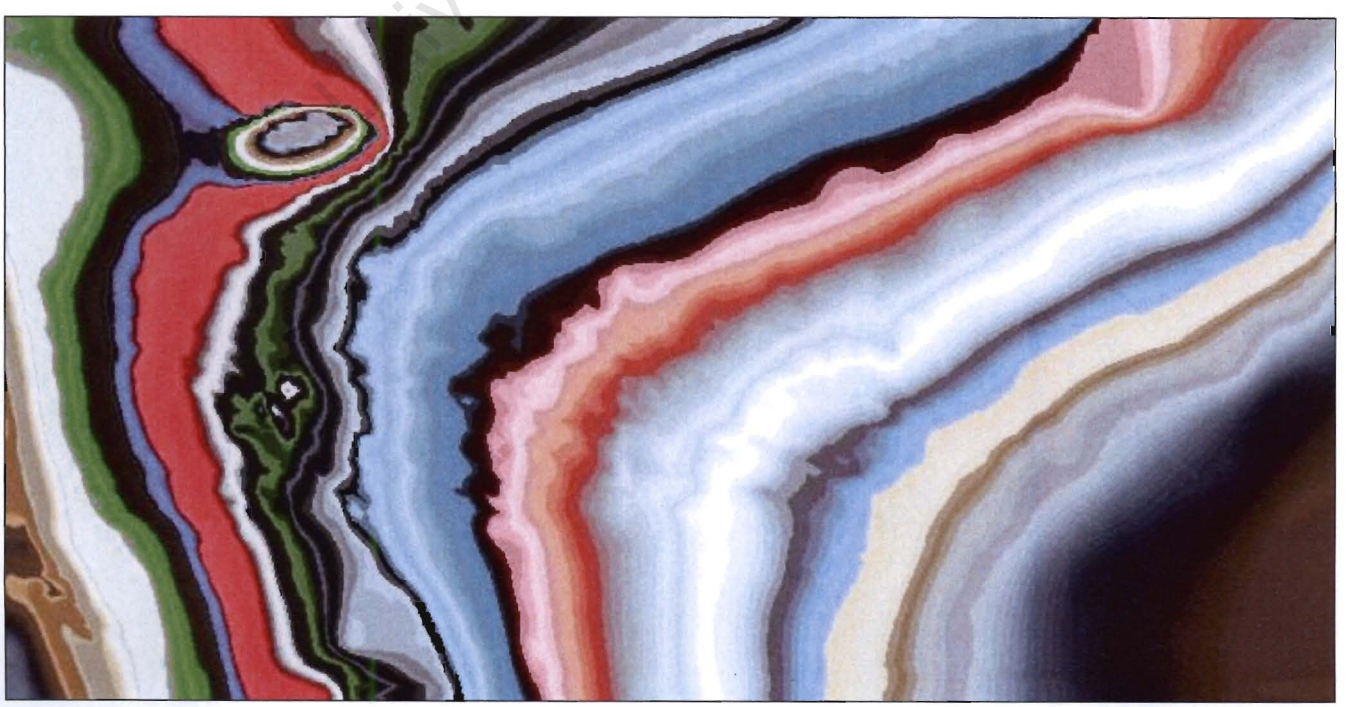
Texture Map: This surface rendering option is used to view the different depth regions in more detail. The ground and wave depths are mapped to colours stored in an image file on disk.
This view is not used to identify high and low points in the model, but rather to view wave and ground depth patterns and formations. The texture map option may be more useful when using the transparency tool to view the sub aqueous ground detail through the wave layer.

Following are sample views of the Wave and Ground layers where the texture map option has been selected.

Wave Layer with Texture Map surface

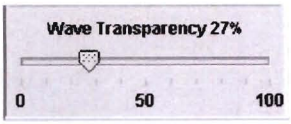


Ground Layer with Texture Map surface



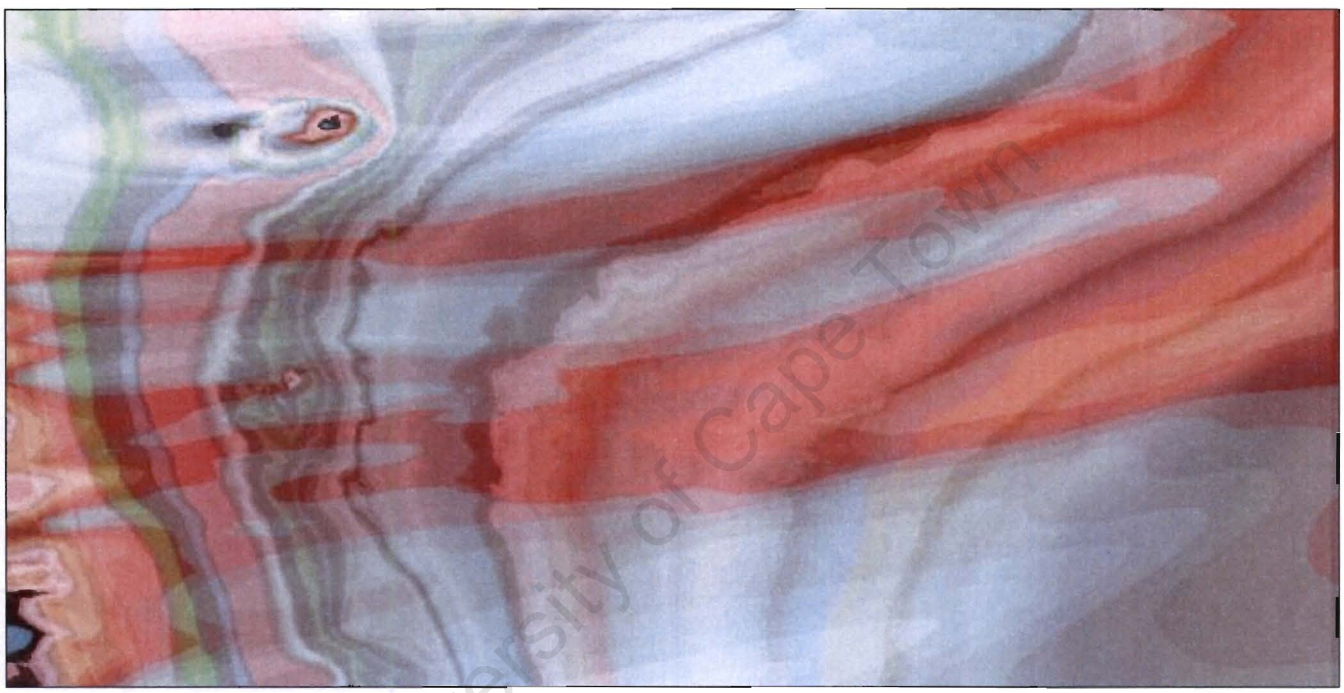
Wave Transparency control

The wave transparency is adjusted by sliding the transparency control slider to reflect the desired percentage transparency:



0% means that the waves are opaque
100% means that the waves are fully translucent
Any value between 0 and 100 sets the transparency to that percentage.

Following is an example of viewing the wave and ground layers together using texture mapping and wave transparency set to 25%:

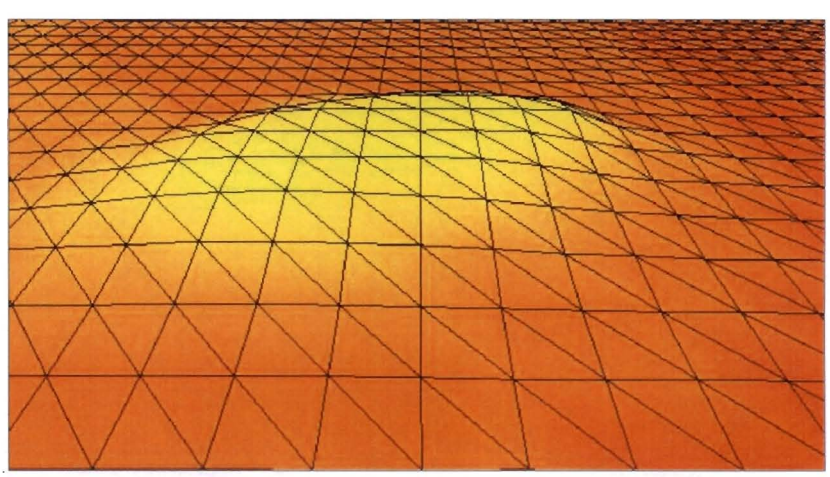


The transparency tool is used to identify sub aqueous ground features that could be causing the formation of certain wave patterns to emerge. This tool allows the ground and wave layers to be overlaid and viewed together so that any relationships between the surfaces can be identified.

Applying a mesh to the view

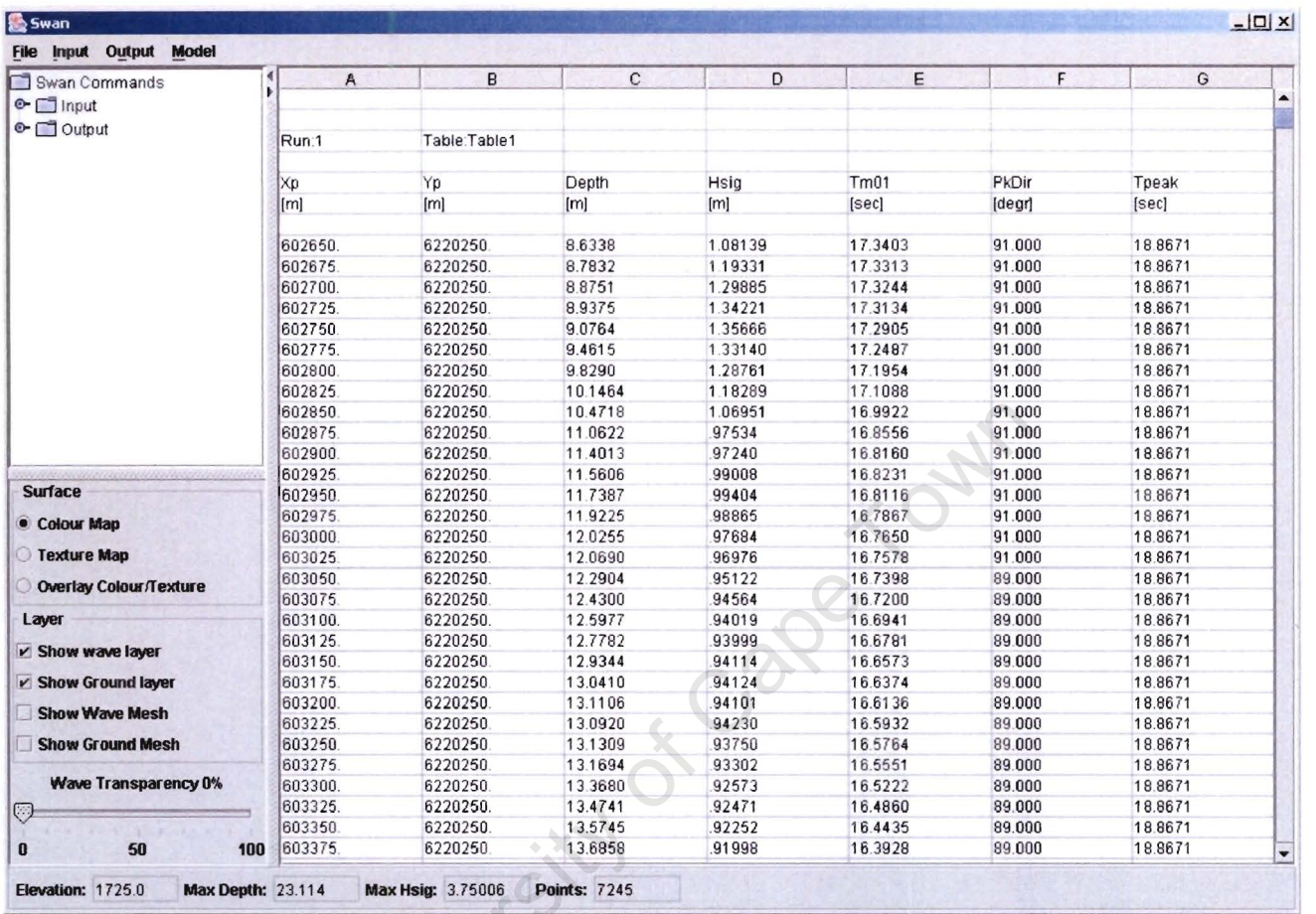
Both ground and wave wire-frame meshes can be viewed by selecting the relevant mesh layer(s) from the model viewing control panel.

This sample view shows the ground layer and ground mesh viewed together. The model view has been rotated and zoomed to highlight the feature identified.



6. Viewing the Model Table

The model table contains coordinate points and details from the Swan output file used to generate the 3-D model. This table can be viewed by selecting "View Table" from the "Model" menu, or using the shortcut <ctrl> T.



A	B	C	D	E	F	G
Run:1	Table:Table1					
Xp [m]	Yp [m]	Depth [m]	Hsig [m]	Tm01 [sec]	PkDir [degr]	Tpeak [sec]
602650.	6220250.	8.6338	1.08139	17.3403	91.000	18.8671
602675.	6220250.	8.7832	1.19331	17.3313	91.000	18.8671
602700.	6220250.	8.8751	1.29885	17.3244	91.000	18.8671
602725.	6220250.	8.9375	1.34221	17.3134	91.000	18.8671
602750.	6220250.	9.0764	1.35666	17.2905	91.000	18.8671
602775.	6220250.	9.4615	1.33140	17.2487	91.000	18.8671
602800.	6220250.	9.8290	1.28761	17.1954	91.000	18.8671
602825.	6220250.	10.1464	1.18289	17.1088	91.000	18.8671
602850.	6220250.	10.4718	1.06951	16.9922	91.000	18.8671
602875.	6220250.	11.0622	.97534	16.8556	91.000	18.8671
602900.	6220250.	11.4013	.97240	16.8160	91.000	18.8671
602925.	6220250.	11.5606	.99008	16.8231	91.000	18.8671
602950.	6220250.	11.7387	.99404	16.8116	91.000	18.8671
602975.	6220250.	11.9225	.98865	16.7867	91.000	18.8671
603000.	6220250.	12.0255	.97684	16.7650	91.000	18.8671
603025.	6220250.	12.0690	.96976	16.7578	91.000	18.8671
603050.	6220250.	12.2904	.95122	16.7398	89.000	18.8671
603075.	6220250.	12.4300	.94564	16.7200	89.000	18.8671
603100.	6220250.	12.5977	.94019	16.6941	89.000	18.8671
603125.	6220250.	12.7782	.93999	16.6781	89.000	18.8671
603150.	6220250.	12.9344	.94114	16.6573	89.000	18.8671
603175.	6220250.	13.0410	.94124	16.6374	89.000	18.8671
603200.	6220250.	13.1106	.94101	16.6136	89.000	18.8671
603225.	6220250.	13.0920	.94230	16.5932	89.000	18.8671
603250.	6220250.	13.1309	.93750	16.5764	89.000	18.8671
603275.	6220250.	13.1694	.93302	16.5551	89.000	18.8671
603300.	6220250.	13.3680	.92573	16.5222	89.000	18.8671
603325.	6220250.	13.4741	.92471	16.4860	89.000	18.8671
603350.	6220250.	13.5745	.92252	16.4435	89.000	18.8671
603375.	6220250.	13.6858	.91998	16.3928	89.000	18.8671

7. Navigating the 3-D model

The mouse can be used to adjust the zoom factor (elevation), rotate the view and move the viewpoint to a particular part of the model as follows: (the mouse pointer should be located inside the viewing window before continuing)

Zooming out (increasing the elevation)
Option 1: Whilst pressing the <alt> key, press the left mouse button and move the mouse pointer upwards.
Option 2: Using the scroll button on the mouse, rotate the button away from you. This will adjust the elevation level by 1 unit. To accelerate the zoom factor, press the <shift> key whilst turning the scroll button.

Zooming in (decreasing the elevation)
Option 1: Whilst pressing the <alt> key, press the left mouse button and move the mouse pointer downwards.
Option 2: Using the scroll button on the mouse, rotate the button towards you. This will adjust the elevation level by 1 unit. To accelerate the zoom factor, press the <shift> key whilst turning the scroll button.

Panning
Whilst pressing the <shift> key, press the left mouse button and move the mouse in the direction that you would like to move. To accelerate the pan, press the <shift> and <alt> keys together

Rotating the view
Press the left mouse button and move the mouse in the desired direction to rotate.

APPENDIX D – EVALUATION RESULTS

USER 1

HEURISTIC EVALUATION OF SWAN VISUALIZATION PROGRAM

COMMENTS:

Guideline Number	Comment.
1.	When selecting "Save" or "Write SWAN Command File", it would make the selection clearer if the required file type (extension) was displayed in the file name box.
3.	Rotating, panning & zooming is a bit difficult to control. An "Undo" command would be useful to return to the previous view.
6.	It is difficult to remember which keys should be pressed for panning, rotating, zooming, etc. A set of buttons on the toolbar would help.

A. R. LLOYD 11-9-03.

USER 2

HEURISTIC EVALUATION OF SWAN VISUALIZATION PROGRAM

COMMENTS:

1. Visibility of system status
   Good.

2. Match between system and real world
   Good

3. User control and freedom
   Good

4. Consistency and standards
   Good

5. Error prevention
   Good.

6. Recognition rather than recall
   Good recognition

7. Flexibility and efficiency of use
   Good

8. Aesthetic and minimalist design
   Good

9. Help users recognise, diagnose and recover from errors
   Acceptable, Good, but difficult to generate an error

10. Help and documentation
    Help and documentation needs to be added

USER 3

HEURISTIC EVALUATION OF SWAN VISUALIZATION PROGRAM

COMMENTS:

1. When opening a data file - should only list 'file types' that can be opened in the program.

2. View Table: The header lines 'xp Yp, depth etc should remain visible when scrolling down the table

3. Commands are not visible as "buttons" on the screen. eg zoom/pan etc..

4. Program closes When hitting ☒ the program closes with prompting to Save.

5. option: When passing the ~~middle~~ key to "Pan a view" the Icon could change to a ✋ hand symbol?

6. option: When moving the mouse over the model - the elevation/depth could reflect current position? — possibly not possible at this stage.

7. Pressing middle button —
   Left + Right buttons have same function in panning.